INTRODUCTION TO THE GRAPHICAL KERNEL SYSTEM (GKS)

P. Palamidese

Rapporto interno C85-13

Introduction to the Graphical Kernel System

Patrizia Palamidese

October 1984

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

The following presentation refers to the document TC97/SC5/WG2 N. 262 that is the final GKS document for ISO Central Segretariat.

At this time GKS has achieved the status of "Draft International Standard" (DIS) and it will not be subject to further technical change before becoming an "International Standard" (IS), the path to which is a formal process.

The Graphical Kernel System (GKS) specifies a set of basic graphics functions that can be used by all applications that produce computer generated pictures. The main goals of a standard in basic computer graphics are:

a. to allow graphics programs to be easily portable between different installations

b. to define a methodology for graphics programming and therefore facilitate programmers

c. to guide manufacturers of graphics equipments in developing best graphics devices

GKS defines functions to generate two dimensional pictures for vector and raster devices, storage tube displays, refresh displays and colour displays.

A fundamental concept in GKS is the workstation that consists of some input devices and one output device. The user can work simultaneously on several workstations and adapt his program to the characteristics of a workstation which are described in the "workstation description table". A workstation can have more intelligent capabilities than those supportable by GKS. In this case, "GENERALIZED DRAWING PRIMITIVES" and "ESCAPE" functions are provided to access these non-standard features (GDP is described in next section).

GKS has also the functions to memorize on and retrieval pictures from a file, the so-called "metafile", in order to transport pictures between different applications and/or devices. The specification of the format and content of the "metafile" is not part of GKS definition but it is still material of discussion into ISO/TC97/SC21/WG2.

2.0 OUTPUT FUNCTIONS

GKS standardizes functions that can be used to draw on any kind of devices, from simple passive plotters to highly interactive and intelligent workstations.

Not every GKS implementation has to support the full set of functions. For this reason some levels are defined that can be implemented separately.

A GKS implementation is therefore characterized by an "output level" and an "input level".

"Output levels" to be used for this work can be "output level 0: minimal ouput" or "output level 1: basic segmentation with full output", where the main difference between the two levels is that the first doesn't implement segmentation.

We can see now how graphical output is generated by GKS. Basic elements are "output primitives" and "primitive attributes". The former perform actions such as drawing lines and writing characters strings, the latter control the aspect of the output such as "linestyle", "character height" and "colour". These attributes are bound to a primitive and cannot be changed afterwards. The only way to do this is to recreate the primitive with different attributes.

We can use POLYLINE to connect several points, POLYMARKER to generates symbols at given positions, TEXT to write a character string at a given position, FILL to fill a polygonal area with uniform colour or with a hatch style, CELL to generate an array of pixels with individual colours.

The way the "FILL primitive" works depends on the value of the "interior style attribute". We consider some most interesting cases:

a. no filling, but draw the boundary polyline using the colour currently selected

b. fill the interior of the polygon using the colour selected (no boundary)

c. fill the interior of the polygon using an hatch style and the colour selected

It should be noted that the last point is always connected to the first point in order to create a closed area.

The interior of a polygon is defined in the following way (See Fig. 1 pag. 4):

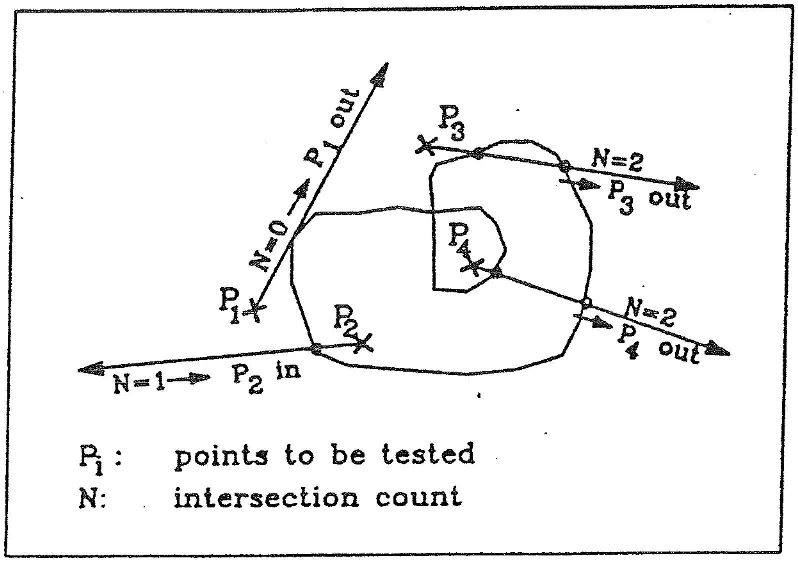


Fig. 1. Area inside a polygon

"For a given point, create a straight line starting at that point and going to infinity. If the number of intersactions between the straight line and the polygon is odd, the point is within the polygon; otherwise it is outside. If the straignt line passes the polygon vertex tangentially, the intersection count is not affected."

What about PICK? We must distinguish among the cases above mentioned. In the case a. only the boundary is detectable, in the case b. any point inside the polygon is detectable, in the case c. any point on any hatch line can be indicated. A hole in the area doesn't identify that area.

GKS defines another kind of output primitive, the GENERALIZED DRAWING PRIMITIVE (GDP) that serves to address in a standard format some special output capabilities of a workstation, such as drawing of arcs, interpoleted curves, ellipses. In this way the application can deliver a set of points to GKS. The interpretation is workstation dependent.

3.0 COORDINATE SYSTEMS AND TRANSFORMATIONS

An application using GKS can have its own coordinate system named "user system" that is not always a cartesian system, as illustrated in Fig. 2. Since GKS understands only cartesian coordinate systems,it is application's responsability to transform the objects to be drawn into a cartesian coordinate system, the "world coordinate system"(WC).

The WC are mapped into a "normalized coordinate space" (NDC) that has value in the range 0-1. A "normalization transformation" (Fig. 3 pag. 6) is defined specifying a "window" in WC that must be mapped into a "viewport" in NDC (Fig. 4 pag. 6).

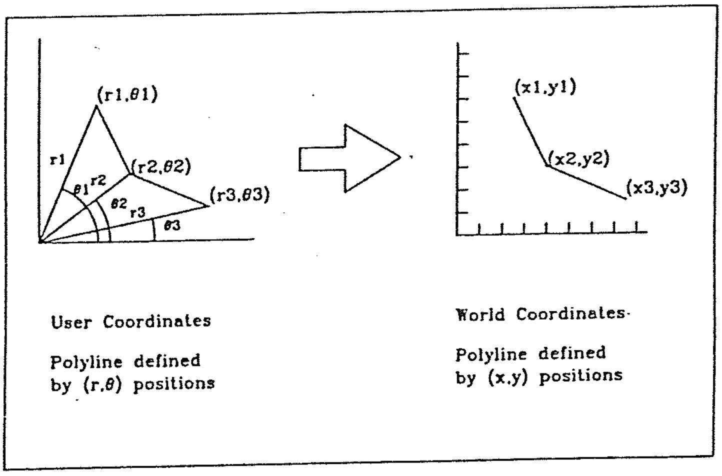


Fig. 2. User coordinates and world coordinates

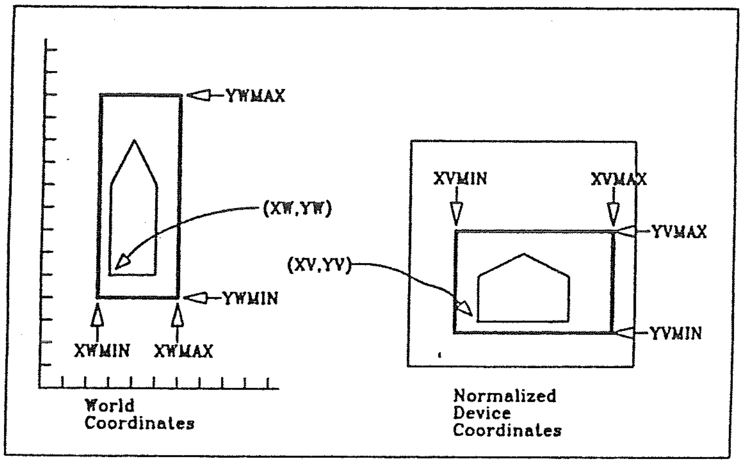


Fig. 3. Normalization transformation

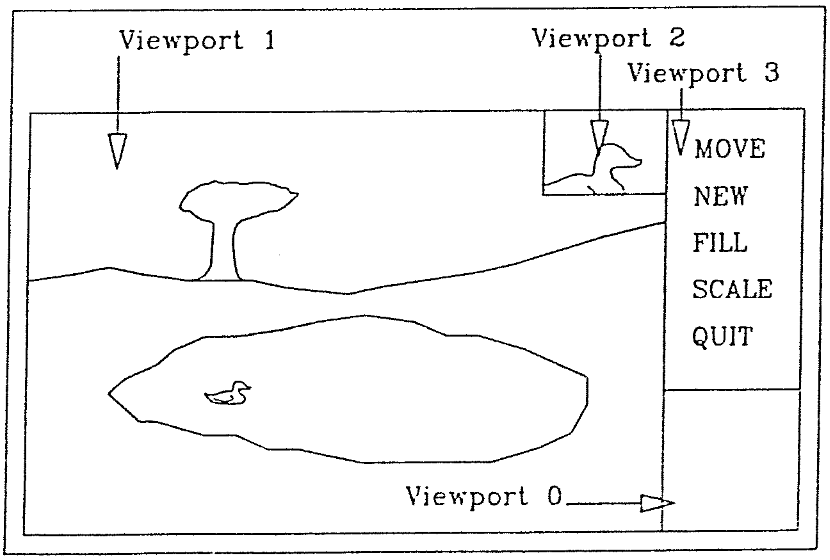


Fig. 4. Using several viewports

4.0 SEGMENTS

Many functions are provided to help the user to perform highly interactive programming. The picture can be described as a collection of "segments" (Fig. 5).

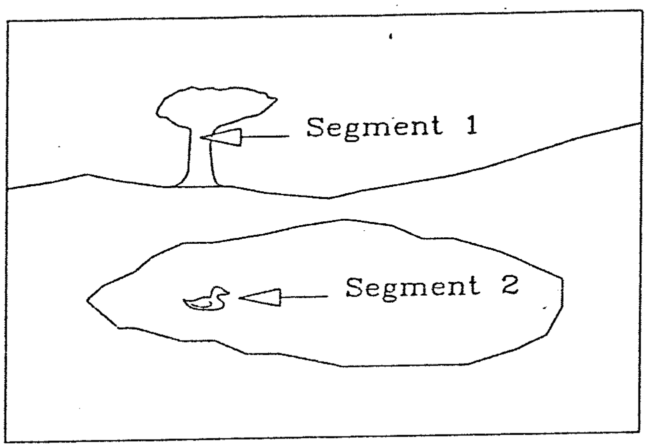


Fig. 5. Defining several segments

Each "segment" is a graphics entity that can be manipulated as a unit: it can be rotated, translated, scaled, can be made visible or invisible, highlited, deleted, renamed, ordered, made detectable or undetectable (that is the user can "pick" it or not). The user can define a transformation matrix (at any time even if the segment has been already created) and associate it to a segment. The segment transformation takes place in NDC space. It doesn't affect the content of the segment and it is not cumulative, i.e. it always applies to the segment as originally created. Therefore applying the same transformation twice to a segment gives identical results.

Segment modifications don't impact directly on the model which belongs to user's world. Each change is lost when GKS is closed. It is user's responsability to save a picture via "metafile" or updating the model.

How can be seen there is a limited number of actions that can be performed on the "segments" and it should be noted that there are many other attributes ("primitive attributes" described above) that cannot be modified after the segment has been created.

Segments can be affected by "segment priority" but "this feature is intended to address appropriate hardware capabilities only. It cannot be used to force software checking of interference between segments on non-raster displays". The "segment priority" is used for picking segments.

5.0 INPUT FUNCTIONS

We can see now breafly some concepts about input functions and levels.

During this work we will deal with "input level b: request input", that is the minimal level to have any kind of input, since the inferior "level a" means "no input" at all.

When in "request mode", GKS waits until the input is entered by the operator or a breack action is performed by the operator. How "request mode" operates can be seen in Fig. 6 where "trigger" means physical device.

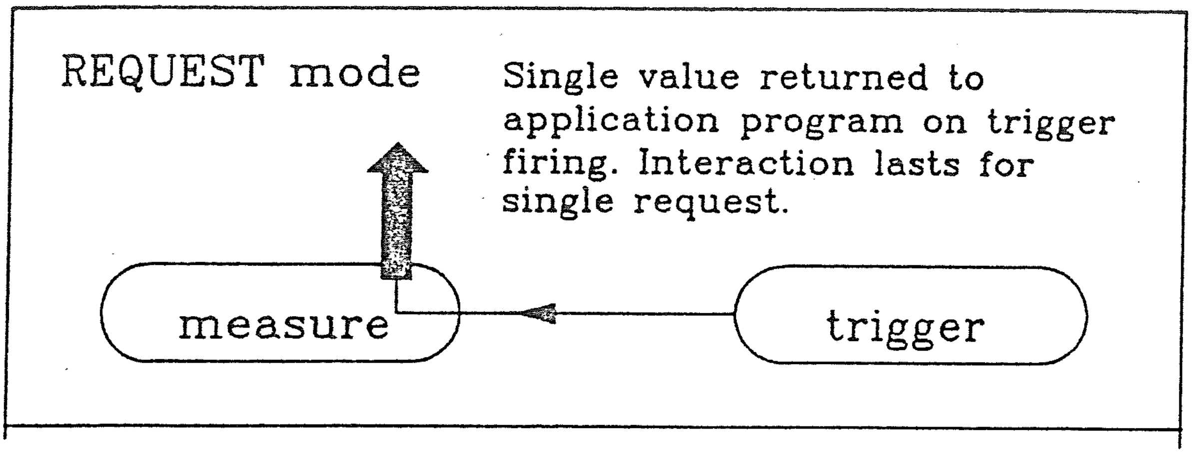


Fig. 6. The relationship between the measure and trigger

GKS defines six logical devices independently of the physical device they are related to.

Each logical input delivers one or more "measures" to the application.

A LOCATOR measure consists of a position in WC and a normalization transformation number (a LOCATOR can be a joystick). The logical input device identifies which viewport the position in NDC lies within and uses this to select the normalization transformation. When some viewports overlap, GKS can take a decision if a "viewport input priority" has been defined previously.

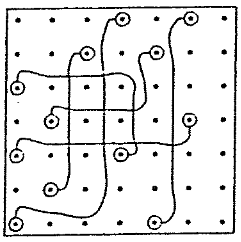
A STROKE measure consists of a sequence of points in WC and a normalization transformation number (a typical stroke can be a digitizer).

A VALUATOR measure provides a real number (it is usually an angle, a length or any type of dimension).

A CHOICE measure provides a value that is an integer in the range 0 and a device dependent maximum (it can be a PF key).

A STRING measure provides a character string with a device dependent maximum length (it is usually a keyboard).

A PICK measure provides a segment name and an output primitive identifier (Fig. 7).



Using pick identifier to identify one of many connections

Fig. 7. Using PICK

As can be seen PICK input can be used only with a GKS implementation that supports segment definition. An output primitive can be picked if an identifier has been associated to it when it was created.

6.0 FIRST ATTEMPT OF DESCRIBING GKS DATA TYPES

simple type

I integer	whole number
R real	floating number
S string	number of character
P point	2 real values specifying the x-y-coordinates of a location in WC, NDC or DC space
N name	identification of a file, a procedure, a segment or a number of primitives (via PICK IDENTIFIER)
E enumeration	a data type comprising a set of values

examples

POLYLINE	(number of points, points)	I ; nxP
TEXT	(text position, character string)	P ; S
SET CHARACTER HEIGHT	(character height)	R
SET CHARACTER UP VECTOR	(char up vector)	2xR
SET PICK IDENTIFIER	(pick identifier)	N
SET VISIBILITY	(segment name, visibility)	N ; E

combination of simple types (the following list contains only examples)

2xR	vector of values
2x3xR	matrix of values
I;E	ordered pair of different types

1.0 TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

6.0 SEMANTICS AND SYNTAX

GKS standard is described in a language independent manner that is in term of abstract functions and data types. Nowadays many efforts are put into defining natural and efficient interfaces between the GKS abstract description and some host languages.

GKS abstract functions and data types which are only tools for describing the semantics of the standard, must be mapped into language functions and actual data types in a way conform to the restrictions of the host language.

One of the most advanced proposal of GKS language binding is contained in the document ISO/TC97/SC5/WG2 N.250 1984 August 7 under the title "Information Processing Systems, Computer Graphics, Graphical Kernel System (GKS) Language Bindings, Part2: Pascal".

We will describe the semantics and the syntax of GKS functions in term of this specification.

GKS data structures are represented by Pascal data structures with the following naming conventions:

types	prefix in Pascal
enumerated and scalar	GT
set	GS
arrays	GA
records	GR
pointer	GPT
constants	GC
procedure	G
global variable	GV

6.1 TYPES APPLICABLE TO GKS DATA STRUCTURES

The following types are applicable to data structures required by the GKS functions explained in the successive paragraphs.

```
GTerrorfile
         = (StdOutput,Othernames);

GTWsId   = INTEGER;

GTWsConnection
         = (WsCon1,WsCon2,WsCon3,WsCon4) ;
           (the number of connection is
           implementation dependent)

GTWsType
```

```
           = (SegstoreWs,MOWs,MIWs,WsType1);
             (Workstation Independent Segment Storage, Metafile
             Output, Metafile Input, any other type)

GTcontrol= (ClearCond,ClearAlways);

GTint0    = 0..MAXINT;

GTint1    = 1..MAXINT;

GTint2    = 2..MAXINT;

GTint3    = 3..MAXINT;

GRbound   = record
              Xlower,Xupper,Ylower,Yupper : REAL;
            end;

GTpair    = array [1..2] of REAL;

GRpoint   = record
              x,y : REAL;
            end;

GTpath    = (RightPath,LeftPath,UpPath,DownPath);

GTprec    = (Stringprec,Charprec,Strokeprec);

GThorizontal
          = (Hnormal,Hleft,Hcenter,Hright);
            (the meaning of Hnormal depends on the value
            of path. See GKS standard for details)

GTvertical
          = (Vnormal,Vtop,Vcap,Vhalf,Vbase,Vbottom);
            (the value of Vnormal depends on the value of
            path. See GKS standard for details)

GTinterior
          = (HolloFill,SolidFill,PatternFill,HatchFill);

GTclip    = (clip,noclip);

GTInputClass
          = (LocatorInput,ChoiceInput,StringInput);

GTstatus =

GRInput   = record
             case InputClass : GTInputClass of
               LocatorInput  : (locator        : GRlocator);
               ChoiceInput   : (choice         : GRChoice);
               StringInput   : (stringlength   : GTint0;
                                inputstring    : GTstring);
           end;

GRlocator
```

```
          = record
              position
                       : GRpoint;
              Ntran    : GTint0;
            end;

GRChoice = record
              status : GTChoiceStatus;
              choice : GTint1;
            end;

GTChoiceStatus
          = (ChoiceOK,NoChoice);

GTstring = packed array [1..GCmaxstring] of char;
```

6.2 CONTROL FUNCTIONS

OPEN GKS

```
procedure GOpenGKS (error: GTerrorfile);
```

Before using any input-output function GKS must be initialized by invocation of OPEN GKS procedure. It can be called only once during a session (that is until a CLOSE GKS is invoked) and need a parameter specifying the name of a file where GKS will put error information. At initialization time some parameters are set up with default values.

CLOSE GKS

```
procedure GCloseGKS;
```

Stop working with GKS. Each workstation must be closed before invoking this procedure.

OPEN WORKSTATION

```
procedure GOpenWs(wsid : GTWsId; connid : GTWsConnection;
                  wstype : GTWsType);
```

The program must identify the workstation that will be used by the application specifying a name, the virtual address used by the operating system and the type. The last parameter describes the I/O capabilities and there is an installation dependent correspondence between a type and a real workstation. For example type 1 could be a storage tube with a keyboard, type 2 could be a plotter and so on.

CLOSE WORKSTATION

```
procedure GCloseWs(wsid : GTWsId);
```

ACTIVATE WORKSTATION

```
procedure GActivateWs(wsid : GTWsId);
```

Opening a workstation only defines the destination of the output but nothing can be displayed until the workstation is activated.

DEACTIVATE WORKSTATION

```
procedure GDeactivateWs(wsid : GTWsId);
```

CLEAR WORKSTATION

```
procedure GClearWs(wsid : GTWsId; control : GTcontrol);
```

This command erases the screen or moves to the next sheet on a plotter. The second parameter permits to choose between "clear only if something has been drawn" and "clear even if nothing has been drawn".

6.3 OUTPUT FUNCTIONS

POLYLINE

```
procedure GPolyline(n : GTint2;
                    p : array[min..max : INTEGER] of GRpoint);
```

It is the basic line drawing primitive. Adjacent points are connected with straight line segments. Points are given in world coordinates.

TEXT

```
procedure GText(position : GRpoint;
                     str : packed array[min..max : GTint1]
                           of CHAR);
```

Text can be written by specifying a position and a string of characters. In order to have a good quality text see the many text attributes defined above.

FILL AREA

```
procedure GFillArea(n : GTint3;
                    p : array[min..max : INTEGER ] of GRpoint);
```

Defining an area is a simple extension of defining a polyline. An array of points must be given. If the area is not closed the last point is automatically connected to the first one. See fill area attributes to define the type of filling.

6.4 OUTPUT ATTRIBUTES

SET LINETYPE

```
procedure GSetLineType(index : Gtint1);
```

The linetypes are implementation dependent. GKS standard defines some raccomanded values: solid, dashed, dotted, dashed-dotted. Default value is (1).

SET LINEWIDTH SCALE FACTOR

```
procedure GSetLineWidth(width : REAL);
```

The width is relative to a standard line. For example a value 0.5 define a line half the thickness of a standard line. Default value is (1.0).

SET POLYLINE COLOUR INDEX

```
procedure GSetLineColour(index : GTint0);
```

This points to a table where the colours are defined as RGB intensity values. For pen plotters an installation dependent correspondence will be established between index and colour pen. Default value is (1).

SET TEXT FONT AND PRECISION

```
procedure GSetTextFontPrec(font : GTint1; prec : GTprec);
```

Different fonts can be used by a GKS application. They can be monospaced or proportionally spaced and the height to width ratio is a property of the font. The first parameter describes the number of the font while precision is used to determine how many text attributes must be used. With STRING precision the text is written from left to wright and only the "character height" can affect the text. A workstation can use its hardware character set. In CHAR mode each individual character can be affected by all the attributes in a workstation dependent way. Clipping is performed on a character by character basis. The preceeding two modes are defined in "output level 0". In STROKE mode the string will be represented precisely using all attributes. Deafult value is (1;STRING). See Fig. 8.

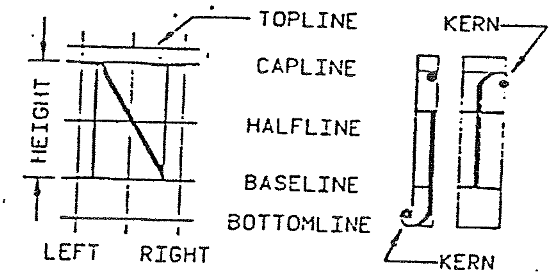


Fig. 8. Font description

SET CHARACTER EXPANSION FACTOR

procedure GSetCharExpansion(expansion : REAL);

The width of the characters can be multiplied by this factor. For example a value of 2 would produce a character twice as wide as it would be. Default value is (1). See Fig. 9 pag. 17 and Fig. 11 pag. 18.

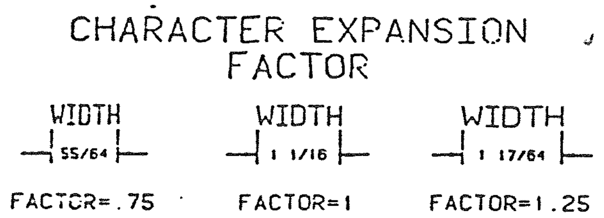


Fig. 9. Character expansion

SET CHARACTER SPACING

procedure GSetCharSpacing(space : REAL);

The space between character bodies is initially assumed to be zero although some space is provided around the characters. Some more space can be defined with this function in term of character height. For example a value of 0.5 would insert a space equal to half the character height. Default value is (0). See Fig. 10 and Fig. 11 pag. 18.

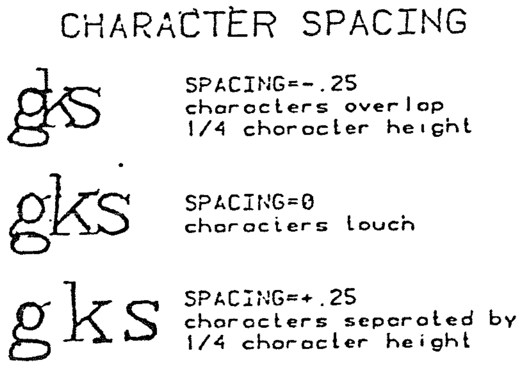


Fig. 10. Character spacing

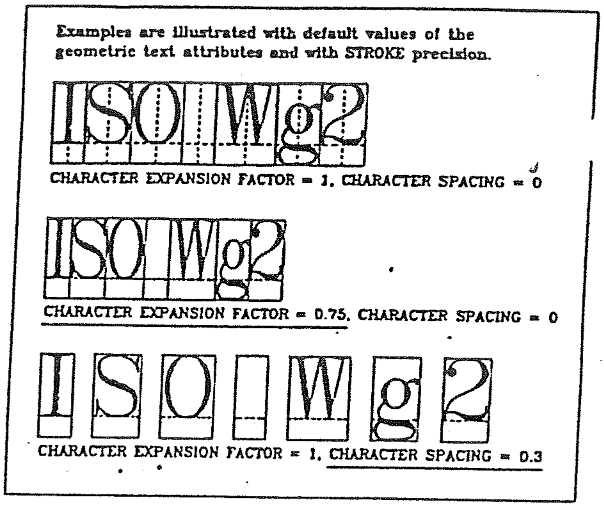


Fig. 11. Non-geometric TEXT attributes

SET TEXT COLOUR INDEX

procedure GSetTextColour(index : GTint0);

This points to a table where the colours are defined as RGB intensity values. For pen plotters an installation dependent correspondence will be established between index and colour pen. Default value is (1).

SET CHARACTER HEIGHT

procedure GSetCharHeight(height : REAL);

Giving the "height" also the "width" is determined since the rate is a property of the "font". This parameter is specified in WC. Default value is (0.01). See Fig. 12 pag. 19.

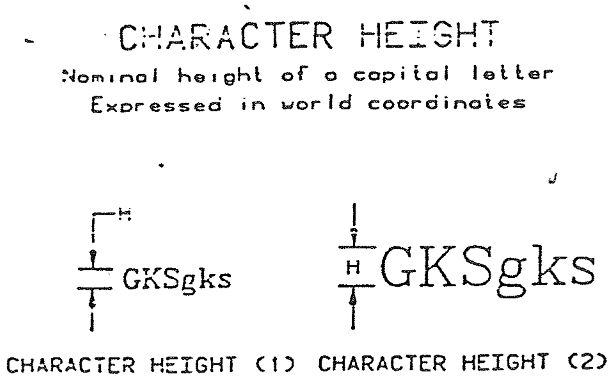


Fig. 12. Character height

SET CHARACTER UP VECTOR

procedure GSetCharUpVector(vector : GTpair);

This parameter specifies the orientation of the character and consequently the orientation of the string which is perpendicular to this vector. Default value is (0.1). See Fig. 13.

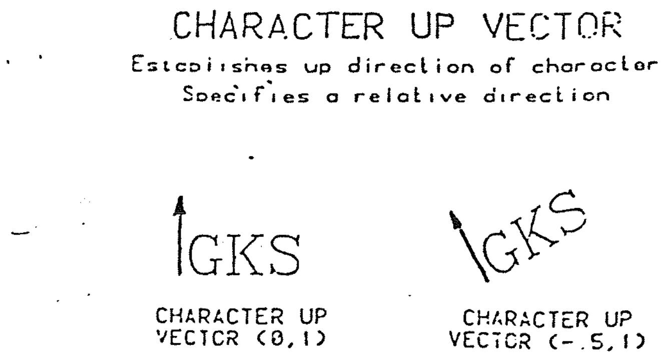


Fig. 13. Character up vector

SET TEXT PATH

procedure GSetTextPath(path : GTpath);

The characters are placed one after the other in the direction specified by "path". Deafult value is (RIGHT). See Fig. 14 pag. 20.

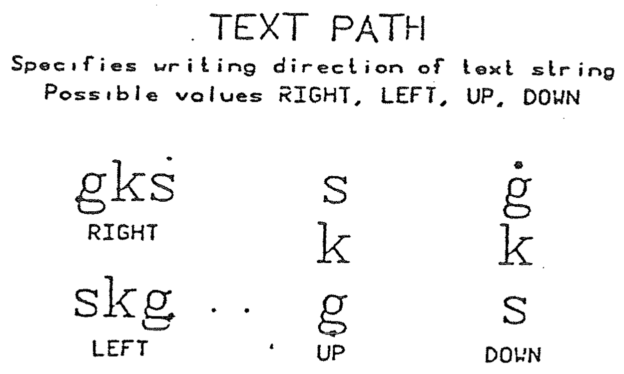


Fig. 14. Possible values of "path"

SET TEXT ALIGNMENT

```
procedure GSetTextAlignment(horiz : GThorizontal;
                             vert : GTvertical);
```

The box containing the string can be positioned in different ways with respect to the "text position". See Fig. 8 pag. 16 for the meaning of the possible values in the orizzontal direction (LEFT,CENTRE,RIGHT) and in the vertical direction (TOP,CAP,HALF,BASE,BOTTOM). Default value is (LEFT;BASE). See also Fig. 15 pag. 21.

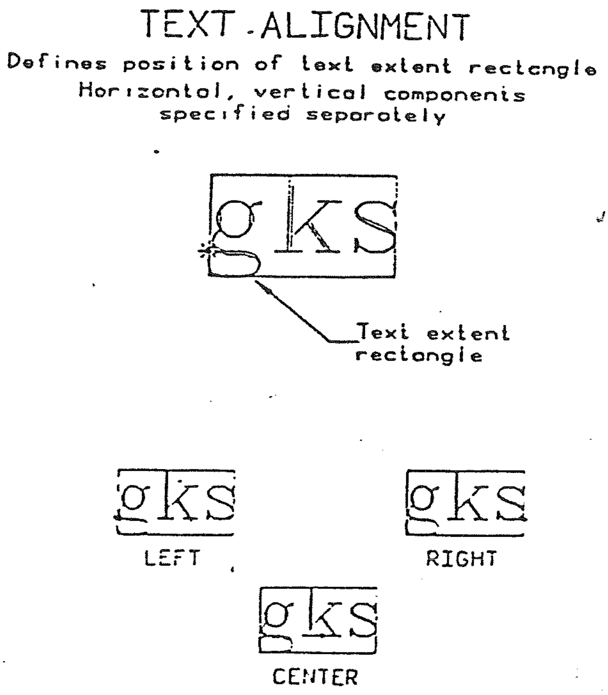


Fig. 15. Some examples of alignment

SET FILL AREA INTERIOR STYLE

```
procedure GSetFillInteriorStyle(style :GTinterior);
```

There are four kinds of filling as illustrated in Fig. 16 pag. 22. Each one is affected by the "colour index". Default value is (HOLLOW).

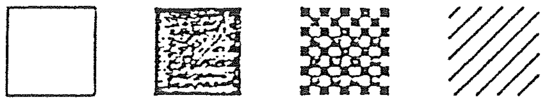


Fig. 16. Values of "fill interior style"

SET FILL AREA STYLE INDEX

procedure GSetFillStyleIndex(index : GTint1);

This parameter has meaning only for PATTERN and HATCH case. It points to a table where different patterns or styles of hatching are defined. Default value is (1).

SET FILL AREA COLOUR INDEX

procedure GSetFillColour(index : GTint0);

This points to a table where the colours are defined as RGB intensity values. For pen plotters an installation dependent correspondence will be established between index and colour pen. Default value is (1).

6.5 TRANSFORMATION FUNCTIONS

SET WINDOW

procedure GSetWindow(nt : GTint1; bound : GRbound);

The application can define a rectangular area in world coordinates giving the limits in the X direction and in the Y direction. The first parameter assign a number to the "window" to "viewport" transformation that is to the "normalization transformation". Default value is (0,1,0,1).

SET VIEWPORT

```
procedure GSetViewport(nt : GTint1; bound : GRbound);
```

The application can define where the "window" will appear in normalized coordinates by specifying the limits in the X direction and in the Y direction. The first parameter assign a number to the "window" to "viewport" transformation that is to the "normalization transformation". Default value is (0,1,0,1).

SELECT NORMALIZATION TRANSFORMATION

```
procedure GSelectNTran(nt : GTint0);
```

The user can define many "window" to "viewport" transformations at the head of the program and select the desired one with this command. The choosen transformation will be in effect until another one is selected. Default value is (0) which means that the "window" will be mapped into the whole NDC space.

SET CLIPPING INDICATOR

```
procedure GSetClipping(clip : GTclip);
```

It is possible to choose between clipping the output to the "viewport" boundary or not. Default value is (CLIP).

6.6 REQUEST OF INPUT DEVICES

REQUEST LOCATOR
REQUEST CHOICE
REQUEST STRING

```
procedure GReqInput(class : GTinputClass;
                      wsid : GTWsId;
                        device : GTint1;
                          var status : GTstatus;
                            var r : GRinput);
```

From this single procedure the GKS abstract functions listed above can be obtained by substituting the first type with the desired input device. The parameters "wsid" and "device" specify the workstation and the desired device of that particular class.
The last two parameters contains the return values: "var status" is the return code and specify the successful of the input action, "var r" has a different meaning depending on the type of input. For LOCATOR it returns a position in world coordinates and the normalization trasformation which was used to map back from NDC to WC. For CHOICE it returns only an integer representing the choice while for STRING it returns the character string that was input and the number of characters.

6.7 EXAMPLE OF USE

In order to facilitate the comprehension of GKS functions it is convenient to write a complete sequence of instrunctions. The simple picture of Fig. 17 pag. 25 is built by dividing the output surface into two parts: an upper one where two dimensional objects are drawn and a lower one where texts are written.

A storage tube (defined as type 2 in our phantom implementation) is used. The logical connection is identifyed by 300. First the upper area is selected to draw a square and a triangle, secondly the triangle is filled with the desired style, finally the lower area is activated in order to write a text.

The position of the string is requested interactively through a LOCATOR device (device 1).

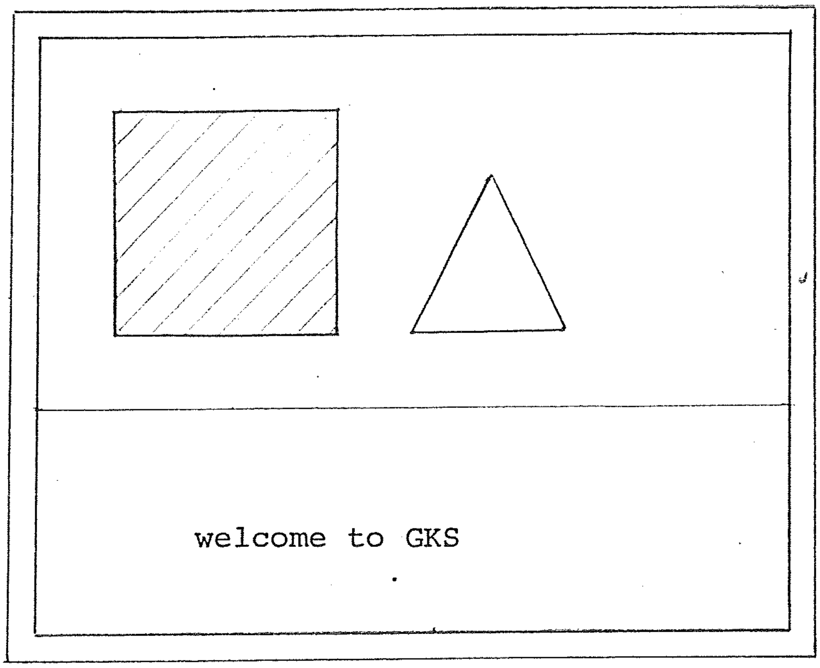


Fig. 17. Simple picture

```
procedure HELLOGKS;

        error       = (errorfile);
        wsid        = 1;
        connid      = 300;
        wstype      = 2;
        nt1         = 1;
        window1     = (0.0,100.0,30.0,80.0);
        viewport1   = (0.0,1.0,0.3,0.8);
        nt2         = 2;
        window2     = (0.0,100.0,0.0,30.0);
        viewport2   = (0.0,1.0,0.0,0.3);
        control     = (ClearAlways);
        n1          = 5;
        square      = (10.0,40.0;40.0,40.0;40.0,70.0;10.0,70.0;10.0,40.0);
        n2          = 4;
        triangle    = (50.0,40.0;70.0,40.0;60.0,60.0;50.0,40.0);
        style       = (Hatch);
        index       = 2;
        string      = (welcome to GKS);
        class       = (LocatorInput);
        device      = 1;

                GOpenGKS(errorfile);
                GOpenWs(wsid;connid;wstype);
                GActivateWs(wsid);

(define normalization transformation for upper viewport);
```

```
                GSetWindow(nt1;window1);
                GSetViewport(nt1;viewport1);

(define normalization transformation for lower viewport);

                GSetWindow(nt2;window2);
                GSetViewport(nt2;viewport2);

                GClearWs(wsid;control);

(draw a square and a triangle in the upper viewport);

                GSelectNtran(nt1);
                GPolyline(n1;square);
                GPolyline(n2;circle);

(fill square with hatch style);

                GSetFillInteriorStyle(style);
                GSetFillStyleIndex(index);
                GFillArea(n3;square);

(select lower viewport and write a text);

                GSelectNtran(nt2);
                GRegInput (class;wsid;device;ntran;position);
                GText(position;string);

(end GKS)

                GDeactivate(wsid);
                GCloseWs(wsid);
                GCloseGKS;
```