AN IMPLEMENTATION OF THE GKS TEXT PRIMITIVE

by

HENRY BRADLEY FOUT III

B.S.,B.A. Loyola College of Baltimore, 1974

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

ACKNOWLEDGEMENTS

There where many individuals involved in the project to implement GKS at Kansas State University. All of them should be recognized for their contributions. Mike Cavins, our project leader, had the somewhat arduous job of melding everyones work together. Mike deserves special thanks for always finding the time to help. Also, Dr. William Hankley for suggesting and sponsoring this project, which was the first summer-on-campus group Masters project.

Aside from the project and report, I feel it proper to mention a couple special people. When I finish the summer of 1984 I will have spent five summers at KSU in pursuit of my Masters Degree in Computer Science. The five weeks each summer are very intense, and at sometimes the pressure seemed overwhelming. There were two people who helped make the summers memorable. They did everything possible to make mine and the other students lives a little bit nicer. I am referring to Mick and Mary Beth Cole. They worked hard to prepare for our arrival each summer, planned picnics and other social events. In general, they were always there to help. Mick and Mary Beth treated us like family. So, to Mick and Mary Beth from the bottom of my heart, thank you.

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 SUMMARY

1.1 Definition and History of the Graphical Kernel System

The Graphical Kernel System (GKS) is currently a proposal for standard by the International Standards Organization (ISO). GKS was developed by a subcommittee of the German Institute for Standardization (DIN). It was submitted to the ISO by DIN in November of 1979. The following year, GKS became a proposal for standard in Germany. In late 1984 GKS is expected to be accepted as the first international graphical standard.

The Graphical Kernel System is an interface between graphics application programs and configurations of graphical input and output devices. GKS is a set of functions callable by application programs that generate two dimensional pictures on line or raster graphic devices [11]. The significance of GKS is that it hides the peculiarities of the graphics hardware from the application program, thus allowing for portability of both the application program and the programmer.

The work which eventually lead to the development of GKS began in 1974 when ACM SIGGRAPH established the Graphics

Standard Planning Committee (GSPC) [9]. This committee was given the charter of developing a set of standard graphics primitives to allow creation and viewing transformations of 2D and 3D line drawings. The main goal of this standardization work was to obtain machine and device independence to protect the software investment of the user. The work of the GSPC resulted in the CORE graphics system. The German Standards Institute, DIN, began the design of a Core graphics system which became known as GKS. Both systems were designed with the same goal in mind, and in many ways are very similar. GKS is being pursued as an international standard mainly due to the fact that it was presented to the ISO before CORE [9].

Extensions and modifications to the basic GKS package are already being planned by standards organizations and user groups. In order to satisfy the needs of the more sophisticated CAD/CAM users, the X3H31 subcommittee of the American National Standards Institute (ANSI) is working on a richer GKS system. Three dimensional graphics capabilities is one of their planned extensions. The X3H31's "rich" system is not expected to reach the final draft stage until 1985 [9].

Concurrent with the efforts to standardize GKS there is work being done to develop a standard hardware interface. Intel, Digital Equipment Corporation, Tektronics, and others are working together to develop a Virtual Device Interface (VDI) standard [13]. It is felt that an increased standardization of the hardware will promote the use of software standards such as GKS.

There are naturally critics of GKS as an international standard. The fact remains that a standard is needed. There will be no requirement for companies to implement GKS, nor will there be any requirement for companies using the CORE system to switch. GKS will have to earn acceptance. Many in the industry feel it will [9].

1.2 KSU Implementation Project

The Kansas State University's Department of Computer Science began a project to study and implement subsets of GKS in the spring semester of 1983. The project continued through the summer semester, from which this report is derived. Besides the interest generated by its pending standardization, the study of GKS, as a project, provided its participants with a valuable educational experience.

The study of GKS by the spring semester students resulted in the framework from which the summer students study and development efforts began. Using the ISO/DIS 7942 (Draft International Standard) [11] and other available literature, the spring semester students designed the general data flow, named and coded most major data structures, developed a device driver for the Chromatics terminal, and partially developed some of the GKS output primitives.

The major goal of the GKS project was for the participants to obtain an educational experience from the study of GKS. In addition to the study of the overall GKS system, each student focused on a specific subset. Using the skeleton developed by the spring semester and the GKS Draft[11] as a base, each student set out to implement their subset(s). Some of the individual implementation projects included working subsets of graphic output primitives, segmentation, device drivers, and the development of a compatible Virtual Device Interface (VDI).

1.3 Report Overview

In addition to the brief definition and history of the

GKS system, this report focuses on the TEXT output primitive. Section 2.0 provides an overview of the TEXT output primitive and its associated attributes, as specified by the GKS ISO Draft[11]. Throughout this section, "COMMENTS" are used to highlight the differences between the ISO Draft[11] and the KSU specification of the TEXT output primitive.

Throughout most of the literature written about GKS, reference is made to the CORE graphics system. Section 3.0 provides a general review of the CORE system. Specifically, it describes the TEXT output primitive as defined in the CORE system. Key differences between GKS and CORE are noted, which leads to an appreciation GKS over CORE.

The study of the GKS TEXT output primitive included the effort to implement a minimal subset of TEXT. General implementation decisions to bundle all attributes, and develop minimal subsets contributed to the design of this subset. The effects of these decisions, and a detailed account of the data structures, user procedures, and overall data flow of the KSU version of TEXT are provided in Section 4.0.

The concept to bundle attributes in GKS appears to

provide many advantages over non-bundling, as found in the CORE system. However, sometimes negative results can occur when a good thing is pushed too far. Thus is the case with the general implementation decision to "bundle all attributes". Section 5.0 presents a discussion of this decision and other implementation concerns.

The project to implement a minimal subset of the GKS TEXT output primitive was not completed. The goal of this project was to generate single lines of text on a graphics output device. The code to accomplish this task was written and compiled, however, there was not enough time to fully test and integrate it into the base system developed by the spring semester. All design changes are discussed in Section 4.0 and included in APPENDIX A.

2.0 GKS TEXT GRAPHICAL OUTPUT PRIMITIVE: A Tutorial and Comparison

2.1 General

This section provides a tutorial of the GKS TEXT output primitive as specified by the GKS ISO/DIS 7942 Draft[11]. Because of basic decisions made early in the KSU project, the KSU specification differs slightly from the Draft[11]. Comparisons of the two specifications are highlighted by **COMMENTS** throughout this section.

The term "workstation" is used throughout this report and other literature on the subject of GKS. A workstation represents a collection of graphical input devices such as a keyboard, joystick, light pen, or a trackball, and/or output devices such as a refresh display, or plotter. GKS treats the whole workstation as one logical unit [5]. For each kind of workstation in a given GKS implementation, an entry exists in a workstation description table. This table defines the capabilities and characteristics of each workstation. These tables are maintained at the GKS level. GKS provides a set of inquire functions that may be used by

the application program to determine the capabilities available on a given workstation. The application program does not have to deal directly with bothersome intricacies and differences of various pieces of hardware. Instead, GKS hides them and provides an abstraction of the hardware, in the form of a workstation, to the application program.

The appearance of the graphic output primitives will vary between workstations, since the capabilities of these facilities may differ significantly. The ISO/DIS Draft [11] specifies facilities to allow for these variations such as POLYLINE REPRESENTATION, TEXT REPRESENTATION, etc.. The KSU implementation supports the same capability by the use of functions such as DEFLINE, DEFTEXT, etc.. These functions are used to define the attribute values in the workstation bundle table of each workstation.

The TEXT output primitive differs from the other output primitives defined under GKS in that its attributes are split into two distinct groups. The first group of attributes are workstation-independent, and are referred to as geometric attributes. These attributes control the geometric aspects of the text representation such as height, direction of character string, its path and alignment. The

second group of attributes are workstation-dependent, and are referred to as non-geometric attributes. These attributes control the aspects of the text representation such as font, precision, expansion factor (relative width), spacing between characters, and color.

The precise control of the appearance of the TEXT output primitive on a given workstation is determined by the combination of the geometric and non-geometric attributes. The extent to which the geometric attributes are applied to the final appearance of TEXT generated depends on the value of the "precision" attribute. The precision attribute determines the closeness of the text representation at the workstation in relationship to the values specified by the geometric attributes. The possible values of the precision attribute, in order of increasing adherence to the defined geometric attribute values are STRING, CHARACTER, and STROKE.

2.2 TEXT an Output Primitive

The graphical output that is generated via GKS is made up of basic pieces called "output primitives". Output primitives are an abstraction of the capabilities of the

graphics device such as line drawing, marker generation , and printing character strings. GKS supports the following graphical output primitives:

POLYLINE

POLYMARKER

TEXT

FILL AREA

CELL ARRAY

GENERALIZED DRAWING PRIMITIVE

One of the basic requirements of any graphics system is the capability to generate lines of text (character strings). Text is essential for both annotating diagrams, and communicating with the console user of an interactive program. Today, the use of presentation graphics, which heavily relies on a variety of character font styles, is growing at an unpresidented rate [13]. Many graphics devices are being manufactured today with a rich set character generators. GKS is designed to take full advantage of such device capabilities.

The appearance of the text output primitive such as color, character height, etc. is determined by the attributes applied to the primitive and the capabilities of

a particular workstation.

2.3 TEXT Attributes

The text output primitive has the largest set of attributes which control its appearance on a workstation. These attributes are divided into geometric and non-geometric attributes as shown in FIGURE 1.

2.3.1 GEOMETRIC

The geometric attributes are also referred to as workstation independent or primitive attributes. These attributes are specified individually and are bound to the text primitive when it is created. The geometric information such as height can be subject to the same transformations as the geometric data that is part of the primitive. The values of the geometric attributes are stored in world coordinates at the GKS level prior to any transformations. The stored values represent the current values of the associated attribute. The current values are unaffected by the transformations. The geometric text attributes are considered workstation independent because they are stored at the GKS level, are bound to the primitive-regardless of the workstation capability, and only

some may affect the appearance of the text-depending on the value of the precision attribute.

The Draft ISO/DIS 7942 [11] specifies that the current values of the geometric attributes for TEXT are stored in the GKS state list. The values are used in the creation of all subsequent text output primitives. The current values are set or changed by the use of the following attribute setting functions:

SET CHARACTER HEIGHT

SET CHARACTER UP VECTOR

SET TEXT PATH

SET TEXT ALIGNMENT

** COMMENT **

The KSU implementation does not allow for the setting of these attributes individually. Instead, they are bundled in much the same way as the workstation dependent attributes. This is a result of a general decision, made early in the project, to bundle all attributes.

The TEXT geometric attributes are stored in an array of records called GKSTEXTTABLE. Each record contains values for character HEIGHT, UP VECTOR, PATH

and ALIGNMENT. The values of the attributes are set by the STOTEXT function (procedure). The current values to be used by the TEXT output primitive depends on the value of the current TEXTINDEX into the GKSTEXTTABLE. These functions and data structures will be covered in more detail in Section 4.0 of this report.

A discussion concerning the decision to bundle all attributes will follow in Section 5.0 of this report.

** END COMMENT **

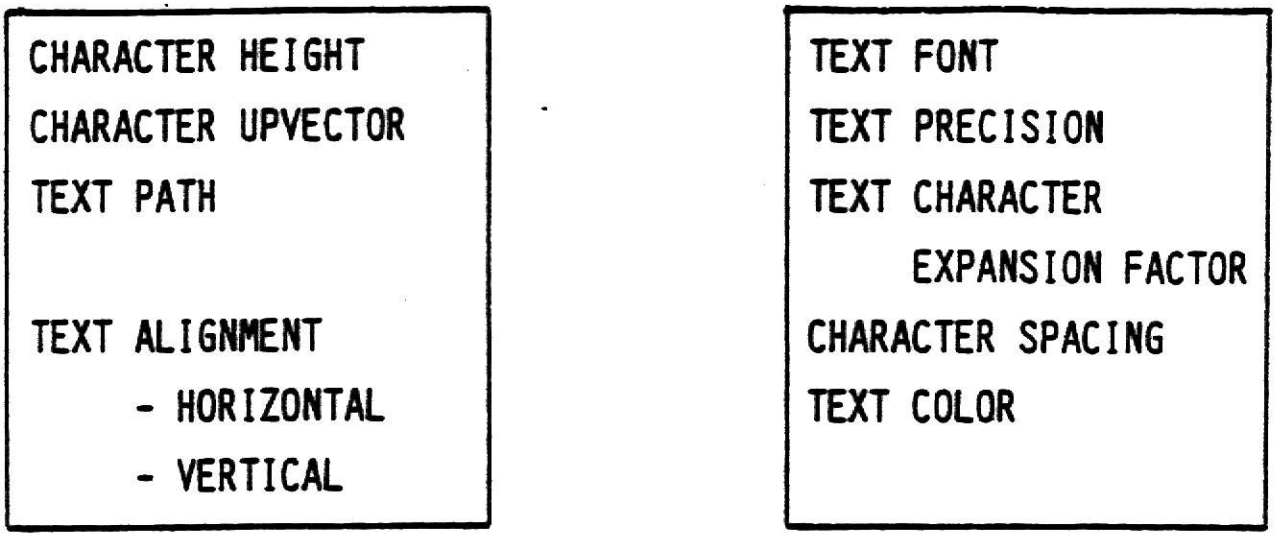
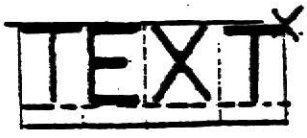


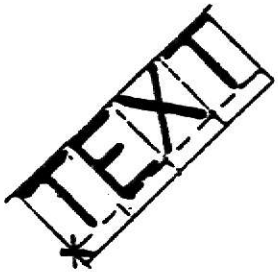
FIGURE 1. TEXT ATTRIBUTES

CHARACTER: HEIGHT = 1.0
UPVECTOR = (0.0, 1.0)
PATH = RIGHT
TEXT ALIGHNMENT -
(NORMAL, NORMAL)

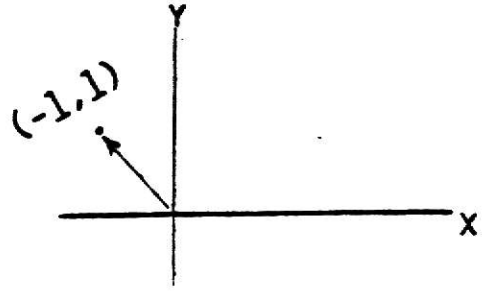
△ HEIGHT = 0.8



△ ALIGNMENT = (RIGHT, TOP)



△ UPVECTOR = (-1.0, 1.0)



X = TEXT POSITION

FIGURE 2. REPRESENTATION OF GEOMETRIC TEXT ATTRIBUTES

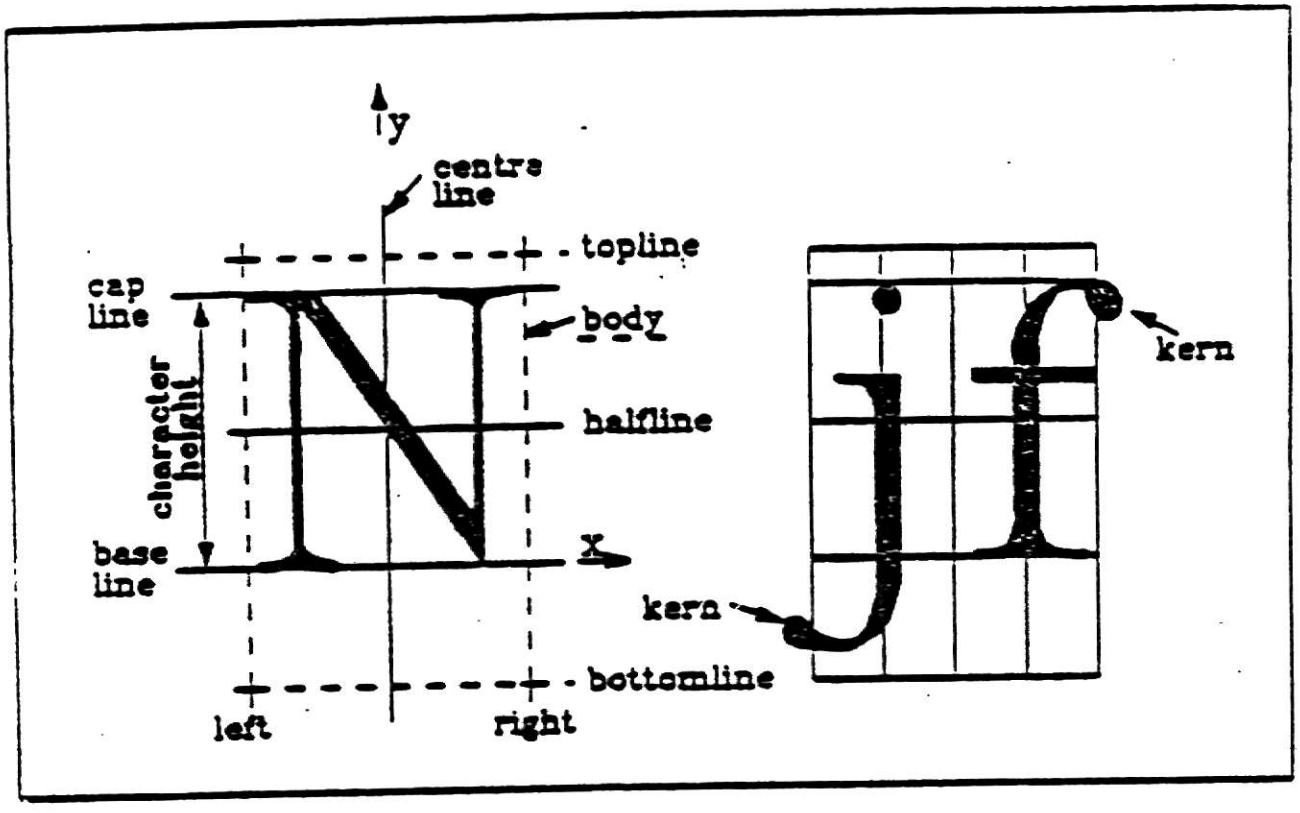


FIGURE 3. FONT DESCRIPTION COORDINATE SYSTEM

The TEXT geometric attributes are further defined below and illustrated in FIGURE 2.

HEIGHT

-Permitted values: Real > 0, specified in World Coordinates.

-Specifies the nominal height of a capital letter. The value of 1.0 (one) would represent default size of the font as specified by the font designer. A change of HEIGHT would uniformly change the size of the character.

-The HEIGHT is the distance between the baseline and the capline of the font coordinate system. See FIGURE 3.

UP VECTOR

-Permitted values: tuple (Real,Real), specified in World Coordinates.

-Specifies the up direction of a character. The UP VECTOR is parallel to the center line and perpendicular to the baseline of the font description coordinate system. See FIGURE 3.

-FIGURE 2 illustrates the relationship between the world coordinate values and the x,y coordinate system.

PATH

-Specifies any one of the enumerated values: right, left, up, or down

-In general, these values represent the direction in which one would read the text in relation to the up vector, i.e. right- means read to the right, down- means read down. See FIGURE 2.

TEXT ALIGNMENT

-Specifies the position of the text extent rectangle in relation to the text position, as shown in FIGURE 2.

-This attribute has a horizontal and vertical component. The horizontal component is specified by one of the enumerated values: left, center, right, or normal. The vertical component is specified by one of the enumerated values: top, cap, half, base, bottom, normal.

** COMMENT **

The KSU implementation specifies the two components as separate attributes as follows:

-HALIGN -specifies any one of the enumerated values: left, center, right, or normal. See FIGURE 3.

-VALIGN -specifies any one of the enumerated values: top, cap, half, base, bottom, normal. See FIGURE 3.

** END COMMENT **

2.3.2 NON GEOMETRIC

The non geometric attributes are also referred to as workstation dependent attributes. These attributes are stored in workstation bundle tables specific for each workstation. Each entry(TEXTINDEX) contains the desired combination of values to apply to the text primitive at the workstation. The values of the non geometric attributes are restricted to the capabilities of the associated workstation.

The Draft ISO/DIS 7942[11] specifies the SET TEXT REPRESENTATION function to build an entry in the workstation bundle table.

** COMMENT **

The KSU project defines the DEFTEXT(define text) function(procedure) to build an entry into an array of records containing the attribute values for a specific workstation identifier and text index, which are parameters of this function.

** END COMMENT **

In addition to specifying the non geometric attributes in the workstation bundle table, the Draft[11] standard allows them to be specified individually in the GKS state list. In this case the non geometric attributes are workstation independent. If an application program desires the use of an individually specified attribute, i.e. color = red, this value will be used in defining the primitive at the workstation, instead of the value for color specified in the workstation bundle table at the current text index. If the individually specified attribute does not conform to the workstation capabilities, i.e. red is not defined in the workstations color table, then a default value is used. This facility allows, within the capabilities of a workstation, consistency of desired attributes when generated on different workstations, independent of the value specified in the bundle table.

The capability to individually specify non geometric attributes was not a part of the original GKS proposal. Pressure from the American National Standards Institute (ANSI) resulted in the ISO adopting it [9]. ANSI did not support GKS's bundling of attributes because the appearance of the primitive was determined at the device level according to the values of the attributes assigned, and the

devices capabilities. More direct programmer control of the attributes was desired.

Of course, if all devices supported the same capabilities, and all workstation bundle tables were built identical, there would be no need for individually specifying attributes. Also, there would not be a need for a standard such as GKS. None of the above is realistic, thus the need for a standard in the first place. This ability gives GKS added flexibility.

GKS specifies the following functions to set the current values of the TEXT non geometric attributes in the GKS state list:

SET TEXT FONT AND PRECISION

SET CHARACTER EXPANSION FACTOR

SET CHARACTER SPACING

SET TEXT COLOR INDEX

The display of the subsequent TEXT output primitives depends on the values of the INDIVIDUAL, BUNDLED or a combination of both sets of non geometric attributes. The value used depends on the setting of the Aspect Source Flag (ASF) associated with each attribute. Each ASF can be assigned the enumerated value INDIVIDUAL or BUNDLED. If

the ASF of an attribute is set to INDIVIDUAL, then the current value of that attribute, specified in the GKS state list, is bound to the primitive at creation, and used in the display of subsequent TEXT output primitives. If the ASF of an attribute is set to BUNDLED, then the value specified in the workstation bundle table, at the current text index, is used. In this case, binding of the bundled attributes occurs at the workstation.

FIGURE 4 illustrates the results of specifying a non geometric attribute as BUNDLED or INDIVIDUAL.

GKS specifies the SET ASPECT SOURCE FLAG function to be used by the application programmer for setting an attribute's ASF in the GKS state list.

** COMMENT **

For reasons of simplicity and the limited scope of the KSU project, it was decided not to provide the capability to individually specify non geometric attributes.

** END COMMENT **

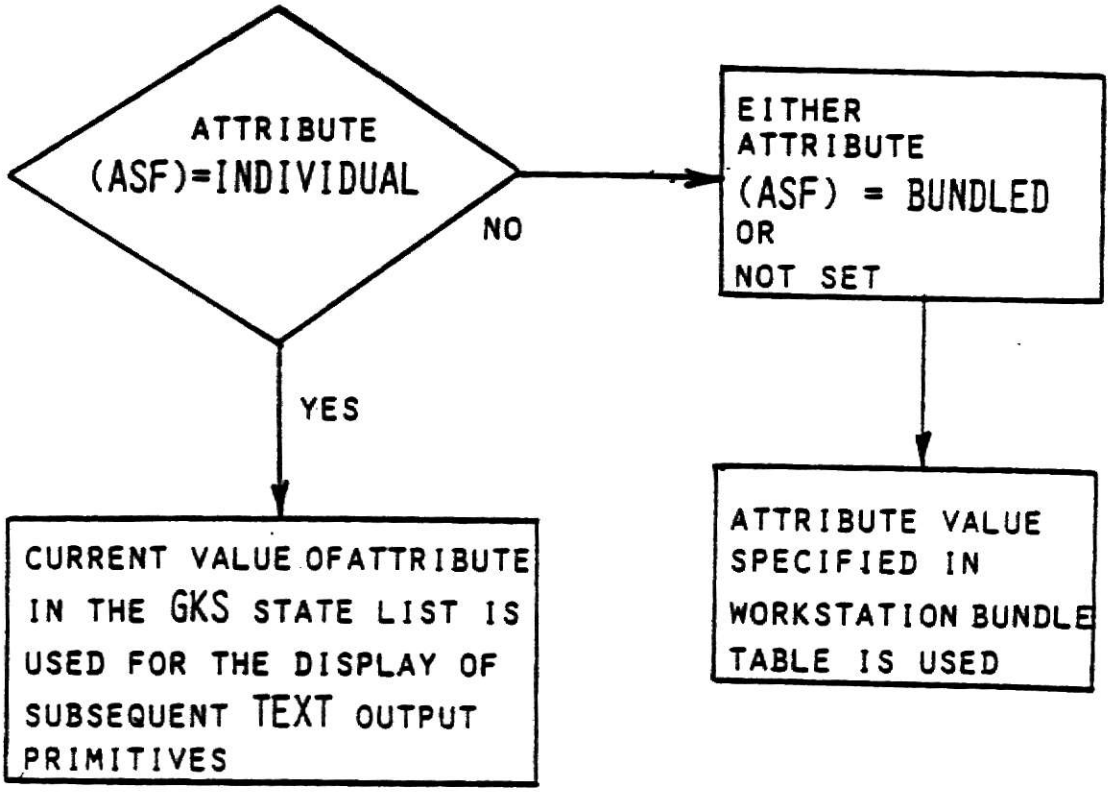


FIGURE 4. ASPECT SOURCE FLAGS (ASF)

The TEXT non geometric attributes are further defined below, and illustrated in FIGURE 5.

FONT and PRECISION

-TEXT FONT and PRECISION will be discussed in section 2.4.

CHARACTER EXPANSION FACTOR

-Permitted values: Real > 0, initial value = 0.0.

-Specifies the deviation of the character width to height ratio from that indicated by the font designer. In effect, a change in the expansion factor is a change in the character width.

-EXAMPLE: If the font design specifies the height to width ratio to be 8:4 (could mean 8 pixels : 4 pixels), then an expansion factor value of 1.0(one) would result in the font being displayed with a height to width ratio of 8:4. If the expansion factor is set to .75, the character would be displayed with a height to width ratio of 8:3 (see FIGURE 5).

CHARACTER SPACING

-Permitted values: Real, initial value = 0.0.

-Specifies the space to be inserted between adjacent character boxes. A value of 0.0(zero) will yield no space

between character boxes. A positive value will yield the specified space between each character box. A negative value will subtract space, causing an overlap of each character box equivalent to the absolute value specified. The value used to specify character spacing is a fraction of the specified CHARACTER HEIGHT.

-EXAMPLE: If the font design specifies the height to width ratio to be 8:4, and the CHARACTER HEIGHT attribute is specified as 1.5, the resultant height to width ratio will become 12:6. If the CHARACTER SPACING is specified as 0.5, the resultant space between character boxes will be 6 (six pixels).

TEXT COLOR INDEX

-Specifies the index into the workstation dependent color table, which points to the desired color.

** COMMENT **

The KSU implementation uses the enumerated color name instead of building a color table for each workstation. The attribute name is TEXTCOLOR. If the color specified is not a capability of the workstation, a default color will be used. In effect, the result is the same as if TEXTCOLOR was an

individually specified workstation attribute.

** END COMMENT **

CHARACTER: EXPANSION FACTOR = 1.0
CHARACTER SPACING = 0.0

EXPANSION FACTOR = 0.75

SPACING = 0.3

FIGURE 5. REPRESENTATION OF NON-GEOMETRIC TEXT ATTRIBUTES

2.4 FONT and PRECISION

The GKS Draft[11] specifies that TEXT FONT and PRECISION constitute one aspect, thus are specified as a tuple and represented by the attribute name: TEXT FONT AND PRECISION.

** COMMENT **

The KSU implementation specifies the two separate attribute names: TEXT FONT, and TEXT PRECISION.

** END COMMENT **

The FONT is basically defined as the shape of the symbol which represents each character. Each character defined in the 2D font coordinate system has an associated character body, font baseline, font halfline, capline, and a centerline (see FIGURE 3). Each character body or symbol represents a specific pattern, generated by the workstation character generator, within the boundary of the character box. Possible font types on a particular workstation may include standard, italic, boldface, etc..

In raster graphic systems, character fonts are defined in terms of a pixel array. Each character is defined as a

pattern of dots(pixels) within the character box [14]. In general, if the character box is 8(eight) pixels by 4(four) pixels, each possible character can be represented as a bit pattern of 32(thirty-two). Typical hardware character generators generate characters in this fashion.

For a graphics application program to have the capability to control such aspects as character height, spacing, etc., it would require much overhead software to handle the hardware generated characters and/or provide its own software character set. GKS provides this overhead by providing an application program the ability to choose the desired TEXT attributes. The control and appearance of the resultant displayed TEXT is dependent on the TEXT FONT specified and the value of the TEXT PRECISION attribute.

The value of the TEXT PRECISION attribute influences the quality of TEXT displayed in relation to other attributes specified. On the low end text will be interpreted very loosely, in a workstation dependent way. On the high end text is displayed at the specified text position with all attributes applied. The possible TEXT PRECISION values, from low to high quality, are STRING CHARACTER, and STROKE. They are briefly defined as

follows:

STRING

-The character string is generated in the desired TEXT FONT starting at the given text position. The specified character HEIGHT and EXPANSION FACTOR are interpreted depending on the capabilities of the specified workstation. The clipping applied to the character string or individual characters is implementation dependent. the values of the character UP VECTOR, TEXT PATH, TEXT ALIGNMENT, and SPACING attributes may not be applied.

-This precision is used to generate efficient low quality text such as a message to the console user, or when normal horizontal text is desired.

-All GKS implementations support at least STRING precision for every defined font.

CHARACTER

-The text character string is generated using the specified font. The position of the resultant text extent rectangle is determined by the value of text alignment and given text position. The aspects of character HEIGHT, UP VECTOR, and EXPANSION FACTOR are evaluated in a workstation dependent way for the representation of each character. Character

SPACING is exact and the character body is ideal, because they are calculated precisely from the text aspects and font dimensions. Clipping is performed on a character by character basis. This means that if part of a character extends beyond the given window or viewport boundary, only that character is clipped or removed.

-CHARACTER precision should be supported by most GKS implementations.

-Not as efficient as STRING, but allows for more graphical uses of the TEXT primitive.

STROKE

-The character string generated reflects the application of all specified attributes, unless those attributes were specified individually and are not within the capabilities of the workstation. The character string is clipped exactly at the clipping rectangle. This means that only the part of the character extending beyond the window is clipped.

-This precision allows the greatest control over the use of the TEXT attributes as they are applied to the TEXT output primitive. It is also the least efficient of the precisions and should only be used when required.

-The GKS Draft [11] specifies that those installations that

support STROKE precision, must support at least one STROKE precision font on every workstation. This font would be the same and designated as font number 1(one) representing a standard character set such as the one used in the writing of this report.

3.0 REVIEW OF THE CORE SYSTEM

3.1 General

As mentioned in the opening summary, the CORE system was a result of the ACM SIGGRAPH's Graphics Standard Planning Committee. Though development began before GKS the GSPC acted tenuously by not submitting CORE to the ISO when they could have. Instead, they submitted it to the ANSI X3H31 subcommittee. Meanwhile DIN submitted GKS to the ISO, which took world attention away from CORE and onto GKS.

Both GKS and CORE were designed with the goal of providing machine and device independence to graphical application software. To achieve this independence the application program calls core/kernel functions, which in turn drive "virtual" graphics I/O devices. A device driver translates the "virtual" instructions into hardware instructions [9]. The CORE system supports a set of output primitives similar to those found in GKS. The major difference between GKS and CORE is the way in which each system applies the associated attributes to define the

output primitives.

The basic difference between CORE an GKS is in the way each generates the appearance of the graphic output primitives. In GKS the appearance is defined in two stages. In the first stage a symbolic attribute(TEXT INDEX) is associated with the primitive. In the second stage this attribute is mapped onto the capabilities of the workstation (attributes set at TEXTINDEX for that workstation). Both stages are under the control of the application program. Thus, GKS has the capability of representing the same output primitive differently on separate workstations without regenerating the primitive. In the CORE system the appearance of the primitive is associated with the primitive itself. To get the same effect as in GKS the associated attribute settings need to be changed and the primitive regenerated [5].

The next section describes how the TEXT output primitive and primitive attributes are defined by the CORE graphics system.

3.2 TEXT as defined in CORE

Since GKS currently has two dimensional capabilities,

the following will only describe the two dimensional capabilities of the CORE system.

In order for the CORE system to generate an output primitive it must be aware of the "current position", which is a reference point defined in world coordinates, from which objects are defined. The first character of the TEXT string must be positioned with its lower left corner at the "current position". Therefore, prior to generating the TEXT output primitive a "MOVE" command must be performed to set the "current position" of where the character string will start. As the character string is generated the value of the "current position", as known to the system, will change to the position of the next character to be generated. To generate another primitive a "MOVE" command must be invoked again, to adjust the "current position".

GKS does not support the concept of "current position". The "text position" is an input parameter used by the TEXT primitive, and is not remembered by the system.

The following primitive attributes are defined for the CORE system TEXT primitive:

CHARSIZE

CHARSPACE

CHARQUALITY

The primitive attributes, CHARSIZE and CHARSPACE may be used in determining the size and extent of the character string [12]. The CHARQUALITY attribute is analogous to the TEXT PRECISION attribute, defined in GKS, in that its value determines the degree to which the other attribute values are adhered to. A brief definition of each of the CORE attributes follows.

CHARSIZE

-The SET_CURRENT_CHARSIZE (a,b) function specifies the size of the character box in world coordinates. The values a(width) and b(height) are used to determine the size of the characters drawn.

-EXAMPLE: If a=3.0 and b=5.0, then each character will be drawn within a rectangle 3(three) world coordinates wide and 5(five) world coordinates high.

CHARSPACE

-This attribute is used to manipulate the spacing between characters and the TEXT orientation.

-The SET_CURRENT_CHARSPACE (x,y) function specifies the x(horizontal) and y(vertical) distance in world coordinates between the setting of the "current position" of each

character to be generated. When the TEXT primitive is invoked, each character in the string advances the "current position" to the next "current position" based on the values of x and y. The orientation of the resultant TEXT character, or string of characters, is a function of the CHARSPACE values and the setting of CHARQUALITY.

-This attribute is analogous to a combination of the GKS functions, CHARACTER SPACING and UP VECTOR.

CHARQUALITY

-This attribute defines how strictly CHARSIZE and CHARSPACE will be interpreted, which will determine the appearance of the character string on the viewing surface [2].

-CHARQUALITY can have the values LOW, MEDIUM, or HIGH, which are analogous to the GKS TEXT PRECISION possible values of STRING, CHARACTER, or STROKE, respectively.

It is not the purpose of this section to give a detailed account of the CORE system, but instead, to touch on the major aspects associated with the CORE TEXT primitive. Much of the detail omitted was not necessary to illustrate the basic similarities and differences between the CORE and GKS systems.

As one can see, there are similarities in the way each

system defines the treatment of the TEXT output primitive. Most of the similarities can be attributed to the similar goals of both systems. On the other hand, the differences can be attributed to the difference in philosophy. The major distinction between GKS and CORE, which has been noted several times, is that the attributes are bundled in GKS and are not in the CORE system. The bundling of attributes in GKS allows for defining a workstations capabilities, and the mapping of these capabilities onto the primitive at the workstation. In CORE the appearance of the primitive is associated with the primitive itself. In CORE the current values of an attribute, i.e. color, are known to the system. If a POLYLINE were just drawn in red, then the subsequent TEXT will be generated in red, unless the "current color" is changed. Bundling allows an application program to generate the same primitive differently on multiple workstations, CORE will not. Proponents of each system will argue a case for each system, but it is obvious that the workstation concept of GKS provides for better order and flexibility.

Taking advantage of the capabilities of each workstations is certainly a characteristic of GKS. The CORE system seems to be deficient in this area. An obvious

example of this difference is that CORE does not define a way to specify the "font" type to be applied to the TEXT primitive, i.e. CORE does not define a font primitive attribute. This obviously leaves the burden of defining the font on the application program. The burden of determining the devices capabilities is on the application programmer. GKS defines the font capabilities of a workstation in its associated bundle table. Software generated fonts are also defined at the GKS level. The application program can "inquire" via GKS in order to determine a workstations capabilities. GKS provides a much better level of abstraction of the capabilities of a graphics device than does CORE.

In general, GKS seems to provide a much more comprehensive and flexible interface between graphics hardware and an application program than does the CORE system.

4.0 KSU IMPLEMENTATION OF TEXT

4.1 General

The implementation of a GKS system at Kansas State University has evolved over a period of time. Therefore, it reflects the efforts of several students over a period of several semesters. The TEXT output primitive was implemented during the 1983 summer semester. At that time the GKS project had only sparse capabilities. Most of the work completed by the prior semester, which was the first, included defining most of the basic data structures, development of minimal LINE and MARKER primitive capabilities on one workstation, and much of the code representing the heart of the GKS system. The 1983 fall semester has the task of melding everyones work and adding enhancements.

The GKS project used an Interdata 3320 host mini under a UNIX* operating system. The Pascal programming language was used for writing the main GKS source code and data structures.

The KSU implementation of the TEXT output primitive differs slightly from the ISO Draft[11]. These deviations

basically are a result of general decisions to bundle all attributes, and provide a minimal subset. Overall, the intent of the ISO Draft was adhered to. The effects of those decisions are discussed below.

Bundle all attributes - The ISO Draft specifies that the non geometric(workstation dependent) attributes are bundled. We do that in WSTEXTTABLE. It also specifies that the geometric (workstation independent) attributes be defined in the GKS statelist, where they represent the current values to be applied to subsequent TEXT primitives. Our implementation bundles the TEXT geometric attributes in GKSTEXTTABLE. The CURTEXTINDEX value is used to index both tables for the attributes to be applied to the TEXT primitive. A discussion of this decision will follow in section 5(five).

Minimal subset of TEXT primitive - In order to provide a minimal subset of the TEXT primitive, certain capabilities defined in the ISO Draft were not incorporated in this implementation. The ability to specify the workstation dependent attributes INDIVIDUALly via the Aspect Source Flag (ASF) capability, was not defined. In addition, the ability

to generate TEXT with the precision of CHARACTER or STROKE was not provided. Though not resolved by this implementation, CHARACTER and STROKE are still valid values for the precision attribute.

4.2 Data Structures

This implementation provides the GKSTEXTTABLE, WSTEXTTABLE and WIS data structures for handling and storage of data and data types associated with the TEXT output primitive. They are described below.

The GKSTEXTTABLE is an array of records of type GKSTEXTREC. Each defined record contains the values of the workstation independent (geometric) attributes CHARHEIGHT, CHARUPVECTOR, CHARPATH, HALIGN and VALIGN.

The data contained in the GKSTEXTTABLE is maintained by the STOTEXT user procedure. See Sec. 4.3 and APPENDIX A & B.

The WSTEXTTABLE is a two dimensional array of records of type WSTEXTREC. The first index into this array represents the workstation identifier. The second index addresses the WSTEXTREC in the specified workstation bundle

table. Each defined record contains the values of the workstation dependent (non geometric) attributes TEXTFONT, TEXTPRECISION, TEXTEXPFACTOR, TEXTSPACING and TEXTCOLOR.

The data contained in the WSTEXTTABLE is maintained by the DEFTEXT user procedure. See Sec. 4.3 and APPENDIX A & B.

The GKSTEXTTABLE and WSTEXTTABLE are initialized by the internal procedures "initgkstables" and "initwstables" respectively. See APPENDIX A.

The Workstation Independent Segment (WIS) is an array of records of type WIELEMENT. These records contain primitive information used by the DISTRIBUTOR in the generation of subsequent LINE, MARKER or TEXT output. Therefore, the WIELEMENT records are of differing types, depending on the type of primitive information stored. For the TEXT primitive, the WIS record stores the value of the CURTEXTINDEX, the length of the text string, the start point of the text string in Normalized Device Coordinates and an array containing the text string.

The WIS records for TEXT are built by the WITEXT procedure, which is called by the DRAWTEXT user procedure.

See Sec. 4.3 and APPENDIX A & B.

4.3 TEXT User Procedures

This section describes the major user procedures developed for the KSU implementation, which allow for the TEXT output capability. These procedures provide for a minimal subset of the ISO definition of TEXT. They can be extended to provide fuller capabilities.

There are seven TEXT user functions(Pascal procedures) defined by the KSU implementation: DRAWTEXT, DEFTEXT, INQTEXT, STOTEXT, QRYTEXT, SETTEXTINDEX, and QRYTEXTINDEX. They are described individually below.

The DRAWTEXT procedure performs an output function. Its purpose is to output character strings to be generated. The start point, specified in world coordinates, and the character string are input. This procedure performs the transformation of the start point from World Coordinates (WC) to Normalized Device Coordinates (NDC), and clips to the world window. Next, DRAWTEXT calls the WITEXT procedure and passes the parameters associated with the current text index, the number of characters that make up the input string, the start point in NDC, and the character string.

The DEFTEXT(DEFine TEXT) procedure performs an attribute setting function. This procedure is used to add or change an entry at the specified text index in the WSTEXTTABLE for the specified workstation identifier. The new entry consists of the desired combination of non geometric attribute values. It should be noted that multiple workstation capabilities were not available, therefore, the workstation identifier was not required input.

The INQTEXT(INQuire TEXT) procedure is used by the application program, or interactive user to determine the attribute settings in the WSTEXTTABLE for the specified text index and workstation. The values for TEXT FONT, PRECISION, EXPANSION FACTOR, SPACING, and COLOR are returned in response to this inquire.

The KSU provides two kinds of "inquire" procedures. The inquire procedures preceded by "INQ" are for inquires at the workstation level. The inquire procedures preceded by "QRY" are for inquiries at the GKS level. In general, inquire procedures are provided to allow the application program or the interactive user to inquire as to the capabilities of a particular workstation, the settings of

GKS level attributes, or system status. Depending on the values returned, the appropriate action is taken.

The STOTEXT(STOre TEXT) procedure performs an attribute setting function. This procedure is used to add or change an entry at the specified text index in the GKSTEXTTABLE. The new entry consists of the desired combination of geometric (workstation independent) attributes.

The QRYTEXT(QueRY TEXT) procedure is used by the application program or interactive to determine the attribute settings in the GKSTEXTTABLE at the specified entry(textindex). The values for CHARACTER HEIGHT, UP VECTOR, PATH, HORIZONTAL ALIGNMENT, and VERTICAL ALIGNMENT are returned in response to this inquire(query).

The SETTEXTINDEX procedure sets the variable "curtextindex" (current text index). If this index is not defined for the GKSTEXTTABLE or WSTEXTTABLE, a default entry is built by this procedure. It should be noted here that the "curtextindex" is the same index into each table. A discussion concerning this will follow in Section 5.

The QRYTEXTINDEX procedure simply returns the current

value of the variable "curtextindex", when invoked.

The input and output parameters for these procedures are discussed in more detail in APPENDIX A.

4.4 Data Flow

This section describes the data flow based on the status of the project when this participant was involved. There were other capabilities being developed concurrent with this effort, however, they will not be included in this description. See FIGURE 6, KSU Data Flow for the TEXT Primitive and APPENDIX B for Code Level Data Flow.

To begin, assume that the GKSTEXTTABLE and WSTEXTTABLE are built, and the current text index has been set by SETTEXTINDEX.

The DRAWTEXT procedure requires the input of the start point, and string of text. DRAWTEXT is invoked by the scanner call (startpoint,textstring). The startpoint is specified in world coordinates, and is the point from which the text string is to begin. The DRAWTEXT procedure performs the normalized transformation of the startpoint from World Coordinates (WC) to Normalized Device Coordinates

(NDC). Clipping to the world window is done here, though not shown in the code section of this report. In addition, this procedure calls the WITEXT procedure and passes the current text index, the number of characters in the text string, the startpoint in NDC, and the text string. At this point the index(curtextindex) is bound to the TEXT primitive.

The WITEXT procedure, which is called by the DRAWTEXT procedure, builds the "WI" record in the Workstation Independent Store (WIS). At this point the "curtextindex" is bound to the TEXT primitive. The index can not be changed, but the attribute values pointed to by it in both the GKSTEXTTABLE and WSTEXTTABLE can be dynamically modified. No further action occurs until the Redraw Work Station (RWS) command is invoked.

The KSU implementation is in the "ASTI" deferral state, as defined by the ISO Draft[11]. This means that the visual effects of output functions such as DRAWTEXT will be generated on the workstation "At Some TIme". "At Some TIme" is when the RWS command is invoked.

Though the data flow, up to this point, does not violate the ISO Draft, there are some basic differences.

The geometric attributes are not yet bound to the TEXT primitive. The Draft specifies that they be bound at creation of the primitive so that they are subject to the same transformations as the primitive. Another difference, is in where clipping is performed. As mentioned, clipping to the world window is currently being resolved in the DRAWTEXT procedure. This was done in order to provide some sort of clipping capability by the initial cutover. Clipping to the screen viewport will be processed by the DISTRIBUTOR at some later date, and will be applied against the primitive just prior to being sent to the workstation. For the purpose of this project, it is not essential that these differences be resolved.

When the RWS command is invoked the WIS records are processed by the DISTRIBUTOR. The WIS records are read, all geometric and non geometric attributes, currently in effect based on the stored "curtextindex", are bound to the TEXT primitive as the Virtual Device Interface (VDI) primitive records are built. The VDI records are then passed on to the workstation device drivers of open workstations - in this case the Chromatics driver, by the DISTRIBUTOR driver. The workstation device driver translates the VDI primitives into Escape Codes(ESC), which are interpreted by the

workstation. The result is the displayed output primitive.

The Data structures and internal procedures which make up the Scanner, Distributor, Distributor Driver and the Chromatics Workstation Driver were developed by the spring semester students. These structures and procedures allowed for, but in some cases were not complete, the generation of a horizontal string of TEXT with only the font type and color attributes applicable. To generate TEXT in this manner was the goal of this implementation project.

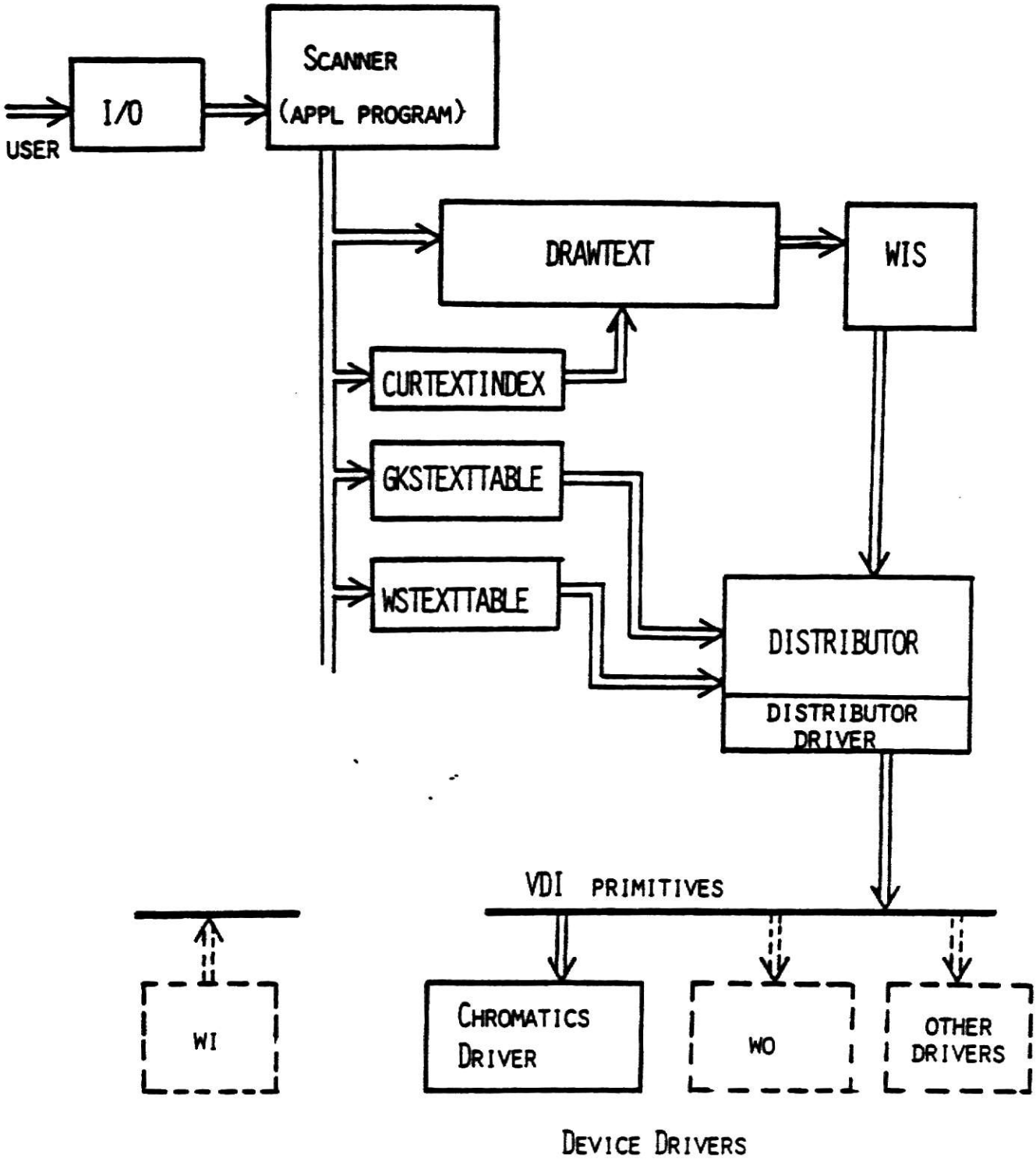


FIGURE 6. KSU DATA FLOW FOR THE TEXT PRIMITIVE

4.5 Possible Enhancements

In order for the KSU version of GKS to generate the TEXT primitive with all attributes applied, much more work is required. Because of the workstation dependent nature of the resultant TEXT generated by the above, it is considered to be of "string" precision by default. No mechanism exists to differentiate between the various precisions, nor does the required software exist to allow the system to respond accordingly. However, with the current design as a base, except for late binding of geometric attributes, the KSU version can be extended to be a richer system.

To achieve a richer system would require some basic changes. The geometric attributes need to be bound at creation. This could be accomplished by providing two text indexes. Early binding is required to create a height vector(parallel to the up vector) and width vector(perpendicular to the up vector) early enough to be influenced by the normalization and workstation transformations. Both vectors are of length equal to the character height. After the vectors have been passed down the viewing pipeline, and end up in device coordinates, they can be influenced by the values of the font coordinate

system. The value of the character expansion factor is used to determine the character width. Additional coding is required to interpret which attributes should be applied, depending on the specified precision attribute, and how they are applied. With these suggested changes, the KSU version of GKS will more closely resemble the TEXT primitive capability defined in the ISO Draft[11].

* UNIX is a trademark of AT&T Bell Laboratories

5.0 DISCUSSION OF IMPLEMENTATION CONCERNS

5.1 General

As with any implementation project there will be deviations from the standard recommendations. Usually such deviations represent a perceived advantage over the standard. However, this is not always the case. Some decisions made early in the KSU project actually produce undesirable effects. A few of these deviations are discussed in this section.

Note that when the KSU project began, the ISO Draft[11] had not yet been released. Therefore, some decisions were made, not as deviations from the standard, but instead, based on other available information.

5.2 Decision to Bundle all Attributes

The bundling of attributes is certainly a major characteristic of GKS. It is also the major difference between GKS and CORE. The ISO Draft[11] specifies that the workstation dependent, non geometric attributes are to be bundled in a workstation bundle table for each workstation. It also specifies, as mentioned in section 4.1, that the

workstation independent, geometric attributes are defined individually in the GKS state list, where they represent the current values to be applied to subsequent primitives. The KSU project took the concept of bundling one step further by requiring all attributes to be bundled. Thus, incorrectly modeling GKS.

While the bundling of workstation dependent attributes provides advantages to the user, over non-bundling, the same advantages are not realized by bundling geometric attributes.

Bundling of workstation dependent attributes takes advantage of the finite capabilities of a particular graphics device. The workstation bundle table contains sets of desirable combinations of these capabilities. As mentioned earlier, the KSU project defines the TEXT workstation dependent attributes in the WSTEXTTABLE. Each record in this table contains a different combination of attributes. The number of these combinations will vary depending on capabilities of a particular workstation. It is likely that not all, but only commonly used combinations are present. Bundling of these attributes provides a beneficial level of abstraction of a devices capabilities to

the application program and programmer.

The "curtextindex" is a variable that is set by the application program or programmer to define the desired combination of attributes that are to be bound to subsequent TEXT primitives. To change the appearance of a primitive becomes merely a matter of changing the value of "curtextindex" into the WSTEXTTABLE, instead of changing the value of each attribute individually, as would be the case in the CORE system.

At first thought, it may seem that the same advantages might be realized by bundling the TEXT geometric attributes. This however, I do not believe will prove to be the case.

The TEXT primitive can be influenced by a large set of geometric attributes. The character HEIGHT and UP VECTOR attributes can take on an almost infinite possibility of values specified in world coordinates. In addition, the character PATH and ALIGNMENT attributes can be assigned a wide range of defined enumerated values. Together, these attributes can provide an infinite number of possible permutations. This could result in an unnecessarily large GKSTEXTTABLE.

If the GKSTEXTTABLE contains a large quantity of records, it would become a burden for the applications programmer to inquire each record in search of an existing desired combination. It would become easier to just establish a new record, thus, increasing the size of an already unmanageable data structure. Any perceived value of bundling the geometric attributes is lost at this point.

One major objective of GKS is to abstract the capabilities of a graphics device or workstation. Bundling the TEXT geometric attributes is an attempt to abstract infinity.

If the KSU project results in a complete system, I recommend that the geometric attributes be unbundled.

5.3 The TEXT Index

Another concern, which compounds the above problem, is that the KSU model specifies only one pointer into both bundle tables. While it is true that the GKS Draft[11] specifies only one textindex, it is also true that the Draft specifies only the bundling of workstation dependent attributes. The value of "curtextindex" is used to index

into the GKSTEXTTABLE and the WSTEXTTABLE, and is bound to the TEXT primitive at creation.

Using the same index for both tables could lead to multiple records containing the same combination of attribute values in each bundle table. As a result, memory is wasted, the data structures become too cumbersome to use properly, and it becomes easier to define new records instead of inquiring. This is a bad situation and definitely distracts from the intent of GKS.

A solution could be to establish a separate index for each table. This would require additional coding and would not completely resolve the problems resulting from the bundling of geometric attributes. The easiest solution would be to unbundle the geometric attributes.

Unbundling the geometric attributes would result in a more manageable and user friendly system. In addition, a system that more closely resembles the GKS standard defined by the ISO Draft[11].

APPENDIX A

SOURCE CODE LISTING

```
(*--------------GRAPHICS KERNEL SYSTEM--------------------

                  procedure drawtext (gks)

-------------------------------------------------------------*)
;  procedure drawtext (startpoint : point
                   ; textstring : tstring )

(* PURPOSE: This function outputs characters to be generated

NEW************
   INPUT PARAMETERS:

       startpoint      - start coordinates (real) for
                         the text string


       textstring      - string of characters

*)
   ; var
       starttext : ipoint;
       scalex,scaley,consx,consy : real

   ; begin (* text *)
         (**** scale text startpoint into unit coordinates ****)

       with gkswintable[curwinindex] do
         begin
           scalex := xmaxndcr - xminndcr;
           scaley := ymaxndcr - yminndcr;
           scalex := scalex / (xmaxwin - xminwin);
           scaley := scaley / (ymaxwin - yminwin);
           consx := xminndcr + scalex * xminwin;
           consy := yminndcr + scaley * yminwin;
                        (* end of scaling *)
           starttext.ix := trunc (startpoint.x * scalex + consx);
           starttext.iy := trunc (startpoint.y * scaley + consy);

      (**** perform clipping if in effect ****)

         end;       (* with gkswintable *)
        witext(curtextindex,maxstring,starttext,textstring)
     end  (* text *)
```

```
(*---------------GRAPHICS KERNAL SYSTEM--------------------

                  procedure deftext (ws)

                      (DEFine TEXT)

!-------------------------------------------------------*)
;  procedure deftext ( textindex : index
                     ; itextfont : fontset
                     ; itextprecision : precisionset
                     ; itextexpfactor : real
                     ; itextspacing : real
                     ; itextcolor : colorset)

(* PURPOSE: This function defines an entry in the WSTEXTTABLE
            of non geometric(workstation dependent) attributes
            function.


   INPUT PARAMETERS:

        wkstation       - specifies which work station's text
 (not in this version)          index table is to be (added to or
                        changed) by this call

        textindex       - specifies where in the text table to
                          (add or replace) the set of text
                          attributes

        itextfont       - this specifies the font to be used
                          for this entry in the text table

        itextprecision  - specifies how the character is to be
                          generated (eg.
                          string - hardware generated writes
                           several characters at a time
                          character - hardware generated
                           writes only one character at a time
                          stroke - all attributes are applied
                           to the text primitive

        itextexpfactor  - specifies the height to width
                          definition of the character fonts

        itextspacing    - this specifies the distance between
                          characters
        itextcolor      - this specifies the color of the text
                          for this entry in the text table

*)
```

```
; begin (* deftext *)

      with wstexttable[textindex]
      do begin (* make entry in WSTEXTTABLE *)
         defined := true
      ;  textfont := itextfont
      ;  textprecision := itextprecision
      ;  textcolor := itextcolor
      ;  textexpfactor := itextexpfactor
      ;  textspacing := itextspacing
      end (* make entry in WSTEXTTABLE *)

  end (* deftext *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM--------------------

                  procedure inqtext (ws)

-------------------------------------------------------------*)
;  procedure inqtext ( textindex          : index
                     ; var otextfont      : fontset
                     ; var otextprecision : precisionset
                     ; var otextexpfactor : real
                     ; var otextspacing   : real
                     ; var otextcolor     : colorset)

(* PURPOSE: This function returns the attributes of a entry
            in the WSTEXTTABLE.


   INPUT PARAMETERS:

      wkstation       - specifies which work station's text
   not in ver 2.0       index table is to be (added to or
                        changed) by this call

      textindex       - specifies where in the text table to
                        (add or replace) the set of text
                        attributes

   OUTPUT PARAMETERS:

      otextfont       - this returns the font type from the
                        set of fonts for the specified entry
                        in the text table

      otextprecision  - this returns the precision type from
                        the set of fonts for the specified
                        index table is to be (added to or
                        changed) by this call

      otextexpfactor  - this returns the height to width
                        definition of the character fonts

      otextspacing    - this returns the distance of the
                        character spacing

      otextcolor      - this specifies the color of the text
                        for this entry in the text table

*)
```

```
; begin (* inqtext *)

      with wstexttable[textindex]
      do if not defined
         then begin (* set return values to default *)
            otextfont      := futura
         ;  otextprecision := textprecision
         ;  otextexpfactor := 1.0 (* check these !! *)
         ;  otextspacing   := 0.0
         ;  otextcolor     := white
              end   (* set return values to default *)

         else begin (* assigns table entry to output vars *)

            otextfont      := textfont
         ;  otextprecision := textprecision
         ;  otextexpfactor := textexpfactor
         ;  otextspacing   := textspacing
         ;  otextcolor     := textcolor
              end   (* assigns table entry to output vars *)

  end (* inqtext *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM--------------------

                  procedure stotext (gks)

-----------------------------------------------------------*)
;  procedure stotext (textindex  : index
                    ; icharheight : real
                    ; icharupvect : point
                    ; icharpath   : textenum
                    ; ihalign     : textenum
                    ; ivalign     : textenum)

(* PURPOSE: This function defines an entry in the device
            independent text table(GKSTEXTTABLE) of attributes
            to be set by the settextindex function.
```

INPUT PARAMETERS:

textindex	- specifies where in the device independent text table to(add to or replace) the set of text attributes
icharheight	- this specifies the height of the chars for this entry in the text table
icharupvect	- specifies the vector of text flow from the alignment point to the point given
icharpath	- specifies the direction that characters are to be generated along the given vector of text flow (eg. up, down, left, right)
ihalign	- the horizontal alignment of the text position
ivalign	- the vertical alignment of the text position

```
*)
```

```
; begin (* stotext *)
      with gkstexttable[textindex]
      do begin  (* make entry in GKSTEXTTABLE *)
         defined := true
      ;  charheight := icharheight
      ;  charupvector := icharupvect
      ;  charpath := icharpath
      ;  halign := ihalign
      ;  valign := ivalign
      end
            (* some atributes may have to default *)

  end (* stotext *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM---------------------

                  procedure qrytext (gks)

-----------------------------------------------------------*)
;  procedure qrytext (textindex      : index
                    ; var ocharheight : real
                    ; var ocharupvect : point
                    ; var ocharpath   : textenum
                    ; var ohalign     : textenum
                    ; var ovalign     : textenum)

(* PURPOSE: This function returns an entry in the device
            independent text table of attributes.


   INPUT PARAMETERS:

       textindex      - specifies which entry in the device
                        independent text table is to be
                        returned

   OUTPUT PARAMETERS:

       charheight     - this returns the height of the chars
                        for an entry in the text table

       charupvect     - the point which specifies the vector
                        of text flow is returned

       charpath       - returns the direction that char-
                        acters are to be generated along
                        the specified vector of text (eg.
                        up, down, left, right)

       halign         - the horizontal alignment value
                        for the text position is returned

       valign         - the vertical alignment value for
                        the text position is returned
*)
```

```
; begin (* qrytext *)
   with gkstexttable[textindex]
   do begin

     ocharheight := charheight
   ; ocharupvect := charupvector
   ; ocharpath   := charpath
   ; ohalign := halign
   ; ovalign := valign
       end
   end   (* qrytext *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM---------------------

                  procedure settextindex (gks)

-----------------------------------------------------------*)
;  procedure settextindex (textindex : index)


(* PURPOSE: This function specifies which entry in the
            text table(GKS and WSTEXTABLES) is to be used
            for the following function calls.
            function calls


   INPUT PARAMETERS:

       textindex      - specifies which entry in the text
                        table is to be used for following
                        calls

*)

;var pt : point
   ; begin (* settextindex *)
       if not wstexttable[textindex].defined
         then
          begin
           deftext(textindex,futura,string,1.0,0.0,white)
           (*textindex,font,precision,expfactor,spacing,color*)
          end
          else begin end
       ; pt.x := 0.0
       ; pt.y := 1.0
       ; if not gkstexttable[textindex].defined
           then
             stotext(textindex,1.0,pt,right,normal,normal)
            (*textindex,charhieght,upvector,path,halign,valign*)
       ; curtextindex := textindex
     end (* settextindex *)
```

```
(*---------------GRAPHIC KERNEL SYSTEM---------------

                procedure qrytextindex (GKS)

-------------------------------------------------------*)
; procedure qrytextindex (var otextindex : index)

(*purpose : This function returns the value of the current
           text index- CURTEXTINDEX

INPUT PARAMETERS :

          NONE !!

OUTPUT PARAMETERS :

     otextindex      - returns the value of the current text
                       index - CURTEXTINDEX.
*)
       ; begin (* qrytextindext *)

          otextindex := curtextindex

         end (* qrytextindex *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM---------------------

                  procedure witext (gks)

-----------------------------------------------------------*)

;procedure witext (textindex:index;
                   nochars  :textcharrange;
                   starttext :ipoint ;
                   textstring :tstring );
  var
    i: integer;

  begin
    if wistop < maxwielements then
      begin
        wistop := wistop +1;
        with wis[wistop] do begin
          opcode := ltext;
          tinx   := textindex; (* fix *)
          tcn  := nochars;
          tp0.ix := starttext.ix;
          tp0.iy := starttext.iy;
          for i := 1 to maxstring do
              tstr[i] := textstring[i];
        end;  (* with wis[wistop] *)

      end  (* if wistop < maxwielements *)
      else begin end
  end  (* witext *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM--------------------

                procedure initgkstables (ws internal)

-----------------------------------------------------------*)
;  procedure  initgkstables

(* PURPOSE: This function initializes all gks tables and
            current pointers to default values. ( some
            entries in the tables are to be set as
            NOT DEFINED )

*)

   ;  var
         count : integer
  ADD -->  ;pt : point

   ;  begin (* initgkstables *)
        curwinindex := 1
      ; stowindow (curwinindex, 0.0, 1.0, 0.0, 1.0)
      ; stoview (curwinindex, 0.0 ,1.0,0.0,1.0)
      ; for count := 2 to minindex
       do gkswintable[count].defined := false
      ;  curtextindex := 1
  ADD -->  ; pt.x := 0.0
  ADD -->  ; pt.y := 1.0
  ADD --> ; stotext (curtextindex, 1.0, pt, right, normal,normal)
     ; for count := 2 to maxindex
       do gkstexttable[count].defined := false
      end (* initgkstables *)
```

```
(*---------------GRAPHICS KERNEL SYSTEM--------------------

                procedure initwstables (ws internal)

-------------------------------------------------------------*)
;  procedure  initwstables

(* PURPOSE: This function initializes all ws tables and
            current pointers to default values. ( some
            entries in the tables are to be set as
            NOT DEFINED )

*)

   ;  var
         count : sindex
      ;  count1 : index

   ;  begin (* initwstables *)

            curlineindex := 1
         ;  curmarkerindex := 1
         ;  curtextindex := 1
         ;  curwinindex := 1
         ;  for count1 := 1 to minindex
            do begin (* set table entries to undefined *)
               wslinetable[count1].defined := false
            ;  wsmarkertable[count1].defined := false
            ;  wstexttable[count1].defined := false
            ;  wsviewtable[count1].defined := false
            end (* set table entries to undefined *)

         ;  defline (curlineindex, solid, 1, white )
         ;  defmarker (curmarkerindex, dot, 1, white )
  ADD --> ;  deftext (curtextindex, futura, string, 1.0, 0.0,
                                                white )
         ;  for count1 := 1 to minindex
            do wstable[count1].state := inactive
         ;  wstable[3].state := active
      end (* initwstables *)
```

```
(*-------------- GRAPHICS KERNEL SYSTEM --------------------

              CONSTANT DECARATIONS for TEXT

-----------------------------------------------------------*)

        ;       MININDEX = 10
        ;       MAXSTRING = 41
        ;       MININDEX = 5

        (* END OF GKS CONSTANT DECLARATIONS *)


(*-------------- GRAPHICS KERNEL SYSTEM --------------------

               TYPE DECLARATIONS  for TEXT

-----------------------------------------------------------*)

        ;       INDEX = 1..MAXINDEX
        ;       SINDEX = 1..MININDEX
        ;       TEXTCHARRANGE = 1..MAXSTRING
        ;       COLORSET = (BLACK, BLUE, GREEN, CYAN, RED,
                                MAGENTA, YELLOW, WHITE)
        ;       FONTSET = (ROMAN, FUTURA, SCRIPT, ITALIC)
        ;       PRECISIONSET = (STRING, CHARACTER, STROKE)
        ;       TEXTENUM = (NORMAL, UP, DOWN, LEFT, RIGHT, CENTER,
                                TOP, BOTTOM, CAP, HALF, BASE)
        ;       OPTYPE = (LTOP, LEND, LCALL, LLINE, LMARKER,
                                LTEXT, LCLEAR, LCONT)

        ;       GKSWINREC =
                        RECORD
                                DEFINED : BOOLEAN
                        ;       XMINWIN : REAL
                        ;       XMAXWIN : REAL
                        ;       YMINWIN : REAL
                        ;       YMAXWIN : REAL
                        ;       XMINNDCR: REAL
                        ;       XMAXNDCR: REAL
                        ;       YMINNDCR: REAL
                        ;       XMINNDC : UNIT
                        ;       XMAXNDC : UNIT
                        ;       YMINNDC : UNIT
                        ;       YMAXNDC : UNIT
                        END   (* GKSWINREC *)

        ;       POINT =
```

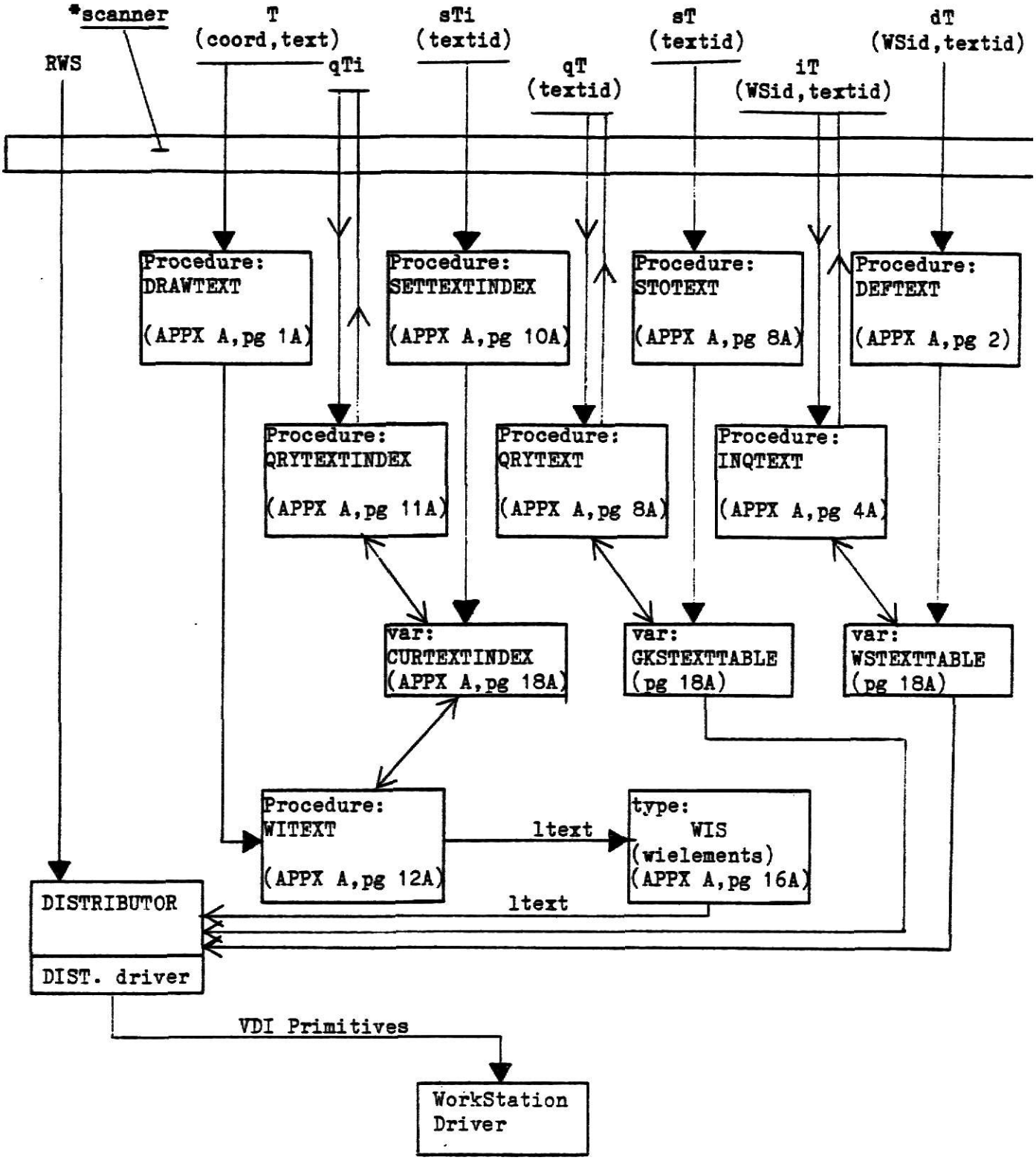
```
                RECORD
                        X : REAL
                ;       Y : REAL
                END  (* POINT *)

;       GKSTEXTREC =
                RECORD
                        DEFINED : BOOLEAN
                ;       CHARHEIGHT : REAL
                ;       CHARUPVECTOR : POINT
                ;       CHARPATH : TEXTENUM
                ;       HALIGN : TEXTENUM
                ;       VALIGN : TEXTENUM
                END  (* GKSTEXTREC *)

;       IPOINT =
                RECORD
                        IX : INTEGER
                ;       IY : INTEGER
                END  (* IPOINT *)

;       WSTEXTREC =
                RECORD
                        DEFINED : BOOLEAN
                ;       TEXTFONT : FONTSET
                ;       TEXTPRECISION : PRECISIONSET
                ;       TEXTEXPFACTOR : REAL
                ;       TEXTSPACING : REAL
                ;       TEXTCOLOR : COLORSET
                END  (* WSTEXTREC *)

;       TSTRING = ARRAY [1..MAXSTRING] OF CHAR

;       WIELEMENT =
                RECORD
                CASE OPCODE: OPTYPE OF
                        LTOP : (A: INTEGER)
                ;       LEND : (B: INTEGER)
                ;       LCALL : (C: INTEGER)
                ;       LLINE : (TNX :INDEX
                                  :N :LINERANGE
                                  :PO,
                                   P1 : IPOINT)
                ;       LMARKER : (MINX : INDEX
                                    :MN : MARKERRANGE
                ;                   :MPO : IPOINT)
                ;       LTEXT : (TINX : INTEGER
                                  :TCN : TEXTCHARRANGE
                                  :TPO : IPOINT
                                  :TSTR : TSTRING)
```

```
                ;       LCLEAR : (G      : INTEGER)
                ;       LCONT : (CP1,
                                 CP2,
                                 CP3     : IPOINT)
              END  (* WIELEMENT *)

;       TGKSTEXTTABLE = ARRAY [INDEX] OF GKSTEXTREC
;       TWSTEXTTABLE = ARRAY [SINDEX] OF WSTEXTREC


        (* END OF GKS TYPE DECLARATIONS for TEXT *)
```

```
(*-------------- GRAPHICS KERNEL SYSTEM ---------------------

                VAR DECLARATIONS  for  TEXT

-----------------------------------------------------------*)

        ;       CURTEXTINDEX : INDEX
        ;       GKSTEXTTABLE : TGKSTEXTTABLE
        ;       GKSWINTABLE : TGKSWINTABLE
        ;       WISTOP : INTEGER
        ;       WSTEXTTABLE : TGKSTEXTTABLE
        ;       TEXTSTRING : TSTRINGE

       (* END GKS VAR DECLARATIONS for TEXT *)
```

APPENDIX B

CODE LEVEL DATA FLOW



CODE LEVEL DATA FLOW

(*scanner for implementation purposes only)

LIST OF REFERENCES

(1) Baily, C. "Graphics standards are emerging-slowly but surely." Electronics Design , Vol 31, 20 Jan. 1983, 103-110.

(2) Bergeron, R.D., Bono, P.R., Folly, J.D. "Graphics Programming using the Core System." ACM Computing Surveys , 10, No.4 (Dec. 1978), 389-443.

(3) Bono, P.R., et.al., "GKS- The First Graphics Standard." Computer Graphics and Applications , 2, No.5 (July 1982), 9-23.

(4) Buttuls, P. "Some Criticisms of the Graphical Kernal System (GKS)." Computer Graphics , 15, No. 4 (Dec. 1981), 302-305.

(5) Encarnacao, J., et. al., "The Workstation concept of GKS and the resulting conceptual differences to the GSPC Core System." Computer Graphics , 14, No. 2 (July 1980), 226-229.

(6) Fichera, R. "Graphics compatibility." Mini-Micro Systems , (July 1983), 189-194.

(7) Hatfield, L. "GKS and the Alphabet Soup of Graphics Standards." Computer Graphics , 16, No. 2 (June 1982), 161,162.

(8) Nicol, C.J. and Kilgore, A.C. "A Pascal Implementation of the GSPC Core Graphics Package." Computer Graphics , 15, No.4 (Dec. 1981), 327-335.

(9) Shrehlo, Kevin B. "ANSI graphics standards coalesce around the international kernel." Mini-Micro Systems , (November 1982), 175-186.

(10) Stluka, F.P., Saunders, B.F., Slayton, P.M., Badler, N.I. "OVERVIEW of the UNIVERSITY OF PENNSYLVANIA CORE SYSTEM STANDARD PACKAGE IMPLEMENTATION." Computer Graphics , 16, No. 2 (June 1982), 177-185.

(11) "Draft International Standard ISO/DIS 7942." Information Processing Graphical Kernal System (GKS) Functional Description, 17 Jan. 1983.

(12) "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH.", Computer Graphics quarterly report of SIGGRAPH-ACM, Vol 11, No 3 (fall 1977), whole issue.

(13) "Graphics that any computer can use." Business Week , 30 Aug. 1982, p.62A.

(14) Foley, J.D. and Van Dam, Andries, Fundamentals of Interactive Computer Graphics , Addison-Wesley Publishing Co., 1982.

AN IMPLEMENTATION OF THE GKS TEXT PRIMITIVE

by

HENRY BRADLEY FOUT III

B.S.,B.A. Loyola College of Baltimore, 1974

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

The Graphical Kernel System is a functional software interface between graphics application programs and configurations of graphical input and output devices. GKS shields the peculiarities of these devices from an application program by providing a set of callable functions which abstract their capabilities. As of the writing of this report, GKS is in the process of being accepted as the first international graphics standard.

Because of the interest generated by this pending standard, the Kansas State University Department of Computer Science began a project to study and evaluate GKS. The project included the implementation of a minimal version of GKS. Each student involved studied GKS in general, then focused their implementation and subsequent report on a subset of GKS. This is one of several reports resulting from the KSU project to study GKS.

This report provides a detailed description of the GKS TEXT graphical output primitive and its associated implementation. After a definition and history of GKS, the report focuses on the TEXT output primitive. Throughout a tutorial on TEXT, comments are used to highlight the areas

where the KSU specification differs from the Draft International Standard (ISO/DIS) 7942[11] specification. In addition, GKS is compared with the CORE graphics system by way of a review of the CORE system. The design and implementation of the KSU version of TEXT is analyzed. Because of decisions made early in the design phase, and deviations from the Draft[11], the KSU version yields special concerns. This report concludes with a discussion of these concerns.

1430-74
CD-53