A VDI DRIVER FOR SOLTEC PLOTTERS

by

EUGENE JANOVITZ

B.E., Pratt Institute of Brooklyn, 1973

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS:

In anticipation of the successful conclusion of the work required to complete the Master's degree in Computer Science, the author wishes to express his gratitude to the many people who helped him with this study.

His committee members, Dr. William J. Hankley, Dr. Virgil E. Wallentine and Dr. Rodney M. Bates deserve praise for their guidance. Dr. Hankley merits special thanks for his patience, understanding, and encouragement while serving as the principal advisor.

Many thanks go to Harvard Townsend, for helping the author to learn the UNIX system initially, and his continued support with the system thereafter. The Cole's, Mary Beth and Mick, for their always cheerful, and timely assistance in the area of administrative support.

Finally, special thanks goes to his friend, Luz Suarez, for her support and for her patience in typing most of the report.

CHAPTER ONE

INTRODUCTION

The growing importance of computer graphics in the past decade has led to several proposals for graphics standards. Two main proposals are close to acceptance as international and national standards: the Graphical Kernel System (GKS) [ISO], which was developed originally by DIN (Deutsches Institut fur Normung), the official standard body of West Germany; and the Core System [Bailey], a de facto standard created by the SIGGRAPH Graphics Standards Planning Committee (GSPC) of the Association for Computing Machinery.

The definition, and basic differences between GKS and Core are given in Chapter Two. Another standard, the Virtual Device Interface (VDI) [ANSI,Aug.27,1982], is being developed in U.S. by ANSI Subcommittee -X3H33 of the X3H3 Computer Graphics Committee. The VDI, which functionally complements GKS, is defined in Chapter Two.

Overview

Together, GKS and VDI provide functional specifications for a device-independent graphics software package. A device-independent graphics system is one that

works with any graphics output devices such as a raster display, a directed beam display, an electrostatic plotter, or a pen plotter without any changes to its requests for graphics output services. A basic subset of a device-independent graphics software system, shown in Figure 1, on page 12, has been implemented at KSU during the Spring term of 1983.

In a typical device-independent system, an application program calls on GKS a standardized set of device-independent graphics functions (GKS) which in turn call on device drivers ("virtual" graphics input and output devices) through VDI which is an interface internal to the graphics support package.

Each device driver is dedicated to a particular graphics input/output device. A device driver accepts virtual commands from the device-independent software through VDI and translates them into a language the device understands.

The Spring term implementation of GKS at KSU supported only one device driver, namely the Chromatics driver. During the five week of the 1983 summer term, additional device drivers and enhancements have been developed. One of these was the development of a VDI driver for SOLTEC plotter, which is presented in this report. The

SOLTEC driver software compiled successfully, however, testing was only partially completed due to the limited time frame, as explained in chapter five.

Paper Organization

This paper is organized in five chapters. The first chapter has summarized this project. In Chapter Two, the history of the graphics standards, is briefly reviewed, and the main differences between GKS and Core are discussed. The current status of the related events are then presented, and the future outlook of the state-of-the-art is evaluated. Chapter Three begins by first defining and then introducing the main features of GKS. Next, it continues by focusing on VDI, and it concludes with a short discussion on two related graphics models. Chapter Four, presents the SOLTEC plotter's hardware characteristics and its firmware capabilities. In Chapter Five, the Driver structure is presented and the design decisions made during the implementation of the SOLTEC driver are evaluated. This last chapter concludes by reporting the results of this project.

CHAPTER TWO

GRAPHICS STANDARDS

The earliest commonly used graphics standards were created by a small number of manufacturers who produced various successful graphics devices and thus established dominance in the field. When these companies implemented software for control of their graphics devices, they "established" the common graphics language for similar applications. Thus de facto standards were born.

Lack of graphics standards has forced consumers to be locked into one vendor for total solution to their graphics application needs. The high cost of such systems have often prohibited the acquisition of graphics where otherwise it would have been highly desirable. On the other hand, few vendors can afford to fund development of graphics projects which have no guarantee of meeting standards that have not yet been established. Hence, an explosive increase in graphics applications which has been expected for a long time did not occur.

Brief History.

As the need for graphics standards became acute,

standards organizations have been formed to provide a framework so that standards that represent a consensus can be developed and approved. A chronology of significant graphics standards development [Bailey] is shown in Table 1, on page 6.

Both GKS and Core grew out of the original proposal efforts by the ACM/Siggraph. The two have branched apart in significant ways, each with its specific features and flaws.

Core is the older and better-known of the two. Core addresses a greater range of issues and features of graphics activity. However, Core is much more a set of recommendations than specifics of implementation. Because of this, many differing products have been developed which are based on the Core recommendations, yet which differ in slight or major ways from one another [Dern].

GKS addresses a smaller portion of the total graphics question but does so in greater detail. One major concern with Core has been the use of "current position" for cursor control. Core utilizes relative positioning of the cursor. Any error, such as jostling a pen-plotter, losing a coordinate, a power spike altering a value, can throw the cursor position off. This causes all subsequent display output to be incorrect, until the cursor position can be reset. GKS avoids this problem by using an absolute

Table 1: Chronology of Significant Graphics Standards Development.

1974	The ACM SIGGRAPH Graphics Standards Planning Committee (GSPC) is formed after a workshop on machine-independent graphics at a National Bureau of Standards meeting.
1976	An international workshop on methodology in computer graphics is held in Seillac, France (Seillac 1).
1977	GSPC Core system and DIN Graphical Kernel System (GKS) specifications are published.
1979	GSPC Core system and GKS specifications are revised; GKS is accepted as International Standards Organization's working draft. Work begins on Virtual Device Interface (VDI) and Virtual Device Metafile (VDM). Canadian government approves Teldon (videotex) field trials.
1981	North American Presentation Level Protocol Syntax (NAPLPS) is announced by American Telephone & Telegraph Co.
1982	GKS is registered as proposed ISO standard; VDI, VDM, and NAPLPS receive industry support.
1983	GKS was accepted as the official ISO standard.
1984	NAPLPS, and VDM are expected to be submitted as U.S. standards.

technique for positioning.

While GKS includes only two-dimensional operations, the viewing operation in the Core system can be two- or three-dimensional. The three-dimensional operations project the image into an X-Y-Z space with either a perspective or a parallel projection. Core's complexity led to problems in satisfying competing viewpoints. When Core was turned over to the ANSI X3H3 Committee on Graphics Standards, it failed to win approval as a national standard, and when it was submitted to the International Standards Organization[1] in 1979, it was again not accepted.

Current Status

At the June 1982 meeting of ISO, GKS was unanimously accepted by ISO. This bring GKS to the status of DIS - Draft International Standard - where the technical contents are frozen, and only editorial clarifications and resulting technical corrections will be made.

In the United States, standards are developed

---

1. ISO (International Standards Organization) is the international equivalent of ANSI, the American National Standards Institute. ISO is in charge of creating and establishing standards for all sorts of things, of which graphics programming languages is one.

jointly by representatives from industry, the government and the affected user communities. Compliance, however, with ANSI is voluntary.

Since its formation in 1977, the ANSI X3H3 committee has been chaired by Dr. Peter Bono and now consists of representatives from more than 40 organizations. Three subtask groups are developing draft proposed American National Standards. The subtask group writes a document listing technical issues, possible resolutions for each issue, pro and con arguments and the recommended resolution for each issue. Once all of X3H3 agrees on the issue resolution, the subtask group writes a draft standard, and it is sent for review by X3H3, then X3, then the general public. A each stage comments must be considered and either incorporated or reasons provided for their rejection.

Expected acceptance of GKS by ISO brings new pressure to ANSI X3H3 to select GKS as the U.S. standard. ANSI policy is to match world standards whenever possible. "No good technical reasons not to accept GKS exist" claims Dr. Bono, who is the chairman of the U.S. delegation to the ISO proceedings as well. "My own personal view is that enough changes have been made to GKS that we (ANSI) should adopt it unchanged as the American national standard" [Dern].

Future Outlook

In late 1983, GKS was accepted as an international standard. This event will result in more vendor development, more system purchases, more emphasis and interest in graphics by new vendors and users, announcements by major vendors of GKS compatible systems, and support for GKS.

The creation of high-level standards plays an important role in speeding efforts and reducing costs in graphics applications programming, concurs Louis Doctor, president of Raster Technologies, Inc., in Billerica, Mass [Dern]. "A high-level interface means it's possible to design an intelligent graphics workstation which can take over more of the sophisticated calculations involved in complex graphics. Without the existence of these interfaces, there wouldn't be a place to take the process from. And definition of standard graphics functions allows creation of subroutine packages which, in turn, can provide those functions from a host computer - and existence of such packages lets programmers who aren't graphics experts put together graphics-using systems" [Dern].

According to one industry watcher, it is most likely that "Core in itself will fade away. Existing Core-based systems may remain, but will not grow. And vendors of Core

based systems will quickly announce support and compatibility for GKS as well, as fast as the coding can be done, and facilities to help convert existing applications to fit into what will be the GKS mainstream. In other words, Core itself may go away, but the vendors, the users, the systems and the applications will stay and prosper" [Dern].

In early summer of 1983, Dr. Bono reported that his committee, in cooperation with international participants, has been working on a "Rich/3D standard" containing as much as possible of both Core and GKS as proper subsets.

This Rich/3D standard would address many issues not tackled fully by either Core or GKS. Furthermore it would provide an economically, politically and technically viable solution to the need for additional graphics standards for applications programming.

CHAPTER THREE

GRAPHICS SYSTEMS

The intent of this chapter is to focus on the Virtual Device Interface. However, before introducing it to the reader, it is of benefit to define and present GKS. To complete the picture, two related models are also presented.

The Graphics Kernel System

Current standards efforts are aimed at the two main interfaces in computer graphics systems: the application programmer interface level, and the device interface level (see Figure 1 on page 12). At the programmer level, GKS provides a standard interface between the application program and the graphics utility programs. The VDI standardizes the interface between graphics utilities and device drivers.

GKS allows portability of graphics application programs between different computer installations by providing a consistent interface implementable in high level portable languages such as FORTRAN or PASCAL. This is achieved by standardizing the way in which graphics functions are accessed and by controlling the appearance of

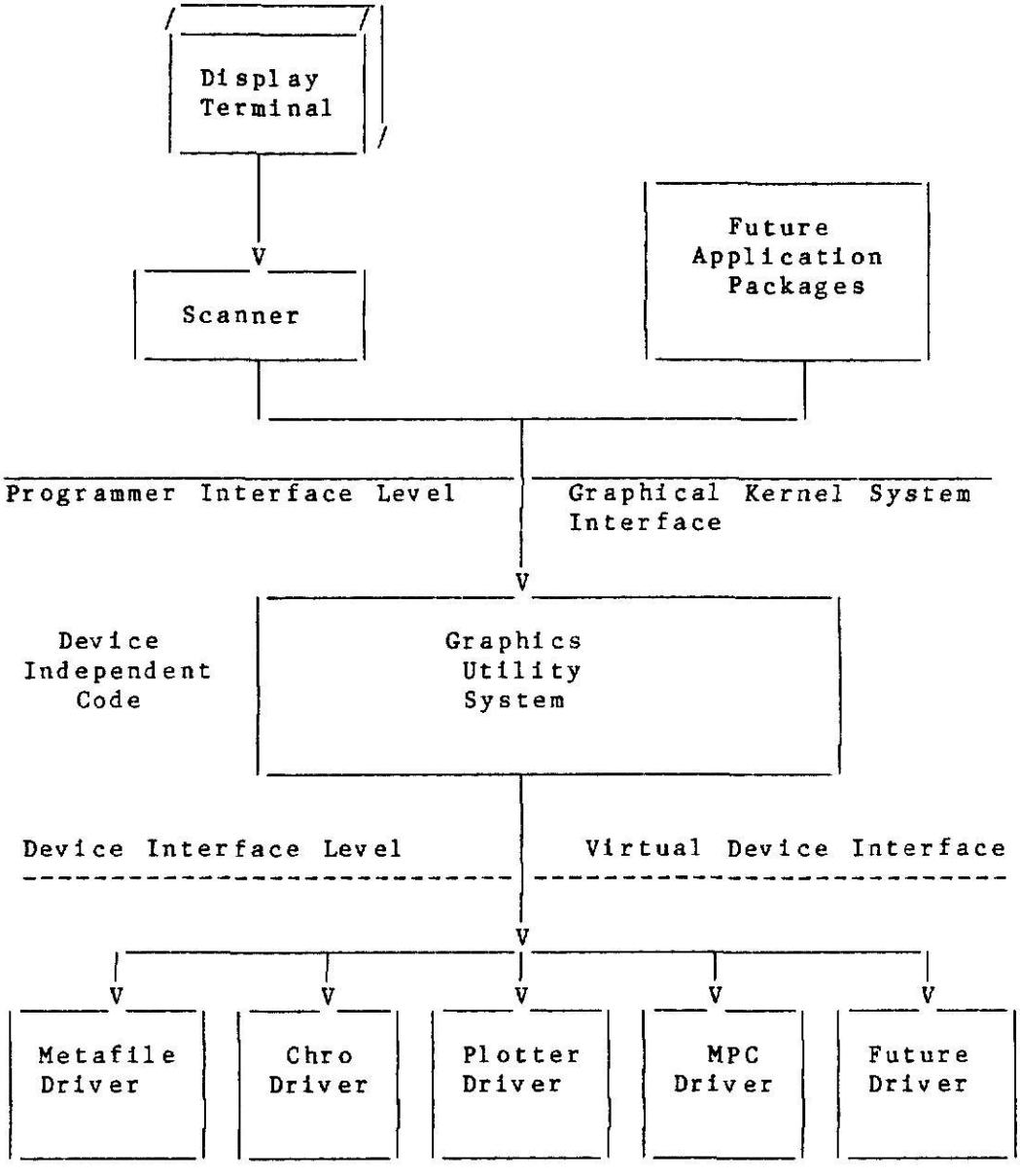

Figure 1: Graphics System Model at KSU.

graphics primitives on a idealized virtual device.

GKS defines only a language independent nucleus of a graphics system. For integration into a language, GKS is embedded in a language dependent layer containing the language conventions, for example, parameter and name assignment. The layer model represented in Figure 2, on page 14 illustrates the role of GKS in a graphical system. Each layer may call the functions of the adjoining lower layers. In general the application program uses the application oriented layer, the language dependent layer, other application dependent layers, and operating system resources. All workstation [Encarnacao] capabilities that can be addressed by GKS functions are used only via GKS.

GKS divides the output of graphics into two distinct parts. The first produces output on a virtual device space called normalized device coordinates. The second allows individual workstations to interpret this virtual space in a way specified by the application program. The workstation translates the normalized coordinates to real device coordinates for display. A GKS workstation is a single display surface and one or more input devices [DIN]. Workstation types are similar to the facilities that would be available at a plotter, storage tube, or refresh display. An operator can have a number of GKS workstations under his control at the same time. Different workstations may be set

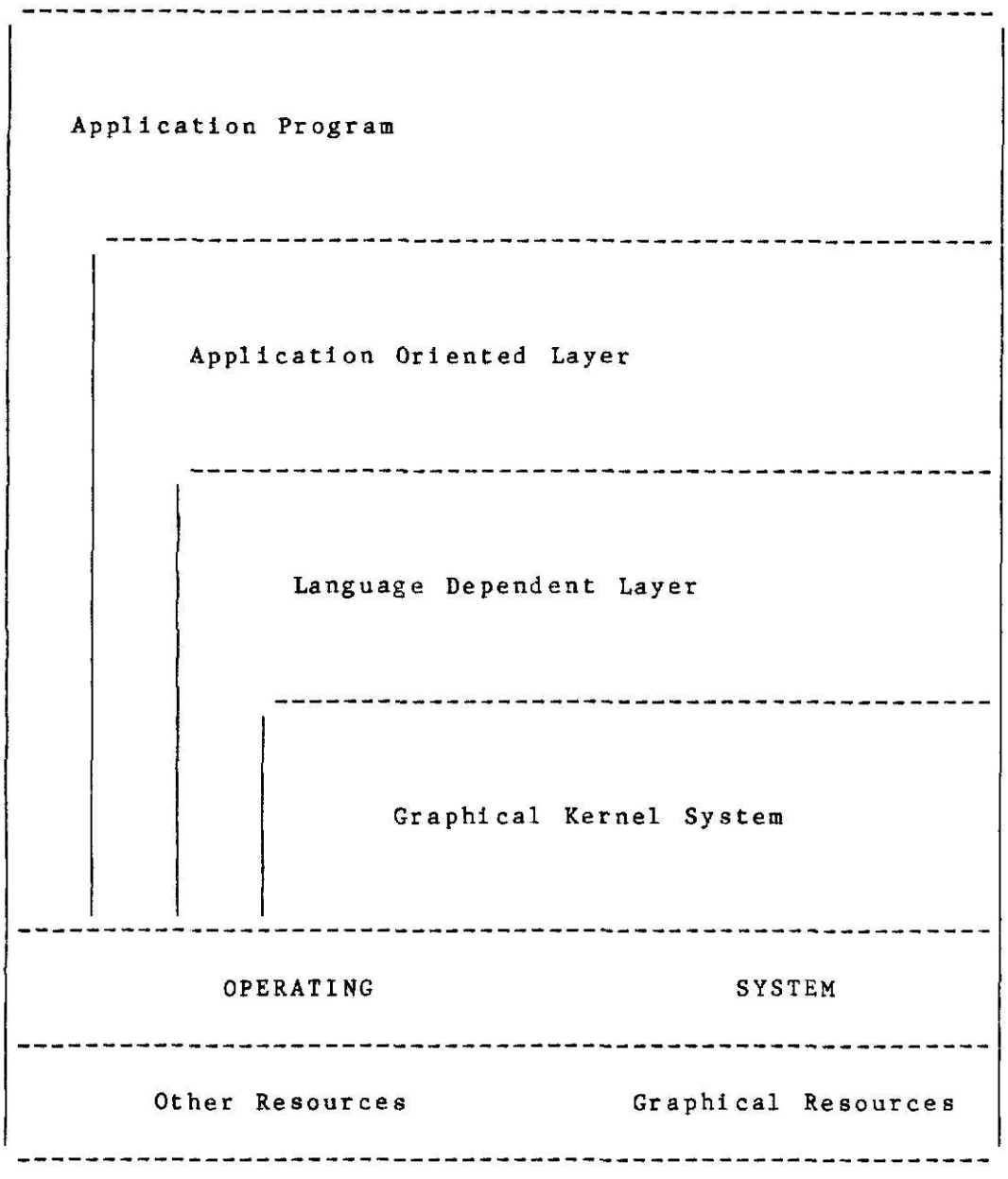

Figure 2: Layer Model of GKS.

to view different parts of the complete virtual picture. Moving from one installation to another requires the application program to redefine the workstation definitions-an operation normally performed at the beginning of a program rather than in the main control flow.

The graphical output that is generated by GKS is built up from two groups of basic elements called output primitives and primitive attributes [Gallop]. The six output primitives provided by GKS are: polyline, polymarker, text, fill area, cell array, and generalized drawing primitive (GDP). The first three form a basic drawing set. The polyline primitive draws a sequence of vectors (straight lines) between pairs of points that form a sequence specified as an array. A single line is just a special case of the polyline, defined by its two endpoints. The polymarker primitive generates marker symbols centered at each specified point. The text primitive allows text strings to be displayed at any position with any orientation.

The remaining three primitives show the increasing importance of raster devices which have fill and pixel-array primitives. The fill operation paints the interior of a closed polyline (polygon) with a specified color or pattern (such as a crosshatch). The pixel-array primitive allows a two-dimensional array of pixels of different colors, called

a cell, to be defined. The cell may then be repeated over an arbitrary area simply by giving the desired boundaries. The final primitive, GDP, is an escape function to allow special geometric primitives such as curves, circular arcs, and elliptic arcs to be defined. The objects are characterized by an identifier, a set of points and additional data.

Associated with each output primitive are attributes that alter the object's appearance. For example, the polyline primitive has line-type (solid, dashed, etc.), width, and color attributes. Polymarkers have attributes of style, size, and color. Text differs from the other primitives in splitting the attributes into two classes. The first type controls the geometric aspects of the text. These attributes are work station independent and are expressed in world coordinates where appropriate. The second type controls the nongeometric aspects of text such as front, precision, and color. The motivation [Bono,July,1982] for this split is that the overall form and shape of the text must fit with the graphic output on all devices and so should be device-independent.

Graphical input functions provided by GKS can be grouped into five classes: choice, locator, pick, string, and valuator. This flexibility allows GKS to support the optimum input device for a particular workstation. All five

classes of input can be obtained from a workstation by use of three mechanisms. These three input types are: request, sample, and event.[1] The result is versatility through which the full potential of the graphics man/machine can be realized.

To ensure the user's view of the system is correct, GKS allows the definition of multiple window/viewports, all existing simultaneously. A picture is defined in the application's conceptual "world space," called world coordinate (WC) space. A window-viewport transformation is done by mapping the world coordinates into a normalized device coordinate (NDC) space by defining a working region (window), in world-coordinate space, and mapping it onto a specified area of the normalized device coordinate space (viewport). The normalization transformation includes translation and scaling. Although NDC space conceptually extends to infinity, the part of NDC space in which the viewport must be located and that can be viewed at a workstation is the closed range [0.1]x[0.1] This space appears identical to all devices in the system, it acts as an intermediary space between applications and devices. In

---

1. It is left to the reader to explore the various combinations between input classes and types. Some of the best sources on this found by the author are [ISO], [DIN], and [Bono,July,1982].

the next step, the workstation transformation, the NDC space is transformed into the device coordinates (DC) of the workstation. When multiple workstations are used, each may have a distinct view of the application by setting its own workstation window. The last transformation allows the workstation to set a viewport, which can be used for scaling and translating the original picture. In this way it is possible, e.g. to use the full display space of an interactive workstation and to have simultaneously a drawing in correct scale on a plotter.

A final concept that deserves mentioning is the GKS segment. In an interactive environment the complete picture frequently needs to be split into a number of objects or segments that can be manipulated independently. One may wish to highlight a particular part of the picture, remove it for some reason, or move parts of it around. This is achieved through a segment transformation matrix, which may be modified after the segment is defined. A picture may be defined on a refresh display, made up of segments, and then when completed one may wish to copy it to a plotter.

This section is intended to give a flavor of the GKS concepts and facilities. It does not cover all features, and many of the facilities are oversimplified. No attempt has been made to describe the GKS level structure, which allows only some of the functionality to be available in

particular implementation.

The Virtual Device Interface

The Virtual Device Interface has the following definition. "The Virtual Device Interface (VDI) is a standard functional and syntactical specification of the control and data exchange between device-independent graphics software and one or more device- dependent graphics device driver or conforming graphics devices."

VDI makes all devices appear as identical virtual graphics devices by defining a standard input/output protocol. The unique characteristics of the physical graphics device are isolated in the device driver software module. VDI includes the following concepts:

Data interface, not a procedural interface.
Partioned into required and nonrequired sets of functions.
Device-independence for both output and input.
Functionally compatible with the Virtual Device Metafile (VDM) standard.
Consistent with the Open System Interconnection (OSI) model.

The VDM is defined in this section three paragraphs below. The Virtual Device Interface functions are divided into two sets: a lean set required to be supported by all device drivers or devices; and a rich set which is not

required to be provided by device drivers or devices. The application package using the VDI can use the lean set of functions without inquiry. Use of the functions in the rich set should be preceding by an inquiry to determine whether the function is supported or not. If the function is not supported, the application can emulate the function using available supported functions. This scheme allows for device-independence while providing access to the richer functions of more powerful (intelligent) devices.

Consistent with GKS, VDI's functional capabilities include, graphical output primitives, associated primitive attributes, segmentation, and input. In addition, VDI control functions include virtual device initialization and termination, virtual device status, coordinate space specification, and addressability selection. For each virtual device a single physical device is required, but the mechanism for directing the processing for multiple devices is left as a responsibility of the application.

To aid the programmer, VDI provides an inquire[2] capability, which falls into the nonrequired category. Such an inquiry must always return a "not available" response for any such function that is not implemented in the VDI.

---

2. This is a GKS capability as well

Similar to the GKS concept of a workstation viewport, the VDI supports a "display surface" viewport. The display surface can be thought of as a region with (0,0) located at the lower left corner of the display surface and (1,1) located in the upper right corner of the display surface. The display surface viewport is then specified by two coordinates that represent the lower left and upper right corners of the viewport. The range of logical device units can be specified by the VDI user. Logical device units are mapped onto the display surface viewport. Logical device units may have an aspect ratio (which is implied by the Logical Device Coordinates range) that is different from the display surface viewport.

The Virtual Device Metafile (VDM) [ANSI,Aug.13,1982] is a way to transport the definition of a picture in a device-independent representation over space and/or time. The metafile consist of the required set of VDI functions and a fixed selection of the nonrequired VDI functions. The metafile virtual device driver sits below the VDI.

Often, the capabilities specified by the VDI standard are not provided by a particular graphics device. In some cases the device-driver software emulates the required function. For example, four line styles are specified by VDI: solid, dashed, dotted, and dashed-dotted. If a graphics device does not have the ability to produce

these directly, their automatic generation is emulated in software. For example, if a dotted line cannot be produced by a device, the required line style may be produced by generating a series of short solid lines with intervening spaces.

It is likely that most systems built on top of the VDI will not have the concept of current position at the programmer's interface. However, the VDI document [ANSI,Aug.27,1982] states that the VDI will support devices that have an inherent current position capability. Therefore, it is natural that the VDI be a current position interface. There are demonstrated benefits to this approach which include the following advantages: compactness of display commands, minimization of data flow across the VDI, and consistency with the many devices that have an inherent current position. If the VDI did not include the concept of current position and a higher level current position based package, such as Core, was built on top of the VDI, the use of current position on a physical device supporting current position would be impossible.

Related Models

North American Presentation Level Protocol Syntax (NAPLPS) or simply videotex, is at the same functional level as the VDI. It was developed by a team at Bell Laboratories

as an extension of graphics developments in the Canadian Telidon videotex system [Fleming]. NAPLPS (pronounced "nap lips") has been adopted by AT&T as a standard for transmitting text and graphics over telecommunication lines. Videotex differs from the work of X3H3 in several ways. First, it is application-specific to videotex, an information distribution system which employs raster displays, i.e., TV sets; second, although graphic capability is part of the videotex repertoire, only the most rudimentary input capability is provided. Finally, it is a data stream definition rather than a programming (software) interface.

The videotex proposal has been standardized by ANSI X3L2, which deals with character set codes, and relies heavily on the Picture Description Instructions produced by the Canadian Telecommunication Research Center. In some computer-graphics implementations, NAPLPS probably will "sit below" the more general device interface VDI.

A graphics extension to CP/M called GSX, for Graphics System Extension (developed jointly by Digital Research Inc., and Graphics Software Systems), provides device-independent graphics based on the Virtual Device Interface [Perry]. The GSX architecture consists of two principal components: the Graphics Device Operating System (GDOS) and the Graphics I/O System (GIOS). The GDOS the

device-independent portion of GSX-is analogous to the basic disk operating system (BDOS) of CP/M. GDOS recognizes and responds to graphics service requests.

The GIOS is analogous to the Basic I/O System (BIOS) in standard CP/M. This device-dependent portion contains the dedicated drivers that match specific devices to the Virtual Device Interface. All graphics capabilities of the devices are accessed through the drivers. A complete specification of the Virtual Device Interface explaining the operation and requirements of each of the 33 operation codes (see Table 2, on page 26) is part of GSX documentation.

Table 2: GSX Graphics Input Output System Operation Codes.

Opcode	Description
1	Open Workstation-initialize a graphics device (load driver if necessary)
2	Close Workstation-stop graphics output to this work station
3	Clear Workstation-clear display device
4	Update Workstation-display all pending graphics on work station
5	Escape-enable special device-dependent operation
6	Polyline-put out a polyline (a graphics primitive composed of one or more connected line segments, or vectors)
7	Polymarker-issue markers
8	Text-issue text starting at specified position
9	Filled Area-display and fill a polygon
10	Cell Array-display a cell (pixel) array
11	Generalized Drawing Primitive-display a generalized drawing primitive ID 1 Bar 2 Arc 3 Pie Slice 4 Circle 5 Print Graphic Characters 6-7 Reserved (for future use) 8-10 Unused (and available)
12	Set Character Height-set text size
13	Set Character Up Vector-set text direction
14	Set Color Representation-define the color associated with a color index

(continued on next page)

Table 2: (continued)

Opcode	Description
15	Set Polyline Linetype-set line style for polylines
16	Set Polyline Linewidth-set width of lines
17	Set Polyline Color Index-set color for polylines
18	Set Polymarker Type-set marker type for polymarkers (a graphics primitive composed of one or more markers)
19	Set Polymarker Scale-set size for polymarkers
20	Set Polymarker Color Index-set color for polymarkers
21	Set Text Front-set device-dependent text style
22	Set Text Color Index-set color of text
23	Set Fill Interior Style-set interior style for polygon fill
24	Set Fill Style Index-set fill style for polygons
25	Set Fill Color Index-set color for polygon fill
26	Inquire Color Representation-return color representation values of index
27	Inquire Cell Array-return definition of cell array
28	Input Locator-return value of locator
29	Input Valuator-return value of valuator
30	Input Choice-return value of choice device
31	Input String-return character string
32	Set Writting Mode-set current writing mode (replace, overstrike, complement, erase)
33	Set Input Mode-set input mode (request or sample)

CHAPTER FOUR

THE SOLTEC 281 PLOTTER

The SOLTEC 281 Plotter, looks like a simple table plotter which can hold up to eight different pens at a time, but when one picks up the approximately 230 pages of the Users Manual, the first thought is that it must be overly documented. Actually, the documentation would be complete if the intent of the manufacturer was to provide a reference manual only. Otherwise, it would benefit from additional detailed explanations and examples, especially in Part II. Needless to say, this plotter comes equipped with an extensive set of firmware functions.

Hardware Characteristics

The SOLTEC 281 Plotter is an interactive intelligent digital plotter built around the ZILOG Z 80 microprocessor. It performs four basic operations: printing of alphanumeric characters, vector generation (linear interpolation) circular interpolation and point digitizing. The Programmable Pen Select feature offers multicolor plots plus various line thicknesses. The accessible plotting size is 338 by 280 mm. The plotting is also programmable and is continuously checked by the microprocessor to provide

coordinate value transmission. The graph paper is standard DIN A3 format. The major firmware characteristics and performance specifications of interest are shown in Table 3, on page 30.

The origin of the rectangular Plotter Coordinate System (PCS) is fixed to its mechanical Zero located at the lower left of the platen (see Figure 3, on page 32). All possible coordinate pairs are restricted to positive values between 0 and 2800. The unit within this system is 0.1 mm for both the X and Y directions. Any programmed plotting action is transformed to this system and therefore its characteristics cannot be altered under program control or locally at the front panel.

The 281 Plotter offers the possibility to shift the origin of a drawing across the physical plotting area and even outside of it. Additionally the plotting units can be set by the user locally at the front panel. This system of coordinates that provides for an alteration of its characteristics by the user is called the "Users Coordinate System" (UCS). The UCS can be shifted either locally (at the front panel) by user interaction, or program controlled.

The plotter has three basic plotting modes: vector generation, positioning, and digitizing.

Table 3: SOLTEC 281 Plotter Characteristics

* Characteristics:
  - Microprocessor ZILOG Z 80
  - Up to 8 pens
  - 800 byte input buffer
  - Electrostatic paper holddown

* Performance Specifications:
  - Plotting area: x-axis 338 mm, Y-axis 280 mm
  - Linearity: better than +0.1%
  - Repeatability: +0.1 mm same pen,
    +0.3 mm different pens
  - Line plotting speed: maximum 30 cm/s
  - Pen lift: 30 actions/s

The most important feature of a plotter is the firmware vector generation that performs a linear interpolation between two points in order to produce a straight line. All drafting plotter actions, regardless of complexity, are finally made up by linear interpolations and thus can be traced back to the vector generator. Vector generation is restricted to drawing plotter movements (with pen down) because any other movement is classified as positioning. An exception is incremental plotting with pen up.

The firmware generates a "straight" line from one point with Xc,Yc coordinates - the current position - to a final point Xf,Yf by alternately incrementing or decrementing the X or Y coordinate, using the smallest incremental unit it can. Exception is of course the drawing of a line where either X or Y is equal in both coordinate pairs. A dashed line would result if the recording system would not be bandwidth limited. Due to the fairly low bandwidth of the system the staircase function is actually smoothed out and the human eye qualifies it as straight.

In a plot program it happens frequently that the drawing tool must be directed to some point without performing a drawing. Normally one wishes to start a drawing or part of it at this point. This plotting action is called "positioning" and the corresponding plot instructions

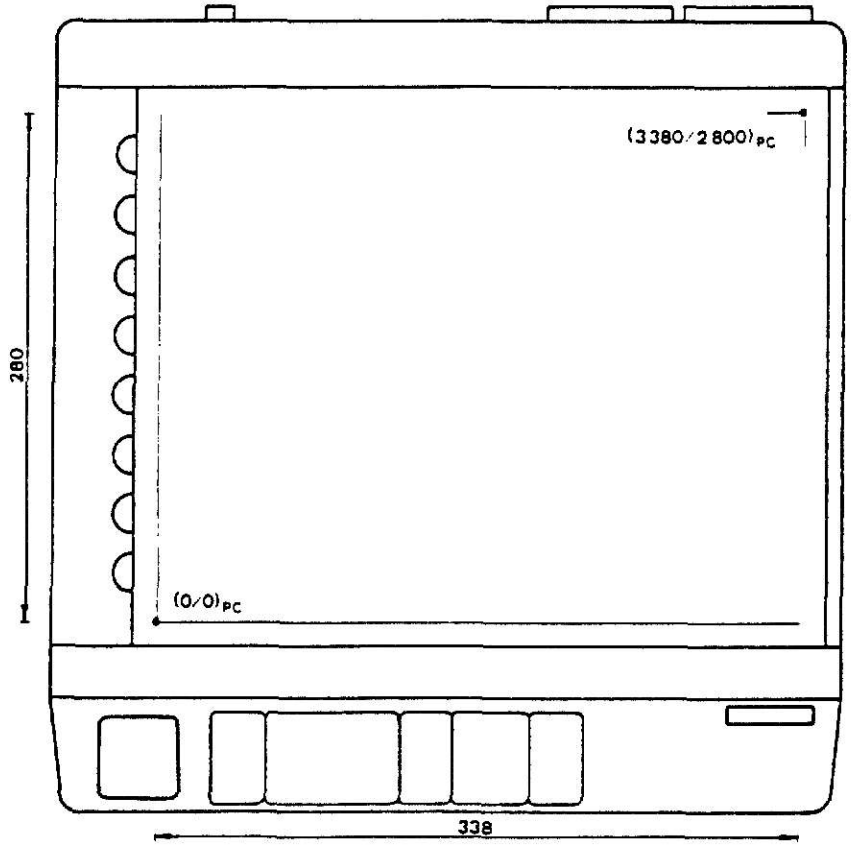

Figure 3: The Plotter Coordinate System (PCS) for the SOLTEC 281 Plotter.

are "move absolute" or "move relative"

The 281 Plotter is *not* an Incremental plotter, however it has an additional feature to move the pen - lifted or lowered - in the same manner as an incremental plotter does. Note that the length of a plotting unit in the X or Y directions (positive or negative) is 0.1, while in any of the other four in-between directions its length is 2*(square root of 2)*0.1 .

Software Capabilities

Based on the vector generation mode, the 281 Plotter has an impressive alphanumerics feature, which provides printing of text on the plot without requiring software supported character generation by a host computer. The firmware character generator offers five fonts of standard characters: standard ASCII, German, Spanish, Swedish or Finnish, Danish or Norvegian. Additionally, the characteristics of alphanumerics can be modified under program control. This is performed by two instructions only. One instruction sets the height and width and the printing plane of characters, the other one sets the character slant to 90 or 75 degrees, respectively. The shape of the characters is predetermined by a 7*7 dot matrix that also provides for the spacing of characters.[1]

The 281 Plotter not only accepts printing characters but also responds to some control characters for convenient formating of text on the plot.

Of special interest is the unusual implementation of circular interpolation, also based on the vector generation mode. The firmware routine provides for arc and circle generation with programmable radius and optionally the starting and final angles. Circles and arcs are drawn counter clockwise (as a default direction) rather than clockwise, as one might expect. Furthermore the current position, becomes the starting point on the circle or arc, rather than the point of origin.

Five special graphic symbols can be plotted with programmable height, width, printing plane and slant. The current pen position is the center and starting point of the mark. These "mark" characters are: X, Y, +, an equilateral triangle, and a square. The last two have a line connecting the point of origin to one of their corners.

Also various types of dashed lines with a programmable length can be selected. The 281 Plotter is

---

1. For details refer to the SOLTEC 281 Plotter Users Manual pp 26-30, 37 of Part I, and pp 25-43 of Part II, showing character height, width, line separation, spacing, and angle of rotation.

able to perform on output to the host computer containing information on the coordinates of the current and/or actual pen position. This feature of point coordinate transmission to the host is known as "digitizing." Two different forms of digitizing are supported: interactive digitizing and immediate digitizing.

When the plotter is set into the interactive digitizing mode (under program control only) the user is requested to move the digitizing sight to the desired point on the platen and to validate data if satisfied.

The Plotter outputs (in ASCII) numeric values for the point coordinates of the actual position in the PCS. X value is transmitted first and separated from Y value by a blank. The X and Y coordinates are represented by 6 bytes each.

Contrary to interactive digitizing this mode does not require user interaction. When the plotter receives the instruction to digitize immediately, it transmits the coordinate values of the current pen position in the UCS. It should be noted that the current pen position not necessarily coincides with the actual (physical) position of the pen because it can be outside the physical plotting area (off scale).

If the application software does not keep track of the current pen position an attempt could be made to direct the pen to a point outside the physical plotting area. Tracking the current pen position in the host's program could be quite cumbersome especially in the cases were plotting functions such as circle and arc generation and alphanumerics are used extensively. Therefore, the 281 Plotter provides for off scale data handling that makes it unnecessary to track the current pen position in the host computer and simultaneously avoids misleading results and undesired portions of a plot.

Whenever a plotting leads to off scale data-points the pen is lifted automatically (if down) at the intercept with the mechanical boundary or graphic limit. As soon as on scale data points are reached again plotting is resumed correctly with the pen status as programmed.

The instructions (input) to the 281 Plotter are buffered in a 800 byte Input Buffer. The handling of this buffer information is different for the types of interface in use. Buffer messages may be requested by the host or may be transmitted to the host automatically; they are also dependent on the type of interface. Buffer messages are presented in the form of one byte, and they include status codes such as "Full" or "Empty." How to perform buffer control is described in Part III of the Users Manual "I/O

Programming and Interfacing."

In general the simplest format of the instruction set of the 281 Plotter consists of only one ASCII-character (one byte) that defines the type of command (one byte instructions). Other, more advanced plotting actions of the plotter are made up by an ASCII-character defining the type of command followed by additional characters representing parameters. (two or more byte instructions). When two or more parameters are present they must be separated by at least one blank.

Let's look at some examples. The one byte instructions "K" and "J" are move absolute and move relative, respectively, and are normally preceded by the X and Y coordinate pair. The instructions "I" and "H" are equivalent to pen down and pen up, respectively. The syntax for plotting a circle or an arc is: "Or a b," where "r" is the radius, "a" is the starting angle, and "b" is the final angle. When "r" is positive the arc is drawn counter clockwise, otherwise clockwise. To draw a circle for example the following set of instructions would be used:

```
800/2000KIO300 0 360H
```

Assuming that the pen was up prior to this instruction, the result is a circle with radius of 3 cm and its origin at

500,2000. Note that if "0-300 0 360" had been specified, the same circle would have been drawn, but with its origin at 1100,2000.

For a complete list and description of each command in the instruction set, the reader is referred to the Part II of the SOLTEC 281 Plotter Users Manual.

To take advantage of as many hardware (firmware) device features as possible, the "optimal" virtual device interface accepts a virtual "vocabulary" representing the capabilities available on graphics devices. Examining the list of capabilities in such a vocabulary [Warner] one finds that the SOLTEC 281 Plotter does not have capabilities such as: polygon fill, device-level segment display file, selective erase, highlighting, background color, or color lookup table. The polygon fill capability for instance, could be emulated in the application software controlling the plotter, namely the device driver.

CHAPTER FIVE

DRIVER IMPLEMENTATION

In the spring semester of 1983, the Kansas State University's Department of Computer Science began a project to study and then implement subsets of GKS. The implementation of a GKS subset at KSU reflects the efforts of many students, guided by Dr. William J. Hankley, over a period of several semesters. The major goal of the GKS project was to provide its participants with a valuable educational experience.

Based on the "GKS Functional Description, Draft International Standard ISO/DIS 7942" [ISO] and other available literature, the spring semester students developed a skeleton representing the heart of the GKS system, defined most of the basic data structures, and implemented a minimum set of (line and marker) primitive capabilities on one workstation. Using the framework developed by the spring semester plus the GKS [ISO], the VDI [ANSI, Aug.27,1982], and the VDM [ANSI, Aug.13,1982] documents as a base, each Western Electric "Summer-On-Campus" student had to study the overall GKS system, and then focus on a specific implementation. The 1983 fall semester students had the task of consolidating everyones work and adding enhancements

to GKS.

As mentioned in the Overview section of Chapter One, the author's individual project was the development of a VDI driver for the SOLTEC 281 Plotter.

Driver Structure

The GKS system was implemented on an Interdata 3220 host minicomputer, under a UNIX[1] operating system. The Pascal programming language was mainly used for writing the main GKS source code and the data structures, plus the components developed by the 1983 summer semester students. The UNIX Shell, and the C languages were used however to write those programs which required "executive" capabilities.

In order to become familiar with the plotter, it was first necessary to establish an interface between the minicomputer and the plotter. Through such an interface, a file which contains a set of valid commands for the ploter, can then be downloaded from the minicomputer into the plotter. The "plttst" program, shown in the Appendix on page 55, is written in Shell and its function is to download

---

1. UNIX is a trademark of AT&T Bell Laboratories.

a file to device tty25, which is the address of the plotter.

One example of the many test files written through the course of that term is "plt.tst.1," shown in the Appendix on page 56. In this file, some of the GKS and VDI output primitives, such as polyline, polymarker, and polytext, plus some circles and arcs which are part of the generalized drawing primitives (GDP), are tested out.

For the remainder of this paper, when referring to these output primitives the "poly" prefix will be dropped, and references to them will be simply stated as the line, the marker, the text, and so on.

The plot generated by the SOLTEC 281 Plotter, resulting from the interpretation of the instructions contained in the "plt.tst.1" file, is shown in Figure 4, on page 42. The original plot is in eight different colors, however due to reproduction limitations the copy in this paper is only in black and white. To understand the relationship between the resulting plot and the set of corresponding instructions, let us examine this file line by line.

The plotter is assumed to be turned on, initialized, and each slot holding a pen, in our example each of different color. The first line instructs the plotter to select pen in slot number 1. Note that the selection of

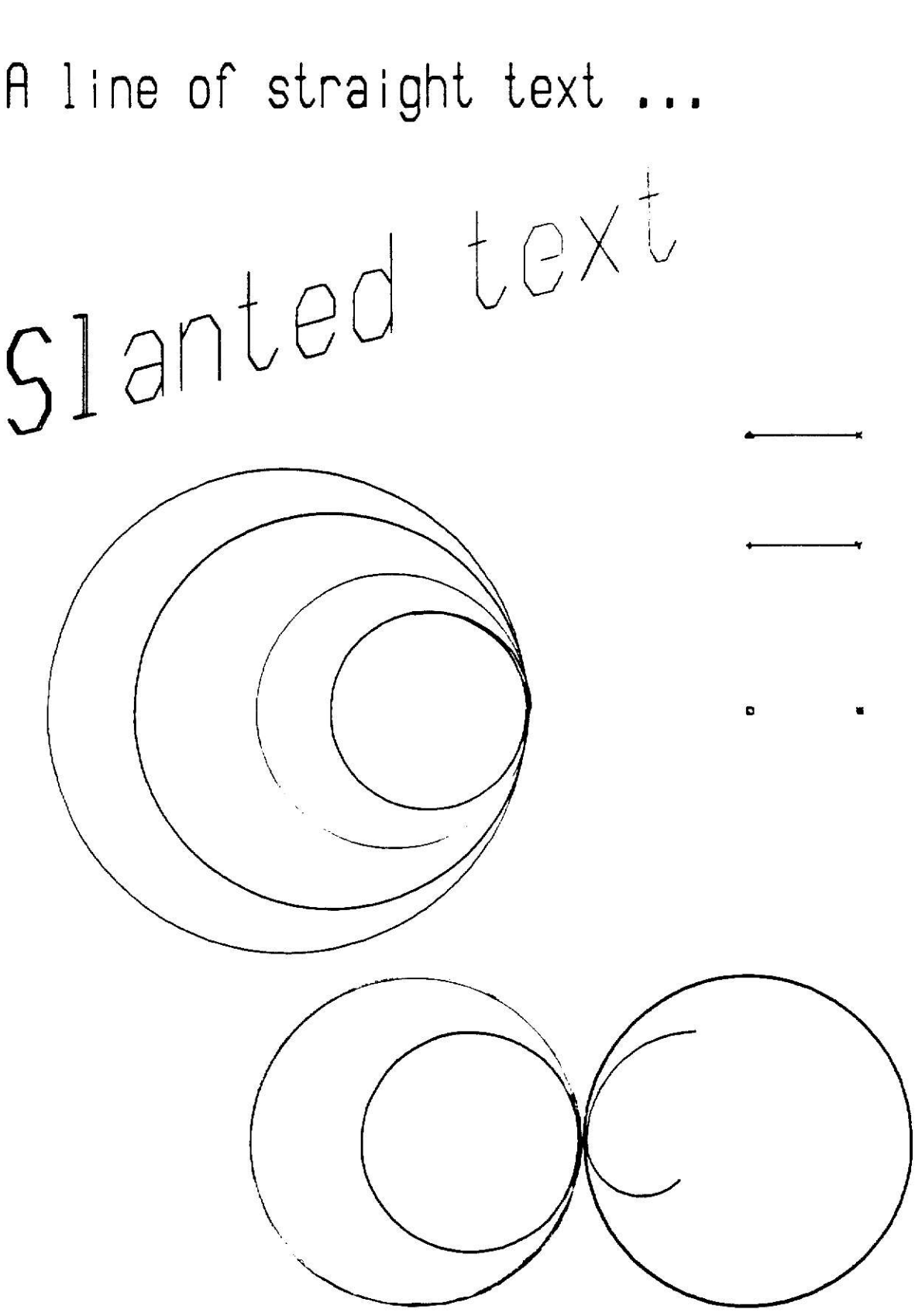

Figure 4: A Sample Plot Generated With The SOLTEC 281 Plotter.

pens is dependent on their physical location rather than their color. Therefore, in order to ensure consistency of color for all primitives on all output devices, it is necessary to standardize the color attributes at the VDI level. In other words, if the color unit number 4 produces red on this plotter, the same unit should produce the same color for all primitives on any other device, even if the metafile is ported to another GKS system.

The second line specifies that all subsequent characters will be plotted with a 90 degree slant. This is the default setting for text plotting on the SOLTEC, and it is equivalent to the GKS Character Upvector (0.0,1.0). The GKS text alignment value here defaults to (normal,normal).

On the third line, three more text primitive attributes are specified, namely: character height = 8mm, text path = right, and character width (expansion factor and spacing) = 5mm, in that order. Actually, the second parameter of this instruction specifies the angle of rotation for each character, thus defining the path attribute. Interestingly, the "Z" instruction bundles these three attributes together, and they cannot be specified separately. An other text attribute, the text font, not having been specified, here defaults to ASCII.

The fourth line contains three instructions: raise

pen, move absolute to coordinate 50,2600 (which is close to the upper left corner of the viewport), and then enable text plotting. The plotting of the text is enabled by the "B" instruction, and it continues until a carriage return character is encountered. The two blank spaces following the "H" instruction are ignored.

The instructions in the fifth through the seventh lines are essentially a repeat of the previous set of instructions, and as it can be seen from the resulting plot, the text is larger, slanted 75 degrees, and each character is rotated 15 degrees.

In the next four lines, the objective is to draw six different marker primitives, connecting the first two pairs with a straight line. The "M" instruction plots the markers, and the single digit integer following it determines the type of marker. Because the markers are treated similarly to text, they are subject to all transformations valid for character plotting. Thus the marker primitives for the SOLTEC 281 Plotter have the same attributes as the text primitives.

Lines twelve through twentyone demonstrate how the SOLTEC 281 Plotter handles the general drawing primitives. All these instructions generate circles, except for the last two instructions, which plot arcs. Note that the first four

circles have the same "starting point" coordinates at (1000,1500). The rest of the circles and arcs also share their coordinates at (1100,720), nevertheless, all of them have different centers of origin.

The VDI document states that "a circle of a given radius is drawn at the specified coordinate," and "the current position is not changed." One can infer that such a circle will have its origin at the specified coordinate, with the given radius. The SOLTEC Driver must therefore calculate the "starting point" for a circle, by increasing the X component of the coordinate with the value of the radius, and then draw the circle with a positive radius.

Having decided to use the Pascal language, the next logical step in the development of the VDI Driver software for the SOLTEC 281 Plotter, was to write the parameter type definitions for this device. This code, was placed in a file called "soltectypes.i," and is shown in the Appendix on page 57. Its structure accommodates the rich set of primitives with their respective attributes, and allows for future expansion of the driver code.

The variable declarations for the driver are in the file "soltecvar.i," shown in the Appendix on page 60. The file "mainvar.i" in the Appendix on page 61, contains the variable declarations for the test program. The

"Pas32/progheading.i" as well as the "progclosing.i" had to be modified to provide for the "plotdev" device. Only the "progclosing.i" file is shown in the Appendix on page 62.

The subroutines which are called by the Driver are in the "soltecsubs.i" file, listed in the Appendix on page 63. The most often called upon subroutine is the "writecoord" procedure, its function being as its name implies, to write a coordinate to the plotter whenever requested to do so.

Briefly, the "getpen" and "returnpen" procedures pick up the pen in slot one and deposit it, respectively. The pen in slot one is always picked up as a default or as the initial pen. The "pendown" and "penup" procedures lower and raise the pen, respectively. The "setcolor" procedure tells the pen holder to pick up an other pen from a specific slot. The "setlinetype," "setcharh," and "soltecupdate" procedures do just that, set the line type, set the character height, and update the device by closing it, respectively. The "movepen" procedure writes a coordinate pair and tells the plotter to move to that absolute address.

In the same file, the remaining code are procedures which call the above mentioned subroutines, and are in turn directly called by the SOLTEC VDI Driver. Procedure "solteclear" resets the current coordinate pair to (0,0).

Procedure "soltecmarker" plots any one of the five markers. Here, resetting the character size and slant remains to be done. The "soltectext" procedure outputs a string of characters to the plotter, setting first the height, direction, and width of the characters.

The "scaletext" and "scaletosoltec" procedures perform the necessary scaling transformations. Finally, the "initsoltec" procedure initializes the required primitive attributes for the plotter.

The "soltecdriver.i" file contains the "soltec" procedure, listed in the Appendix on page 68. This procedure is the actual VDI Driver for the SOLTEC 281 Plotter, and is executed from the GKS through that part which at KSU was named the "Distributor." The following VDI commands were implemented in this procedure: vdiopen, vdiclose, vdiclear, vdiupdate, vdiline, vdimark, vditext, vdilinecolor, vdimarktype, vdimarkcolor, and vditextcolor. The rest remain to be implemented by perhaps future KSU students.

The main, or rather the test program for the Driver, "main.i," is listed in the Appendix on page 71. In this test program the vdiopen, vdiclear, vdimarkcolor, vdilinecolor, vditextcolor, vdiline, vdimarktype, and vdiupdate commands are exercised.

Now, to pull all this code together, the "pascomp" Shell, listed in the Appendix on page 73, is invoked with the "comp" parameter. The Shell reads the "comp.c" file, shown in the Appendix on page 75, which contains all the names of the GKS files plus the files which make up the SOLTEC Driver. To run GKS, the "rungks" Shell, shown in the Appendix on page 74, must be executed with a valid device name specified as a runtime parameter. In the case of the SOLTEC Driver, the device name is "plot."

Project Results

The major goal of the GKS project, providing its participants with a valuable educational experience: a general knowledge of the state-of-the-art of the graphics standards, plus an in depth knowledge of GKS, VDI and VDM, was definitely achieved. With respect to the implementation portion of this project, the objective was not so much to produce "deliverable" software, but to learn even more about GKS and to leave behind "readable" code.

The VDI driver for the SOLTEC 281 plotter compiled successfully. However, there was insufficient time left to test the GKS system with the driver software included. It is the author's belief however, that the test would have been successful when applied to the version of the GKS system as it existed at the end of the 1983 summer term.

Since there were other capabilities being developed concurrent with this effort, the GKS code has gone through several changes afterwards. As of this time, the following GKS files are known to this participant to have been modified since then:

```
/usr/src/gks/gks.c
/usr/src/gks/GKS/initws.i
/usr/src/gks/Gks/initgks.i
/usr/src/gks/Scanner/interpreter.i
/usr/src/gks/Scanner/Interpreter/testparms.i
/usr/src/gks/Scanner/Interpreter/prtrcallinf.i
/usr/src/gks/Glob/gkstype.i
```

Also, all occurrences of index (type) need to be changed to sindex (type).

Had there been more time available, it would have been quite rewarding to complete the plotter's driver code, implement it, and then fully test it. During this five week long semester, each participant in the GKS project was also taking an advanced graduate course, and had to prepare and deliver the "final orals" exam. In this participant's case, due to the hardware dependency of the SOLTEC 281 plotter driver code, it was not possible to continue the implementation part of the project after leaving the KSU campus.

SELECTED BIBLIOGRAPHY

ANSI. "Proposal for an ANSI X3 Standards Project for the Computer Graphics Virtual Device Interface," Aug. 27, 1982.

ANSI. "Draft Proposed American National Standard for the Virtual Device Metafile," Aug. 13, 1982.

Ansell, Ian O. "A Practical Introduction to Computer Graphics," John Wiley & Son , New York, 1981.

Bailey, Chris. "Graphics Standards are emerging slowly but surely," Electronic Design, pp. 103-110,January 20, 1983.

Bono, Peter R. et. al. "GKS- The First Graphics Standard," Computer Graphics and Applications, Vol. 2, No.5, pp. 9-23, July, 1982.

Bono, Peter R. "The GKS Impact on Graphics Standardization," Computer Graphics World, Vol. 5, No.9, p. 47, Sept., 1982.

Borufka, H. G., and Pfaff, G. "The Design of a General -Purpose Command Interpreter for Graphical Man-Machine Communication," Man-Machine Communications in CAD/CAM, North-Holl and Publishing Co., pp. 161-175, 1981.

Brown, Marlene. "Graphics Standards Advancing on Many Fronts," Software News, Vol. 3, No.6, pp. 28-36, June 1983.

Bruns, Bob., and Warner, James R. "A Discussion of Software Standards," Computer Graphics World, pp. 60-63, Aug., 1982.

Buttuls, P. "Some Criticisms of the Graphics Kernal System (GKS)," COMPUTER GRAPHICS, VOL. 15, NO. 4, PP. 302-305, DEC. 1981.

Card, Chuck., Prigge, Donald R., Walkowicz, Josephine L., Hill, Marjorie F. "The World of Standards," BYTE, pp. 130-142, Feb., 1983.

Caruthers, L. C. et. al. Stand-Alone and Satellite Graphics," pp. 112-119.

DIN. "The Graphical Kernal System (GKS). The Standard for Computer Graphics Proposed by the German Institute for Standardization (DIN)," Erlangen-Nurnberg Univ., Germany, Computer Graphics, State of The Art Report, pp. 219-248, 1980.

Dern, Daniel P. "Graphics Boom Relies on Standards," Software News, Vol. 3, No.6, pp. 24-27, June, 1983.

Encarnacao, J., Enderle, G., Kansy, K., Nees, G., Schlechtendahl, E. G., Weiss, J., and Wibkirchen, P. "The workstation concept of GKS and the resulting conceptual differences to the GSPC core system," Proceedings of SIGGRAPH'80 in Computer Graphics, Vol. 14, No.2, pp. 226-230, July 1980.

Gallop, J. R. "Graphical Output facilities of GKS," Rutherford Applenton Laboratory.

Green, Mark. "A Catalogue of Graphical Interaction Techniques," Computer Graphics. Vol. 17, No. 1, pp. 46-52, Jan., 1983.

Fleming, Jim. "NAPLPS: A New Standard for Text and Graphics," "Part 1: Introduction, History, and Structure," BYTE, pp. 203-254, Feb., 1983, "Part 2: Basic Features," BYTE, pp. 152-188, March, 1983 "Part 3: Advanced Features," BYTE, pp. 190-206, April, 1983 "Part 4: More Advanced Features and Conclusions," BYTE, pp. 272-284, May, 1983.

Foley, James D., and Van Dam, Andries. "Fundamentals of Interactive Computer Graphics," Addison-Wesley Publishing Co., Reading, Mass., 1982.

Hatfield, Lansing., and Herzog, Bertram. "Graphics Software - from Techniques to Principles," IEEE Computer Graphics and Applications, pp. 59-80, Jan., 1982.

ISO. "Graphics Kernal System (GKS) Functional Description, Draft International Standard ISO/DIS 7942," Nov. 14, 1982.

Langhorst, Fred E., and Clarkson III, Thomas B. "Realizing Graphics Standards for Microcomputers," BYTE, pp. 256-268, Feb., 1983.

Mudur, S. P. et. al. "Environmental Independence in a Graphics Programming System," Proceedings of the International Conference on Techniques in Computer Aided Design, pp. 241-248, Sept., 1978.

Newman, William M., and Sproull, Robert F. "Principles of Interactive Computer Graphics," 2nd Ed., McGraw-Hill, Inc., New York. Chap 27: Device-Independent Graphics Systems, 1979.

Perry, Burt. "Graphics frees itself from device-dependence," Electronic Design, pp. 167-173, January 20, 1983.

Puk, Richard F. "The Background of Computer Graphics Standardization," Computer Graphics, Vol. 12, No. 1-2 , pp. 2-6, June, 1978.

Reed, Theodore N. "A Metafile for Efficient Sequential and Random Display of Graphics," Proceedings of SIGGRAPH'82 in Computer Graphics, Vol. 16, No. 3, pp. 39-43, July, 1982.

Reed, John S. "Computer Graphics," Mini-Micro Systems, Vol. 15, No . 12, pp. 210-221, Dec. , 1982.

Rosenthal, David S. H. et. al. "The Detailed Semantics of Graphics Input Devices," Proceedings of SIGGRAPH'82 in Computer Graphics, Vol. 16, No. 3, pp. 33-38, July ,1982.

Rosenthal, David S.H. "Managing Graphical Resources," Computer Graphics, pp. 38-45, January 1983.

Sonderegger, Elaine L. "Recent ANSI X3H3 (Computer Graphics) Activity," Computer Graphics, Vol. 16, No. 4, pp. 246-248, Dec., 1982.

Sonderegger, Elaine L. "Report of the SIGGRAPH Representative to ANSI X3H3 Computer Graphics," Computer Graphics, Vol. 15, No. 4, pp. 276-277, Dec., 1981.

Vinberg, Anders. "Device intelligence is part of the picture," Data Management, Vol. 21, No.5, pp. 18-24, May 1983.

Warner, James R., and Kiefhaber, Nikolaus J. "Implementing standard device - independent graphics," Mini-Micro, pp.201-208, July 1982.

Weller, D. L. et. al. "Software Architecture for Graphical Interaction," IBM Systems Journal, Vol. 19, No. 3. pp. 314-330, 1980.

Wisskirchen, Peter et. al. "Implementation of the Core Graphics System GKS in a Distributed Graphics

Environment," Proceedings of the International Conference on Interactive Technique in Computer Aided Design, pp. 249-254, 1978.

APPENDIX A

SOURCE CODE LISTING

Jul 18 16:24 1983 plttst Page 1

```
(*****************************************************************)
(* SHELL  TO  TEST  THE  SOLTEC  PLOTTER                         *)
(*                                                               *)
(* Purpose:  Download a file from the Interdata 3220             *)
(*           into the SOLTEC 281 Plotter.                        *)
(*           The /dev/tty25 is the plotter's address.            *)
(*                                                               *)
(* Written by: Gene Janovitz                                     *)
(* Date:  June 1983                                              *)
(*                                                               *)
(*****************************************************************)


echo "Downloading file $1 to the SOLTEC 281 Plotter, "
echo "please stand by ..."
cat $1 > /dev/tty25
if test $? -eq 0
then echo "File $1 downloaded successfully"
else echo "Warning: file $1 may not have downloaded successfully"
fi
```

```
Jul 18 16:24 1983  plt.tst.l Page 1


F1
%0
Z80 0 50
H  50/2600KBA line of straight text ...
F2
Z180 15 110
H50/2000K%1BSlanted text
%0Z10 0 10
H1400/2000KIM1 1600/2000KM2
H1400/1800KIM3 1600/2000KM4
H1400/1500KM0 1600/2000KM2M3
F3
H1000/1500KI0180
F4
H1000/1500KI0250
F5
HI0360
F6
HI0440
F7H1100/720KI0300F80200F10-300 180 180F20-200 180 90
HF4KI0100 180 315H
```

```
(*****************************************************************)
(* PARAMETER TYPE DEFINITIONS FOR SOLTEC VDI DRIVERS             *)
(*****************************************************************)

(* proc soltec( cmd: vdicmdtype;
                parm: vdiparm  ;
            var oparm: vdioutparm) *)

;type vdicmdtype =
               ( vdiopen,       (* f1 Open Workstation *)
                 vdiclose,      (* f2 Close Workstation *)
                 vdiclear,      (* f3 Clear Workstation *)
                 vdiupdate,     (* f4 Update Workstation *)
                 vdiescape,     (* f5 Escape *)
                 vdiline,       (* f6 Polyline *)
                 vdimark,       (* f7 Polymarker *)
                 vditext,       (* f8 Text *)
                 vdiarea,       (* f9 Filled Area *)
                 vdigdp,        (* f11 Generalized Drawing
                                       Primitive *)
                 vdicharh,      (* f12 Set Character Height *)
                 vdicharupv,    (* f13 Set Character Up Vector *)
                 vdilinetype,   (* f15 Set Line Type *)
                 vdilinewidth,  (* f16 Set Line Width *)
                 vdilinecolor,  (* f17 Set Line Color *)
                 vdimarktype,   (* f18 Set Marker Type *)
                 vdimarksize,   (* f19 Set Marker Size *)
                 vdimarkcolor,  (* f20 Set Marker Color *)
                 vditextfont,   (* f21 Set Text Font *)
                 vditextcolor,  (* f22 Set Text Color *)
                 vdifillcolor,  (* f25 Set Fill Color *)
                 vditextpath,   (* f-- Set Text Path *)
                 vdiinlocator,  (* f28 Input Locator *)
                 vdiinvaluator,(* f29 Input Valuator *)
                 vdiinchoice,   (* f30 Input Choice *)
                 vdiinstring,   (* f31 Input String *)
                 vdiinmode)     (* f33 Set Input Mode *)

;type vdiparm = record
             case cmdcode: vdicmdtype of
               vdiopen,
               vdiclose,
               vdiupdate,
               vdiclear     :();
               vdiescape       :(msg:tstring);
```

```
                vdiline,
                vdimark,
                vdiarea      :(numpts : pointsrange;
                               pts  : upoints );
                vditext      :(textpos: ipoint;
                               numchar : textcharrange ;
                                 string: tstring );
                vdigdp       :(gdpcmd:index;
                               numgdppts: pointsrange;
                               gdppts   : upoints );
                vdicharh:(height:integer);
                vdicharupv:(upvec:ipoint);
                vdilinetype,vdimarktype,vditextfont
                             :(kind:index); (* 1=deflt .. maxkind *)
                vdilinecolor,vdimarkcolor,vditextcolor,vdifillcolor
                             :(color:unit);(* 0..7 = blk,blu,grn,cyan,
                                                red,magenta,yel,
                                                white;
                                              >7 =other*)
                vdilinewidth,vdimarksize
                             :(size:integer);
                vditextpath :(path:ipoint);
                vdiinlocator:(locdev :index; (*1=dflt, 2=cursor,
                                              3=tab, 4=joystk,
                                              5=ltpen, 6=plotter,
                                              7=mouse, 8=ball
                                              >8 = other *)
                              locpos:ipoint);(* initial pos*)
                vdiinvaluator:(valdev:index);
                vdiinchoice :(chdev:index); (*1=dflt, 2=funkey,
                                                >2=other*)
                vdiinstring :(strdev:index; (* 1=keybd *)
                               strlen:textcharrange);(* max length*)
                vdiinmode   :(indev:index; (* 1=loc, 2=val, 3=choic,
                                              4=strng*)
                                mode:index) (* 1=request (driver waits
                                                for indev)
                                             2=default, sampled *)


                 end   (*  vdiparm  *)

;type vdioutparm = record
                 status:unit; (* 0=no 1=ok *)
                 case cmdcode: vdicmdtype of
```

```
            vdiinlocator :( locpos:ipoint);
            vdiinchoice  :( choice:integer );
            vdiinstring  :( len:textcharrange;
                            strng:tstring);
            vdiinvaluator : (value:real)
             end (* vdioutparm *)

(*****************************************************************)
(* END OF SOLTEC VDI TYPE DEFINITIONS                            *)
(*****************************************************************)
```

```
(*****************************************************************)
(* SOLTEC VARIABLE DECLARATIONS                                  *)
(*****************************************************************)


;t1, t2 : ipoint
;telem  : vdiparm
;twielem : upoints
;ii: integer
;plotison : boolean
;solteccolor,
solteclinetype,
solteechrh,
solteechrdir,
solteechrw,
solteechrslant,
soltecfont,
currlinecolor,
currtextcolor,
currmarkercolor,
typeofmark: integer


(*****************************************************************)
(* END OF SOLTEC VARIABLE DECLARATIONS                           *)
(*****************************************************************)
```

```
(*****************************************************************)
(* S O L T E C    T E S T   P R O G R A M   V A R I A B L E S   *)
(*****************************************************************)


   ;ip :vdiparm
   ;op :vdioutparm
   ;iiii :integer


(*****************************************************************)
```

```
; procedure programclosing
  ; begin (* program closing *)
      (* close   output *)      close ( output )
    ; (* close   error *)       close ( driver3  )
    ; (* close   segment *)     close ( segment)
    ; (* close   plotdev  *)     close ( plotdev )
    end (* programclosing *)
```

```
(**************  GRAPHICS  KERNAL  SYSTEM  ****************
*                                                          *
*  SOLTEC VDI Driver Subroutines                           *
*                                                          *
*  Purpose:     This set of procedures provide the         *
*               necessary subroutines to be called         *
*               from the SOLTEC VDI Driver.                *
*               Also, the SOLTEC VDI Primitives            *
*               procedures are included, which call        *
*               these subroutines, and are in turn         *
*               called by the SOLTEC VDI Driver.           *
*                                                          *
*  Written by: Gene Janovitz                               *
*  Date: July 1983                                         *
*                                                          *
************************************************************)

;procedure writecoord (coordinate : integer);
  var thousands,
      hundreds,
      tens,
      coord,
      ones : integer;
      ch   : char;

  begin
    coord := coordinate;
    if coord > 999 then
     begin
       thousands := coord div 1000;
       coord := coord - (thousands * 1000);
       ch := chr(thousands+48);
       writechar (plotdev,ch)
     end
    else writechar (plotdev,'0');
    if coord > 99 then
     begin
       hundreds := coord div 100;
       coord := coord - (hundreds * 100);
       ch := chr(hundreds+48);
       writechar (plotdev,ch)
     end
    else writechar (plotdev,'0');
    if coord > 9 then
     begin
```

```
        tens := coord div 10;
        coord := coord - (tens * 10);
        ch := chr(tens+48);
        writechar (plotdev,ch)
      end
    else writechar (plotdev,'0');
    if coord < 10 then
      writechar (plotdev,chr(coord +48))
  end;

procedure pendown;
  begin
    writechar(plotdev,'I');
    plotison := true
  end;  (*  pendown  *)

procedure penup;
  begin
    writechar (plotdev,'H');
    plotison := false
  end;  (*  penup  *)

procedure returnpen;
  begin
    writechar (plotdev,'H');
    writechar (plotdev,'F');
    writechar (plotdev,'0');
    plotison := false
  end;  (*  returnpen  *)

procedure getpen;
  begin
    writechar (plotdev,'H');
    writechar (plotdev,'F');
    writechar (plotdev,'1');
    plotison := false
  end;  (*  getpen  *)

procedure setcolor (colornum : integer);
  begin
    writechar(plotdev,'F');
    writechar(plotdev,chr(colornum + 1 + ord('0')));
    solteccolor := colornum
  end;  (*  setcolor  *)
```

```
procedure setlinetype (linetpnum : integer);
  begin
    writechar(plotdev,'L');
    writechar(plotdev,chr(linetpnum - 1 + ord('0')));
    solteclinetype := linetpnum
  end;  (*  setlinetype  *)

procedure setcharh (charhnum : integer);
  begin
    writechar(plotdev,'5');
    writechar(plotdev,'0');
(*    writechar(plotdev,chr(charhnum - 1 + ord('0')));
*)
    solteccharh := charhnum
  end;  (*  setcharh  *)

procedure soltecupdate (eomchr : char);
  begin
    writechar(plotdev,eomchr)
(*    writeline(plotdev)     *)
  end;  (*  soltecupdate  *)

procedure movepen (xycoords: ipoint);
  begin
    writecoord (xycoords.ix);
    writechar(plotdev,'/');
    writecoord (xycoords.iy);
    writechar(plotdev,'K');
  end;  (*  movepen  *)



(*****************************************************************)
(* SOLTEC VDI PRIMITIVES                                         *)
(*****************************************************************)

procedure solteclear;

  begin
    writecoord(0);
    writechar(plotdev,'/');
    writecoord(0);
  end  (* solteclear *) ;

procedure soltecmarker;
```

```
  begin
(*    Set the character size and slant  *)
    case typeofmark of
      1: writechar (plotdev,'.');
      2: writechar (plotdev,'+');
      3: writechar (plotdev,'*');
      4: writechar (plotdev,'o');
      5: writechar (plotdev,'x')
    end  (* case typeofmark *)
(*    Reset the character size and slant  *)
  end;   (* soltecmarker *)


procedure soltectext (nofchars: integer;string: tstring);
  var
    i: integer;
  begin
    writechar(plotdev,'Z');
    writechar(plotdev,'5');
    writechar(plotdev,'0');
    writechar(plotdev,' ');
    writechar(plotdev,'0');
    writechar(plotdev,'3');
    writechar(plotdev,'0');
    for i := 1 to nofchars do
      writechar (plotdev,string[i])
  end;   (*  soltectext  *)

procedure scaletosoltec (num: integer;xyin: upoints;var xyout:
                                                        upoints);

  const
    scalefactor = 12  (* 2800/32000=0.0875 or 1/11.428571 *) ;

  var
    i,n: integer;
  begin
    for i:= 1 to num do begin
      xyout[i].ix := xyin[i].ix div scalefactor;
      xyout[i].iy := xyin[i].iy div scalefactor
    end (* for i *)
  end;  (*  scaletosoltec  *)

procedure scaletext ( xyin: ipoint;var xyout: ipoint);

  const
```

```
    scalefactor = 12  (* 2800/32000=0.0875 or 1/11.428571 *) ;

  begin
    xyout.ix := xyin.ix div scalefactor;
    xyout.iy := xyin.iy div scalefactor
  end;  (*  scaletext  *)

(*****************************************************************)
(* INIT  ROUTINE  FOR  THE  SOLTEC  VDI  DRIVER                  *)
(*****************************************************************)
procedure initsoltec;
  var
    color: colorset;
  begin
    getpen;
    penup;
    solteccclear;
    setcolor (0);
    setlinetype (1);
    currlinecolor := 0;
    currtextcolor := 0;
    currmarkercolor := 0;
    typeofmark := 1 ;
    soltecchrh := 50;
    soltecchrdir := 0;
    soltecchrw := 30;
    soltecchrslant := 0;
    soltecfont := 0

  end;    (* initsoltec *)




(*****************************************************************)
(* END OF SOLTEC VDI PRIMITIVES AND SUBROUTINES                  *)
(*****************************************************************)
```

```
(***************   GRAPHICS  KERNAL  SYSTEM    ******************
*                                                                 *
*   procedure soltec                                              *
*                                                                 *
*   Purpose:  This procedure is the VDI Driver for the SOLTEC     *
*             table PLOTTER 281. This procedure is executed       *
*             from the GKS through the Distributor.               *
*                                                                 *
*                                                                 *
*   Input Parameters:  cmd    - VDI Command                       *
*                      parm   - The record of input parameters    *
*                               associated with the VDI command   *
*                                                                 *
*   Output Parameters: oparm  - The record of output values       *
*                               requested by the VDI command.     *
*                                                                 *
*   Procedure Calls:                                              *
*                                                                 *
*   Written by: Gene Janovitz                                     *
*   Date: July 1983                                               *
*                                                                 *
*                                                                 *
*******************************************************************)

procedure soltec  (   cmd: vdicmdtype;
                      parm: vdiparm;
                    var oparm: vdioutparm );

const
   nl = '(:10:)' ;
   ff = '(:12:)' ;
   cr = '(:13:)' ;
   em = '(:25:)' ;

var
  i, xcrd, ycrd : integer;
  icoord : ipoint;
  ucoords: upoints;

begin    (*  driver  *)
 with parm  do
  begin
   case cmdcode of
    vdiopen : begin getpen end;
```

```
    vdiclose : begin returnpen end;

    vdiclear : begin solteccclear end;

    vdiupdate : begin
                  soltecupdate (em)
                end;


    vdiescape : begin (* not implemented *) end;



    vdiline : begin
                if plotison then penup;
                if currmarkercolor <> solteccolor
                   then setcolor (currmarkercolor);
                scaletosoltec (numpts, pts, ucoords);
                      movepen (ucoords[1]);
                      pendown;
                  for i := 2 to numpts do
                    begin
                      movepen (ucoords[i]);
                      soltecmarker;
                  penup
                    end   (* for i *)
              end;   (* vdiline *)

    vdimark : begin
                if plotison then penup;
                if currmarkercolor <> solteccolor
                   then setcolor (currmarkercolor);
                scaletosoltec (numpts, pts, ucoords);
                  for i := 1 to numpts do
                    begin
                      movepen (ucoords[i]);
                      soltecmarker
                    end   (* for i *)
              end;   (* vdimark *)

    vditext : begin
                if plotison then penup;
                if currtextcolor <> solteccolor
                   then setcolor (currtextcolor);
                xcrd := parm.textpos.ix;
```

```
                ycrd := parm.textpos.iy;
                scaletext (textpos, icoord);
                movepen (icoord);
                soltectext (numchar, string)
              end; (* vditext *)


    vdigdp : begin (* not implemented *) end;

    vdicharh : begin (* not implemented *) end;

    vdicharupv : begin (* not implemented *) end;

    vdilinetype : begin (* not implemented *) end;

    vdilinewidth : begin (* not implemented *) end;

    vdilinecolor : begin
                     currlinecolor := color
                   end; (* vdilinecolor *)

    vdimarktype : begin
                    typeofmark := kind
                  end;  (* vdimarktype *)

    vdimarksize : begin (* not implemented *) end;

    vdimarkcolor : begin
                     currmarkercolor := color
                   end;  (* vdimarkcolor *)

    vditextfont : begin (* not implemented *) end;

    vditextcolor : begin
                     currtextcolor := color
                   end; (* vditextcolor *)

    vditextpath : begin (* not implemented *) end;

    vdiinlocator : begin (* not implemented *) end;

  end;   (* case cmdcode *)
 end;
end;     (* soltec *)

(*****************************************************************)
(* END  OF  THE  SOLTEC  VDI  DRIVER                             *)
(*****************************************************************)
```

```
begin (* main *)
programheading;
(*****************************************************************)
(* TEST  ROUTINE  FOR  THE  SOLTEC  VDI  DRIVER                  *)
(*****************************************************************)

soltec(vdiopen,ip,op);   (* Open Workstation *)
soltec(vdiclear,ip,op);  (* Clear Display *)

ip.color := 7; (* White *)
soltec(vdimarkcolor,ip,op);
soltec(vdilinecolor,ip,op);
soltec(vditextcolor,ip,op);
ip.color := 4; (* Red *)

ip.size := 160;

ip.height := 160;

ip.upvec.ix := 0;
ip.upvec.iy := 1;

ip.path.ix := 1;
ip.path.iy := 0;

ip.indev := 1;
ip.mode  := 1;

ip.numpts := 3;
iiii := 0;
repeat
begin
ip.pts[1].ix := iiii;
ip.pts[1].iy := 0;
ip.pts[2].ix := iiii;
ip.pts[2].iy := iiii;
ip.pts[3].ix := 0;
ip.pts[3].iy := iiii;
soltec(vdiline,ip,op);
iiii := iiii + 100
end
until iiii > 2800;
soltec(vdiupdate,ip,op);

soltec(vdiclear,ip,op);
```

```
ip.kind := 1;
soltec(vdimarktype,ip,op);

(*****************************************************************)
programclosing

end (* main *)
```

```
case $# in
     1) if   /bin/test -f $1.c
        then echo
        else
           echo "File `$1.c` not found, in this directory "
           exit 1
        fi
        echo `* processing includes `
        cc -P -I"/usr/src/gks" $1.c
        mv $1.i $1.p
        echo `* processing includes complete `
        echo `* compiling `
        /usr/src/gks/Tool/pas32 $1 SO  > list
        mv a.out $1.out
        grep "PASS" list
        pasxref $1.p > $1.xref
        echo `* compile complete`;;
     *) echo `incorrect number of parameters `
        echo `**********************************`
        echo `* pascomp fn1                    *`
        echo `*================================*`
        echo `* fn1 - file to compile,         *`
        echo `*       must be fn1.c extension  *`
        echo `*                                *`
        echo `* outputs listing to file "list" *`
        echo `**********************************`
        echo ` `;;
     esac
```

```
dev=`tty`
files="-f1 $dev "
if /bin/test $# -eq 0 ]
then
   echo '****************************************'
   echo '* rungks dev1 dev2 dev3 file1 file2  *'
   echo '****************************************'
   echo '* to run the gks system at this time *'
   echo '* you must specify the devices and   *'
   echo '* files to be used. Following is a   *'
   echo '* list of the devices that are       *'
   echo '* defined:                           *'
   echo '*      wis - for output wis to screen *'
   echo '*      chro - output graphics to the  *'
   echo '*             chromatics              *'
   echo '*      plot - output graphics to the  *'
   echo '*             SOLTEC plotter          *'
   echo '****************************************'
   exit 1
fi
flag=1
while /bin/test $1w != w
   do
        case $1 in
                wis);;
                chro)  files="$files -f3 /dev/tty19" ;;
                plot)  files="$files -f4 /dev/tty25" ;;
                -mo)   shift;files="$files -f5  $1";;
                -mi)   shift;files="$files -f6  $1";;
                pc1)   files="$files -f7 /dev/tty19" ;;
                pc2)   files="$files -f8 /dev/tty19" ;;
                tvir1) files="$files -f9 /dev/tty28" ;;
                tvir2) files="$files -f10 /dev/tty17" ;;
                *)     echo "unknown device: $1"; exit 1;;
        esac
        shift
    done
comp.out -k190k $files
```

```
(* ------------------------------
    begin prefix definition
------------------------------- *)
#include "Pas32/preprogram.i"
(* ------------------------------
    begin constdeclarations
------------------------------- *)
#include "Pas32/preconst.i"
#include "Glob/gksconst.i"
#include "Scanner/scanconst.i"
/*#include "meta.const.i"*/
(* ------------------------------
    begin type declarations
------------------------------- *)
#include "Pas32/pretype.i"
#include "Glob/gkstype.i"
#include "Scanner/scantype.i"
(* #include "vtypes.i" *)
#include "soltectypes.i"
(* ------------------------------
    begin variable declarations
------------------------------- *)
#include "Pas32/prevar.i"
#include "Glob/gksvar.i"
#include "Scanner/scanvar.i"
(* #include "Chro/chrovar.i" *)
#include "soltecvar.i"
#include "mainvar.i"
/*#include "meta.var.i"*/
(* ------------------------------
    begin pas32 io routines
------------------------------- *)
#include "Pas32/get.i"
#include "Pas32/put.i"
#include "Pas32/eof.i"
#include "Pas32/eoln.i"
#include "Pas32/writechar.i"
#include "Pas32/writeline.i"
#include "Pas32/shortmsg.i"
#include "Pas32/longmsg.i"
#include "Pas32/regularmsg.i"
#include "Pas32/writeint.i"
#include "Pas32/writereal.i"
#include "Pas32/readchar.i"
#include "Pas32/readline.i"
```

```
#include "Pas32/open.i"
#include "Pas32/close.i"
#include "Pas32/progheading.i"
#include "progclosing.i"
(* ------------------------------
    gks functions and procedures
------------------------------- *)
(*      Global GKS procedures    *)

(*#include "Gks/clipit.i"*)
(*#include "Gks/errorhandler.i"*)

(*      Wi procedures            *)

(*#include "Gks/Wis/wiline.i"*)
(*#include "Gks/Wis/wimarker.i"*)

(*      Line                     *)

(*#include "Gks/Line/drawline.i"*)
(*#include "Gks/Line/defline.i"*)
(*#include "Gks/Line/inqline.i"*)
(*#include "Gks/Line/setline.i"*)
(*#include "Gks/Line/qryline.i"*)

(*      Marker                   *)

(*#include "Gks/Marker/drawmarker.i"*)
(*#include "Gks/Marker/defmarker.i"*)
(*#include "Gks/Marker/inqmarker.i"*)
(*#include "Gks/Marker/setmarker.i"*)
(*#include "Gks/Marker/qrymarker.i"*)

(*      View/Window              *)

(*#include "Gks/View/fstoview.i"*)
(*#include "Gks/View/stowindow.i"*)
(*#include "Gks/View/stoview.i"*)
(*#include "Gks/View/setwindowind.i"*)
(*#include "Gks/View/qrywindow.i"*)
(*#include "Gks/View/qrywindowind.i"*)
(*#include "Gks/View/qryview.i"*)
(*#include "Gks/View/fdefwsview.i"*)
(*#include "Gks/View/defwswindow.i"*)
(*#include "Gks/View/defwsview.i"*)
```

```
(*#include "Gks/View/inqwswindow.i"*)
(*#include "Gks/View/inqwsview.i"*)

(*    drivers                    *)

/*#include "chrodriver.i"*/

(*  init gks and ws procedures   *)


(*#include "Gks/initws.i"*)
(*#include "Gks/initgks.i"*)
/*#include "metafile.i"*/
/*#include "distdriver.i"*/
/*#include "Gks/distributor.i"*/
/*#include "interpret.i"*/
#include "soltecsubs.i"
#include "soltecdriver.i"


(* ------------------------------
    begin scanner definition
------------------------------  *)
/*#include "scanner.i"*/
(* ------------------------------
    begin main program
------------------------------  *)
#include "main.i"
```

```
;  begin (* main *)
      programheading
   ;  initwstables
   ;  initgkstables
   ;  initmetaout( mofile )
   end (* main *)
```

A VDI DRIVER FOR SOLTEC PLOTTERS

by

EUGENE JANOVITZ

B.E., Pratt Institute of Brooklyn, 1973

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Abstract

The Graphical Kernel System (GKS) provides a set of functional specifications for computer graphics programming which can be used by the majority of applications that produce computer generated pictures. A graphics device driver represents that portion of the graphics system software which translates commands and data from the device-independent Virtual Device Interface (VDI) into a form required by a particular input/output medium.

Expansion and further definition of these two international standards, GKS and VDI, is an ongoing activity. Ambiguity and some inconsistencies present in both standard documents, remain to be worked out. This paper, after introducing the existing graphics standards, discusses the GKS and the VDI standards on which this project is based on.

Each device driver is dependent on the respective physical device, hence the hardware and software capabilities of the SOLTEC 281 Plotter are then discussed rather in detail. Finally, the design decisions made by the author, as problems were encountered during implementation of this driver, are presented. In conclusion, the author presents the results of this project.