A VDI INTERFACE FOR A MICROPROCESSOR GRAPHICS SYSTEM

by

PAUL L. STEVENS

B. S., V. P. I. & S. U., 1978

A MASTER'S REPORT

submitted in parital fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

William Hankley
Major Professor

TABLE OF CONTENTS

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.

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

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.

LIST OF FIGURES

Chapter 1

INTRODUCTION

BACKGROUND

The Computer Graphics Virtual Device Interface (VDI) is a computer graphics standard proposal under development by the American National Standards Institute (ANSI) Virtual Device Interface Task Group (X3H33) for adoption as an American National Standard. The task group [19] defines the VDI as:

> "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 drivers."

The goal of the task group is to develop a standard interface between device-independent software packages and device driver routines. This allows the software package to interface with each device driver without regard to details of the physical characteristics of the device or the particular graphics primitive operations implemented on each device. Calls to a particular device driver contain the same parameters, in the same format, as every other device

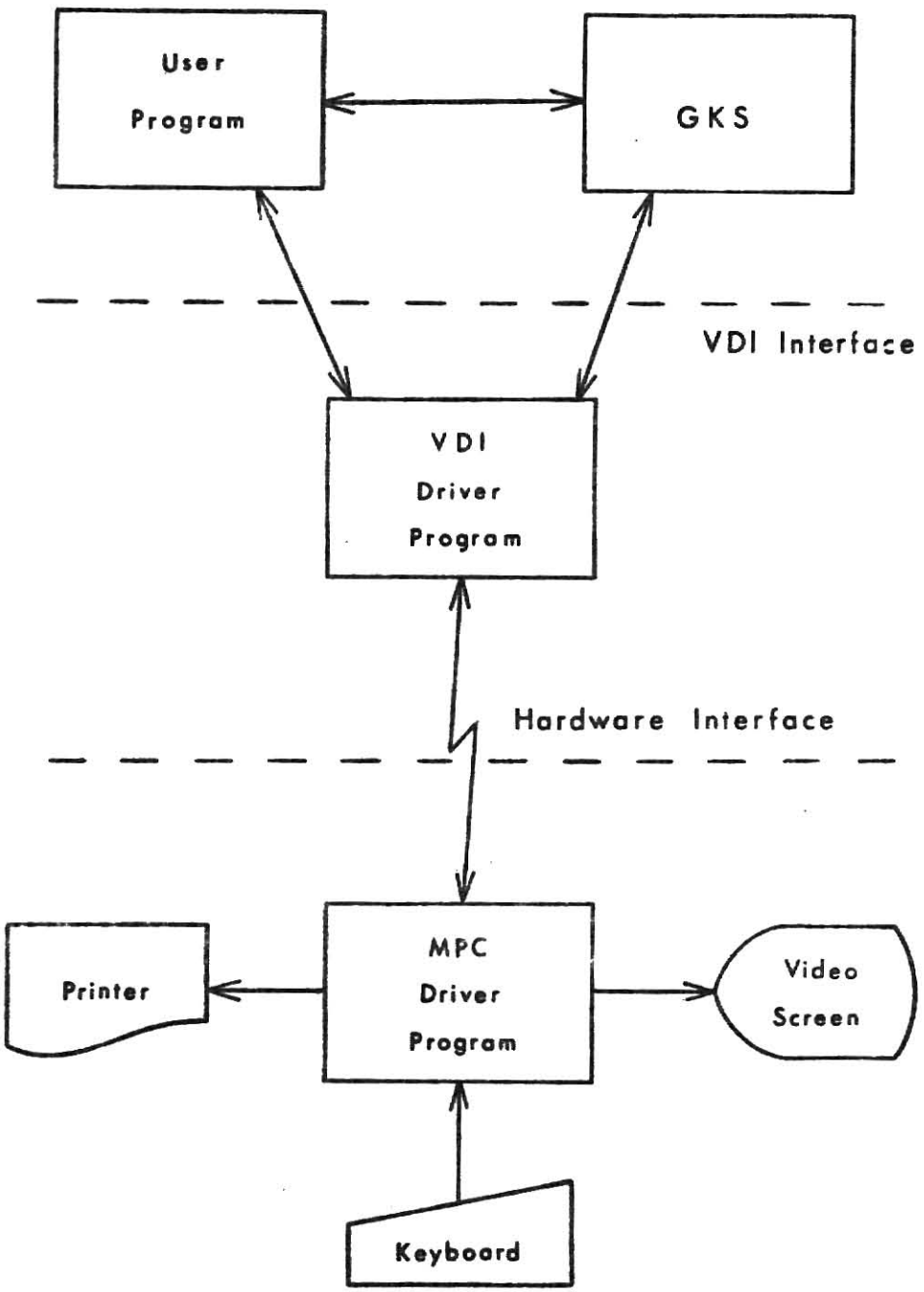
driver. Each device driver implements a standard set of primitive operations for use in generating and inputing graphical information from the device.

Use of a Virtual Device Interface is advantageous to both computer graphics systems' implementors and applications programmers. Both classes of users will generate less code to implement a computer graphics program and will spend less time maintaining the code if VDI is used to interface to graphics devices. Computer graphics hardware vendors will also benefit if the VDI standard is accepted and becomes widespread. Given a VDI hardware interface standard, vendors will be able to produce graphics devices that are interchangeable - a plotter could be replaced with a graphics terminal with little or no modification required to the graphics software.

The VDI task group works closely with and under ANSI's task group (X3H3) that is developing the standard for the Graphical Kernel System (GKS). First proposed by the German Institute of Standardization (DIN), the GKS standard [4] provides a functional specification of a set of functions or subroutines which implement "a basic graphics system that can be used by the majority of applications that produce computer generated pictures." The VDI concept is integral to the design of the GKS, providing a standard,

device-independent interface to each device accessed by the system. Figure 1.1 contains a diagram which illustrates the relationship between GKS and the VDI.

Figure 1.1 System Structure of Project



SCOPE OF WORK

Implementation of a version of the Graphical Kernel System is in progress at Kansas State University. During the spring semester of 1983, the initial design and implementation work was begun. The development and refinement of the GKS is a continuing effort. Recently, additions and enhancements to the system have been the subject of numerous master's projects. This report documents the effort of one of such master's projects.

The scope of this particular master's project is limited to the development of a VDI interface for the Columbia Data Products Multi-Personal Computer (MPC). This interface is designed to link the GKS system, running on a Perkin-Elmer 32/20 computer, to the MPC. Since the MPC is not a terminal, but a personal computer with computer graphics capabilities, implementing this interface requires that software be written for both machines. This software must not only execute the required graphics functions, but also the data communications between the two machines. The VDI and MPC Driver Programs of Figure 1.1 were designed to implement this VDI interface.

During the development phase of of the project, all

efforts failed to establish a data communications link between the 32/20 and the MPC running a BASIC program. Rather than suspend development until the interface was working, it was deemed necessary to continue development and testing of the software in order to complete as much of the project as possible before leaving the campus. The problem is discussed further in the Testing Section of Chapter 3. Chapter 2 of this report presents information introducing the Graphical Kernal System and the Virtual Device Interface and their relevance to the project. Also discussed in Chapter 2 is the Association for Computing Machinery's Core System - a related computer graphics standard proposal. The first implementation of GKS and VDI, called the Graphics System Extension, is also presented.

Chapter 3 presents a detailed discussion of the software design and implementation of the VDI and MPC Driver Programs. The final section of Chapter 3 discusses the details of software testing and the final project status. Chapter 4 provides recommendations for further work concerning this project, and discusses deficiencies in the GKS and VDI standards' proposals.

Chapter 2

DEVICE INDEPENDENT COMPUTER GRAPHICS

DEVICE INDEPENDENCE AND PORTABILITY

Device-independence in computer graphics systems has been a popular topic in recent years. Much of the literature talks about device-independence as a means of achieving portability [7,10,14,17,21,22]. The concept of portability implies that something can be carried from one place to another while maintaining its functionality. In computer graphics, two types of portability are desired: Application Program Portability and Application Programmer Portability [14,17].

In order to achieve portability of application programs, the graphics software packages which are used by the program must present a uniform interface to the applications programmer regardless of the graphics hardware that is used. The package must provide a common set of graphics primitives which can be used for interaction with all graphics devices. In this way, each device can then be viewed by the programmer as having the same capabilities as

every other device - the graphics package must be device-independent. As defined in [10], a graphic software system *can be called device-independent* "if it can be used in conjunction with two or more different graphics terminals ... without the need to modify the application programs which use the system." Thus a graphics software system which provides device-independence also provides Application Program Portability within the system. If the same graphics software system also exists in another installation, Application Program Portability is achieved between installations also. This is where the use of common graphics standards such as the Graphical Kernel System [4] become important.

Use of graphics software systems which result in portable and device-independent programs has many advantages. Duplication of effort is eliminated because programmers need not write similar programs for each device in order to implement the same graphics application, the same code can be used for each device. The program is also insulated from changes in hardware. If a display is updated with a newer model, the program is insulated from any differences between the two hardware devices.

The programmer then, only has to worry about the interface into the graphics software system and not individual graphics devices. Figure 2.1 contains a diagram which illustrates this point using the GKS system as an example. The software modules at each level of the diagram can only access the functions provided to it by the level immediately below it. In this system, the User Program can only access the functions provided by GKS and the Operating System, it has no way to access the functions of a particular device or even knowing what hardware functions are available on the device.

Portability of graphics application programmers is also possible. By utilizing a device-independent software system, the programmer no longer has to become intimately familiar with the functions and properties of each device used by his software. This improves the programmer's productivity since he will not have to experience a learning period each time use of a new device is required. In addition, if the applications are written using a standard graphics software system, such as GKS, the programmer may then write software for other installations which use the system without having to learn the details of a new software system or the devices supported by the installation.

Figure 2.1 Isolation of Device Functions

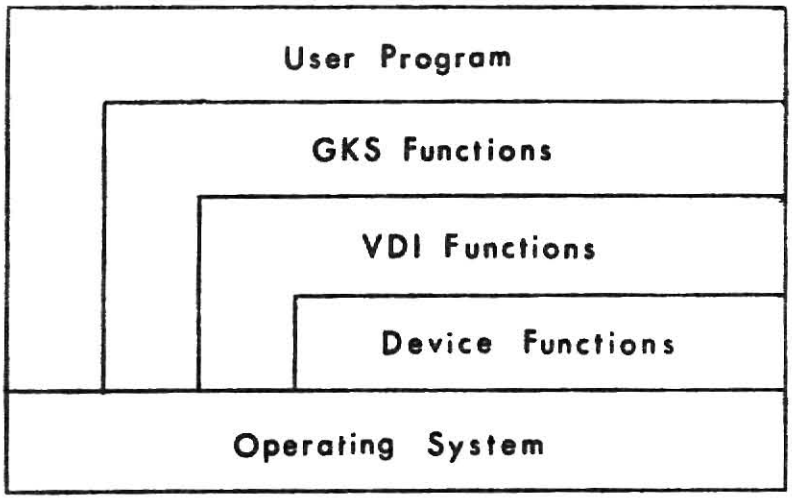
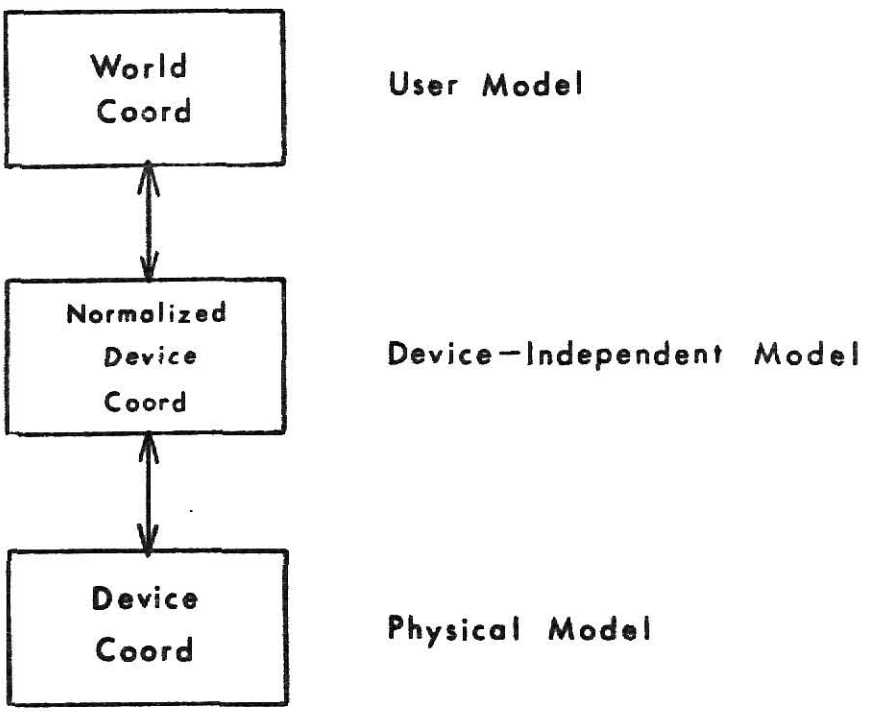


Figure 2.2 Device Independent Coordinates



One of the properties of a device-independent graphics system is that it uses a Device-Independent Coordinate System for specifying graphical information. Use of a device-independent Coordinate System frees the user from being concerned with the physical viewsurface characteristics of each device. Typically, modern graphics systems allow the user himself to define the units of the coordinate system [4]. This is made possible by using an intermediate coordinate system internally in the graphics software system. Figure 2.2 is a diagram illustrating this technique. The user provides the graphics system with a definition of the coordinate system that he wants to use to specify graphical information, this user's view of the data is called the World Coordinate System. The graphics system uses a different coordinate system internally, called Normalized Device Coordinates. The graphics system must be able to apply the appropriate transformations to convert data back and forth between the two coordinate systems. The graphics system must also be able to apply transformations to convert back and forth from Normalized Device Coordinates (NDC) to the physical coordinates of the device, called Device Coordinates (DC).

Use of NDC coordinates allows the graphics system to maintain data that is both device-independent and independent of the World Coordinate model specified by the user.

THE GRAPHICAL KERNEL SYSTEM

The Graphical Kernel System (GKS) is a device-independent computer graphics software system defined by a draft standard proposed by the International Standards Organization (ISO). GKS is also being considered for adoption by the American National Standards Institute (ANSI) as an American National Standard. The GKS provides a common, device-independent interface to graphics devices which insulate the user from Device-Dependent features of graphics devices. Figure 1.1 is a diagram of the logical design of GKS. The figure shows the hierarchical layering of the software used to insulate the user from the physical device characteristics (shown in another form in Figure 2.1).

There are two major software layers in GKS, the Device-Independent Layer and the Device-Dependent Layer. The Device-Dependent Layer is comprised of the software drivers for each of the graphics workstations and the Device Supervisor. The Device Supervisor is responsible for providing the mapping between logical devices in the Device-Independent Layer and physical devices in the Device-Dependent Layer. The Device-Independent Layer is comprised of the rest of the GKS and the User Program. The

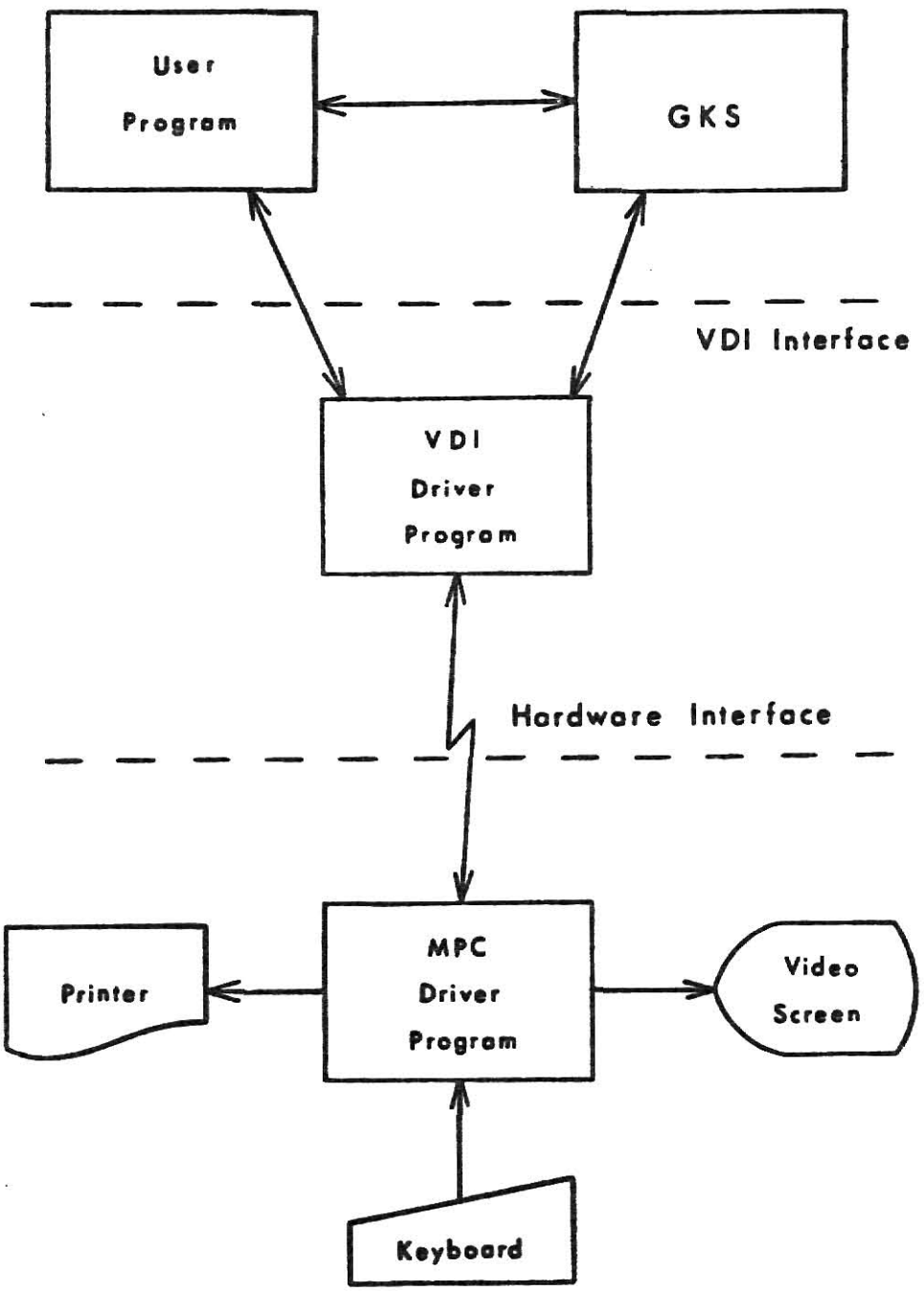
Device-Independent Layer, as the name suggests, operates on device-independent data and data structures. This layer handles the mapping between the user-defined World Coordinate system and NDC coordinates while the Device-Dependent Layer maps between NDC and Device Coordinates.

While GKS provides a set of common graphics primitives for manipulating graphical information (Polyline, Polymarker, Text etc.), it also provides two mechanisms which allow the user to access non-standard graphics functions. The first mechanism is actually a GKS graphics primitive that is designed to provide optional extensions to the basic functions of GKS. Called the Generalized Drawing Primitive (GDP), this primitive is used to execute additional common primitives such as circle and rectangle drawing functions which may or may not exist in hardware devices. The second mechanism is called the Escape Function and was defined to allow the applications programmer to use features supported by a device but not by GKS or the GDP. The Escape Function allows the user to circumvent normal graphics processing and pass a text string directly to the device to invoke a desired hardware function. Use of the Escape Function, however, makes the application program device-dependent and may hinder the porting of the program to another device or installation.

THE VIRTUAL DEVICE INTERFACE

As described in Chapter 1, the Virtual Device Interface (VDI) is a specification of the software interface between a device-independent graphics software system and device-dependent graphics device drivers. The term "Device Driver" is used to mean that portion of the graphics software which translates VDI commands into the form required by the device(s) supported by the driver. In the VDI, as with GKS, each device driver handles the I/O for one workstation. A workstation is a logical grouping of devices comprised of a device containing some type of viewsurface, coupled with zero or more associated graphical input devices. Figure 2.3 is a block diagram of the VDI Device Driver implemented in this master's project. This diagram shows that the driver (comprised of the VDI Driver Program and the MPC Driver Program) actually controls three devices: the Video Screen; the Keyboard and a Graphics Printer used for making hardcopies of the screen.

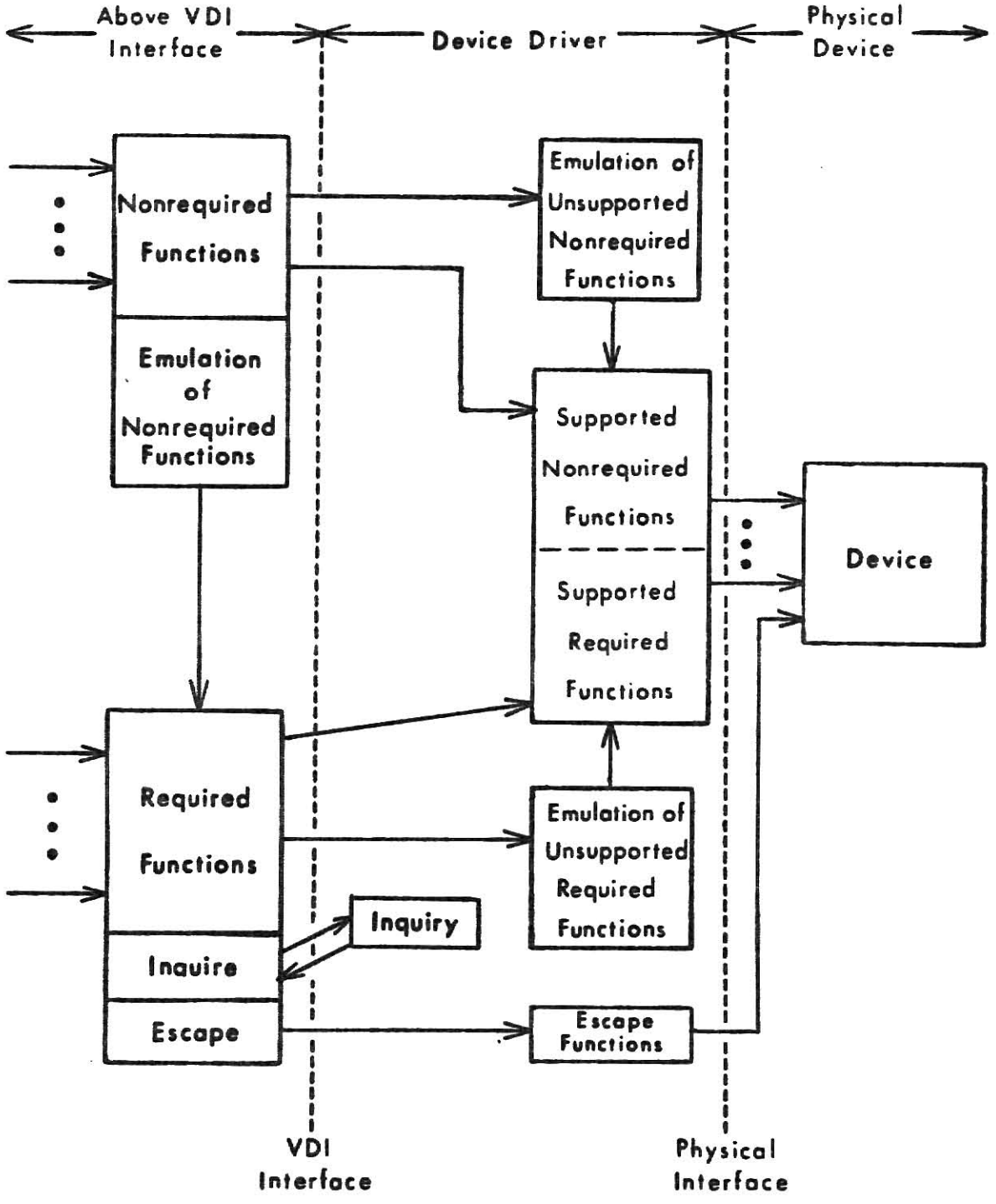
Figure 2.3 VDI Device Driver Design



The VDI provides a common interface to each device driver used by a graphics system or application program. Figure 2.4 contains a diagram of a model of the Virtual Device Interface [20]. The X3H33 Task Group specifies that each VDI workstation (driver) must support a minimal set of graphics functions that it calls Required Functions. These functions may be supported by the graphics hardware or emulated by the device driver. Functions available on the device(s) are called Supported Functions while those which must be emulated by the driver are called Unsupported Functions.

The Task Group [20] also identifies a set of graphics functions, called Nonrequired Functions, which represent a standard set of extensions to the VDI and which corresponds to the functions available under the GKS Generalized Drawing Primitive. The Nonrequired Functions are defined by the VDI specification (for example, CIRCLE) but the device driver is not required to implement them. Before using a Nonrequired Function, a program must query the driver to determine if the function is available.

Figure 2.4 Virtual Device Interface Model



Nonrequired Functions may be "Supported" by the graphics device(s) or they may be emulated in the driver or the program which invokes the driver if they are "Unsupported." The VDI also includes an Escape Function which is a lower level implementation of the GKS Escape Function described in the previous section.

In [20] the VDI Task Group defines the graphics functions which it proposes for the VDI Standard. Graphics Software Systems Inc. has developed a VDI for the CP/M operating system, called the Graphics System Extension (GSX), which is marketed by Digital Research. The GSX system defines a somewhat different set of graphics functions for the VDI [13]. This is to be expected however, since the VDI Standard is still under development. Thus it was necessary to define, for the purposes of this project, a set of VDI commands which might be implemented by the device driver. Figure 2.5 is a PASCAL type definition which specifies the VDI commands which were chosen as a reasonable command set for implementation. Figures 2.6 and 2.7 are type definitions which define the input and output parameters for the various VDI commands.

Figure 2.5 - VDI Commands

```
type vdicmdtype =
   ( vdiopen,        (* Open Workstation *)
     vdiclose,       (* Close Workstation *)
     vdiclear,       (* Clear Workstation *)
     vdiupdate,      (* Update Workstation *)
     vdiescape,      (* Escape Function *)
     vdiline,        (* Draw Polyline *)
     vdimark,        (* Draw Polymarker *)
     vditext,        (* Draw Text *)
     vdiarea,        (* Draw Filled Area *)
     vdigdp,         (* Generalized Drawing Primitive *)
     vdicharh,       (* Set Character Height *)
     vdicharupv,     (* Set Character Up Vector *)
     vdilinetype,    (* Set Line Type *)
     vdilinewidth,   (* Set Line Width *)
     vdilinecolor,   (* Set Line Color *)
     vdimarktype,    (* Set Marker Type *)
     vdimarksize,    (* Set Marker Size *)
     vdimarkcolor,   (* Set Marker Color *)
     vditextfont,    (* Set Text Font *)
     vditextcolor,   (* Set Text Color *)
     vdifillcolor,   (* Set Fill Color *)
     vditextpath,    (* Set Text Path *)
     vdiinlocator,   (* Input Locator *)
     vdiinvaluator,  (* Input Valuator *)
     vdiinchoice,    (* Input Choice *)
     vdiinstring,    (* Input String *)
     vdiinmode)      (* Set Input Mode *)
```

Figure 2.6 VDI Driver Input Parameters

```
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);
      vdilinecolor,
      vdimarkcolor,
      vditextcolor,
      vdifillcolor:(color:unit);
      vdilinewidth,
      vdimarksize  :(size:integer);
      vditextpath  :(path:ipoint);
      vdiinlocator:(locdev :index;
                     locpos:ipoint);(* initial pos*)
      vdiinvaluator:(valdev:index);
      vdiinchoice  :(chdev:index);
      vdiinstring  :(strdev:index;
                     strlen:textcharrange);
      vdiinmode    :(indev:index;
                     mode:index)
  end   (*  vdiparm  *)
```

Figure 2.7 VDI Driver Output Parameters

```
;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 *)
```

ANOTHER GRAPHICS STANDARD

The proposed GKS and VDI Standards are by no means alone in the world of computer graphics standards, several other related standards have been developed and proposed for adoption by ANSI or the ISO. Probably the most widely known is the Core System developed under the auspices of the Association for Computing Machinery (ACM) Special Interest Computer Group on Graphics (SIGGRAPH) [9]. The Core System, like GKS, is a standard proposed for providing a device-independent interface to graphics devices via use of a suite of special graphics subroutines.

The proposed Core System standard, developed by the ACM/SIGGRAPH Graphic Standards Planning Committee (GSPC), defines a general-purpose, three-dimensional graphics system designed to facilitate program portability. Probably due to its early beginnings in the development of graphics standards, the Core System is a Pen-Oriented system which makes use of the Current Drawing (Pen) Position concept widely used in software developed for plotters. The GKS and VDI, which are oriented more toward raster devices, do not use the Current Drawing Point technique for specifying coordinate information.

While the software organization of the Core system is similar to that of GKS (see Figure 1.1), one important difference exists in the Device Dependent Layer of the system. The Core System does not support the Workstation Concept as does GKS. Instead, each physical device is treated as a separate entity with its own device driver. While the Core system does support logical input device types such as Pick and Valuator Devices, it is the applications programmer who determines which physical devices are available to the program, not the graphics system. Since all device types are not required to be available in a (programmer-defined) workstation, as is the case with GKS, portability problems can arise due to the unavailability of a particular input device in a new installation. Also, since there is not a standard for the interface between the Core System and its device drivers, such as the VDI standard, porting an implementation of the Core System would probably be much more difficult than porting GKS.

THE GRAPHICS SYSTEM EXTENSION

The Graphics System Extension (GSX) is a software upgrade package for the CP/M and CP/M-86 operating systems [13]. Developed by Graphics Software Systems Inc. and marketed by Digital Research, GSX is one of the first attempts to implement the VDI standard. GSX provides a device-independent graphics interface for a user or graphics software package running under CP/M. Calls to GSX routines are made through the standard CP/M function-call mechanism - branching to a location in the base-page jump table. GSX is comprised of three software components: the Graphics Device Operating System (GDOS), the Graphics I/O System (GIOS), and the Gengraf utility routine.

The Graphics Device Operating System component of GSX is analogous to the Basic Disk Operating System (BDOS) in CP/M and provides the device-independent graphics interface for the user. When the GSX system is set up, the normal BDOS entry in the base-page jump table is replaced by the entry-point address of GDOS. The user is then able to transfer to either the GDOS or the BDOS (via GDOS) through the same jump-table entry. All function calls to GDOS specify a parameter list which includes an Operation Code, a Control Array, a Parameter Array, and a Point Array with

coordinates specified in the range -32767 to 32767 (Integer NDC). The function of the GDOS actually corresponds to the Device Supervisor in GKS (See Figure 1.1), providing: coordinate scaling for the device drivers; control over which device driver is resident in memory; and interfacing with the workstation device driver (in the GIOS) via subroutine calls. Thus, the user interface with GSX is actually on a higher level than that defined for the VDI, the GDOS-GIOS interface is the actual VDI-level interface.

The Graphics I/O System (GIOS) is made up of the device-dependent device drivers which actually interface with the physical graphics devices. Each GDOS call to the GIOS specifies the following parameters: an Operation Code; an Input Control Parameter Array; an Input Parameter Array; an Input Point Coordinate Array; an Output Control Array; an Output Parameter Array; and an Output Point Coordinate Array. This uniform parameter passing approach differs from that defined in the proposed VDI standard, the standard defines a command-specific set of parameters for each VDI command.

The GSX Gengraf program is used by the application programmer to configure his program for use with GSX. Gengraf appends a special loader routine to the user program which sets up the GDOS and GIOS at run time.

Several incompatibilities exist between the GSX and the current VDI standard proposal. Since GSX was developed concurrently with the development of the standard, some incompatibilities were virtually assured. Examples are the different parameter passing strategy and the assignment of different VDI operation code numbers. Another difference results from the fact that GSX is designed to be compatible with any CP/M program. In effect, this binds the GSX implementation to the operating system and its conventions. The VDI and GKS standard proposals, on the other hand, are designed to be bound to high-level computer languages. User calls to GDOS must be accomplished via assembly or machine language which puts a burden on many applications programmers. This difference in binding explains many of the incompatibilities between GSX and the VDI standard.

Chapter 3

DEVICE DRIVER DESIGN AND IMPLEMENTATION

VDI DEVICE DRIVER DESIGN

The original software design developed for this project provides a VDI-level interface between GKS software and the MPC computer. As shown in Figure 2.3, the driver is designed as two software modules. The first, called the VDI Driver Routine, is a PASCAL subroutine which is callable by the GKS system. This routine provides the VDI interface to the GKS system. The VDI Driver Routine supports the VDI commands specified in Figure 2.5. The input and output parameters for the routine are shown in Figures 2.6 and 2.7. The second software module, called the MPC Driver Program, is a BASIC program which runs on the MPC computer itself. Appendix A contains a source listing of the MPC Driver Program. The MPC Driver Program executes the actual graphics functions in the MPC based on directives from the VDI Driver Routine. The VDI Driver Routine and the MPC Driver Program are designed to communicate via a serial communications link between the 32/20 and the MPC. The two

routines communicate via the MPC Interface Command Language described in the next section.

As shown in the source listing contained in Appendix B, the VDI Driver Routine is modular in design, the subroutine is made up of a single PASCAL Case statement. The routine has a case entry for each of the possible VDI commands. This modularity makes the routine design simple and the code easy to modify and maintain. The VDI Driver Routine contains calls to a suite of support routines which execute lower-level functions. Source listings of these subroutines is contained in Appendix C.

MPC INTERFACE AND SOFTWARE

The Columbia Data Products MPC is capable of displaying 16 different colors when equipped with the color adapter card and a color monitor. The MPC can display color graphics when the screen is in Medium Resolution Graphics Mode. This mode provides 320 by 200 pixel screen resolution as shown in Figure 3.1. The MPC has an additional graphics mode called High Resolution Mode which has twice the resolution, but this mode is monochrome only. The Medium Resolution Mode was chosen for use in the project due to the ability to display color. Unfortunately the MPC uses only two bits per pixel in this mode, this means that though there are 16 possible colors, only four may be displayed on the screen at one time. Figure 3.2 contains a list of pertinent characteristics of the MPC video display.

Figure 3.1 MPC Medium Resolution Screen Format

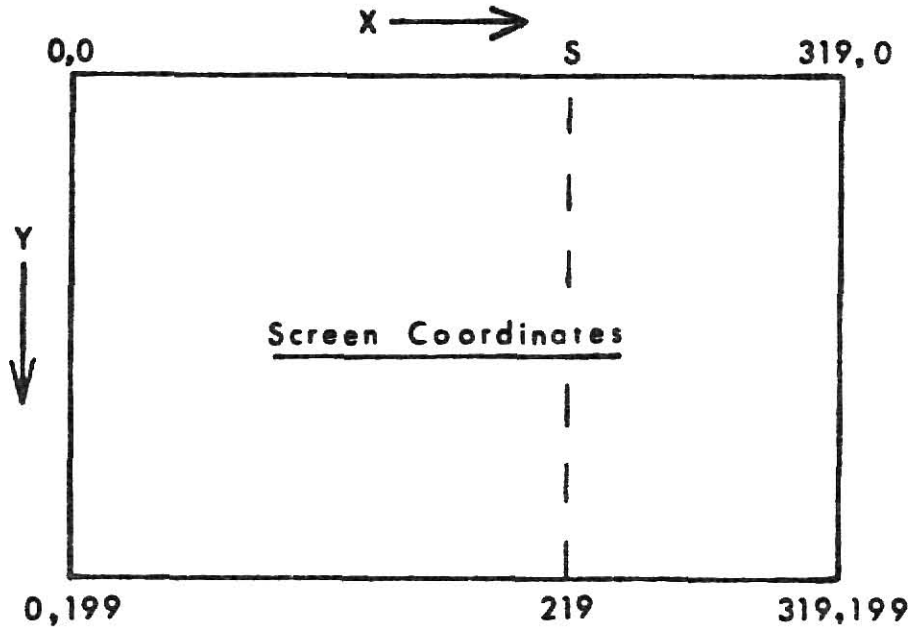


Figure 3.2 - Display Screen Characteristics

Screen Size = 320 Pixels (X) x 200 Pixels (Y)
= 22 cm Wide (X) x 15.1 cm High (Y)
= 40 Characters (X) x 25 Characters (Y)

$$\text{Aspect Ratio} = \frac{200/15.1}{320/22} = 0.910596 \text{ Height/Width}$$

Character Size = 8 Pixels (X) x 8 Pixels (Y)
= 0.55 cm (X) x 0.604 cm (Y)

Largest Square = 220 Pixels (X) x 200 Pixels (Y)
= 15.1 cm (X) x 15.1 cm (Y)
= 27.5 Char. (X) x 25 Char. (Y)

The MPC Driver Program handles the communications interface between it and the VDI Driver Routine and executes commands sent over the interface by the VDI Driver. The MPC Driver Program is written in the BASIC programming language. While BASIC was not the first choice for this application, it was chosen because it was the only language available on the MPC which incorporates extensions for graphics. This was especially important since the only graphics primitives supported by the MPC hardware are reading and writing pixels.

A well-defined command language was developed for use between the VDI Driver and the MPC Driver. The command set is similar to that specified for the VDI but on a lower, less sophisticated level. The command formats were designed to be consistent and easily parsed in BASIC. Figures 3.3 and 3.4 contain the command language specification in Backus-Naur Form. Figure 3.3 contains a specification of the grammar of the commands which may be issued by the VDI Driver and the responses (if any) returned by the MPC Driver. Figure 3.4 contains the specifications for the various command and response fields identified in Figure 3.3.

Figure 3.3 MPC Interface Command Language Definition

[...] (n) Specifies that n Repetitions of the String in Brackets are required.

Command	Format
Set Color	C <color> <ret>
Erase Screen	E <ret>
Print Screen	P <ret>
Display Text	T <up_vec> <path_vec> <text> <ret> <x_coord> , <y_coord> <ret>
Draw Lines	L <ret> <no_coords> <ret> [<x_coord> , <y_coord> <ret>] (<no_coords>)
Draw Markers	M <mark_type> <ret> <mark_size> <ret> <no_coords> <ret> [<x_coord> , <y_coord> <ret>] (<no_coords>)
Fill Area	F <bound_color> <ret> <x_coord> , <y_coord> <ret>
Draw Arc (or Circle)	A <connect_flag> <ret> <x_coord> , <y_coord> , <radius> , <start_deg> , <end_deg> <ret>
Draw Rectangle	R <fill_flag> <ret> <ll_x_coord> , <ll_y_coord> , <ur_x_coord> , <ur_y_coord> <ret>
Input String	Command: IS <ret> Response: <text> <ret>

Figure 3.3 MPC Interface Command Language Definition (Continued)

Command	Format
Input Cursor	Command: IC <ret> Response: <x_coord> , <y_coord> <ret>
Sample Cursor	Command: SC <ret> Response: <x_coord> , <y_coord> <ret>
Sample Enable	SE <ret>
Sample Disable	SD <ret>
Sample Keys	Command: SK <ret> Response: <key> <ret>

Figure 3.4 MPC Interface Command Language Field Definitions

{ n .. m } specifies the Range of Legal Values

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<color> ::= <digit> { 0 .. 7 }

<ret> ::= ASCII Carriage Return Character (Decimal 13)

<coord> ::= <digit> | <digit> <digit> |
            <digit> <digit> <digit>

<x_coord> ::= <coord> { 0 .. 319 }

<y_coord> ::= <coord> { 0 .. 199 }

<ll_x_coord> ::= <x_coord>

<ll_y_coord> ::= <y_coord>

<ur_x_coord> ::= <x_coord>

<ur_y_coord> ::= <y_coord>

<vector> ::= U | R | D | L

<up_vec> ::= <vector>

<path_vec> ::= <vector>

<text> ::= A String of ASCII Characters (40 Max.)

<no_coords> ::= <coord> { 1 .. 999 }

<bound_color> ::= <color>

<mark_type> ::= <digit> { 1 .. 4 }

<mark_size> ::= <coord> { 1 .. 199 }

Figure 3.4 MPC Interface Command Language Field Definition (Continued)

```
<fill_flag> ::= 0 | 1

<connect_flag> ::= 0 | 1

<radius> ::= <coord>

<start_deg> ::= <coord>    { 0 .. 360 }

<end_deg> ::= <coord>    { 0 .. 360 }

<key> ::= <digit> | <digit> <digit>
```

These commands are issued by the VDI Driver in the 32/20 as requests to the MPC Driver running in the MPC. The VDI Driver is responsible for interpreting the VDI commands it inputs from the Device Supervisor and issuing the appropriate command(s) to the MPC Driver to execute the primitive(s). A diagram illustrating the relationship between these three (3) programs is contained in Figure 2.3.

Since the MPC Interface is shielded from the VDI Interface, where access by applications programs is possible, certain assumptions can be made about the correctness of the data passing over the interface. For example, the coordinates contained in the various commands are specified in Screen Coordinate Units since the VDI Driver has already done the appropriate clipping and scaling that is required before the data is passed over the interface. This means that the interface is device dependent as opposed to the VDI interface which is device independent.

The MPC Driver retains only three (3) status variables, these are the Current Drawing Color, Current Key and the Cursor Position. The Current Drawing Color is set by the Set Color command and remains in effect until explicitly modified by another invocation of the Set Color command. The Current Key is the number of the last function key hit

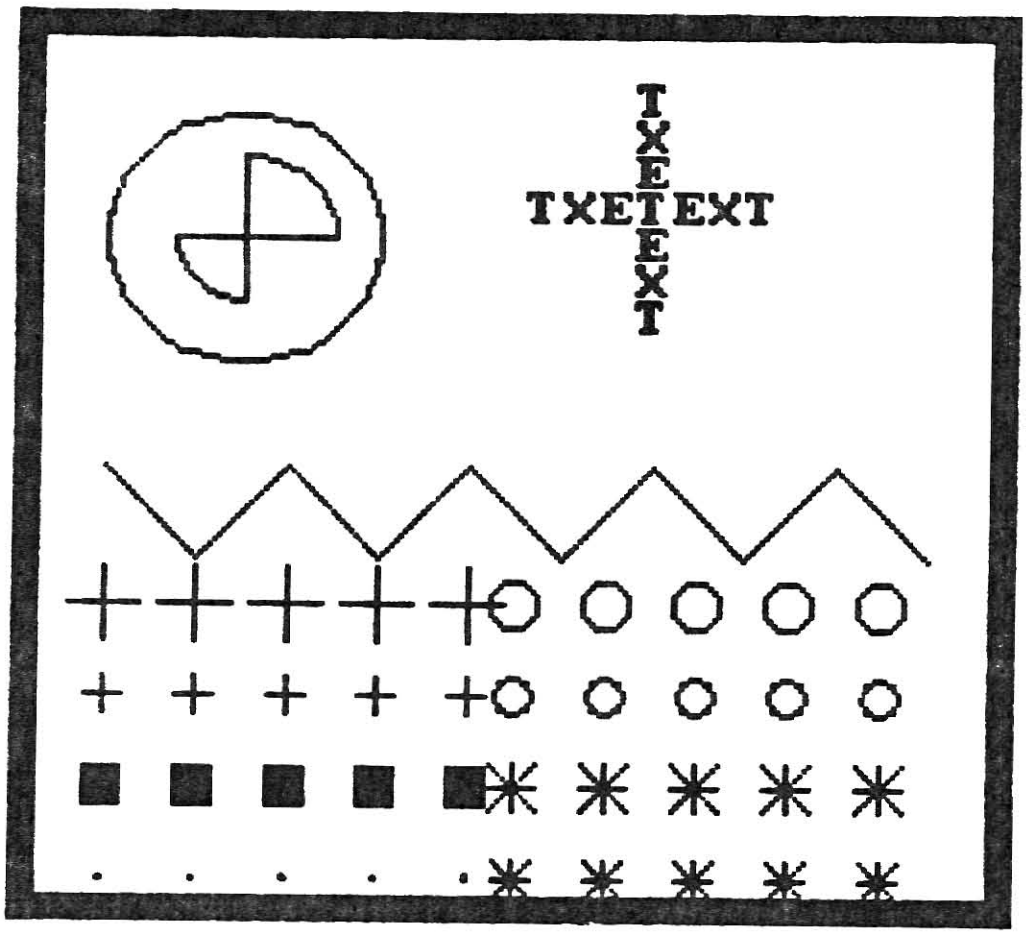
on the keyboard. If no valid key has been hit since the last Sample Key Command was executed, the Current Key will have the value of the ASCII Bell Character (Decimal 7). The Cursor Position is affected by each command which specifies an X, Y coordinate pair. The position of the cursor is maintained at the most recently specified coordinate except after execution of cursor control commands.

The Set Color Command sets the Current Drawing Color to the value specified by the <color> field in the command. The Current Drawing Color is set to white as the default during initialization. The color numbers and their associated colors are shown in the table below.

Number	Color
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Yellow
7	White

The Print Screen Command causes a copy of the video screen to be produced on the Prism 80 printer connected to the MPC computer. Figure 3.5 is a sample of the hardcopy of the MPC screen generated on the Prism 80 printer.

Figure 3.5 Sample Prism 80 Graphics Hardcopy



A listing of the MPC Interface Commands used to generate this figure is contained in Appendix D. The MPC Driver Program uses a subroutine called "Color-It", obtained from an outside vendor, to copy the graphics screen to the printer. This machine language subroutine must be loaded into the memory of the MPC prior to loading the MPC Driver Program.

The Display Text Command is used to generate text on the video screen. The <text> command parameter specifies a string of up to forty (40) ASCII characters to be displayed. The command contains four (4) parameters besides the text string which are used to control the generation of the text. Two of the command parameters represent vectors which specify the orientation of individual characters and the text string itself. The Character Up Vector, <up_vec>, specifies the orientation of individual characters. This vector specifies the direction in which the top of the character will point. The vector may specify Up (0,1), Right (1,0), Down (0,-1) or Left (-1,0) as the Character Up Vector. The vector is specified by using the letters U, R, D or L for Up, Right, Down or Left in the command string. The default Character Up Vector value is Up. Figure 3.6 illustrates the use of the Character Up Vector.

Figure 3.6 Illustration of Character Up Vector Concept

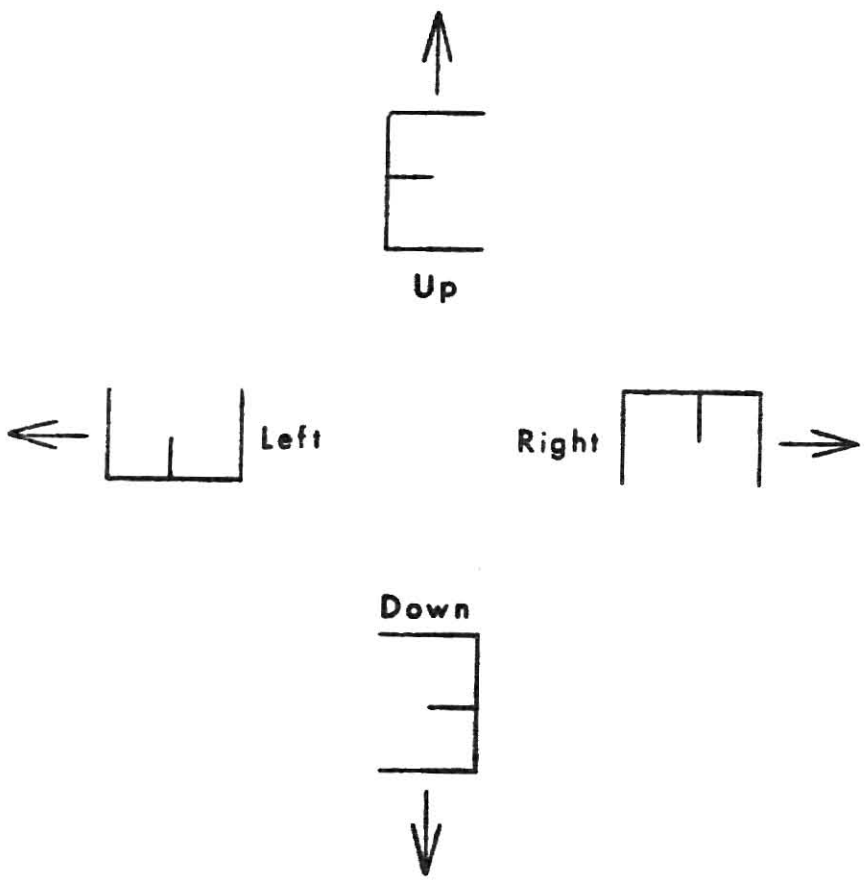
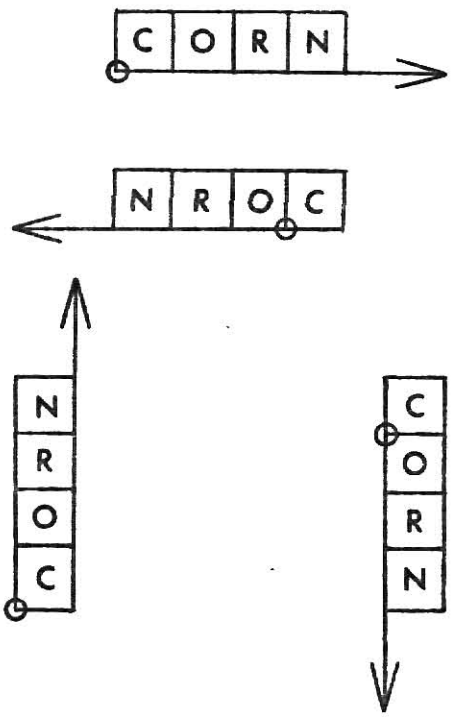


Figure 3.7 Illustration of Text Path Vector Concept



The second vector specified in a Display Text Command is the Text Path Vector, <path_vec>, which specifies the direction to propagate the text string on the video screen. As with the Character Up Vector, the Text Path Vector may take the values U, R, D or L. Figure 3.7 illustrates the use of the Text Path Vector. The default value of the Text Path Vector is Right.

The third and fourth command parameters, <x_coord> and <y_coord>, specify the starting screen coordinate for displaying the text string. The coordinate specifies the location of the lower-left hand corner of the first character position on the screen as shown by the circles in Figure 3.7.

The Draw Line command is used to draw lines on the video screen in much the same way as the Polyline command in GKS. Since this command draws solid lines only, all other line styles must be simulated by the VDI Driver.

The Draw Markers Command is used to draw markers on the video screen in much the same way as the Polymarker command in GKS. The <mark_type> parameter specifies the type of marker to be displayed - either Dot, Plus, Star or Circle.

The actual value of <mark_type> is a marker number 1 - 4 as shown in the table below.

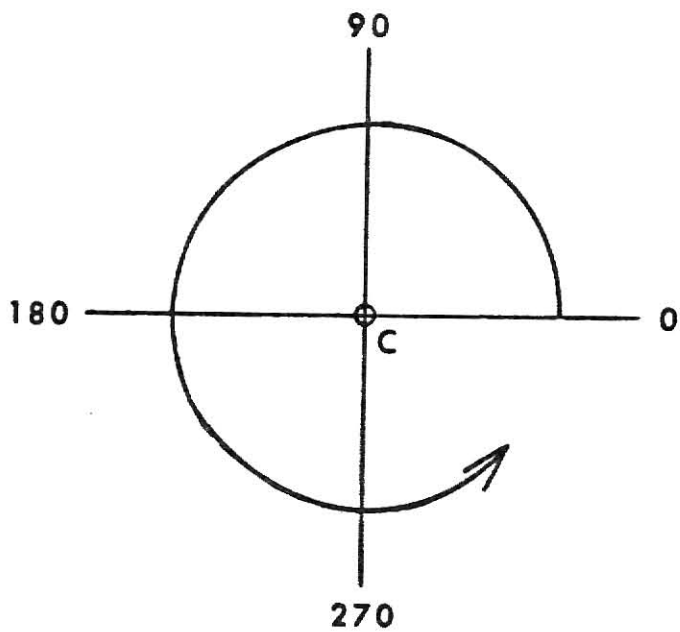
Number	Marker	
1	Dot	(.)
2	Plus	(+)
3	Star	(*)
4	Circle	(o)

The <mark_size> parameter specifies the size of the marker in Y screen units. A marker is drawn, centered around the X, Y coordinate, for each coordinate pair in the parameter list.

The Fill Area Command is used to fill a bounded polygon already displayed on the video screen with the Current Drawing Color. The color of the polygon boundary must be specified in the <bound_color> parameter. The <x_coord>, <y_coord> pair specifies an X, Y coordinate of a point in the interior region of the polygon to be used as a seed point for the fill algorithm.

The Draw Arc Command is used for drawing arcs or circles on the video screen. The arc is centered at the screen coordinates specified by the <x_coord>, <y_coord> parameters and has a radius determined by the <radius> parameter. The length of the arc is defined by the Start and End Angles in the <start_deg> and <end_deg> parameters.

Figure 3.8 Angle Definition for Drawing Arcs



The angles are measured such that zero degrees is right of the arc center and the angle increases counter-clockwise about the center. Figure 3.8 illustrates how the angles are determined for drawing arcs and circles. If the <connect_flag> parameter is set to one (1), the arc is connected to its center.

The Draw Rectangle Command is used to draw filled or unfilled rectangles on the video screen. If the <fill_flag> is set to one (1), the rectangle is filled with the Current Drawing Color. The four coordinate parameters, <ll_x_coord>, <ll_y_coord>, <ur_x_coord> and <ur_y_coord>, specify the lower-left and upper-right hand screen coordinates which define the size and location of the rectangle on the screen.

The Input String Command is used to request a string of text from MPC Driver. When the MPC Driver receives this command it makes the cursor visible at the bottom of the video screen, prompting the user to enter a text string at the keyboard. The MPC Driver then reads in the text string, makes the cursor invisible and sends the text string to the Device Driver in the response parameter <text>.

The Input Cursor Command is used to read the cursor with wait. When this command is received, the MPC Driver displays the cursor in the center of the screen and allows

the user to position the cursor using the cursor control keys. Once any of the Function Keys is hit, the cursor position is read and sent to the VDI Driver in the <x_coord> and <y_coord> fields of the response. The MPC Driver then blanks the cursor and sets the Current Key to the value of the Function Key pressed, and sets the Current Cursor Position to the value read when the key was pressed.

The Sample Enable Command is used to direct the MPC driver to display the cursor and allow the user to move the cursor about the screen with the cursor control keys. The cursor is initially placed in the center of the screen. A Sample Enable Command must be executed before a Sample Cursor Command in order to inform the user that he may move the cursor. The Sample Disable Command directs the MPC Driver to blank the cursor and terminate movement of the cursor via the cursor control keys.

The Sample Cursor Command is used to sample or poll the position of the cursor on the screen. The Cursor Position is returned to the VDI Driver immediately upon receipt of the command by the MPC Driver. The Cursor Position is returned to the Device Driver in the <x_coord> and <y_coord> fields of the response.

The Sample Keys Command requests the MPC Driver to

return the value of the Current Key. The value of the Current Key is the number of the Function Key last struck. If a Function Key has not been struck since the last Sample Keys Command or since the MPC Driver was initialized, the MPC Driver returns the value of the Bell Character (ASCII decimal 7 code).

TESTING

A particularly annoying problem encountered during project development was the failure to establish a working communications interface between the MPC and the 32/20 computer. The project design called for a serial (RS-232-C) interface between the two computers. While a terminal emulator program for the MPC communicated properly with the 32/20, the MPC Driver Program could not be made to communicate using the appropriate BASIC commands. Assistance was solicited from administrators of the 32/20 system. Still, the interface remained inoperable. Either the proper combination of interface parameters was not found or BASIC has bugs in its communications software.

Despite the problems with the serial interface, a method for testing the MPC Driver Program was devised. The program was modified to read its input from a disk file. The disc file contained MPC Interface Commands (See Figure 3.3) which exercised the various graphics command functions. The MPC Driver Program was tested by comparing the graphics display generated with results expected from the input file used, the graphics display shown in Figure 3.5 was generated in this manner using the MPC Interface Command File listed in Appendix D. Due to time limitations, the following MPC

Interface Commands were not implemented in the MPC Driver Program: Input String; Input Cursor; Sample Enable; Sample Disable; Sample Cursor; and Sample Keys.

The VDI Driver Routine was tested using a test program which invoked the driver (via subroutine calls), passing the appropriate parameters, to exercise each of the VDI commands. MPC Interface Commands output from the routine were directed to a text file for verification. Several functions of the driver were not implemented because of the limited capabilities of the MPC. Each of the non-implemented commands returns a bad status indication when invoked. The Set Character Height VDI command was not implemented since the MPC does not support multiple character sizes and this feature would be difficult to provide in software. The Set Text Font, Input Valuator and Set Linewidth VDI commands were not implemented for similar reasons. The Escape and Generalized Drawing Primitive commands were not implemented due to a lack of time. However, it would be a simple task to implement the Arc, Circle and Rectangle drawing primitives as part of the GDP. The Draw Polyline command supports only solid lines, however the driver can be modified to emulate other linestyles.

Chapter 4

RECOMMENDATIONS

During the development of the VDI driver for this master's project, several other students were also working on projects involving the GKS system. Numerous problems resulted due to the constant state of change in the system. Changes that one student would make often affected the work of others. Especially frustrating were changes made to global PASCAL type definitions that affected another student's software. This was particularly aggravating since all PASCAL software modules had to be compiled at the same time. Mid-way through the term, a method was devised to enable separate compilations of subroutines. This helped improve compilation times but could have resulted in conflicting type, variable and parameter definitions at run time. The problems encountered with multiple users making changes to the system may have been reduced somewhat if the GKS system were implemented in a language, other than PASCAL, which offered separate compilation facilities. However, use of such a language would negate the advantages of PASCAL's strong type checking and possibly defer type and parameter mismatch error detection to run time instead of

compilation time.

The GKS system at Kansas State is a single user system, the GKS routines must be compiled and loaded together with the application program. This implies that all graphics hardware supported by the GKS system is dedicated to one user. In a small system this may not pose a problem, but in a larger installation much would be gained by making GKS a multi-user GKS system. Concurrent PASCAL [11] might be used to implement a multi-user system. In such a system, devices could be allocated to users dynamically, at run time, rather than being dedicated to one user.

The definition of the Virtual Device Interface [18,19,20] specifies that graphics coordinates are passed to the device driver in Device Coordinates. This seems to conflict with the objective of providing device-independence. It would be more reasonable to allow the device driver itself to handle the mapping between Normalized Device Coordinates and Device Coordinates, as has been done in this project. This approach migrates the device-dependent code to a lower level of the software hierarchy. Thus the Device Supervisor need not know about the physical coordinate system used by any particular device. This also simplifies the software design of the

Device Supervisor.

During the development of this project it became evident that a deficiency exists in the specification of the GKS and VDI Fill Area command. The Fill Area command accepts a coordinate list which defines a polygon and the software is supposed to draw the polygon and fill the interior region with a color or pattern. However, there is no specification of how the software is to determine where the interior region is on the graphics viewsurface. Many filling algorithms require that an interior point be specified as a parameter of the fill algorithm. A Parity Check Algorithm [15] might be used to attempt to determine the interior region. However, this approach will not always work if the polygon is irregular or convex. Therefore an ambiguity exists in the Fill Area command in both the GKS and the VDI standard proposals.

This master's report has described the implementation of a Virtual Device Interface device driver for the Columbia Data Products MPC. The driver will be used to interface the MPC to the Graphical Kernal System installation at Kansas State University. Though the implementation of certain capabilities of the device driver is incomplete, the author hopes that the work begun under this project will be continued by other students who work on

the GKS project. The completion of the GKS implementation is certainly a worthy project for future work.

REFERENCES

[1] Bono, Peter R., and others, "GKS-The First Graphics Standard," IEEE Computer Graphics and Applications",2, No. 5, July, 1982, 9-23.

[2] _____, "The GKS Impact on Graphics Standardization," Computer Graphics World, 5, Sept., 1982, 47.

[3] Cahn, Deborah U., and others, "A response to the 1977 GSPC Core Graphics System," Computer Graphics, 13, No. 2, Aug., 1979, 57-62.

[4] Draft International Standard ISO/DIS 7942 Information Processing Graphical Kernel System (GKS) Functional Description, International Standards Organization, ISO Document ISO TC97/SC5/WG2 N163, Nov. 14, 1982.

[5] Encarnacao, J., and others, "The workstation concept of GKS and the resulting conceptual differences to the GSPC core system," Computer Graphics, 14, No. 2, July, 1980, 226-30.

[6] Fleming, Jim, and Frezza, William, "NAPLPS A New Standard for Text and Graphics," BYTE, 8, No.s 2-5, Feb.-May, 1983.

[7] Foley, J. D., and Van Dam, A., Fundamentals of Interactive Computer Graphics, Reading: Addison-Wesley, 1982.

[8] Giloi, Wolfgang K., Interactive Computer Graphics Data Structures, Algorithms, Languages, Englewood-Cliffs: Prentice-Hall, 1978.

[9] "Graphics Standards Planning Committee State of the Art Subcommittee Graphics System Comparison Document," Computer Graphics, 12, No.s 1-2, June, 1978, 129-40.

[10] Guttman, Herbert, and Weiss, Johann, "Device Independent and Decentralized Graphic Systems," Computer Graphics, 13, No. 4, Feb., 1980, 288-302.

[11] Hansen, Per Brinch, The Architecture of Concurrent Programs, Englewood Cliffs: Prentice-Hall, 1977.

[12] Hatfield, Lansing, "GKS and the Alphabet Soup of Graphics Standards," Computer Graphics, 16, No. 2, June, 1982, 161-2.

[13] Langhorst, Fred E., and Clarkson III, Thomas B., "Realizing Graphics Standards for Microcomputers," BYTE, 8, Feb., 1983, 256-68.

[14] Newman, William M., and Sproull, Robert F., Principles of Interactive Computer Graphics, 2d ed. New York: McGraw-Hill, 1979.

[15] Pavlidis, Theo, Algorithms for Graphics and Image Processing, Rockville: Computer Science Press, 1982.

[16] Prester, F., "The Graphical Kernel System, the Standard for Computer Graphics Proposed by the German Institute for Standardization," Computer Graphics, State of the Art Report, INFOTECH Publ., 1980.

[17] Reed, Jon S., "Computer Graphics," Mini-Micro Systems, 15, Dec., 1982, 210-21.

[18] Reed, Theodore n., Activities of the ANSI X3H33 Virtual Device Interface Task Group, American National Standards Institute, Aug. 27, 1982.

[19] _____, and others, Proposal for an ANSI X3 Standards Project for the Computer Graphics Virtual Device Interface, American National Standards Institute, Aug. 27, 1982.

[20] _____, and others, Virtual Device Interface and Virtual Device Metafile, Position Paper by the X3H33 Task Group of the American National Standards Institute, Dec. 28, 1981.

[21] Rosenthal, David S. H., and others, "The Detailed Semantics of Graphics Input Devices," Computer Graphics, 16, No. 3, July, 1982, 33-38.

[22] Thomas, James J., and others, "Graphical Input Interaction Technique (GIIT) Workshop Summary," Computer Graphics, 17, No. 1, Jan., 1983, 5-30.

[23] Weller, D. L., and others, "Software architecture for graphical interaction," IBM Systems Journal, 19, No. 3, 1980, 314-30.

APPENDIX A

Source Listing of MPC Driver Program

```
10  DIM COLTAB(9)
11  DIM X(999),Y(999)
12  PI2=2*3.1415927# : ASPECT = .910596
15 DATA 0,1,2,3,4,5,14,7,7,7
20 FOR I = 0 TO 9:READ COLTAB(I):NEXT  'INIT COLOR TABLE
30 COL=7               'SET CURRENT COLOR DEFAULT=WHITE
40 K$=CHR$(7)          'SET CURRENT KEY DEFAULT=BELL CHARACTER
50 CURX=159:CURY=99 'SET CURSOR DEFAULT=CENTER OF THE SCREEN
60 SAMPON=0            'CURSOR SAMPLING DEFAULT=OFF
65 KEY OFF             'TURN OFF DISPLAY OF FUNCTION KEYS
70 SCREEN 1,1          'SET MED-RESOLUTION COLOR GRAPHICS MODE
75 DEF SEG = 0         'SET UP SO COLOR-IT ROUTINE MAY BE USED
76                     'TO COPY SCREEN TO PRINTER
80 POKE &H200,&HCD:POKE &H201,&H5:POKE &H202,&HCD
77 DEF USR5 = &H200
80 CLS                 'CLEAR SCREEN
90 OPEN "KYBD:" FOR INPUT AS #2  'OPEN KEYBOARD
95 OPEN "PCTEST.DAT" FOR INPUT AS #3   'DATA FILE
100 '
110 ' INPUT COMMAND STRING INTO C$
120 '
130 '
135 IF EOF(3)>=0 THEN 138
136 CLOSE
137 STOP
138 LINE INPUT #3,C$
140 '
150 ' CHECK COMMAND AND JUMP TO PROPER CODE TO PROCESS IT
160 '
170 A$=LEFT$(C$,1)  'FIRST CHAR OF COMMAND STRING
180 B$=MID$(C$,1,2) 'FIRST TWO CHAR OF COMMAND STRING
190 '
200 IF A$="C" THEN 1000          'SET COLOR COMMAND
210 IF A$="E" THEN 2000          'ERASE SCREEN COMMAND
220 IF A$="P" THEN 3000          'PRINT SCREEN COMMAND
230 IF A$="T" THEN 4000          'DISPLAY TEXT COMMAND
240 IF A$="L" THEN 5000          'DRAW LINES COMMAND
250 IF A$="M" THEN 6000          'DRAW MARKERS COMMAND
260 IF A$="F" THEN 7000          'FILL AREA COMMAND
```

```
270 IF A$="A" THEN 8000           'DRAW ARC COMMAND
280 IF A$="R" THEN 9000           'DRAW RECTANGLE COMMAND
290 IF B$="IS" THEN 10000         'INPUT STRING COMMAND
300 IF B$="IC" THEN 11000         'INPUT CURSOR COMMAND
310 IF B$="SE" THEN 12000         'SAMPLE ENABLE COMMAND
320 IF B$="SD" THEN 13000         'SAMPLE DISABLE COMMAND
330 IF B$="SC" THEN 14000         'SAMPLE CURSOR COMMAND
340 IF B$="SK" THEN 15000         'SAMPLE KEYS COMMAND
350 '
360 ' IF HERE WE HAVE READ AN INVALID COMMAND, IGNORE IT
370 ' AND GO BACK TO REAAD ANOTHER COMMAND
380 '
390 GOTO 130
400 '
410 ' STATUS VARIABLE DEFINTION
420 '
430 ' COL    = CURRENT DRAWING COLOR
440 ' CURX   = CURSOR X
450 ' CURY   = CUSROR Y
460 ' K$     = CURRENT KEY
470 '
900 '
910 '*********************
920 ' SET COLOR COMMAND
930 '*********************
1000 COL=COLTAB(ASC(MID$(C$,2,1))-48)  'SET CURRENT COLOR
1005 COLOR 0,COL
1010 GOTO 130                     'READ ANOTHER COMMAND
1900 '
1910 '*********************
1920 'ERASE SCREEN COMMAND
1930 '*********************
1940 '
2000 CLS
2010 GOTO 130
2900 '
2910 '*********************
2920 'PRINT SCREEN COMMAND
2930 '*********************
2940 '
3000 DEF SEG = 0:MAP = USR5(0)   'INVOKE COLOR-IT ROUTINE
3001                             'TO PRINT SCREEN
3010 GOTO 130
3900 '
3910 '*********************
3920 'DISPLAY TEXT COMMAND
3930 '*********************
3940 '
4000 U$=MID$(C$,2,1) 'TEXT UP VECTOR
```

```
4010 B$=MID$(C$,3,1) 'TEXT BASE VECTOR
4020 L=LEN(C$)-3
4030 T$=MID$(C$,4,L) 'TEXT STRING
4040 INPUT #3,CURX,CURY 'TEXT X,Y STARTING POINT
4050 CURX=CURX8          'FIX STARTING POINT TO EVEN
4060 CURY=CURY8           'CHARACTER POSITION
4070 LOCATE CURY,CURX,0  'MOVE CURSOR INTO POSITION
4080 C$=MID$(T$,1,1)
4090 PRINT C$;
4100 FOR I = 2 TO L
4110 IF B$="U" THEN CURY=CURY-1
4120 IF B$="R" THEN CURX=CURX+1
4130 IF B$="D" THEN CURY=CURY+1
4140 IF B$="L" THEN CURX=CURX-1
4150 LOCATE CURY,CURX
4160 PRINT C$;
4180 NEXT
4185 CURX=CURX*8:CURY=CURY*8
4190 GOTO 130
4900 '
4910 '*********************
4920 'DRAW LINES COMMAND
4930 '*********************
4940 '
5000 INPUT #3,NO:J=NO-1
5010 FOR I = 0 TO J : INPUT #3,X(I),Y(I):NEXT
5020 FOR I = 1 TO J
5030 K=I-1
5040 LINE (X(K),Y(K))-(X(I),Y(I)),COL
5050 NEXT
5060 GOTO 130
5900 '
5910 '*********************
5920 'DRAW MARKERS COMMAND
5930 '*********************
5940 '
6000 TYPE=ASC(MID$(C$,2,1))-48  'MARKER SIZE
6010 INPUT #3,SIZE              'MARKER SIZE
6020 INPUT #3,NO                'NO MARKERS
6030 J=NO-1 : K=SIZE2
6040 FOR = 0 TO J : INPUT #3,X(I),Y(I) :NEXT
6045 CURX=X(J) : CURY=Y(J)
6050 IF NOT TYPE=1 THEN 6200
6052 '
6053 ' DRAW DOTS
6054 '
6060 FOR I = 0 TO J
6070 X1=X(I)-K : X2=X(I)+K : Y1=Y(I)-K : Y2=Y(I)+K
6080 FOR B = Y1 TO Y2
```

```
6090 FOR A = X1 TO X2
6095 PSET(A,B),COL
6100 NEXT : NEXT : NEXT
6110 GOTO 130
6120 '
6200 IF NOT TYPE=2 THEN 6400
6210 '
6220 'DRAW PLUS SIGNS
6230 '
6240 FOR I = 0 TO J
6250 X1=X(I)-K : X2=X(I)+K : Y1=Y(I)-K : Y2=Y(I)+K
6260 LINE (X(I),Y1)-(X(I),Y2),COL
6270 LINE (X1,Y(I))-(X2,Y(I)),COL
6280 NEXT
6290 GOTO 130
6300'
6400 IF NOT TYPE=3 THEN 6600
6410 '
6420 'DRAW ASTERISKS
6430 '
6440 FOR I = 0 TO J
6450 X1=X(I)-K : X2=X(I)+K : Y1=Y(I)-K : Y2 =Y(I)+K
6460 LINE (X1,Y1)-(X2,Y2),COL
6470 LINE (X1,Y2)-(X2,Y1),COL
6480 LINE (X1,Y(I))-(X2,Y(I)),COL
6490 LINE (X(I),Y1)-(X(I),Y2),COL
6500 NEXT
6510 GOTO 130
6520 '
6600 IF NOT TYPE=4 THEN 130
6602 '
6603 'DRAW CIRCLES
6604 '
6610 FOR I = 0 TO J
6620 CIRCLE (X(I),Y(I),K,COL,0,PI2,ASPECT
6630 NEXT
6640 GOTO 130
6900 '
6910 '*********************
6920 'FILL AREA COMMAND
6940 '*********************
6950 '
7000 BCOL=COLTAB(ASC(MID$(C$,2,1))-48)   'BOUNDARY COLOR
7010 INPUT #3,CURX,CURY
7020 PAINT (CURX,CURY),COL,BCOL
7030 GOTO 130
7900 '
7910 '*********************
7920 'DRAW ARC COMMAND
```

```
7930 '*********************
7940 '
8000 FFLAG=ASC(MID$(C$,2,1))-48      'CONNECT FLAG
8010 INPUT #3,CURX,CURY,RADIUS,SDEG,EDEG  'X, Y, RADIUS,
8011                                      ' S DEG, E DEG
8020 IF FFLAG=1 THEN SDEG=-SDEG
8025 IF FFLAG=1 THEN EDEG=-EDEG
8030 SDEG=SDEG*PI2/360!          'CONVERT ANGLES TO RADIANS
8040 EDEG=EDEG*PI2/360!
8070 CIRCLE (CURX,CURY),RADIUS,COL,SDEG,EDEG,ASPECT
8080 GOTO 130
8900 '
8910 '*********************
8920 'DRAW RECTANGLE COMMAND
8930 '*********************
8940 '
9000 FFLAG=ASC'MID$(C$,2,1))-48  'FILL FLAG
9010 INPUT #3,X1,Y1
9020 INPUT #3,CURX,CURY
9030 IF FFLAG=1 THEN LINE(X1,Y1)-(CURX,CURY),COL,BF
9040 IF FFLAG=0 THEN LINE(X1,Y1)-(CURX,CURY),COL,B
9050 GOTO 130
9900 '
9910 '*********************
9920 'INPUT STRING COMMAND
9930 '*********************
9940 '
10000 GOTO 130
10900 '
10910 '*********************
10920 'INPUT CURSOR COMMAND
10930 '*********************
10940 '
11000 GOTO 130
11900 '
11910 '*********************
11920 'SAMPLE ENABLE COMMAND
11930 '*********************
11940 '
12000 GOTO 130
12910 '
12920 '*********************
12930 'SAMPLE DISABLE COMMAND
12940 '*********************
12950 '
13000 GOTO 130
13910 '
13920 '*********************
13930 'SAMPLE CURSOR COMMAND
```

```
13940 '**********************
13950 '
14000 GOTO 130
14910 '
14920 '**********************
14930 'SAMPLE KEYS COMMAND
14940 '**********************
14950 '
15000 GOTO 130
```

APPENDIX B

Source Listing of VDI Driver Routine

```
(* 4 july 83 *)
(*************************************************************************)
(*************************************************************************)
(* P C   D R I V E R   P R O C E D U R E
(*************************************************************************)
(*************************************************************************)
(* Function: This procedure is the VDI Driver for the Columbia Data
(*           Products MPC. The procedure is executed from the GKS
(*           system through the Distributor.
(* Input: cmd   - VDI Command
(*        parm  - Record of Input Parameters associated with the
(*                VDI Command
(* Output: oparm - Record of Output Value(s) requested by the
(*                 VDI Command. Status is set by each command as Good or Bad
(* History: 18 JUN 83 - Initial Version
(*          28 JUN 83 - Modify to fit new VDI Type Definition
(*           4 JUL 83 - Add Status to Output
(*                      Add Fill Area
(* Author: Paul L. Stevens
(*************************************************************************)

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

const
      good = 1;
      bad  = 0;
      request = 1;
      sample  = 2;

var
   i,sumx,sumy:integer;
   ip:ipoint;
   pc_locator : ipoint;       (* Current Cursor Position *)
   pc_choice : unit;          (* Current Choice Index *)
   pc_dc : upoints;           (* Current Array of DC Coordinate Points *)
   pc_ndc : upoints;          (* Current Array of NDC Coordinate Points *)
   pc_nolines : pointsrange;    (* Number of Lines in pc_dc or pc_ndc *)
   pc_nomarks : pointsrange; (* Number of Markers in pc_dc or pc_ndc *)
   pc_nochar  : textcharrange; (* Number of Characters in String *)
```

```
   pc_string : tstring;          (* Current Text String *)
   mark_color : unit;          (* Current Marker Color *)
   line_color : unit;          (* Current Line Color *)
   fill_color : unit;          (* Current Fill Area Color *)
   text_color : unit;          (* Current Text Color *)
   line_type  : index;         (* Current Line Style *)
   mark_type  : index;         (* Current Marker Type *)
   mark_size  : integer;       (* Current Marker Size *)
   mark_size_dc : integer;     (* Current Marker Size in Screen Y Units *)
   char_upvec : ipoint;        (* Current Character Up Vector *)
   char_updir : textenum;      (* Current Character Up Direction *)
   text_path  : ipoint;          (* Current Text Path *)
   text_pathdir: textenum;     (* Current Text Path Direction *)
   input_mode : array [index] of index;       (* Current Input Mode *)
   input_dev  : index;         (* Current Input Device *)
   no_points  : pointsrange;   (* Temporary Number of Points *)

begin

case cmd of    (* Execute the Appropriate VDI Command *)

vdiclear: begin      (* VDI Clear Workstation Command *)

          (* Erase the Video Screen to Black *)

          pcerase;   (* Erase Video Screen to Black *)
          oparm.status := good
          end;

vdiupdate: begin   (* VDI Update Workstation Command *)

           (* Force Flushing of PC Command Output Buffer *)

           oparm.status := good;
           writechar( pcldev, em )
           end;

vdiline: begin   (* VDI Polyline Command *)

         (* Draw Lines in Current Line Color with Current Line Style *)

         (**** Note: Linestyles other than solid need to be simulated *)
         (*          by this routine, this is not currently done.     *)

         pc_nolines := parm.numpts;
         for i := 1 to pc_nolines do
           begin
           pc_ndc[i].ix := parm.pts[i].ix;
```

```
            pc_ndc[i].iy := parm.pts[i].iy
            end;
          pcndctodc(pc_nolines,pc_ndc,pc_dc);  (* NDC to DC Scaling *)
          pcsetcolor(line_color);
          pcline(pc_nolines,pc_dc);
          oparm.status := good
          end;

vdimark: begin   (* VDI Polymarker Command *)

          (* Draw Markers of Current Marker Type with Current Marker Color *)
          (*    and Current Marker Size                                    *)

          pc_nomarks := parm.numpts;
          for i := 1 to pc_nomarks do
            begin
            pc_ndc[i].ix := parm.pts[i].ix;
            pc_ndc[i].iy := parm.pts[i].iy
            end;
          pcndctodc(pc_nomarks,pc_ndc,pc_dc);  (* NDC to DC Scaling *)
          pcsetcolor(mark_color);
          pcmark(mark_type,mark_size_dc,pc_nomarks,pc_dc);
          oparm.status := good
          end;

vditext: begin   (* VDI Text Command *)

          (* Draw Text String in Current Text Color with Current Character *)
          (*    Up Direction and Current Text Path Direction               *)

          pcsetcolor(text_color);
          pc_ndc[1].ix := parm.textpos.ix;
          pc_ndc[1].iy := parm.textpos.iy;
          pcndctodc(1,pc_ndc,pc_dc);          (* NDC to DC Scaling *)
          pctext(char_updir,text_pathdir,pc_dc[1],parm.numchar,parm.string);
          oparm.status := good
          end;

vdicharh: begin   (* VDI Set Character Height Command *)
                  (****     Not Implemented        ****)
           oparm.status := bad
           end;

vdicharupv: begin   (* VDI Set Character Up Vector Command *)

             (* Set Current Character Up Direction to Left, Right, *)
             (* Up or Down based on the Character Up Vector Input  *)
             (* Parameter. Since the MPC can support only these    *)
             (* four directions, the direction closest to the Up   *)
```

```
            (* Vector is used.                                          *)

            char_upvec.ix := parm.upvec.ix;
            char_upvec.iy := parm.upvec.iy;
            if (char_upvec.ix > 0)
              then if (char_upvec.iy > 0)
                     then if (char_upvec.ix > char_upvec.iy)
                            then char_updir := right
                            else char_updir := up
                     else if (char_upvec.ix > -char_upvec.iy)
                            then char_updir := right
                            else char_updir := down
              else if (char_upvec.ix = 0)
                     then if (char_upvec.iy < 0)
                            then char_updir := down
                            else char_updir := up
                     else if (char_upvec.iy > 0)
                            then if (-char_upvec.ix > char_upvec.iy)
                                   then char_updir := left
                                   else char_updir := up
                            else if (-char_upvec.ix > -char_upvec.iy)
                                   then char_updir := left
                                   else char_updir := down;
            oparm.status := good
            end;

vdilinetype: begin   (* VDI Set Polyline Type Command *)

             (* Set Current Line Type to Input Parameter Value *)

             line_type := parm.kind;
             oparm.status := good
             end;

vdilinewidth: begin   (* VDI Set Polyline Width Command *)
                      (****   Not Implemented        ****)
              oparm.status := bad
              end;

vdilinecolor: begin   (* VDI Set Polyline Color Command *)

              (* Set Current Polyline Color to Value of Input Parameter *)

              line_color := parm.color;
              oparm.status := good
              end;

vdimarktype: begin   (* VDI Set Polymarker Type Command *)
```

```
            (* Set Current Marker Type to Value of Input Parameter *)

            mark_type := parm.kind;
            oparm.status := good
            end;

vdimarksize: begin   (* VDI Set Polymarker Size Command *)

             (* Set Current Marker Size to Value of Input Parameter    *)
             (* in NDC. Scale to Y Screen value.                       *)
             (* The Default Marker Size is 8 Pixels High & Wide.       *)

             mark_size := parm.size;
             mark_size_dc := mark_size * 200 div 32001;
             oparm.status := good
             end;

vdimarkcolor: begin   (* VDI Set Polymarker Color Command *)

             (* Set the Current Marker Color to the Value of the *)
             (* Input Parameter.                                 *)

             mark_color := parm.color;
             oparm.status := good
             end;

vditextfont: begin   (* VDI Set Text Font Command *)
                     (****   Not Implemented   ****)
            oparm.status := bad
            end;

vditextcolor: begin   (* VDI Set Text Color Command *)

             (* Set the Current Text Color to the Valu of the *)
             (* Input Parameter.                              *)

             text_color := parm.color;
             oparm.status := good
             end;

vditextpath: begin   (* VDI Set Text Path Command *)

            (* Set the Current Text Path according to the value *)
            (* of the input parameter.                          *)

            text_path.ix := parm.path.ix;
            text_path.iy := parm.path.iy;

            if abs(text_path.ix) > abs(text_path.iy)
```

```
                then if text_path.ix > text_path.iy
                       then text_pathdir := right
                       else text_pathdir := left
                else if text_path.ix > text_path.iy
                       then text_pathdir := down
                       else text_pathdir := up;
              oparm.status := good
              end;

vdigdp: begin   (* VDI Generalized Drawing Primitive Command *)

                (**** Not Implemented Now, But            ****)
                (* Should include: Arc, Circle, Rectangle    *)
                (* when implemented.                         *)
        oparm.status := bad
        end;

vdiinlocator: begin   (* VDI Input Locator Command *)

              (* Input the Current Position of the Cursor from the MPC. *)
              (* If in Request Input Mode, the MPC waits until a Func.  *)
              (* Key is hit before returning the cursor position. If in *)
              (* Sample Input Mode, the current cursor position is      *)
              (* strobed and returned immediately.                      *)

              if input_mode[1] = request
                then begin
                     pcinpcursor(oparm.locpos);
                     oparm.status := good
                     end
                else if input_mode[1] = sample
                       then begin
                            pcsampcursor(oparm.locpos);
                            oparm.status := good
                            end
                       else oparm.status := bad
               end;

vdiinchoice: begin   (* VDI Input Choice Command *)

              (* Input Choice from MPC            *)
              (*   returns Function Key No. last  *)
              (*   hit on keyboard.               *)

              pcsampkeys(oparm.choice);
              oparm.status := good
              end;

vdiinstring: begin   (* VDI Input String Command *)
```

```
            (* Input Text String from MPC        *)
            (* Executed with wait only           *)

            pcinpstring(oparm.len,oparm.strng);
            oparm.status := good
            end;

vdiopen: begin   (* VDI Open Workstation Command *)

        (* Initialize Status Variables & Clear Video Screen *)

         pc_locator.ix := 109;    (* Screen Center X *)
         pc_locator.iy :=  99;    (* Screen Center Y *)
         pc_choice     :=   1;    (* Choice 1 *)
         pc_dc[1].ix   := 109;    (* Screen Center X *)
         pc_dc[1].iy   :=  99;    (* Screen Center Y *)
         pc_ndc[1].ix  := 16000;    (* NDC Center X *)
         pc_ndc[1].iy  := 16000;    (* NDC Center Y *)
         pc_nolines    := 1;    (* 1 Coordinate *)
         pc_nomarks    := 1;    (* 1 Marker *)
         pc_nochar     := 0;    (* Min. Text String Length *)
         mark_color    := 7;    (* White *)
         line_color    := 7;    (* White *)
         fill_color    := 7;    (* White *)
         text_color    := 7;    (* White *)
         line_type     := 1;    (* Solid Line Style *)
         mark_type     := 1;    (* Dot *)
         mark_size     := 8;    (* Marker Scale *)
         mark_size_dc  := mark_size * 8;
         char_upvec.ix  := 1;    (* Char. Up Vector (Up) *)
         char_upvec.iy  := 0;
         text_path.ix   := 0;    (* Text Path Vector (Right) *)
         text_path.iy   := 1;
         text_pathdir   := right;
         input_mode[1]    := request;    (* Locator - Request Input Mode *)
         input_mode[2]    := request;    (* Valuator - Request Input Mode *)
         input_mode[3]    := request;    (* Choice - Request Input Mode *)
         input_mode[4]    := request;    (* String - Request Input Mode *)
         pcerase;    (* Clear Screen to Black *)
         oparm.status := good
         end;

vdiclose: begin   (* VDI Close Workstation Command *)

          (* Update the Video Screen *)

          writechar( pcldev, em );  (* Force flush of output buffer *)
          oparm.status := good
```

```
          end;

vdiescape: begin   (* VDI Escape Command *)
                   (**** Not Implemented ****)
           oparm.status := bad
           end;

vdiarea: begin   (* VDI Fill Area Command *)
         pc_nolines := parm.numpts;
         for i := 1 to pc_nolines do
            begin
            pc_ndc[i].ix := parm.pts[i].ix;
            pc_ndc[i].iy := parm.pts[i].iy
            end;
         pcndctodc(pc_nolines,pc_ndc,pc_dc);
         pcsetcolor(line_color);
         pcline(pc_nolines,pc_dc);
         pcsetcolor(fill_color);
         (* Compute Fill Seed Pixel Point as Average of All Points *)
         sumx := 0;
         sumy := 0;
         for i := 1 to pc_nolines do
            begin
            sumx := sumx + pc_dc[i].ix;
            sumy := sumy + pc_dc[i].iy
            end;
         ip.ix := sumx div pc_nolines;
         ip.iy := sumy div pc_nolines;
         pcfill(line_color,ip);
         oparm.status := good
         end;

vdiinmode: begin   (* VDI Set Input Mode Command *)

           (* Set the Current Input Mode for a Logical Input Device *)

           input_mode[parm.indev] := parm.mode;
           oparm.status := good
           end;

vdifillcolor: begin (* VDI Set Fill Color Command *)

              (* Set the Current Fill Area Color to Input Value *)

              fill_color := parm.color;
              oparm.status := good
              end;

vdiinvaluator: begin (* VDI Input Valuator Command *)
```

```
                (**** Not Implemented          ****)
                oparm.status := bad
                end;
end;  (* case of cmd *)

end;  (* pcdevdriver *)
```

APPENDIX C

Source Listing of VDI Driver Subroutines

```
(* 4 july 83 *)
(*************************************************************************)
(* C O N V E R T   S T R I N G   C O O R D I N A T E S   T O
(*
(* I N T E G E R   F O R M A T   P R O C E D U R E
(*************************************************************************)
(* Function: This procedure converts an X, Y coordinate pair in a text
(*           string to an integer coordinate pair.
(*           The text string should have the format:
(*           <X coord> , <Y coord> <carriage return>
(* Input: text - text string
(*        num  - number of characters in the string
(* Output: oxy - the integer coordinate pair
(* History: 21 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
;procedure pcconvcoord(num:pointsrange;text:tstring;var oxy:ipoint);
var i, power : integer;
begin
  i := num-1;
  power := 1;
  oxy.iy := 0;
  while (ord(text[i]) > 47 ) and ( ord(text[i]) < 58) do
  begin
    oxy.iy:=oxy.iy+((ord(text[i])-48)*power);
    power := power * 10;
    i := pred(i)
  end;

  power := 1;
  oxy.ix := 0;
  i := pred(i);
  while ( i>0 ) do
  begin
    oxy.ix:=oxy.ix+((ord(text[i])-48)*power);
    power := power * 10;
    i := pred(i)
  end
end;  (* pcconvcoord *)
```

```
(*****************************************************************)
(* P C   W R I T E   C O O R D I N A T E   R O U T I N E       *)
(*****************************************************************)
(* Function: This routine outputs a coordinate value to the
(*           MPC.
(* Input: Coordinate value in range 0-999
(* Output: 3 ASCII characters representing the input value
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*****************************************************************)

procedure pcwritecoord ( coordinate : integer );
var digit, coord : integer;
begin
  coord := coordinate;
  digit := 0;

  if coord > 99
    then begin
         digit := coord div 100;
         coord := coord - ( digit * 100 );
         writechar ( pcldev, chr( digit+48 ) )
         end;

  if coord > 9
    then begin
         digit := coord div 10;
         coord := coord - ( digit * 10 );
         writechar ( pcldev, chr ( digit+48) )
         end
    else if digit > 0
           then writechar( pcldev, '0' );

  writechar( pcldev, chr( coord+48 ) )
end;  (* pcwritecoord *)


(*****************************************************************************)
(* P C   W R I T E   C O O R D I N A T E   &   R E T U R N   R O U T I N E
(*****************************************************************************)
(* Function: This procedure writes an integer no. plus a carriage return
(*           to the MPC.
(* Input: Integer in range 1-999
(* Output: 1-3 ASCII characters representing the no. plus
(*         a carriage return.
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
```

```
(*************************************************************************)
procedure pcwritecoordret ( nopoints : integer );
begin
  pcwritecoord ( nopoints );
  writechar( pcldev, cr )
end;  (* pcwritecoordret *)


(*************************************************************************
(* P C   W R I T E   C O O R D I N A T E   P A I R   R O U T I N E
(*************************************************************************
(* Function: This procedure writes a coordinate X, Y pair to the
(*           MPC.
(* Input: xcoord, ycoord are X, Y coordinates in range 0-999
(* Output: Two coordinates as 1-3 ASCII characters, separated by a
(*         comma and a carraiage return as shown below.
(*
(*         <xcoord> , <ycoord > <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)

procedure pcwritexy ( xcoord, ycoord : integer );
begin
  pcwritecoord ( xcoord );
  writechar( pcldev, ',' );
  pcwritecoordret ( ycoord )
end;  (* pcwritexy *)


(*************************************************************************)
(* O U T P U T   P C   S E T   C O L O R   C O M M A N D   P R O C
(*************************************************************************)
(* Function: This procedure outputs a Set Color Command to the MPC.
(* Input: Integer color number
(* Output: Co MDnd in format: C <colorno> <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcsetcolor( colorno : index );
begin
  writechar( pcldev, 'C' );
  writechar( pcldev, chr(colorno+48) );
  writechar( pcldev, cr )
end;  (* pcsetcolor *)


(*************************************************************************)
(* O U T P U T   P C   E R A S E   S C R E E N   C O M M A N D
```

```
(*************************************************************************)
(* Function: This procedure outputs the PC Erase Screen Command to the MPC.
(* Input: None
(* Output: Command in format: E <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcerase;
begin
  writechar( pcldev, 'E' );
  writechar( pcldev, cr )
end;  (* pcerase *)


(*************************************************************************)
(* O U T P U T   P C   P R I N T   S C R E E N   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs a Print Screen Command to the MPC.
(* Input: None
(* Output: Command in format: P <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcprintscreen;
begin
  writechar( pcldev, 'P' );
  writechar( pcldev, cr )
end;  (* pcprintscreen *)


(*************************************************************************)
(* O U T P U T   P C   S A M P L E   E N A B L E   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the Sample Enable Command to the MPC.
(* Input: None
(* Output: Command in format: SE <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcsampenable;
begin
  writechar( pcldev, 'S' );
  writechar( pcldev, 'E' );
  writechar( pcldev, cr );
  writechar( pcldev, em )     (* force output buffer flush *)
end;  (* pcsampenable *)


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

```
(* O U T P U T   P C   S A M P L E   D I S A B L E   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the PC Sample Disable Command to
(*           the MPC.
(* Input: None
(* Output: Command in format: SD <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcsampdisable;
begin
  writechar( pcldev, 'S' );
  writechar( pcldev, 'D' );
  writechar( pcldev, cr );
  writechar( pcldev, em )     (* force output buffer flush *)
end;  (* pcsampdisable *)


(*************************************************************************)
(* O U T P U T   P C   D R A W   L I N E   C O M M A N D
(*************************************************************************)
(* Function: This routine outputs the PC Draw Line Command to the MPC.
(* Input: nocoord - no. of coordinates in line
(*        xycoord - X, Y line coordinates
(* Output: Command in format:
(*         L <carriage return>
(*         <no. coordinates> <carriage return>
(*         <X coord.> , <Y coord.> <carriage return>
(*         ..... repeated for all coordinates
(* History: 19 JUN 83 - Intial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcline(no:pointsrange;xy:upoints);
var i : integer;
begin
  writechar( pcldev, 'L' );
  writechar( pcldev, cr );
  pcwritecoordret( no );
  for i := 1 to no do
     pcwritexy ( xy[i].ix, xy[i].iy )
end;  (* pcline *)


(*************************************************************************)
(* O U T P U T   P C   D R A W   M A R K E R   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the PC Draw Markers Command to
(*           the MPC.
(* Input: nocoord - No. of Markers
```

```
(*        marktype - type of marker (1-4)
(*        marksize - size of marker in Y screen units
(*        xy       - X, Y Coordinates for Markers
(* Output: Command in format:
(*          M <marker type> <carriage return>
(*          <marker size> <carriage return>
(*          <no. coord.s> <carriage return>
(*          <X coor.> , <Y coord.> <carriage return>
(*          .... repeated for all coordinates
(* History: 19 JUN 83 - Inital Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcmark(marktype:index;marksize:integer;
                 nocoord:pointsrange;xy:upoints);
var i : integer;
begin
  writechar( pcldev, 'M' );
  writechar( pcldev, chr(marktype+48) );
  writechar( pcldev, cr );
  pcwritecoordret ( marksize );
  pcwritecoordret( nocoord );
  for i := 1 to nocoord do
     pcwritexy( xy[i].ix, xy[i].iy )
end;  (* pcmark *)


(*************************************************************************)
(* O U T P U T   P C   F I L L   A R E A   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the PC Fill Area Command to the MPC.
(* Input: boundcolor - boundary color
(*        ixy        - fill seed point coordinate
(* Output: Command in format:
(*          F <boundary color> <carriage return>
(*          <X coord.> , <Y coord.> <carriage return
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcfill(boundcolor:unit;ixy:ipoint);
begin
  writechar( pcldev, 'F' );
  writechar( pcldev, chr(boundcolor+48) );
  writechar( pcldev, cr );
  pcwritexy( ixy.ix, ixy.iy )
end;  (* pcfill *)


(*************************************************************************)
(* O U T P U T   P C   D I S P L A Y   T E X T   C O M M A N D
```

```
(*************************************************************************)
(* Function: This procedure outputs the PC Display Text Command to the MPC.
(* Input: upvec   - Character Up Vector
(*        basevec - Text Base Vector
(*        text    - Text String to display
(*        ixy     - X, Y Coordinate for first character
(* Output: Command in format:
(*         T <upvec> <basevec> <textstring> <carriage return>
(*         <X coord.> , <Y coord.> <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pctext(upvec,basevec:textenum;ixy:ipoint;
                 nochar:textcharrange;text:tstring);
var i : integer;
begin
  writechar( pcldev, 'T' );

  if upvec=down then writechar( pcldev, 'D' )
    else if upvec=left then writechar( pcldev, 'L' )
            else if upvec=right then writechar( pcldev, 'R' )
                    else writechar( pcldev, 'U' );

  if basevec=down then writechar( pcldev, 'D' )
    else if basevec=left then writechar( pcldev, 'L' )
           else if basevec=up then writechar( pcldev, 'U' )
                   else writechar( pcldev, 'R' );

  for i := 1 to nochar do
     writechar( pcldev, text[i] );

  writechar( pcldev, cr );
  pcwritexy( ixy.ix, ixy.iy )
end;  (* pctext *)


(*************************************************************************)
(* O U T P U T   P C   D R A W   A R C   C O M M A N D
(*************************************************************************)
(* Function: This routine outputs the PC Draw Arc Command to the MPC.
(* Input: connect - if true, connects arc to center with lines
(*        ixy     - X, Y coordinate for center of arc
(*        radius  - Radius of the arc
(*        sdeg    - Starting degrees of arc
(*        edeg    - Ending degrees of arc
(* Output: Command in format:
(*         A <connect> <carriage return>
(*         <X coord.> , <Y coord.> , <radius> ,
(*         <start deg> , <end deg> <carriage return>
```

```
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcarc(connect:boolean;ixy:ipoint;radius,sdeg,edeg:integer);
begin
  writechar( pcldev, 'A');
  if connect then writechar( pcldev, '1' )
             else writechar( pcldev, '0' );
  writechar( pcldev, cr );

  pcwritecoord( ixy.ix );
  writechar( pcldev, ',' );
  pcwritecoord( ixy.iy );
  writechar( pcldev, ',' );
  pcwritecoord( radius );
  writechar( pcldev, ',' );
  pcwritecoord( sdeg );
  writechar( pcldev, ',' );
  pcwritecoord( edeg );
  writechar( pcldev, cr )
end;  (* pcarc *)


(*************************************************************************)
(* O U T P U T   P C   D R A W   R E C T A N G L E   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the PC Draw Rectangle Command
(*           to the MPC.
(* Input: fill - If True, the rectangle will be filled
(*        llxy - X, Y coordinate of lower-left corner
(*        urxy - X, Y coordinate of upper-right corner
(* Output: Command in format:
(*          R <fill> <carriage return>
(*          <ll X coord> , <ll Y coord> ,
(*          <ur X coord> , <ur Y coord> <carriage return>
(* History: 19 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcrect(fill:boolean;llxy,urxy:ipoint);
begin
  writechar( pcldev, 'R' );
  if fill then writechar( pcldev, '1' )
          else writechar( pcldev, '0' );
  writechar( pcldev, cr );

  pcwritecoord( llxy.ix );
  writechar( pcldev, ',' );
  pcwritecoord( llxy.iy );
  writechar( pcldev, ',' );
```

```
  pcwritecoord( urxy.ix );
  writechar( pcldev, ',' );
  pcwritecoord( urxy.iy );
  writechar( pcldev, cr )
end;  (* pcrect *)


(*************************************************************************)
(* P C   N D C - T O - D C   C O O R D I N A T E   S C A L I N G
(*                      P R O C E D U R E
(*************************************************************************)
(* Function: This routine converts an array of coorinate points specified
(*           in Normalized Device Coordinates and scales them to be
(*           specified in the Device Coordinate System for the MPC.
(*           The input coordinate point values are in the range
(*           of 0-32,000. The DC coordinates are in the range of
(*           0-219 for X and 0-199 for Y. The DC ranges are different
(*           due to the aspect ratio of the video screen.
(* Input: noccord - Number of Coordinates to Scale
(*        icoord  - Array of NDC coordinates
(* Output: ocoord - Array of DC coordinates.
(* History: 21 JUN 83 - Inital Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcndctodc(nocoord:pointsrange;icoord:upoints; var ocoord:upoints);
var i : integer;
begin
  for i := 1 to nocoord do
    begin
      ocoord[i].ix := icoord[i].ix * 220 div 32001;
      ocoord[i].iy := icoord[i].iy * 200 div 32001
    end
end;  (* pcndctodc *)


(*************************************************************************)
(* P C   D C - T O - N D C   C O O R D I N A T E   S C A L I N G
(*
(*                      P R O C E D U R E
(*************************************************************************)
(* Function: This routine converts an array of coordinate points specified
(*           in PC Device Coordinates to be in Normalized Device Coordinates.
(*           The input coordinate point values are in the range of 0-219
(*           for X and 0-199 for Y. The output coordinates are
(*           in the range 0-32000.
(* Input: nocoord - the no. of coordinates to convert to NDC
(*        icoord  - the array of Device Coordinates
(* Output: ocoord - the array of Normalized Device Coordinates
(* History: 22 JUN 83 - Inital Version
```

```
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcdctondc(nocoord:pointsrange;icoord:upoints; var ocoord:upoints);
var i : integer;
begin
  for i := 1 to nocoord do
    begin
      ocoord[i].ix := icoord[i].ix div 220 * 32001;
      ocoord[i].iy := icoord[i].iy div 200 * 32001
    end
end;  (* pcdctondc *)


(*************************************************************************)
(* P C   I N P U T   S T R I N G   C O M M A N D   P R O C E U R E
(*************************************************************************)
(* Function: This procedure outputs the PC Input String Command to the
(*           MPC and reads the string from the MPC.
(* Input: Text String from the MPC.
(* Output: Command in the format:
(*         IS <carriage return>
(* History: 21 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcinpstring(var nochar:textcharrange; var text:tstring);
begin
  writechar( pcldev, 'I' );
  writechar( pcldev, 'S' );
  writechar( pcldev, cr );
  writechar( pcldev, em );     (* force output buffer flush *)
  nochar := 0;
  repeat
    begin
      nochar := succ( nochar );
      readchar( pcldev, text[nochar] )
    end
  until ((text[nochar] = cr) or (nochar = maxstring))
end;  (* pcinpstring *)


(*************************************************************************)
(* P C   I N P U T   C U R S O R   C O M M A N D   P R O C E D U R E
(*************************************************************************)
(* Function: This procedure outputs the Input Cursor Command to the MPC,
(*           reads in the response and coverts it to integer X, Y
(*           coordinate values.
(* Input: ASCII response from the MPC
(* Output: Input Cursor Command to the MPC in the format:
(*         IC <carriage return>
```

```
(*         ixy - the X, Y coordinates in integer format
(* History: 21 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcinpcursor(var ixy:ipoint);
var text : tstring; i : integer;
begin
  writechar( pcldev, 'I' );
  writechar( pcldev, 'C' );
  writechar( pcldev, cr );
  writechar( pcldev, em );    (* force output buffer flush *)

  i := 0;
  repeat
    begin
      i := succ( i );
      readchar( pcldev, text[i] )
    end
  until ((text[i] = cr) or (i = maxstring));

  pcconvcoord(i,text,ixy)  (* convert string to integers *)
end;  (* pcinpcursor *)


(*************************************************************************)
(* P C   O U T P U T   S A M P L E   C U R S O R   C O M M A N D
(*************************************************************************)
(* Function: This procedure outputs the Sample Cursor Command to the
(*           MPC, reads the response string and converts the X, Y
(*           coordinates in the string to integers.
(* Input: Character string from MPC
(* Output: Sample Cursor Command in format:
(*         SC <carriage return>
(*         ixy - integer X, Y coordinate
(* History: 21 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcsampcursor(var ixy:ipoint);
var text:tstring; i:integer;
begin
  writechar( pcldev, 'S' );
  writechar( pcldev, 'C' );
  writechar( pcldev, cr );
  writechar( pcldev, em );    (* force output buffer flush *)

  i := 0;
  repeat
    begin
      i := succ( i );
```

```
      readchar( pcldev, text[i] );
    end
  until ((text[i] = cr) or (i = maxstring));

  pcconvcoord( i, text, ixy )   (* convert string to integers *)
end;  (* pcsampcursor *)


(*************************************************************************)
(* P C   O U T P U T   S A M P L E   K E Y S   C O M M A N D
(*************************************************************************)
(* Function: This routine outputs a Sample Keys Command to the MPC,
(*           and reads the response character.
(* Input: Key character plus carriage return from MPC
(* Output: Command in format:
(*         SK <carriage return>
(*         Function Key No. last hit in Key parameter
(* History: 21 JUN 83 - Initial Version
(* Author: Paul L. Stevens
(*************************************************************************)
procedure pcsampkeys(var key:integer);
var dummy:char;
begin
  writechar( pcldev, 'S' );
  writechar( pcldev, 'K' );
  writechar( pcldev, cr );
  writechar( pcldev, em );    (* force output buffer flush *)

  readchar( pcldev, dummy );
  key := ord(dummy)-48;
  if key < 0 then key := 0;
  readchar( pcldev, dummy )   (* for return *)
end;  (* pcsampkeys *)
```

APPENDIX D

Sample Listing of MPC Interface Commands

```
;
; PCGKS TEST INPUT FILE
;
; 4 JULY 83 - PAUL L. STEVENS
;
;
;DRAW 2 BOXES ON THE SCREEN
;
R0
0,0
219,199
R0
5,5
214,194
;
;DRAW CIRCLE AND 2 ARCS
;
A0
50,50,30,0,360
A1
50,50,20,1,90
A1
50,50,15,180,270
;
;DRAW MARKERS
;
;DOTS
;
M1
1
5
20,190
40,190
60,190
80,190
100,190
M1
8
5
```

```
20,170
40,170
60,170
80,170
100,170
;
;PLUSES
;
M2
8
5
20,150
40,150
60,150
80,150
100,150
M2
16
5
20,130
40,130
60,130
80,130
100,130
;
;ASTERISKS
;
M3
8
5
110,190
130,190
150,190
170,190
190,190
M3
10
5
110,170
130,170
150,170
170,170
190,170
;
;CIRCLES
;
M4
8
5
```

```
110,150
130,150
150,150
170,150
190,150
M4
10
5
110,130
130,130
150,130
170,130
190,130
;
;TEXT
;
TURTEXT
150,50
TULTEXT
150,50
TUUTEXT
150,50
TUDTEXT
150,50
;
;FILL AREA
;
F3
2,2
;
;DRAW LINES
;
L
10
20,100
40,120
60,100
80,120
100,100
120,120
140,100
160,120
180,100
200,120
;
;PRINT SCREEN
;
P
```

A VDI INTERFACE FOR A MICROPROCESSOR GRAPHICS SYSTEM

by

PAUL L. STEVENS

B. S., V. P. I. & S. U., 1978

AN ABSTRACT OF A MASTER'S REPORT

submitted in parital fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

The Virtual Device Interface (VDI) is a computer graphics interface standard proposal under development by the ANSI Virtual Device Interface Task Group (X3H33). The task group defines the Virtual Device Interface as "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 drivers." This report documents the design and implementation of a project whose purpose was to provide a VDI level interface between the Graphical Kernel System (GKS) and a microprocessor-based color graphics system. As both VDI and GKS are still evolving standards proposals, various omissions and inconsistencies exist in the specifications. This report also investigates issues related to these proposed standards and documents the design decisions and assumptions made during the course of the project.