

REVIEW OF COMPUTER GRAPHICS
STANDARDIZATION EFFORTS WITH EMPHASIS ON
GKS, VDI, and VDM

by

DEBRA MAE HERRING

B.S., Ohio State University, 1979

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE


Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1983

Approved by:


Major Professor

LD
2668
.R4
1983
H47
C.2

A11202 617812

CONTENTS

1. Introduction.....	1
1.1 Overview.....	1
1.2 Evaluation of Proposed Standards.....	2
2. Brief History of Computer Graphics Standards.....	3
3. Current Standard Proposals.....	10
3.1 The Graphical Kernel System (GKS).....	10
3.2 The Virtual Device Interface (VDI).....	20
3.3 The Virtual Device Metafile (VDM).....	23
4. Implementation.....	25
References.....	31
Appendixes	
1 - Implementation - VDI Driver Code.....	34
2 - Application Program Examples.....	51
3 - Application Program Output..(Pictures).....	70

**THIS BOOK
CONTAINS SEVERAL
DOCUMENTS THAT
ARE OF POOR
QUALITY DUE TO
BEING A
PHOTOCOPY OF A
PHOTO.**

**THIS IS AS RECEIVED
FROM CUSTOMER.**

LIST OF FIGURES

Figure 1.	GKS Main Features.....	11
Figure 2.	GKS L0a Functions.....	12
Figure 2.	GKS L0a Functions continued.....	13
Figure 3.	GKS Coordinate System Transformations.....	15
Figure 4.	GKS / GKSM Interface.....	20
Figure 5.	VDI Required Functions.....	21
Figure 6.	VDI Context Model.....	22
Figure 7.	VDM Required Functions.....	24
Figure 8.	Implementation Diagram.....	25
Figure 9.	VDI Implemented Functions.....	26
Figure 10.	Metafile Format.....	29
Figure 10.	Metafile Format continued.....	30

1. Introduction

1.1 Overview

This paper examines the recent proposals for computer graphics standards, the differences between them, and the reasons behind them. These proposals are the application-level Graphical Kernel System (GKS), the device-level Virtual Device Interface (VDI) and Virtual Device Metafile (VDM). GKS was developed in Germany by a subcommittee of the German Institute for Standardisation (DIN) and is the first computer graphics standard proposed by the International Standards Organization. The American National Standards Institute (ANSI) has made GKS available for public review to determine if it will be adopted as the U.S. standard also. The VDI and VDM are being developed by ANSI committees to standardize the interface to graphics device drivers and allow portability of files containing picture descriptions.

There are four sections in this paper. The Introduction continues with an evaluation of the standards. The second section covers a brief history of computer graphics standards. The third section goes into more detail concerning GKS, VDI and VDM. The fourth and final section is a discussion of an implementation relating to the VDI standard. A small graphics device driver was developed based on a subset of the VDI functions. The implementation, written in C language,¹ provides subroutine calls to the device driver which will draw the requested picture or text on the graphics device screen.

1. C is a high level programming language that was developed by Bell Telephone Labs.

1.2 Evaluation of Proposed Standards

This section assumes that the reader is already familiar with the general concepts of GKS, VDI and VDM. It summarizes my opinions of the stated proposals based upon my study of them.

GKS is far too complicated for most simple, single device applications. There are too many unneeded functions required at the lowest level, (LOa). It seems likely that this impending standard will just be considered a guideline because the documentation is not explicit enough and GKS is too complex for most implementations.

The VDI and VDM concepts are hard to differentiate. They have not been widely publicized since they are not as far along the standardization process as is GKS. The Virtual Device Interface needs to be defined more explicitly to be implemented correctly. In the future, more of the graphics software at the VDI level will be incorporated into the hardware, thereby providing more time for concentration on higher level graphics application packages at the device independent level. The VDM actually allows portability of the graphical information whereas the application level GKS package merely enhances graphics programmer and application program portability.

It is surprising how long it has taken to realize the need for standards. It seems that standards would have been developed in conjunction with the growth in the graphics hardware technology. Also, much time has been spent in developing and gaining acceptance for the standards.

Many industries do follow the standards efforts to implement them as soon as possible and the standards usage is then advertised as part

of the product quality [4]. Precision Visuals, one of the leading graphics software vendors, plans to update their current software to the GKS and VDI standards as soon as possible. A microcomputer VDI package, the Graphics System Extension GSX-86, has been developed jointly by Digital Research and Graphics Software Systems, Inc.

2. Brief History of Computer Graphics Standards

Interest in computer graphics standards started in the mid-70's due mainly to the growth of so many varied graphics applications. Some of these applications were forgotten when the special hardware they were based on became outdated or useless. The graphics hardware technology advanced rapidly, but software had to be rewritten for the new devices. The advances in graphics technology date from dedicated host systems with high powered refresh displays to the more common use today of low cost raster devices and high powered single user systems [16]. Eventually, more people began to see the importance of the concept of device independence to enhance portability of both graphics programs and the programmers between various host processors and graphics devices thereby reducing software costs and graphics programmer training costs [2,4]. Device independence refers to hiding the hardware characteristics from the application users by allowing the graphics package to work on various types of graphics devices with only minimal changes being made to the device driver interface.

There are different standards levels to be considered. The two discussed here are the programmer level and the code generator level. The programmer level interface is the incorporation of graphics functions into an application program and has taken on the form of subroutine packages, though other ideas have been studied including (1)

programming language extensions to include new syntactic constructs for graphics functions and (2) a new language just for graphics. The latter two haven't had as much support and aren't as practical as the subroutine package [15,23]. The code generator level is sometimes referred to as the DI/DD interface standing for the "device-independent to device-dependent interface" [2]. It is essentially a high level device driver interface.

The following information concerning standards history was basically derived from the BONO, HATFIELD, LANGHORST, PUK, and SHREHLO references [2,15,20,26,29]. The beginning of formal U.S. standards efforts can be traced to a workshop on Device Independent Computer Graphics, sponsored by the ACM Special Interest Group on Graphics, SIGGRAPH, held in 1974 at the National Bureau of Standards in Gaithersburg, Maryland. The chairman of SIGGRAPH at the time, Robert M. Dunn, appointed a Graphics Standards Planning Committee (GSPC) to study existing standards or proposals to determine if a standard covering certain areas could be clearly defined [15]. Also, in August, 1974 at an IFIP² WG5.2³ meeting in Sweden, Richard Guedj was asked to set up a subcommittee to explore standards for computer graphics. The committee thought it was too early to look at standards and first needed to look at computer graphics concepts so they organized a "Workshop on Graphics Standards Methodology" [20]. The workshop was held in Seillac, France in May, 1976 and marked the first graphics standards workshop which is now known as Seillac I. About 30 graphics authorities from Europe and

2. IFIP stands for International Federation for Information Processing.

3. Working Group 5.2's title is Computer Applications in Technology - Computer Aided Design.

the U.S. tried to resolve some differences between the various graphics users. The two main results of this meeting were the following.

1. The decision was made to separate the viewing from the modeling concepts. These concepts involved the differences between the operations to look at a picture and those operations used to define a picture [26].
2. Enough issues were clarified to pave the way for the design of a standard graphics system.

One of the topics that spurred the viewing versus modeling concept, concerned the "current position" concept in output drawing. Some arguments against the current position said that there was confusion as to the definition of the current position after a coordinate system is transformed or after a hardware vector generator draws a character string [29].

Seillac I motivated U.S. attendees to revitalize the Graphics Standards Planning Committee. In 1977, the GSPC published its first Status Report, GSPC77 [31], covering its survey of existing graphics packages and its device independent graphics standard proposal called the GSPC Core System, or more commonly, the Core. The final GSPC review of existing graphics packages was published in 1978 [7]. The Core was influenced mainly by the GPGS package from the Netherlands [12]. The December 1978 issue of ACM Computing Surveys [1,21,22] gives a good overview of the first Core draft. In the next few years, many objections

-
4. Current position is the reference point where the graphics cursor resides after drawing a section of the picture. The next primitive could specify relative values of how far to draw or where to start relative to the current position.

and suggestions required modification to the first report. The second GSPC Status Report, GSPC79 [32], was presented at the Seillac II workshop in 1979. This report was basically the functional specification of the revised Core proposal which included raster graphics extensions, a description of the Metafile, and a model for distributed graphics systems.

The Core, at least in the U.S., has essentially become the de facto standard for device-independent graphics applications. There have been many graphics packages developed based on the GSPC Core. Since 1979, more than twenty organizations have implemented some levels of the Core. Some leading manufacture-provided systems including some firmware versions, are the PGL from Hewlett-Packard, Template from Megatek and other systems from Ramtek Corp. and Tektronix. There have also been independent software products such as the DI-3000 system from Precision Visuals Inc. [34]. One of the most complete Core systems was the George Washington University's GWCORE [9]. A more recent university Core system implementation was the result of a senior project at the University of Pennsylvania [33]. A microcomputer Core based implementation for the Apple II is described in [10].

Later in 1979, GSPC was disbanded and its graphics standards activity was turned over to the formal standards body in the U.S., the American National Standards Institute (ANSI). ANSI established the X3H3, Technical Committee on Computer Graphics Programming Languages. This ANSI group is currently the major graphics-standardization body in the U.S.[20].

A standards effort in Germany similar to the U.S. efforts, had begun in 1977 because the government was spending too much money on

information processing systems. The Graphical Kernel System (GKS) was proposed by the German standards organization DIN under a group chaired by José Encarnação. This system also underwent many changes due to discussions at the Seillac workshops and input from the GSPC. GKS doesn't offer as much as does the GSPC Core System. GKS is only a 2-D system and doesn't include the current position concept. Another difference between GKS and the Core is the workstation concept in GKS that allows application programs to better use features of a specific output device [2].

The growing interest in graphics standards and mainly the proposal to the International Standards Organization (ISO) of the GINO-F [11] system by the British Computer Society in 1976 led to the formation of an appropriate ISO working group (ISO TC97/SC5/WG2) to study proposals for international computer graphics standards.

There were several ISO meetings and various standards submitted including the GSPC Core System, the German DIN GKS and IDIGS, a successor to GPGS, from Norway. IDIGS was late for a proposal deadline so an ISO committee studied differences between the GKS and GSPC drafts with the objective to bring the two proposals closer together. At the October 1979 ISO meeting in Budapest, it was decided that only GKS would be considered as the graphics standards proposal because DIN was ready to sponsor it and the 2-D system could be adopted faster than the more elaborate 3-D Core system. There were a few more ISO meetings ironing out details for improvements to GKS and various working drafts were produced. As a result of an October 9, 1981 ISO TC97/SC5 meeting, GKS Version 7.0 (ISO DP 7942) was accepted as a draft proposal. GKS was registered as a Draft International Standard on March 11, 1983. The document can be updated to an International Standard after it is

translated to French and again approved by the ISO member nations. ANSI is studying whether the GKS standard will also be adopted as an American National Standard.

Universities and companies in Germany, of course, were the first to implement GKS packages. A university implementation in Austria based on GKS is described in [13]. A command interpreter was built on top of GKS in Germany [3]. An early implementation based on the combination of GKS and Core functions is described in [35]. U.S. companies that had planned to use or were using Core have already started implementing GKS-based systems. With ISO and ANSI approval, most vendors will switch to promote this latest standard. Pertaining to microcomputers, Digital Research provides a GSS-Kernel package that looks like a GKS interface to the programmer that in turn accesses the Graphics System Extension (GSX). The GSX was developed jointly by Digital Research and Graphics Software Systems, Inc. and is an extension of the CP/M operating system family [20].

ANSI work is progressing on a 'superset of GKS' to provide 3-D and other support. Until that is defined, users with those requirements will most likely keep using the Core systems.

The device independent to device dependent interface also is being studied by various groups now. The VDI and VDM concepts were originally going to be part of the GKS specification, but broke off to be studied as a separate standard. Germany is developing standards at this level, but only the U.S. efforts are examined here. The Virtual Device Interface (VDI) standard is being developed by the ANSI X3H33 Technical Committee. The chairman of the task group, Ted Reed, of Los Alamos National Laboratory, foresees a draft VDI standard to be submitted for

American National Standards approval by late 1983. The VDI is defined 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" [38]. A device driver here refers to the part of the graphics software that translates commands and data from the VDI into the form required by a particular input/output device. A position paper [38], though not being maintained any longer, provides useful information on VDI and the Virtual Device Metafile (VDM). A document explicitly describing the VDI will be produced after the VDM standard is complete. The VDI is expected to be driven by application packages or other graphics software such as the GKS. A graphics device driver could directly support the VDI and the long-range possibilities include implementation of more of the software in the hardware such as the VDI interface in the graphics device itself. This would make it unnecessary for graphics utility software vendors to write software device drivers for new hardware. The VDI helps make graphics software products compatible with many different devices by isolating the unique characteristics of the physical graphics device in the device driver software. This is similar to efforts currently made by individual vendors to make their products compatible with various devices. The UNIX⁵ and CP/M operating systems are examples of systems which are designed to interface to various configurations easily [20].

The Virtual Device Metafile (VDM) is a particular metafile standard also being developed by the ANSI X3H33 Task Group and is defined as "a

5. UNIX is a trademark of Bell Telephone Labs.

mechanism for retaining and/or transporting graphics data and control information" [37]. There is a working draft proposal document on the VDM [39] available from ANSI. Annex E of the GKS document [18] is a guide for metafile usage (GKSM) until standards are accepted.

3. Current Standard Proposals

3.1 The Graphical Kernel System (GKS)

The main sources for the GKS overview were the BONO [2] and PRESTER [25] articles and the draft document [18] which is available for a fee from ANSI.

GKS is a device independent graphics system at the programmer level interface. It basically consists of a set of routines to be accessed by application programs for computer graphics programming. The GKS package itself can be implemented in any high level language making it and the application program using it portable between systems that support the particular language. Mainly only the device driver code (not part of GKS) and the application code definition of workstation attributes should have to change when using the package with a different device.

GKS uses a 'workstation concept' to refer to support of graphical output on one or more workstations concurrently. A workstation consists of a console that includes one display device and/or one or more input devices. The display surface is usually a CRT (cathode-ray-tube) screen, but could be a simple plotter. Some possible input devices include alphanumeric keyboards, function keys, joysticks, control balls or light pens [25].

The main features of GKS are listed in Figure 1. There are control

functions, output functions, functions to set input and output attributes, transformation functions, input functions, inquiry functions and error handling functions in GKS.

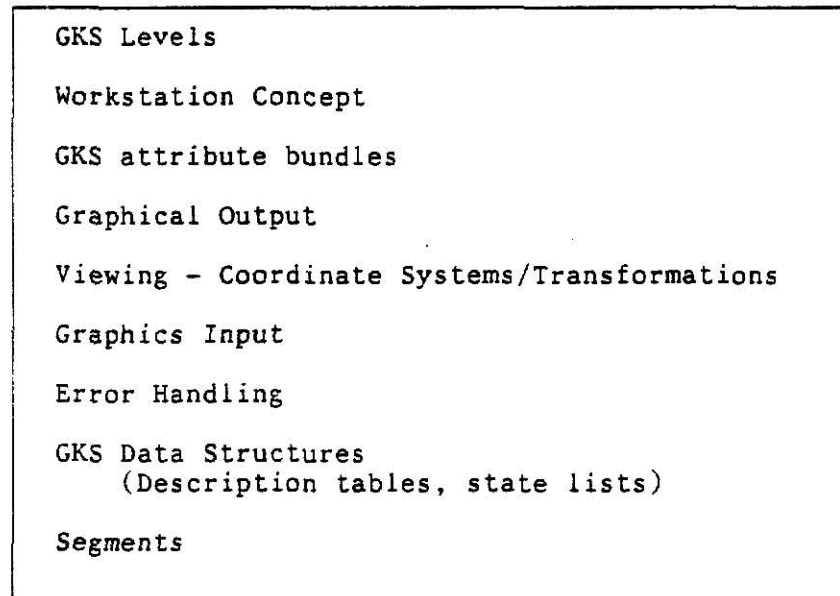


Figure 1. GKS Main Features

GKS doesn't recognize the current position concept that is used in the GSPC Core System so all input and output functions must specify all starting and ending coordinates. In other words, relative coordinates aren't used.

There are seven different levels of GKS that can be implemented depending on the input/output capabilities available and required for a particular application [18]. The lowest level (LOa) is the minimal required set of functions for a GKS implementation. This level doesn't include input functions. The Level 0a functions are listed in Figure 2.

GKS supports two-dimensional input and output primitives. These primitives refer to graphical data that can be obtained from any open

Inquiry Functions

```

INQUIRE OPERATING STATE VALUE
INQUIRE LEVEL OF GKS
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES
INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION NUMBER
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES
INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES
INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS
INQUIRE NORMALIZATION TRANSFORMATION
INQUIRE CLIPPING INDICATOR
INQUIRE WORKSTATION CONNECTION AND TYPE
INQUIRE WORKSTATION STATE
INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES
INQUIRE TEXT EXTENT
INQUIRE LIST OF COLOUR INDICES
INQUIRE COLOUR REPRESENTATION
INQUIRE WORKSTATION TRANSFORMATION
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION CLASSIFICATION
INQUIRE MAXIMUM DISPLAY SURFACE SIZE
INQUIRE POLYLINE FACILITIES
INQUIRE PREDEFINED POLYLINE REPRESENTATION
INQUIRE POLYMARKER FACILITIES
INQUIRE PREDEFINED POLYMARKER REPRESENTATION
INQUIRE TEXT FACILITIES
INQUIRE PREDEFINED TEXT REPRESENTATION
INQUIRE FILL AREA FACILITIES
INQUIRE PREDEFINED FILL AREA REPRESENTATION
INQUIRE PATTERN FACILITIES
INQUIRE PREDEFINED PATTERN REPRESENTATION
INQUIRE COLOUR FACILITIES
INQUIRE PREDEFINED COLOUR REPRESENTATION
INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES
INQUIRE GENERALIZED DRAWING PRIMITIVE
INQUIRE PIXEL ARRAY DIMENSIONS
INQUIRE PIXEL ARRAY
INQUIRE PIXEL

```

Figure 2. GKS L0a Functions continued

workstation (input) and graphical information that is generated by GKS and routed to all active workstations (output). The output primitives are:

polyline

polymarker
 text
 fill area
 cell array
 generalized drawing primitive (GDP)

The polyline routine is a line drawing routine which draws a set of connected lines described by the set of points connecting them. The polymarker routine generates a specific symbol, such as a plus sign, centered on given points. It is used mainly to identify points on plotted curves [20]. The text function generates a given string of characters on the graphics device. The fill area function is used on raster graphics devices to fill a closed area with a solid color or pattern of colors given a point within that area. The last two items are normally only supported at the higher levels of GKS. The cell array is also a raster graphics function and is a complex fill that fills areas based on a 2-dimensional array addressing individual pixel⁶ colors. This is useful for imaging applications such as cartography [20]. The GDP is used for general pictures not available on all devices like drawing circular arcs, bars or spline curves. The GDP is a mechanism to escape from GKS to allow access to specific capabilities of a particular device.

The applications program defines all graphical primitives with its own world coordinate system which is based on the two dimensional Cartesian coordinate system. The world coordinates are mapped into device coordinates through normalization and workstation

6. A pixel represents the smallest addressable point (picture element) on a raster device display surface.

transformations. Figure 3 illustrates the concepts of the GKS coordinate system transformations.⁷

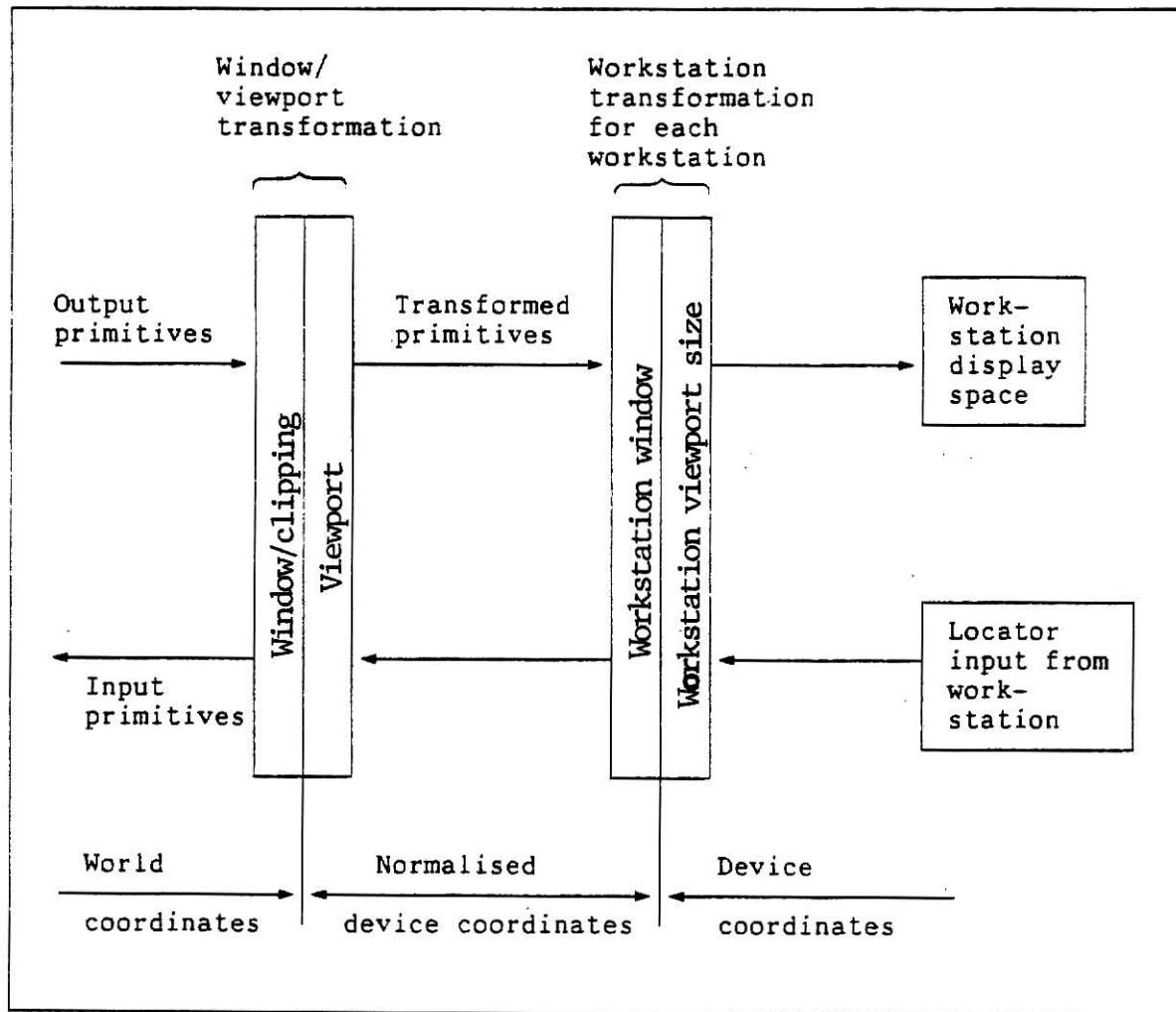


Figure 3. GKS Coordinate System Transformations

The GKS functions define output primitives in normalized device coordinates (NDC) in the range $(0,1) \times (0,1)$. The normalized device coordinate space is essentially an abstract viewing surface independent

7. This figure was obtained from the Prester article [25].

of the various workstation viewing surfaces. A 'window' is specified by defining the limits of a rectangular area parallel to the coordinate axes in world coordinates and the 'viewport' is represented by the two corresponding points in NDC. A normalization transformation is applied to each output primitive to map the window into the NDC viewport by using translation and scaling formulas. A clipping rectangle is defined based on the particular NDC viewport. If the clipping function is enabled, the parts of the picture that lie outside of this rectangle will be clipped (or deleted) so that they won't show up on the display surface.

Each workstation has its own recognized coordinates so each workstation must be associated with a part of the NDC space to be displayed on the workstation display surface. Output primitives in NDC are transformed to workstation (device) coordinates by the corresponding workstation transformation. The limits of the device coordinates are available in a workstation dependent table and the workstation transformation is specified any time after the workstation is opened via the SET WORKSTATION WINDOW and SET WORKSTATION VIEWPORT functions. GKS always clips at the workstation boundaries so that the routines do not try to draw pictures outside the display surface.

Graphical input in GKS, as well as in the GSPC Core System, is based on the concept of virtual (or logical) input devices. This concept promotes graphics application portability because the interactive graphics application need only recognize values returned by virtual input devices and therefore not be concerned with the device hardware. GKS graphical input is accessed by an application program by controlling logical input devices which get the input from the operator of the device and return logical input values to the program. The

actual input is from hardware such as a light pen, digitizer or mouse. The input class refers to the type of logical input values. The six classes of input primitives and their input values provided are:

Locator - provides a position in world coordinates.

Stroke - provides a sequence of points in world coordinates.

Valuator - provides a real number.

Choice - provides an integer selection from a number of choices.

Pick - is only applicable to segments and provides a segment name and a pick identifier and status.

String - provides a character string read from a keyboard device.

Input from the devices is obtained based on three 'operating modes' which are set by the various SET MODE functions. These modes are REQUEST, SAMPLE and EVENT. For one of the input classes in the REQUEST mode, logical input values are read by GKS one at a time, each time, waiting for operator input or break action. The SAMPLE function for a given input class doesn't wait for operator action, but merely returns the current logical input value for the device. The sampled devices are represented by the Locator and Valuator classes and the values returned represent the point selected possibly by a cursor (Locator), or the current value of a continuous valuator device such as a potentiometer (Valuator class). The EVENT mode causes GKS to build an input queue which holds input primitives from various input devices in this mode ordered by the time they are generated. The application can obtain the latest input data from the queue. The Choice and Pick classes are represented by the EVENT mode.

The input and output functions have attributes associated with them that specify the characteristics of the picture or text to be drawn. The attribute primitives affect the object's appearance. For example, the text function has associated attributes that allow for specification of the direction the text should read, the width of the characters, the spacing between the characters, the size of the characters, the text font to be used and the color of the characters. The attribute information is specified via an attribute function call which sets program variables within the GKS routines for specific devices to be used by GKS. GKS provides attribute bundling which was not provided in the GSPC Core System. The bundling refers to choosing all attributes for a primitive by selecting an index into a table containing entries linking all possible combinations of the various attributes for a particular workstation. This eliminates the requirement of setting the attributes individually via separate functions although attributes can also be set individually.

A segment facility allows subdivision of pictures into subparts and short term storage of these subparts. These subparts are represented by a collection of output primitives and are identified by a segment name. Segments can be created and deleted, transformed, made visible or invisible, highlighted, renamed and inserted into other segments.

The Inquiry functions return values from various 'state lists' which represent general information about the current state of the system, such as the level of GKS being used, the operating state of GKS, the geometrical transformation parameters, the list of available workstations, or specific workstation information, such as the current attributes for the polymarker output primitive for a given workstation. All of this information is obtained from variables in various structures

in GKS that keep track of the current state of the system. This information is accessed as needed by the specific routines. For example, when the 'polyline' routine is called, the current attribute values of the polyline set from the 'set polyline index' routine affect the color, line type and line width of the polyline displayed.

There are specified error situations for each GKS function that invoke the ERROR HANDLING procedure. All GKS implementations should provide this error handling. The ERROR HANDLING procedure can be provided by the application program in which case GKS functions would instead call the user-supplied routine to handle errors.

GKS also provides an interface to external sequential files called GKS Metafiles (GKSMs). These are used for long-term storage and exchange of graphical information. A GKSM is treated as another workstation. Functions are provided to write to the GKSM and to read and interpret the information from it. Reading and interpreting the information from the GKSM causes the other GKS functions to be called to generate the picture on an opened output workstation. The format and content of a metafile aren't part of the GKS standard, but a suitable format is covered in Annex E of the GKS draft document [18]. Figure 4 shows the simplified GKS to GKSM interface.⁸ Metafiles will be discussed more thoroughly later.

8. This figure was obtained from the Prester article [25].

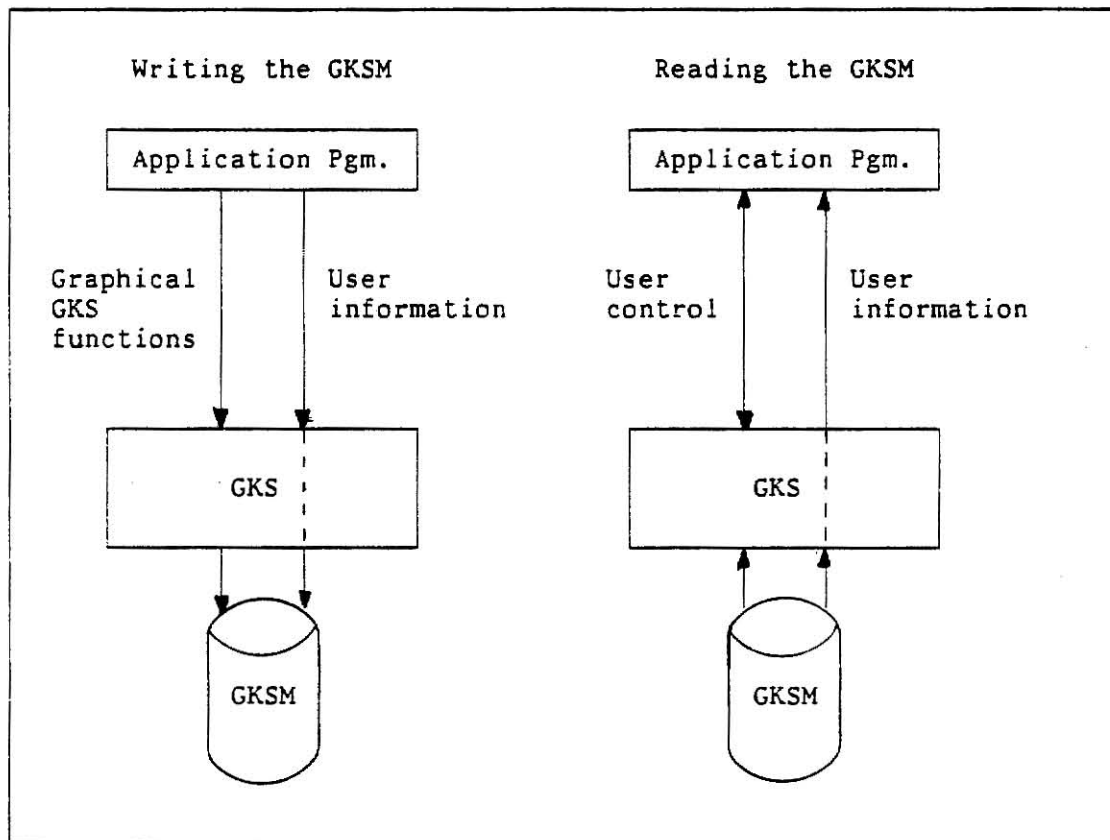


Figure 4. GKS / GKSM Interface

3.2 The Virtual Device Interface (VDI)

In general, a virtual device interface is a definition of operations (commands and parameters) that interface to the device dependent software. These commands represent a device-driver protocol that provides most capabilities available on various graphics devices [20]. These commands provide essentially the same graphics capabilities as the GKS functions discussed. The difference is that the VDI represents a step closer to the hardware in the software architecture.

The VDI standard, being developed by an ANSI task group, will include: an inquiry and response mechanism for graphics device capabilities, characteristics and states; a graphics escape mechanism to

access non-standard graphics device capabilities; support of multiple concurrent graphics device drivers; and a rich as well as a lower level mode of operation. The lean set will be required to be supported by all device drivers conforming to VDI. It will support most functions that are commonly used by most graphics devices today, but will not require input support. The VDI will support devices that have the current position capability. Figure 5 shows the lean set of VDI functions. The commands passed through VDI will be sequential and will affect the state of the graphics display as they are issued.

<u>Function</u>	<u>Parameters</u>
<u>Control Functions</u>	
INQUIRE	input: nonrequired function name output: (NO, EMULATED or YES)
ESCAPE	n, data stream, device id
INITIALIZE VDI	
TERMINATE VDI	
RETURN VDI STATUS	(status vector returned)
CLEAR VIRTUAL DEVICE	
SET MAPPING MODE	mapping mode indicator
SET LDC RANGE	xmin, ymin, xmax, ymax
SET MAPPING REF. POINTS	xmin, ymin, xmax, ymax
<u>Output Functions</u>	
MOVE	x, y
POLYLINE	n, x, y values
POLYMARKER	n, x, y values
RETURN CURRENT POSITION	(x, y coordinates returned)
<u>Text:</u>	
TEXT	n, n character codes
SET TEXT HEIGHT	height of graphic symbol
SET TEXT ALIGNMENT	horizontal, vertical

Figure 5. VDI Required Functions

The standard can support a metafile (data format for a picture description) to facilitate picture transfer and will be functionally compatible with the Virtual Device Metafile standard. A metafile following the VDM standard can be read by a VDI driver and the resulting picture will be drawn on the open workstation(s).

One of the reference models presented in the position paper on VDI and VDM is shown in Figure 6. This is called the context model [38] and it represents a conceptual view of the VDI and the VDM and their relationship to a larger graphics system.

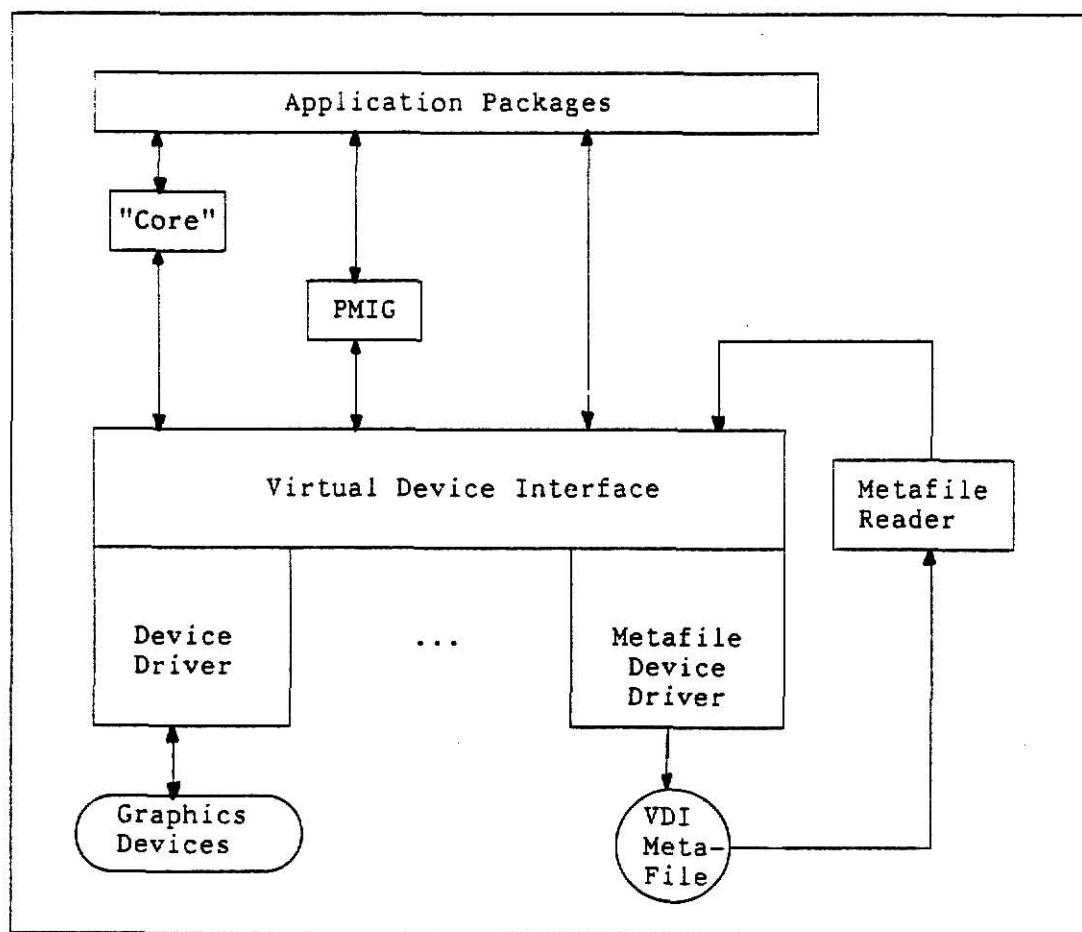


Figure 6. VDI Context Model

As of November 1982, fifteen companies have announced technical and personnel support for the VDI standardization process and will provide products based on the standard [29]. The leaders in this effort are Digital Equipment Corp., Intel Corp. and Tektronix. A microcomputer system based on the emerging VDI standard is the Graphics Input/Output System (GIOS) portion of the GSX CP/M upgrade discussed under the GKS section [20].

3.3 The Virtual Device Metafile (VDM)

A graphics metafile is generally a file containing a device-independent representation of a picture that can be accessed and displayed on various graphics output devices [27].

The VDM, being developed by an ANSI task group, is intended for use as a long term storage and retrieval mechanism. Looking at the Context Model (Figure 6) again, it shows that input from the VDM occurs at the VDI level.

The VDM also will support two levels of functionality where the minimal set includes the VDI required set of functions (output and control) and a header containing 1) identification information 2) the VDM level and version number 3) the format and precision of the VDM (formats for real and integer included) and 4) bounds on the function set used by the particular metafile. An Extended Metafile (EVDM) could also include support of nonrequired VDI functions such as fill area and segmentation. The header provides information on the contents and capability of the VDM so the application programs or VDIs will know whether they can interpret the VDM. The VDM functions will be two-dimensional. Functions that could have variable amounts of data such as points on a connected set of lines, have parameters indicating the

amount of data preceding the data itself. The device driver then knows how much data to read in and associate with the given function. Figure 7 shows required functions for the VDM.

<u>Function</u>	<u>Parameters</u>
VDM Header	n, values
Control Functions:	
BEGIN VDM	
END VDM	
CLEAR VIRTUAL DEVICE	
SET MAPPING MODE	
SET LDC RANGE	xmin,ymin,xmax,ymax
SET MAPPING REF. POINTS	xmin,ymin,xmax,ymax
ESCAPE	n, data stream device id
Output Primitives:	
MOVE	x,y
POLYLINE	n, x,y values
POLYMARKER	n, x,y values
Text:	
TEXT	n, n character codes
SET TEXT HEIGHT	height of graphic symbol
SET TEXT ALIGNMENT	horizontal, vertical

Figure 7. VDM Required Functions

The EVDM includes more primitives, attributes and segmentation and is discussed further in [39].

The VDM coding format hasn't yet been approved, but the structure should be compatible with the proposed higher level GKS standard. Also, some functions of the proposed North American Presentation Level Protocol Syntax (NAPLPS) encoding scheme (which provides a super set of ASCII) have also been included. Details on how the different coding standards are encoded into the VDM coding format are discussed in the VDM draft proposal [39]. The actual implementation and support of a

metafile could likely be done using linked lists or simple ASCII files. Los Alamos National Laboratories has been using a graphics metafile since 1977 and had developed theirs after studying a few other metafiles that were in use at that time. The VDM is based on a study of four existing metafiles, one being the Los Alamos BGP metafile [27].

4. Implementation

The implementation for this report was designed to make a particular graphics device accessible by implementing an application program level interface to it. By basing this interface on one of the proposed standards, two goals were achieved. These goals were to further my understanding of the principles of the latest standards and to see how the standards are related to the portability issue.

Figure 8 represents a diagram of the implementation.

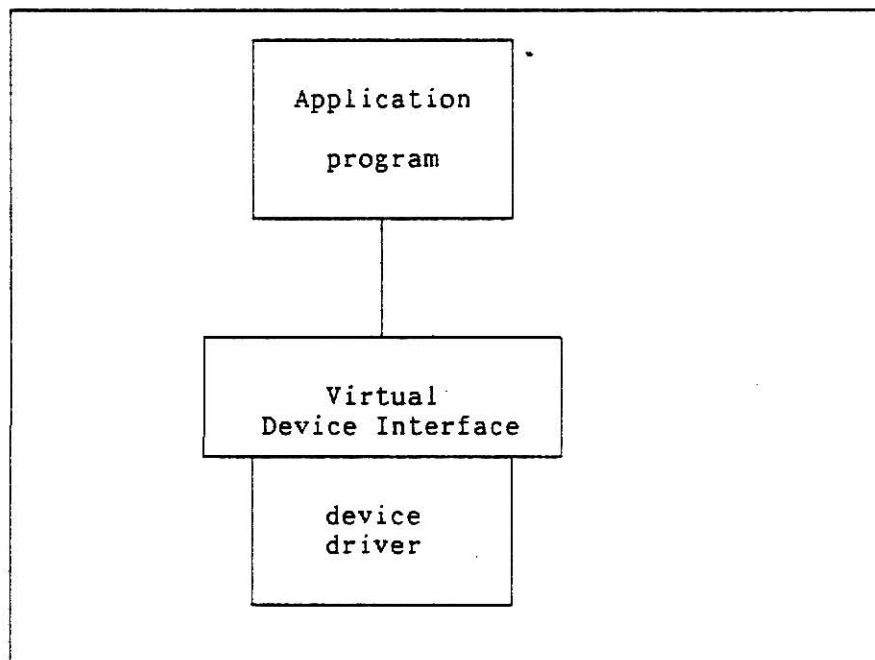


Figure 8. Implementation Diagram

The VDI standard was used as a guideline for the application level graphics system interface. The Virtual Device Interface was implemented as standard calls to the VDI device driver. This implementation does not include all of the required minimal set of functions, but it does also include some non-required functions. The set of functions in Figure 9 was implemented. The operation codes used to reference the functions follow those used in the GSX implementation [20].

<u>Control Functions</u>	<u>Implementation OPCODE Names</u>
INITIALIZE VDI	VDIOPEN
TERMINATE VDI	VDICLOSE
CLEAR VDI	VDICLEAR
ESCAPE (Circle)	VDIGDP
<u>Output Functions</u>	
POLYLINE	VDILINE
POLYMARKER	VDIMARKER
TEXT	VDITEXT
FILL AREA	VDIFILL
<u>Primitive Attributes</u>	
SET POLYLINE COLOR	VDILNCOLOR
SET POLYMARKER COLOR	VDIMKCOLOR
SET MARKER TYPE	VDIMKTYPE
SET TEXT COLOR	VDITXCOLOR
SET TEXT FONT	VDITXFONT

Figure 9. VDI Implemented Functions

The graphics device driver code was written in the C language using the UNIX operating system. The host processor was a Digital Equipment Corporation 11/70 minicomputer and the graphics device was an Industrial Data Terminal (IDT) Model 2200, which is manufactured in Westerville, Ohio. It is a basic color graphics raster scan terminal.

An application program has access to the device driver at the VDI level by calling the driver with command codes and parameters for a particular VDI function. The code for the driver is in Appendix 1. Note, that only output functions are supported. Each of the functions sends the output to the open workstation. The IDT driver translates the data into the ASCII escape sequences supported by the IDT and sends the translated commands to the device for generation of the picture. An application program must be compiled with the device driver. Three example application programs illustrate, in Appendix 2, how the VDI driver is accessed. The appendix also includes the compilation UNIX shell files that set up the source code to be compiled. Note the sequence of calls in the programs. VDI must first be initialized and then the output commands are used to draw the picture and finally the VDI terminate function is accessed. The control functions set certain information that the VDI and workstation routines access later to control the characteristics of the output. The 'vdi.h' included header file is used to pass the required graphical information from the application program to the IDT driver. The 'idt.h' header file contains device-dependent information required to access the IDT device. The resulting screen pictures of the first two application examples are shown in Appendix 3.

For general use and simpler application program interface, another graphics level package such as the GKS would normally be built on top of the VDIs so that the application program would make calls to the GKS routines and GKS would then call the VDI routines. This implementation allows the programs to call the device driver directly via the VDI function calls providing a standard interface to the graphics capabilities at a lower level than the GKS. The VDI coordinates should

be in the range 0-32767 and are then scaled down to the device coordinates within the device driver. The coordinate system interface isn't as convenient to the application program as it could be if the application could use its own coordinate system, such as that of the specific device. In this case, another set of calculations would need to be done to transform the application coordinate system to the VDI coordinate system. This would be done, for instance, if GKS was built on top of the VDI.

In addition to the VDI implementation, metafiles were studied and a sample metafile following the formats in Figure 10 was created and sent to the graphics group at KSU for further study. The format was selected by Professor Hankley and his Spring CS898 class and was based on the ASCII/integer format specified in Annex E of the GKS draft document [18]. The GKSM is currently easier to understand and implement than the proposed VDM standard.

File:

file header	item 1	...	item N	end item
-------------	--------	-----	--------	----------

File Header:

GKSM	N	D	V	H	T	L	I	R	F	RI	ZERO	ONE
------	---	---	---	---	---	---	---	---	---	----	------	-----

where:

'GKSM' - 4 bytes
 N - 40 bytes - name, author of picture
 D - 8 bytes - YY/MM/DD
 V - 2 bytes - picture number
 H - 2 bytes - 02 - data items start with "GK"
 T - 2 bytes - 02 - item id. number
 L - 2 bytes - 03 - max data item = 999 bytes
 I - 2 bytes - 03 - I3 format for integers
 R - 2 bytes - 06 - I6 format for NDC's
 F - 2 bytes - 01 - all numbers stored in ASCII
 RI - 2 bytes - 02 - all numbers stored as integers
 ZERO - 11 bytes - 00000000000 - zero representation
 ONE - 11 bytes - 00000032000 - [0,32000] - NDC range

Ex. File Header:

GKSMBike Meeting, Debbie Herring, Cols.,OH 83/06/05
 01020203030601020000000000000000032000

Figure 10. Metafile Format

Item:

item header	item data record
-------------	------------------

Item

Header:

'GK'	identif. no.	length of item record
------	--------------	-----------------------

items possible are:

#0	-	end item
#11	-	polyline
#12	-	polymarker
#21	-	set polyline index
#22	-	line type
#23	-	line width
#24	-	polyline colour
#25	-	set polymarker index
#26	-	set marker type
#27	-	set marker size scale factor
#28	-	set polymarker colour index
#29	-	set text index
#30	-	set text font and precision
#31	-	set character expansion factor
#32	-	set character spacing
#33	-	set text colour index
#34	-	set character height and up vector
#35	-	set text path
#36	-	set text alignment
#51	-	set polyline representation (GKSM-OUT)
#52	-	set polymarker representation (GKSM-OUT)
#53	-	set text representation (GKSM-OUT)
#56	-	set colour representation (GKSM-OUT)

Item Ex.:

GK113806120300100100200100120300270300200100

Figure 10. Metafile Format continued

References

1. Bergeron, R.D., P.R. Bono and J.D. Foley. "Graphics Programming Using the Core System." *ACM Computing Surveys* 10(4), December 1978. pp.389-444.
2. Bono, Peter R., Jose L. Encarnacao, F. Robert A. Hopgood and Paul J.W. tenHagen. "GKS - The First Graphics Standard." *IEEE Computer Graphics and Applications* July 1982. pp.9-23.
3. Borufka, H.G. and G. Pfaff. "Design of a General-Purpose Interpreter for Graphical Man-Machine Communication." *Man-Machine Communication in CAD/CAM*. T. Sata, E. Warman (eds.) (North-Holland, 1981.) pp.161-175.
4. Bruns, Bob and James R. Warner. "A Discussion of Software Standards." *Computer Graphics World* August 1982. pp.60-63.
5. Buttuls, Peter. "Some Criticisms of the Graphical Kernel System (GKS)." *Computer Graphics* 15(4), December 1981. pp.301-305.
6. Caruthers, L.C., J. van den Bos, and A. van Dam. "A Device-Independent General Purpose Graphics System for Stand-alone and Satellite Graphics." *Computer Graphics* 11(2), Summer 1977. pp.112-119.
7. Ewald, R.H. and R. Fryer (eds.) "Final Report of the GSPC State-of-the-Art Subcommittee." *Computer Graphics* 12(1-2), June 1978. pp.14-169.
8. Foley, James D. "The SIGGRAPH Core System Today." *Computer Graphics World* August 1982. pp.27-30.
9. Foley, James D. and Patricia A. Wenner. "The George Washington University Core System Implementation." *Computer Graphics* 15(3), August 1981. pp.123-132.
10. Freiden, Alan. "A Two-Dimensional, Level 2 Core System for the Apple II." *Computer Graphics* 14(4), March 1981. pp.127-152.
11. *GINO-F, The General Purpose Graphics Package Reference Manual*, CAD Centre, Cambridge, England, July 1975.
12. *GPGS-F User's Guide*, RUNIT Computer Centre, University of Trondheim, Norway, September 1975.
13. Guttman, Herbert and Johann Weiss. "Device Independent and Decentralized Graphic Systems." *Computer Graphics* 13(4), February 1980. pp.288-302.
14. Hatfield, Lansing. "GKS and the Alphabet Soup of Graphics Standards (An Informal Commentary)." *Computer Graphics* 16(2), June 1982. pp.161-162.

15. Hatfield, Lansing. "Graphics Software - from Techniques to Principles." *IEEE CG&A* 2(1), January 1982. pp.59-80.
16. Hopgood, F.R.A. "The Road to Graphics Standards." *Computer-Aided Design* 14(4), July 1982. pp.221-225.
17. *IDT-2000 Users Manual Version 12.*
18. *ISO DIS 7942 (GKS) - Version 7.2 "Information Processing - Graphical Kernel System (GKS) - Functional Description."* 12/27/82.
19. Kernighan, Brian W. and Dennis M. Ritchie. *The C Programming Language.* (Prentice-Hall, Inc. 1978.)
20. Langhorst, Fred E. and Thomas B. Clarkson. "Realizing Graphics Standards for Microcomputers." *Byte* February 1983. pp.256-268.
21. Michener, J.C. and J.D. Foley. "Some Major Issues in the Design of the Core Graphics System." *ACM Computing Surveys* 10(4), December 1978. pp.445-464.
22. Michener, J. C. and A. van Dam. "A Functional Overview of the Core System with Glossary." *ACM Computing Surveys* 10(4), December 1978. pp.381-388.
23. Mudur, S.P., S.C. Gupta, C.U. Sharma and S. Ramesh. "Environmental Independence in a Graphics Programming System." *Proc. International Conf. Interactive Techniques in Computer Aided Design - Sept.21-23.* 1978. pp.241-248.
24. Newman, W.M. and A. Van Dam. "Recent Efforts Toward Graphics Standardization." *ACM Computing Surveys* 10(4), December 1978. pp.365-380.
25. Prester, F. "The Graphical Kernel System (GKS). The Standard for Computer Graphics Proposed by the German Institute for Standardisation (DIN)." *Computer Graphics, State of the Art Report INFOTECH Publ.* 1980. pp.219-248.
26. Puk, Richard. "The Background of Computer Graphics Standardization." (ANSI Study Group Working Paper) *Computer Graphics* 12(1-2), June 1978. pp.2-6.
27. Reed, Theodore N. "A Metafile for Efficient Sequential and Random Display of Graphics." *Computer Graphics* 16(3), July 1982. pp.39-43.
28. Rosenthal, Davis S.H. "Managing Graphical Resources." *Computer Graphics* 17(1), January 1983. pp. 38-45.
29. Shrehlo, Kevin B. (Assoc. Ed.). "ANSI graphics standards coalesce around international kernel." *Mini-Micro Systems.* 15(11), November 1982. pp.175-186.
30. Sonderegger, Elaine L. "Report on ANSI Activities." *Computer Graphics* 16(4), December 1982. pp.246-249.

31. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH. Part II: General Methodology and Proposed Standard." *Computer Graphics* 11(3), Fall 1977. 117 pages.
32. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH." *Computer Graphics* 13(3), August 1979.
33. Stluka, Frederick P., Brian F. Saudners, Paul M. Slayton and Norman I. Badler. "Overview of the University Of Pennsylvania Core System Standard Graphics Package Implementation." *Computer Graphics* 16(2), June 1982. pp.177-186.
34. Warner, James R. and Nikolaus J. Kiefhaber. "Implementing Standard Device-Independent Graphics." *Mini-Micro Systems* 15(7), July 1982. pp.201-208.
35. Wisskirchen, Peter, Karl-Heinz Klein, Peter Seuffert, and Gerd Woetzel. "Implementation of the Core Graphics System GKS in a Distributed Graphics Environment." *Proc. Int. Conf. Interactive Techniques in Computer Aided Design* Sept.21-23 1978. pp.249-254.
36. X3H33 80-34 SD-3 "Proposal for an ANSI X3 Standard Project for the Computer Graphics Virtual Device Interface.", November 11, 1980.
37. X3H33 80-52 SD-3 "Proposal for an ANSI X3 Standard Project for the Computer Graphics Virtual Device Metafile.", November 17, 1980.
38. X3H33 81-27R5 "Virtual Device Interface and Virtual Device Metafile X3H33 Task Group Position Paper.", December 28, 1981.
39. X3H33 82-15 "Draft Proposed American National Standard for the Virtual Device Metafile." August 13, 1982.
40. Yen, Elizabeth, "A Graphics Glossary." *Computer Graphics* 15(2), July 1981. pp.208-229.

A p p e n d i x 1

VDI Driver Code

Aug 6 23:53 1983 ddidt.c Page 1

```

/*****
/**          ddidt.c          **/
*****/
/* Virtual Device Interface (VDI) driver for */
/* IDT 2200 color raster scan device          */
*****/

#include <stdio.h>
#include <sys/ioctl.h>
#include "idt.h"

idt_dvr(opcode, vdiptr)
int opcode;
struct VDI_REC *vdiptr;
{
    switch (opcode)
    {
        case VDIOPEN:
            op_idt();
            break;
        case VDICLOSE:
            cls_idt();
            break;
        case VDICLEAR:
            clr_idt(vdiptr->gen);
            break;
        case VDILINE:
            poly1_idt(vdiptr);
            break;
        case VDIMARKER:
            polym_idt(vdiptr);
            break;
        case VDITEXT:
            text_idt(vdiptr);
            break;
        case VDIFILL:
            fill_idt(vdiptr);
            break;
        case VDIGDP:
            switch (vdiptr->gen)
            {
                case CIRCLE:
                    cir_idt(vdiptr);
                    /* Later could add support for bar graphs, polygons, */
                    /* grids, RAMPICs - segments stored in RAM.          */
                    /* and ROMPICs - segments stored in ROM,              */
                    break;
                default:
                    fprintf(stderr, "*** ERROR - unsupported GDP ***");
                    return(BAD);
            } /* end gdp type switch */
        case VDIHRHGT:
            /* could later add scaled size if use MACROFONT */
            /* or can use MICROFONT for tiny characters      */
    }
}

```

Aug 6 23:53 1983 ddidt.c Page 2

```

        tidt.txstyle.size = vdiptr->gen;
        break;
    case VDICHRUP:
        /* could later use MICROFONT and adjust direction */
        break;
    case VDILNTYPE:
        /* can't do with firmware - not emulated now */
        break;
    case VDILNCOLOR:
        lnstyle.color = vdiptr->gen;
        break;
    case VDIMKTYPE:
        mkstyle.type = vdiptr->gen;
        break;
    case VDIMKCOLOR:
        mkstyle.color = vdiptr->gen;
        break;
    case VDITXFONT:
        /* could use MACRO or MICROFONT, with 1st - can use shadows also */
        tidt.txstyle.font = 1;
        break;
    case VDITXCOLOR:
        tidt.txstyle.color = vdiptr->gen;
        break;
    default:
        fprintf(stderr, "*** ERROR - unsupported opcode ***");
        return(BAD);
} /* end of opcode switch */
return(GOOD);
} /* end of idt_dvr */

```

```

op_idt()
{
    int fd;
    /* next 3 structures for I/O control */
    struct ttiocb ttiocb;
    struct ttiothcb ttiothcb;
    struct termcb termcb;

    /* initialize attribute values */
    lnstyle.type = 0;
    lnstyle.width = 0;
    lnstyle.color = BLUE;

    mkstyle.type = DOT;
    mkstyle.size = 0;
    mkstyle.color = BLUE;

    tidt.back = BLACK;
    tidt.foreblink = OFF;
    tidt.backblink = OFF;
    tidt.txstyle.font = 0;
    tidt.txstyle.size = T_REG;
    tidt.txstyle.color = BLUE;
}

```

Aug 6 23:53 1983 ddidt.c Page 3

#ifdef OUTPUT

/* open (IDT) line for output */

idt = fopen(IDT_FILE,"w");

fd = fileno(idt);

if (idt == NULL)

{

fprintf(stderr,"\n*** ERROR on OPEN of IDT line ***\n");

return(BAD);

}

ioctl(fd,TIOCGTEP,&ttiocb);

ttiocb.ioc_ispeed = ttiocb.ioc_ospeed = B2400;

ttiocb.ioc_flags |= RAW;

ioctl(fd,TIOCTEP,&ttiocb);

ioctl(fd,TIOCGTEO,&ttiothcb);

ttiothcb.ioth_flags |= TANDEMO|TANDEMI;

ioctl(fd,TIOCTEO,&ttiothcb);

ioctl(fd,DIOCGT,&termcb);

termcb.st_term = TERM_NONE;

ioctl(fd,DIOCT,&termcb);

/*****

/* on keyboard, hit RESET &/or CNTL/RESET keys if run into trouble */

*****/

/* change escape char to ESC (~ now) & term char to TERM (| now) */

fputc(033,idt); fprintf(idt,"E%c",ESC);

fflush(idt);

fprintf(idt,"%cJ%c",ESC,TERM);

fflush(idt);

res_idt(); /* set idt communication */

clr_idt(BLACK);

#endif

return(GOOD);

} /* op_idt */

res_idt()

/* set baud rate, raw input mode, */

/* send cntl reset to delete RAM buffers, etc. */

{

#ifdef DEBUG

printf("\nIn res_idt\n");

Aug 6 23:53 1983 ddidt.c Page 4

#endif

```
/* clear RAM area used for storage of MACROGRAPHICS or RAMPICs */
fprintf(idt,"%c0",ESC);
fflush(idt);
```

```
/* turn cursor off */
fprintf(idt,"%cCC",ESC);
fflush(idt);
```

```
/* set normal character size */
fprintf(idt,"%c)",ESC);
fflush(idt);
```

```
/* turn blinks off for foreground, background and character */
fprintf(idt,"%cCBF%cCBB%cCBC",ESC,ESC,ESC);
fflush(idt);
```

```
/* correct circle aspect ratio */
fprintf(idt,"%cCM",ESC);
fflush(idt);
```

```
} /* res_idt */
```

```
cls_idt()
{
```

```
#ifdef OUTPUT
    /* close (IDT) line for output */
    res_idt();
    fclose(idt);
#endif
```

```
}
```

```
clr_idt(color)
```

```
int color;
{
#ifdef DEBUG
    printf("\nIn clr_idt\n");
#endif
```

```
    tidd.back = color; /* set character background to back. */
#ifdef OUTPUT
    fprintf(idt,"%cP%d",ESC,color);
    fflush(idt);
#endif
}
```

Aug 6 23:53 1983 ddidt.c Page 5

```

polyl_idt(vdiptr)
struct VDI_REC *vdiptr;
{
    int px,py,pxt,pyt,i;

    if (vdiptr->gen < 2)
    {
        fprintf(stderr,
            "\n** ERROR - less than 2 points specified for line **\n");
        return(BAD);
    }

    /******
    /* Convert ndc to device coordinates. */
    /******

    ndc_dc(vdiptr->gen,vdiptr->parm.line);

    px = vdiptr->parm.line[0].x;
    py = vdiptr->parm.line[0].y;
    pxt = vdiptr->parm.line[1].x;
    pyt = vdiptr->parm.line[1].y;
    vec_idt(px,py,pxt,pyt);

    for (i=2; ((i < vdiptr->gen) && (i < MAXPTS)); i++)
    {
        /* starting point of new vector from where left off */
        px=pxt; py=pyt;
        pxt = vdiptr->parm.line[i].x;
        pyt = vdiptr->parm.line[i].y;
        vec_idt(px,py,pxt,pyt);
    }
    return(GOOD);
} /* poly_idt */

vec_idt(x,y,xt,yt)
int x,y,xt,yt;
{
#ifdef DEBUG
    printf("\nIn polyl_idt:\n");
    printf("x=%d,y=%d,xt=%d,yt=%d,color=%d\n",x,y,xt,yt,lnstyle.color);
#endif

#ifdef OUTPUT
    fprintf(idt,"%cV%1d%cVA%3d%3d%3d%3d",ESC,lnstyle.color,ESC,x,y,xt,yt);

```

Aug 6 23:53 1983 ddidt.c Page 6

```

    fflush(idt);
#endif
}

```

```

polym_idt(vdiptr)

```

```

struct VDI_REC *vdiptr;

```

```

/*****
/* Use IDT MACROPLOT AND MACROGRAPHIC utilities for all but */
/* DOT and OH */
*****/

```

```

{
    int i;

```

```

    /*****
    /* Convert ndc to device coordinates. */
    *****/

```

```

    ndc_dc(vdiptr->gen,vdiptr->parm.mark.c);

```

```

#ifdef DEBUG

```

```

    printf("\nIn polym_idt: ");
    printf("\n      marker=%d, color=%d, #points=%d\n",
        vdiptr->parm.mark.type,
        mkstyle.color,
        vdiptr->gen);

```

```

#endif

```

```

    fprintf(idt,"%cV%d",ESC,mkstyle.color); /* set color */
    fflush(idt);

```

```

    switch (vdiptr->parm.mark.type)
    {

```

```

        case DOT:

```

```

        case OH:

```

```

            for (i=0; ((i < vdiptr->gen) && (i < MAXPTS)); i++)
            {

```

```

#ifdef DEBUG

```

```

                printf("      i=%d, Location: x=%d, y=%d\n",i,
                    vdiptr->parm.mark.c[i].x,
                    vdiptr->parm.mark.c[i].y);

```

```

#endif

```

```

                fprintf(idt,"%cV@%3d%3d",ESC,
                    vdiptr->parm.mark.c[i].x,
                    vdiptr->parm.mark.c[i].y);
                fflush(idt);
                if (vdiptr->parm.mark.type == OH)
                {
                    fprintf(idt,I_OH);
                    fflush(idt);
                }
            }

```


Aug 6 23:53 1983 dddt.c Page 7

```

        else /* DOT */
        {
            fprintf(idt,I_DOT,
                vdiptr->parm.mark.c[i].x,
                vdiptr->parm.mark.c[i].y);
            fflush(idt);
        }
        break;

case PLUS:
case ASTERISK:
case XXX:
    if (vdiptr->parm.mark.type == PLUS)
        fprintf(idt,I_PLUS);
    else if (vdiptr->parm.mark.type == ASTERISK)
        fprintf(idt,I_AST);
    else fprintf(idt,I_XXX);
    fflush(idt);
    fprintf(idt,"%cV$%2d",ESC,vdiptr->parm.mark.type);
    fflush(idt);
    for (i=0; ((i < vdiptr->gen) && (i < MAXPTS)); i++)
    {
#ifdef DEBUG
        printf("    i=%d,    Location: x=%d, y=%d\n",i,
            vdiptr->parm.mark.c[i].x,
            vdiptr->parm.mark.c[i].y);
#endif
        fprintf(idt,"%3d%3d",
            vdiptr->parm.mark.c[i].x,
            vdiptr->parm.mark.c[i].y);
        fflush(idt);
    }
    fprintf(idt,"%c",TERM);
    fflush(idt);
    break;

default:
    fprintf(stderr,"** ERROR - unsupported GDP **");
    return(BAD);
    } /* end switch */
return(GOOD);
} /* end polym_idt */

text_idt(vdiptr)

struct VDI_REC *vdiptr;
{
    int line,col,size,num;
    char tit[MAXSTRING];
    struct XY xy[2];

```

Aug 6 23:53 1983 ddidt.c Page 8

```

float fcol,fline;

/*****
/* Convert ndc to device coordinates. */
*****/

xy[1].x = xy[1].y = 0;
xy[0].x = vdiptr->parm.text.x;
xy[0].y = vdiptr->parm.text.y;
num = 1;

ndc_dc(num,xy);
size = tidt.txstyle.size;
strcpy(tit,vdiptr->parm.text.stg);

#ifdef DEBUG
printf("\nIn text_idt:\n");
printf("tit=%s\nline=%d, col=%d, color=%d, size=%d\n",
      tit,xy[0].y,xy[0].x,tidt.txstyle.color,size);
#endif
/*****
/* now convert for reverse character screen image */
*****/
/* line = int[(512-y)/10]   col = int[85 * x / 512] for regular text*/
/* line = int[(512-y)/21]   col = int[42 * x / 512] for large text */
*****/

if (size == T_LARGE)
{
    fcol = 42.0 * xy[0].x / 512.0;
    fline = (512.0 - xy[0].y) / 21.0;
}
else
{
    fcol = 85.0 * xy[0].x / 512.0;
    fline = (512.0 - xy[0].y) / 10.0;
}

col = fcol;
line = fline;

#ifdef DEBUG
printf("\nIn text_idt:\n");
printf("tit=%s\nline=%d, col=%d, color=%d, size=%d\n",
      tit,line,col,tidt.txstyle.color,size);
#endif

#ifdef OUTPUT
if (size == T_LARGE)
{
    fprintf(idt,"%cC%1d%1d",
        ESC,tidt.txstyle.color,tidt.back);
    fflush(idt);
    fprintf(idt,"%c(%c%2d%2d%s%c)",
        ESC,ESC,line,col,tit,ESC);
}

```

Aug 6 23:53 1983 ddidt.c Page 9

```

        fflush(idt);
    }
    else /* regular */
    {
        fprintf(idt,"%cC%1d%1d%c@%2d%2d%s",
            ESC,tidt.txstyle.color,tidt.back,ESC,line,col,tit);
        fflush(idt);
    }
#endif
}

```

```
fill_idt(vdiptr)
```

```
struct VDI_REC *vdiptr;
```

```

{
    int num;
    struct XY xy[2];
    /***/
    /* Convert ndc to device coordinates. */
    /* ndc_dc(px,py,&xd,&yd); */
    /***/

    xy[1].x = xy[1].y = 0;
    xy[0].x = vdiptr->parm.fill.x;
    xy[0].y = vdiptr->parm.fill.y;
    num = 1;
    ndc_dc(num,xy);

#ifdef DEBUG
    printf("\nIn fill_idt\n");
    printf("x=%d,y=%d\n",xy[0].x,xy[0].y);
#endif

#ifdef OUTPUT
    fprintf(idt,"%cV%3d%3d%cG",ESC,xy[0].x,xy[0].y,ESC);
    fflush(idt);
#endif
}

```

```
cir_idt(vdiptr)
```

```
struct VDI_REC *vdiptr;
```

```

{
    int x,y,rad,color;

    /***/
    /* Convert ndc to device coordinates. */

```

Aug 6 23:53 1983 ddidt.c Page 10

```

/*****/

vdiptr->parm.gdp.c[1].x = vdiptr->parm.gdp.gp; /* radius */
ndc_dc(2, vdiptr->parm.gdp.c);

    x = vdiptr->parm.gdp.c[0].x;
    y = vdiptr->parm.gdp.c[0].y;
    rad = vdiptr->parm.gdp.c[1].x;
    color = lnstyle.color;

#ifdef DEBUG
    printf("\nIn cir_idt:\n");
    printf("x=%d,y=%d,rad=%d,color=%d\n",x,y,rad,color);
#endif

#ifdef OUTPUT
    fprintf(idt,"%cV@%3d%3d%cV%1d%c' %3d",
        ESC,x,y,ESC,color,ESC,rad);
    fflush(idt);
#endif

} /* cir_idt */

```

```

ndc_dc(n, coord)
int n;
struct XY coord[];

/*****/
/* scale normalized device coordinates (NDC) to */
/* device coordinates (DC). */
/* dcx = int[ndcx/63] dcy = int[ndcy/63] */
/*****/
{
    int i;

    for (i=0; ((i < n) && (i < MAXPTS)); i++)
    {
        if (coord[i].x > DC_MALL)
            coord[i].x = DC_MAX;
        else
            coord[i].x = coord[i].x / S_FACTOR;
        if (coord[i].y > DC_MALL)
            coord[i].y = DC_MAX;
        else
            coord[i].y = coord[i].y / S_FACTOR;
    }
}

```

Aug 6 23:54 1983 vdi.h Page 1

```

/*****
/*      vdi.h      */
/*****
/* Header for VDI device drivers      */
/* To be included in appl. pgm. & driver */
/*      */
/* Opcodes used follow those used in GSX */
/* (Langhorst article - VDI by      */
/* Digital Research and      */
/* Graphics Software Systems Inc.) */
/*****

/* Normalized device coordinates for x and y */

#define NDC_MIN 0
#define NDC_MAX 32767

/*****
/* VDI opcodes */
/*****

#define VDIOPEN      1      /* initialize a graphics workstation */
#define VDICLOSE     2      /* stop graphics output to this workstation */
#define VDICLEAR     3      /* clear display device */
#define VDILINE      6      /* output a polyline */
#define VDIMARKER    7      /* output a polymarker */
#define VDITEXT      8      /* output text starting at a given location */
#define VDIFILL      9      /* fill a closed polygonal figure */
#define VDIGDP      11      /* display Generalized Drawing Primitive */
#define VDICHRHGT    12      /* set text size */
#define VDICHRUP     13      /* set text direction */
#define VDILNTYPE    15      /* set polyline type */
#define VDILNCOLOR   17      /* set polyline color */
#define VDIMKTYPE    18      /* set marker type for polymarkers */
#define VDIMKCOLOR   20      /* set polymarker color */
#define VDITXFONT    21      /* set text font */
#define VDITXCOLOR   22      /* set text color */

/* from GSX required CRT opcode list, am missing only 4,5,10,14,25,26 */
/*****
/* for IDT: */
/* VDIESCAPE      5 - could later add some device dependent operations */
/* VDICELL        10 - could probably do - use COLOR MAP (or pattern fill) */
/* VDIFILLCOLOR   25 - not used cause IDT can only fill with outlined color */
/*****

/* possible colors */

#define BLACK      0
#define BLUE       1
#define GREEN      2
#define CYAN       3
#define RED        4
#define MAGENTA    5
#define YELLOW     6

```

Aug 6 23:54 1983 vdi.h Page 2

```
#define WHITE 7

/* Maximum points for polymarker and polyline */
#define MAXPTS 50

/* Maximum size of text string */
#define MAXSTRING 42

/* Polymarker types */
#define DOT 1 /* . */
#define PLUS 2 /* + */
#define ASTERISK 3 /* * */
#define OH 4 /* o */
#define XXX 5 /* x */

/* possible VDIGDP values */
#define CIRCLE 0

/* possible VDICHRHGT values (character size) */

#define T_REG 0
#define T_LARGE 1

/*****
/* global variables, structure definitions */
*****/

struct VDI_LINE
{
    int x;
    int y;
};

struct VDI_MARK
{
    int type;
    struct
    {
        int x;
        int y;
    } c[MAXPTS];
};

struct VDI_TEXT
{
    int x;
    int y;
    char stg[MAXSTRING];
};

struct VDI_FILL
{
    int x;
    int y;
};
```

Aug 6 23:54 1983 vdi.h Page 3

```

};

struct VDI_GDP
{
    int gp;          /* general purpose, e.g. CIRCLE - radius */
    struct
    {
        int x;
        int y;
    } c[MAXPTS];
};

union VDI_PARM          /* opcode dependent parameters to pass to driver */
{
    struct VDI_LINE
    {
        line[MAXPTS];
    };
    struct VDI_MARK
    {
        mark;
    };
    struct VDI_TEXT
    {
        text;
    };
    struct VDI_FILL
    {
        fill;
    };
    struct VDI_GDP
    {
        gdp;
    };
};

struct VDI_REC          /* record of paramaters to pass to driver */
{
    int gen;          /* general - #points or color #, etc. */
    union VDI_PARM
    {
        parm;
    };
};

/* Add later - to prompt user for ascii color string */
struct COLOR
{
    char *str;
    int value;
};

struct LINE_STYLE
{
    int type;          /* polyline type */
    int width;          /* polyline width */
    int color;          /* polyline color */
};

struct MARK_STYLE
{
    int type;          /* polymarker type */
    int size;          /* polymarker scale factor */
    int color;          /* polymarker color */
};

struct TEXT_STYLE

```

Aug 6 23:54 1983 vdi.h Page 4

```
{
  int font;      /* text font */
  int size;      /* text size */
  int color;     /* text color */
};
```


Aug 6 23:55 1983 idt.h Page 1

```

/*****
/*          idt.h          */
/*****
/* Header for IDT 2200 driver (ddidt.c) */
/*****

#include "vdi.h"

#define IDT_FILE "/dev/ln66"
FILE *idt;

#define ESC '^'
#define TERM '|

/* device coordinates for x and y */

#define DC_MIN 0
#define DC_MAX 512

#define DC_MALL 32256

/*****
/* scale factor - highest divisor NDC_MAX/DC_MAX = 63.998046 */
/* used to change from ndc to dc */
/* here use 63 so can enter integer ndc so max ndc value */
/* actually allowed is 32,256, if too large, use this (512) */
/*****
#define S_FACTOR 63

/*****
/* Character coordinates for x and y (col and line) */
/*
/* In character mode, coordinates run from one to
/* the max where 1,1 starts in the upper left-hand
/* corner and 51,85 or 25,42 are the max values in
/* the lower right area of the screen - opposite of
/* regular graphics mode
/*****
#define MAX_LX 42 /* Large char. max x or col. */
#define MAX_LY 25 /* Large char. max y or line */
#define MAX_RX 85 /* Reg. char. max x or col. */
#define MAX_RY 51 /* Reg. char. max y or line */

/* actually should scale to exact as per VDI - so would */
/* use scale factor 0.0156255 - DC_MAX/NDC_MAX */

#define GOOD 0
#define BAD -1

#define OFF 0
#define ON 1

/* Marker types - later don't explicitly define so can adjust size */

#define I_DOT "^^005^V@%3d%3d^G"

```

Aug 6 23:55 1983 idt.h Page 2

```
#define I_PLUS "~0~MA02444488888888884444111122222222|"
#define I_AST "~0~MA034444888888888844441111222222221115555AAAAAAA5555666699999999|"
#define I_OH "~0~MA055555AAAAAAA5555666699999999|"
#define I_XXX "~0~MA055555AAAAAAA5555666699999999|"
```

```
struct XY
{
    int x;
    int y;
};
```

/* Note: the vdi.h header is included for the following */

```
static struct LINE_STYLE lnstyle;
static struct MARK_STYLE mkstyle;
```

```
static struct TEXT_IDT
{
    int foreblink;    /* foreground blink off or on */
    int back;         /* background color */
    int backblink;    /* background blink off or on */
    struct TEXT_STYLE txstyle;
} tidt;
```

A p p e n d i x 2

Application Program Examples

Aug 6 23:44 1983 compbike Page 1

: 'Compile application program bike.c with the idt device driver'

```
ncc -O -DOUTPUT bike.c \  
      ddidt.c \  
      -l3 \  
      -o bike
```

: 'run Ex. bike 0 4'

: 'for black background and red general drawing color'

Aug 6 23:46 1983 bike.c Page 1

```

/*****
/**      bike.c      **/
/*****
/* Advertisement for Bike Meeting at work */
/* Use ndc 0 - 32256 (63*512) */
*****/

#include <stdio.h>
#include "vdi.h"      /* virtual device interface header */

#define END -1

struct VDI_REC vdirec; /* record to pass to VDI driver */

main(argc,argv)
int argc;
char *argv[];
{
    int sc_color,color,frame[MAXPTS*2],i;

    /* convert to integer before use */
    /* later - use string-value table & pass color string */
    i=atod(argv[1],&sc_color);
    i=atod(argv[2],&color);

    /* open workstation */
    idt_dvr(VDIOPEN,&vdirec);

    /* clear screen */
    vdirec.gen = sc_color;      /* background color */
    idt_dvr(VDICLEAR,&vdirec);

    /***/
    /* draw left tire */
    /***/

    vdirec.gen = color;
    idt_dvr(VDILNCOLOR,&vdirec);

    circle(6300,6300,4410);
    circle(6300,6300,5040);
    fill(6300,11025);

    /***/
    /* draw right tire */
    /***/

    circle(18900,6300,4410);
    circle(18900,6300,5040);

    fill(18900,11025);

```

Aug 6 23:46 1983 bike.c Page 2

```

/*****
/* draw frame outline */
*****/

vdirec.gen = BLUE;
idt_dvr(VDILNCOLOR,&vdirec);

frame[0]=7560; frame[1]=18900; frame[2]=6300; frame[3]=6300;
frame[4]=12600; frame[5]=6300; frame[6]=7560; frame[7]=18900;
frame[8]=17010; frame[9]=18900; frame[10]=12600; frame[11]=6300;
frame[12]=END;
polyline(frame);
frame[0]=17010; frame[1]=18900; frame[2]=18900; frame[3]=6300;
frame[4]=END;
polyline(frame);

/* 1 - back stay (seat post to back axle) */

frame[0]=7875; frame[1]=18081; frame[2]=6489; frame[3]=6300;
frame[4]=END;
polyline(frame);

/* 2 - back tire to bottom bracket */

frame[0]=6300; frame[1]=6930; frame[2]=12348; frame[3]=6930;
frame[4]=END;
polyline(frame);
fill(6615,6804);

/* 3 - crank up to seat post */

frame[0]=8190; frame[1]=18900; frame[2]=12915; frame[3]=7056;
frame[4]=END;
polyline(frame);
fill(12474,7371);

/* 4 - top tube */

frame[0]=8190; frame[1]=18270; frame[2]=16821; frame[3]=18270;
frame[4]=END;
polyline(frame);
fill(16380,18585);

/* 5 - down tube (from headset) */

frame[0]=16380; frame[1]=18900; frame[2]=11970; frame[3]=6300;
frame[4]=END;
polyline(frame);
fill(12915,8190);

/* 6 - front fork */

frame[0]=16380; frame[1]=18900; frame[2]=18270; frame[3]=6300;
frame[4]=END;
polyline(frame);
frame[0]=18270; frame[1]=6300; frame[2]=18900; frame[3]=6300;

```

Aug 6 23:46 1983 bike.c Page 3

```

frame[4]=END;
polyline(frame);
fill(18396,6930);
fill(17640,11655);

/* fill for back stay */
fill(7119,12600);
fill(6804,10080);

/* fill for 2 */
fill(10710,6804);

/*****/
/* left spokes */
/*****/

vdirec.gen = YELLOW;
idt_dvr(VDILNCOLOR,&vdirec);

frame[0]=6300; frame[1]=6300; frame[2]=6300; frame[3]=1890;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=3780; frame[3]=3780;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=3150; frame[3]=6300;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=3465; frame[3]=8820;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=6300; frame[3]=10710;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=8946; frame[3]=8946;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=9450; frame[3]=6300;
frame[4]=END;
polyline(frame);
frame[0]=6300; frame[1]=6300; frame[2]=9135; frame[3]=3780;
frame[4]=END;
polyline(frame);

/*****/
/* right spokes */
/*****/

frame[0]=18900; frame[1]=6300; frame[2]=18900; frame[3]=1890;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=16380; frame[3]=3780;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=15750; frame[3]=6300;
frame[4]=END;

```

Aug 6 23:46 1983 bike.c Page 4

```

polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=16065; frame[3]=8820;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=18900; frame[3]=10710;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=21546; frame[3]=8946;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=22050; frame[3]=6300;
frame[4]=END;
polyline(frame);
frame[0]=18900; frame[1]=6300; frame[2]=21735; frame[3]=3780;
frame[4]=END;
polyline(frame);

/******/
/* handlebars */
/*****/

vdirec.gen = BLUE;
idt_dvr(VDILNCOLOR,&vdirec);

frame[0]=17010; frame[1]=18900; frame[2]=18900; frame[3]=18900;
frame[4]=19530; frame[5]=18270; frame[6]=19530; frame[7]=17325;
frame[8]=16380; frame[9]=17010; frame[10]=END;
polyline(frame);
frame[0]=17010; frame[1]=18585; frame[2]=18585; frame[3]=18585;
frame[4]=19215; frame[5]=17955;
frame[6]=19215; frame[7]=17640; frame[8]=16695; frame[9]=17325;
frame[10]=16380; frame[11]=17010; frame[12]=END;
polyline(frame);
fill(17640,18711);

/*****/
/* seat */
/*****/

vdirec.gen = YELLOW;
idt_dvr(VDILNCOLOR,&vdirec);

frame[0]=7875; frame[1]=18900; frame[2]=7560; frame[3]=19530;
frame[4]=END;
polyline(frame);
frame[0]=10395; frame[1]=19530; frame[2]=6615; frame[3]=19530;
frame[4]=6300; frame[5]=20160; frame[6]=6300; frame[7]=20475;
frame[8]=7245; frame[9]=20790; frame[10]=8190; frame[11]=20790;
frame[12]=10395; frame[13]=19530; frame[14]=END;
polyline(frame);
fill(8820,19845);

/*****/
/* crank */
/*****/
circle(12348,6930,1890);

```


Aug 6 23:46 1983 bike.c Page 5

```

/*****/
/* pedals */
/*****/

frame[0]=12348; frame[1]=6930; frame[2]=12348; frame[3]=4410;
frame[4]=END;
polyline(frame);
frame[0]=12726; frame[1]=4410; frame[2]=11970; frame[3]=4410;
frame[4]=END;
polyline(frame);
frame[0]=12348; frame[1]=6930; frame[2]=12348; frame[3]=9450;
frame[4]=END;
polyline(frame);
frame[0]=12726; frame[1]=9450; frame[2]=11970; frame[3]=9450;
frame[4]=END;
polyline(frame);

/*****/
/* write headings */
/*****/

vdirec.gen = color;
idt_dvr(VDITXCOLOR,&vdirec);

vdirec.gen = T_LARGE;
idt_dvr(VDICHRRHGT,&vdirec);

text(28287,7623,"CCC Meeting Today"); /* dc- line 3, col 10 */
text(25641,5355,"11:15 South Blue Room"); /* dc- line 5, col 7 */

/* close workstation */
idt_dvr(VDICLOSE,&vdirec);

} /* end of bike main */


circle(x,y,rad)
int x,y,rad;
{

    vdirec.gen = CIRCLE;
    vdirec.parm.gdp.c[0].x = x;
    vdirec.parm.gdp.c[0].y = y;
    vdirec.parm.gdp.gp = rad;
    idt_dvr(VDIGDP,&vdirec);

}

```

Aug 6 23:46 1983 bike.c Page 6

```

fill(x,y)
int x,y;
{
    vdirec.parm.fill.x = x;
    vdirec.parm.fill.y = y;
    idt_dvr(VDIFILL,&vdirec);
}

```

```

polyline(pts)
int pts[];
{
    int i,j;

    j = 0;

    for (i=0; ((pts[i] != END) && (pts[i+1] != END)); i=i+2)
    {
        vdirec.parm.line[j].x = pts[i];
        vdirec.parm.line[j].y = pts[i+1];
        j++;
    }
    vdirec.gen = j;
    idt_dvr(VDILINE,&vdirec);
} /* polyline */

```

```

text(y,x,stg)
int x,y;
char *stg;
{
    strcpy(vdirec.parm.text.stg,stg);
    vdirec.parm.text.x = x;
    vdirec.parm.text.y = y;
    idt_dvr(VDITEXT,&vdirec);
} /* text */

```

Aug 6 23:47 1983 comppool Page 1

: 'Compile application program pool.c with the idt device driver'

```
ncc -O -DOUTPUT pool.c \  
                ddidt.c \  
                -13 \  
                -o pool
```

: 'run Ex. pool 0 2'

: 'for black background and green general drawing color'

Aug 6 23:49 1983 pool.c Page 1

```

/*****
/**          pool.c          **/
*****/
/* Draw a pool table, set up the balls, */
/* rack them, roll the cue ball, erase */
/* other balls, add title.             */
*****/

#include <stdio.h>
#include "vdi.h"

#define END -1

struct VDI_REC vdirec; /* record to pass to VDI driver */

main(argc,argv)
int argc;
char *argv[];
{
    int sc_color,color,frame[MAXPTS*2],i;

    /* convert to integer before use */
    /* later - use string-value table & pass color string */
    i=atod(argv[1],&sc_color);
    i=atod(argv[2],&color);

    /* open workstation */
    idt_dvr(VDIOPEN,&vdirec);

    /* clear screen */
    vdirec.gen = sc_color; /* background color */
    idt_dvr(VDICLEAR,&vdirec);

    /* draw table outline */
    vdirec.gen = color;
    idt_dvr(VDILNCOLOR,&vdirec);
    frame[0]=1890; frame[1]=5040; frame[2]=30366; frame[3]=5040;
    frame[4]=30366; frame[5]=27216; frame[6]=1890; frame[7]=27216;
    frame[8]=1890; frame[9]=5040; frame[10]=-1;
    polyline(frame);
    frame[0]=2835; frame[1]=6363; frame[2]=29421; frame[3]=6363;
    frame[4]=29421; frame[5]=25893; frame[6]=2835; frame[7]=25893;
    frame[8]=2835; frame[9]=6363; frame[10]=-1;
    polyline(frame);
    fill(16065,16065);

    /* 1 - make pockets */
    vdirec.gen = WHITE;
    idt_dvr(VDILNCOLOR,&vdirec);
    circle(2835,25893,504);
    fill(2835,25893);

    circle(16128,25893,504);
    fill(16128,25893);

```

Aug 6 23:49 1983 pool.c Page 2

```

circle(29421,25893,504);
fill(29421,25893);

circle(29421,6363,504);
fill(29421,6363);

circle(16128,6363,504);
fill(16128,6363);

circle(2835,6363,504);
fill(2835,6363);

/* 2 - rack balls */
frame[0]=8505; frame[1]=19656; frame[2]=8505; frame[3]=12474;
frame[4]=11466; frame[5]=16065; frame[6]=8505; frame[7]=19656;
frame[8]=-1;
polyline(frame);

vdirec.gen = BLUE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(10710,16065,315);
fill(10710,16065);
vdirec.gen = RED;
idt_dvr(VDILNCOLOR,&vdirec);
circle(10269,16695,315);
fill(10269,16695);
vdirec.gen = YELLOW;
idt_dvr(VDILNCOLOR,&vdirec);
circle(10269,15435,315);
fill(10269,15435);
vdirec.gen = MAGENTA;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9828,17325,315);
fill(9828,17325);
vdirec.gen = BLACK;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9828,16065,315);
fill(9828,16065);
vdirec.gen = CYAN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9828,14805,315);
fill(9828,14805);
vdirec.gen = YELLOW;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9387,17955,315);
fill(9387,17955);
vdirec.gen = CYAN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9387,16695,315);
fill(9387,16695);
vdirec.gen = MAGENTA;
idt_dvr(VDILNCOLOR,&vdirec);
circle(9387,15435,315);
fill(9387,15435);
vdirec.gen = BLUE;

```

Aug 6 23:49 1983 pool.c Page 3

```

idt_dvr(VDILNCOLOR,&vdirec);
circle(9387,14175,315);
fill(9387,14175);
vdirec.gen = CYAN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(8946,18585,315);
fill(8946,18585);
vdirec.gen = RED;
idt_dvr(VDILNCOLOR,&vdirec);
circle(8946,17325,315);
fill(8946,17325);
vdirec.gen = BLUE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(8946,16065,315);
fill(8946,16065);
vdirec.gen = YELLOW;
idt_dvr(VDILNCOLOR,&vdirec);
circle(8946,14805,315);
fill(8946,14805);
vdirec.gen = RED;
idt_dvr(VDILNCOLOR,&vdirec);
circle(8946,13545,315);
fill(8946,13545);
vdirec.gen = GREEN;
idt_dvr(VDILNCOLOR,&vdirec);
frame[0]=8505; frame[1]=19656; frame[2]=8505; frame[3]=12474;
frame[4]=11466; frame[5]=16065; frame[6]=8505; frame[7]=19656;
frame[8]=-1;

```

```

polyline(frame);

```

```

/* roll cue ball */
vdirec.gen = WHITE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(26271,16065,315);
fill(26271,16065);
circle(22428,16065,315);
fill(22428,16065);
vdirec.gen = GREEN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(26271,16065,315);
fill(26271,16065);
vdirec.gen = WHITE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(18648,16065,315);
fill(18648,16065);
vdirec.gen = GREEN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(22428,16065,315);
fill(22428,16065);
vdirec.gen = WHITE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(14868,16065,315);
fill(14868,16065);
vdirec.gen = GREEN;

```

Aug 6 23:49 1983 pool.c Page 4

```

idt_dvr(VDILNCOLOR,&vdirec);
circle(18648,16065,315);
fill(18648,16065);
vdirec.gen = WHITE;
idt_dvr(VDILNCOLOR,&vdirec);
circle(11340,16065,315);
fill(11340,16065);
vdirec.gen = GREEN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(14868,16065,315);
fill(14868,16065);

```

```

/* erase balls */
vdirec.gen = GREEN;
idt_dvr(VDILNCOLOR,&vdirec);
circle(10710,16065,315);
fill(10710,16065);
circle(10269,16695,315);
fill(10269,16695);
circle(10269,15435,315);
fill(10269,15435);
circle(9828,17325,315);
fill(9828,17325);
circle(9828,16065,315);
fill(9828,16065);
circle(9828,14805,315);
fill(9828,14805);
circle(9387,17955,315);
fill(9387,17955);
circle(9387,16695,315);
fill(9387,16695);
circle(9387,15435,315);
fill(9387,15435);
circle(9387,14175,315);
fill(9387,14175);
circle(8946,18585,315);
fill(8946,18585);
circle(8946,17325,315);
fill(8946,17325);
circle(8946,16065,315);
fill(8946,16065);
circle(8946,14805,315);
fill(8946,14805);
circle(8946,13545,315);
fill(8946,13545);

```

```

/* write headings */
vdirec.gen = color;
idt_dvr(VDITXCOLOR,&vdirec);

```

```

vdirec.gen = T_LARGE;
idt_dvr(VDICHRRHGT,&vdirec);

```

```

text(30933,7623,"Pool Anyone?"); /* dc -line 1, col 10 */

```

Aug 6 23:49 1983 pool.c Page 5

```

    /* close workstation */
    idt_dvr(VDICLOSE,&vdirec);
} /* end of bike main */

```

```

circle(x,y,rad)
int x,y,rad;
{
    vdirec.gen = CIRCLE;
    vdirec.parm.gdp.c[0].x = x;
    vdirec.parm.gdp.c[0].y = y;
    vdirec.parm.gdp.gp = rad;
    idt_dvr(VDIGDP,&vdirec);
}

```

```

fill(x,y)
int x,y;
{
    vdirec.parm.fill.x = x;
    vdirec.parm.fill.y = y;
    idt_dvr(VDIFILL,&vdirec);
}

```

```

polyline(pts)
int pts[];
{
    int i,j;

    j = 0;

    for (i=0; ((pts[i] != END) && (pts[i+1] != END)); i=i+2)
    {
        vdirec.parm.line[j].x = pts[i];
        vdirec.parm.line[j].y = pts[i+1];
        j++;
    }
    vdirec.gen = j;
    idt_dvr(VDILINE,&vdirec);
} /* polyline */

```


Aug 6 23:49 1983 pool.c Page 6

```
text(y,x,stg)
int x,y;
char *stg;
{
    strcpy(vdirec.parm.text.stg,stg);
    vdirec.parm.text.x = x;
    vdirec.parm.text.y = y;
    idt_dvr(VDITEXT,&vdirec);
} /* text */
```

Aug 6 23:49 1983 compdeplot Page 1

: 'Compile application program plot.c with the idt device driver'
: 'Watch trace statements here while running'

```
ncc -O -DDEBUG -DOUTPUT plot.c \  
                        ddidt.c \  
                        -l3 \  
                        -o plotd
```

: 'run Ex. plotd 0 5 3'
: 'for black background, magenta drawing color, asterisk marker'

Aug 6 23:50 1983 plot.c Page 1

```

/*****
/**          plot.c          **/
*****/
/* Part of a graph testing polymarkers. */
*****/

#include <stdio.h>
#include "vdi.h"          /* virtual device interface header */

struct VDI_REC vdirec; /* record to pass to VDI driver */

main(argc,argv)
int argc;
char *argv[];
{
    int sc_color,color,mktype,i;

    /* convert to integer before use */
    /* later - use string-value tables for friendlier interface */
    i=atod(argv[1],&sc_color);
    i=atod(argv[2],&color);
    i=atod(argv[3],&mktype);

    /* open workstation */
    idt_dvr(VDIOPEN,&vdirec);

    /* clear screen */
    vdirec.gen = sc_color;          /* background color */
    idt_dvr(VDICLEAR,&vdirec);

    /***/
    /* write headings */
    /***/

    vdirec.gen = color;
    idt_dvr(VDITXCOLOR,&vdirec);

    vdirec.gen = T_LARGE;
    idt_dvr(VDICHRRHGT,&vdirec);

    text(19026,1575,"POLYMARKER");

    /***/
    /* plot points */
    /***/

    vdirec.gen = color;
    idt_dvr(VDIMKCOLOR,&vdirec);

    vdirec.parm.mark.c[0].x = 3150; vdirec.parm.mark.c[0].y = 14490;
    vdirec.parm.mark.c[1].x = 6300; vdirec.parm.mark.c[1].y = 25200;

```

Aug 6 23:50 1983 plot.c Page 2

```

vdirec.parm.mark.c[2].x = 19450; vdirec.parm.mark.c[2].y = 28350;
vdirec.parm.mark.c[3].x = 12600; vdirec.parm.mark.c[3].y = 23940;
vdirec.parm.mark.c[4].x = 15750; vdirec.parm.mark.c[4].y = 18900;
vdirec.parm.mark.c[5].x = 18900; vdirec.parm.mark.c[5].y = 27090;
vdirec.parm.mark.c[6].x = 22050; vdirec.parm.mark.c[6].y = 28980;
vdirec.parm.mark.c[7].x = 25200; vdirec.parm.mark.c[7].y = 31500;
vdirec.parm.mark.c[8].x = 28350; vdirec.parm.mark.c[8].y = 31500;

vdirec.gen = 9;
vdirec.parm.mark.type = mktype;

idt_dvr(VDIMARKER,&vdirec);

/* close workstation */
idt_dvr(VDICLOSE,&vdirec);

} /* end of plot main */

text(y,x,stg)
int x,y;
char stg[MAXSTRING];
{
    strcpy(vdirec.parm.text.stg,stg);
    vdirec.parm.text.x = x;
    vdirec.parm.text.y = y;
    idt_dvr(VDITEXT,&vdirec);
} /* text */

```

Aug 6 23:51 1983 plotdout Page 1

Plot.c trace (DEBUG) on.

In res_idt

In clr_idt

In clr_idt

In text_idt:

tit=POLYMARKER

line=302, col=25, color=5, size=1

In text_idt:

tit=POLYMARKER

line=10, col=2, color=5, size=1

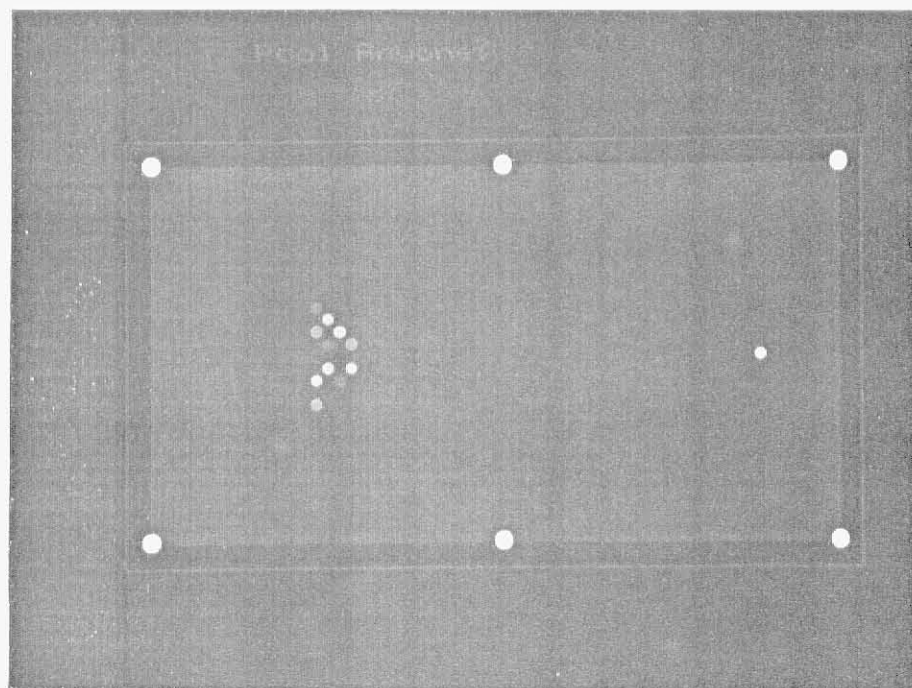
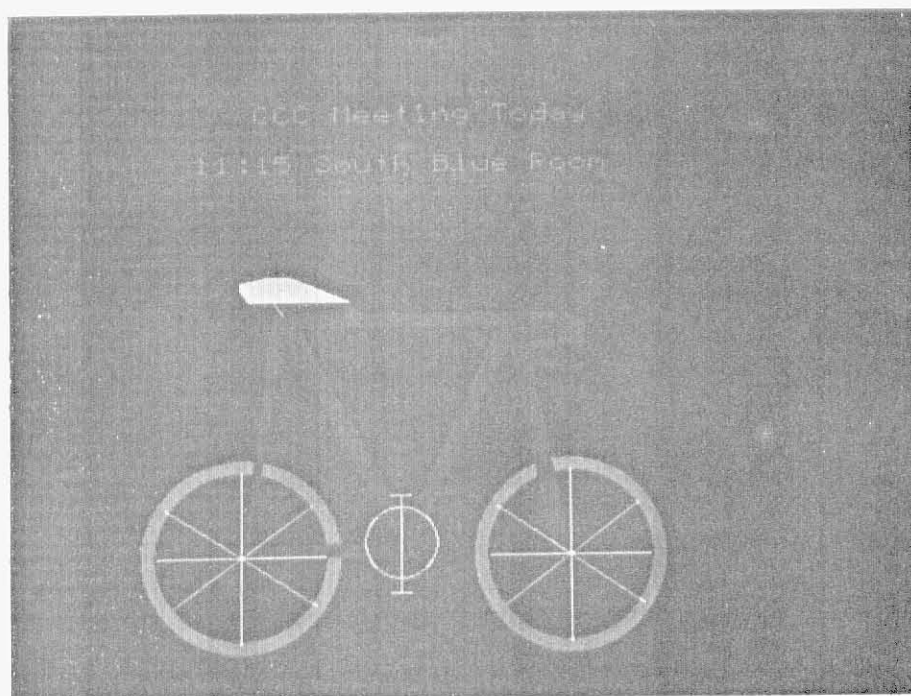
In polym_idt:

	marker=3, color=5, #points=9
i=0,	Location: x=50, y=230
i=1,	Location: x=100, y=400
i=2,	Location: x=308, y=450
i=3,	Location: x=200, y=380
i=4,	Location: x=250, y=300
i=5,	Location: x=300, y=430
i=6,	Location: x=350, y=460
i=7,	Location: x=400, y=500
i=8,	Location: x=450, y=500

In res_idt

A p p e n d i x 3

Application Program Outputs



REVIEW OF COMPUTER GRAPHICS
STANDARDIZATION EFFORTS WITH EMPHASIS ON
GKS, VDI, and VDM

by

DEBRA MAE HERRING

B.S., Ohio State University, 1979

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1983

Abstract

Recent proposals in computer graphics software standards are investigated with emphasis on the problem of computer graphics software portability. The specific standards studied include the application level Graphical Kernel System (GKS), the device level Virtual Device Interface (VDI), Virtual Device Metafile (VDM) and GKS Metafile. GKS, developed in Germany, is a draft proposal for the first international graphics standard. The VDI and VDM are being developed by the American National Standards Institute to standardize the interface to graphics device drivers and allow portability of files containing picture descriptions. As part of the study, a small graphics device driver was developed based on a subset of the VDI functions. The implementation, written in C language, provides subroutine calls to the device driver which will draw the requested picture or text on the graphics device screen. The metafile concept was studied and a simple metafile was written and transferred to a KSU computer to be read by the KSU metafile reader and redrawn by the KSU GKS package.