An Examination of Computer Graphics Standards

An Honors Thesis (ID 499)

by

Kenneth L. Mikeworth

Thesis Director

Michael McNew

Ball State University
Muncie, Indiana
May, 1986

Expected date of Graduation: Winter, 1986-1987
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>HISTORY AND BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>Early Beginnings: Concept Development</td>
<td>4</td>
</tr>
<tr>
<td>Development Of A Standard</td>
<td>4</td>
</tr>
<tr>
<td>Selection Of GKS.</td>
<td>5</td>
</tr>
<tr>
<td>The Goal Is Reached.</td>
<td>5</td>
</tr>
<tr>
<td>PRESENT STATUS</td>
<td>6</td>
</tr>
<tr>
<td>Various Standards Available</td>
<td>6</td>
</tr>
<tr>
<td>The Case Study: Introduction</td>
<td>8</td>
</tr>
<tr>
<td>3D Graphics</td>
<td>9</td>
</tr>
<tr>
<td>Coordinate Systems.</td>
<td>10</td>
</tr>
<tr>
<td>Windows &amp; Clipping.</td>
<td>13</td>
</tr>
<tr>
<td>Current Position.</td>
<td>14</td>
</tr>
<tr>
<td>Picture Segments.</td>
<td>16</td>
</tr>
<tr>
<td>Priorities.</td>
<td>17</td>
</tr>
<tr>
<td>Input &amp; Output.</td>
<td>18</td>
</tr>
<tr>
<td>Interaction.</td>
<td>20</td>
</tr>
<tr>
<td>Text.</td>
<td>22</td>
</tr>
<tr>
<td>Workstations.</td>
<td>24</td>
</tr>
<tr>
<td>Language Bindings.</td>
<td>25</td>
</tr>
<tr>
<td>PROJECTIONS</td>
<td>26</td>
</tr>
<tr>
<td>The Future Of Computer Graphics Standards.</td>
<td>26</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>28</td>
</tr>
<tr>
<td>Computer Graphics Standards.</td>
<td>28</td>
</tr>
<tr>
<td>AN IMPLEMENTATION OF THE GKS GRAPHICS STANDARD</td>
<td>29</td>
</tr>
<tr>
<td>A Project In GKS.</td>
<td>29</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
</tr>
</tbody>
</table>
An Examination of Computer Graphics Standards

INTRODUCTION

The computer age. High technology. Rapid advances. Home computers. Sometimes one really wonders what this world is coming to. No one will dispute that rapid advances, high technology and the computer age, especially home computers or PCs are impacting the lives of everyone and touching every realm of knowledge. To aid in these advances, computer scientists must work on developing special interfaces, easy-to-use, understandable methods of communication between the increasingly powerful machine and the inexperienced and unknowledgeable user. These interfaces take many forms and are present in nearly every computer application to some degree. All of the most popular computer software packages contain very good interfaces. Examples are the menu-driven lotus with one-keystroke commands, word processors which include an overlay for the keyboard pointing out special keys and combinations, and the icon and mouse concepts made popular by the introduction of the MacIntosh. By looking at these examples, it is interesting to note that menu-driven programs and the icon and mouse concepts both require use of computer graphics in one form or another in order to implement the interface fully. Indeed, generating on the screen an attractive display of options for the user is one of the best methods of creating a feeling of understanding and enjoyment in a computer application.
Why A Computer Graphics Standard?

Computer graphics can take many forms, from merely displaying a row of asterisks on the screen to drawing multicolor, complex designs on a color monitor. The hardware used in generating computer graphics is also widely varied, with many hardware manufacturers developing their own products and many different options available. The list, which includes light pens, mice, graphics tablets, keyboards, touch-screen monitors, printers, plotters, and high-resolution graphics screens, is incredibly vast. Due to this great diversity in graphics uses, hardware, and development, over time the many separate companies began to diverge. Divergence in this type of field means that companies using equipment from one vendor cannot obtain replacements or even equipment from another vendor which will work with their present equipment or software. Once a company had a computer system working, they were locked into that system. If the equipment became obsolete or expanded equipment was needed, the company had to start from scratch in both hardware and software because nothing could be obtained that would work with their present system. This motivated national organizations in several countries to pursue the development of an international graphics standard which would specify reasonable expectations for graphics equipment and programs and would allow for hardware from different vendors to work together with minor software
adjustments. Typical expectations for graphics programs were set down so that each would reference the hardware identically. So began the progress of a graphics standard. Although the technology for most of the equipment available today has been around for approximately twenty-five years, the completion of a graphics standard is still very young, about 12 years.[9, p.1] The area is still growing and changing to meet the needs of companies and organizations today.

GKS has the distinction of laying the groundwork for the development of graphics standards. Some feel that the development of GKS as the international graphics standard has begun a new and significant area of study and development:

For the first time, a methodological framework for the various concepts within the field of computer graphics has been developed. This is the base for a common understanding and a common terminology for creating computer graphics systems, for using computer graphics, for talking about computer graphics, and for educating students in computer graphics methods, concepts, and applications.[8, p.4]

First let us examine the background and development of a graphics standard. Next, by examining the present status of several graphics standards, we can project trends and look at the future growth, adaptation, and development of computer graphics standards. Then we can examine an example of the GKS graphics standard, and close by examining the impact of the computer and the use of graphics, as well as graphics standards, on the everyday life of individuals.
HISTORY AND BACKGROUND

Early Beginnings: Concept Development.

The first international graphics standards conference, called Seillac I, was held in May 1976. This was the culmination of planning starting in August 1974, and was the foundation for all subsequent research and development of the graphics standards. Although several nations including the United States, Germany, and Norway had begun developing some form of graphics standard prior to this conference, most were indefinite and incomplete compared to the combined work which resulted from this and subsequent conferences. At this first conference, general concepts were discussed as well as requirements that graphics standards would need in order to be satisfactory and meet the needs of the customers.

Development Of A Standard.

In September of 1978 the first working group, called ISO/TC97/SC5/WG2, of the International Standards Organization for computer graphics standards met. This working group was a subcommittee of the computer programming languages committee. At the time of this meeting, three national standards were proposed for submission as an international standard. These three were the Core from the United States, GKS from Germany, and IDIGS from Norway. At the first formal meeting the Norwegian standard was not available, and so was dropped. The Core and GKS were
compared and each was sent back with suggestions for updating so that each would become more like the other, incorporating the better ideas of both.

Selection Of GKS.

In 1979, the working group met again and selected GKS over the Core as being the more complete standard, despite the fact that the Core was a three dimensional standard and GKS was limited to the two dimensional realm. IDIGS was submitted at this meeting, but was relatively incomplete having lacked the benefits of the previous meeting. With this decision, concentration began to focus on adapting GKS to solve those problems handled by the Core. ANSI/X3H3, the subcommittee of ANSI working on the Core, proposed more than two hundred of the three hundred changes to GKS over this development. At this point, GKS 6.2 was developed. It was over 130 pages in length.[9, p.5]

Two more meetings concerning the development of GKS were set, and progress continued. Each time more problems were raised but even more were resolved. A complete list of the issues under discussion was kept and used as documentation and rationalization for the inclusion or exclusion of various features in GKS.

The Goal Is Reached.

Eventually, full agreement was reached within WG2 and the proposal was passed on to SC5 to be accepted as a
working draft. The acceptance of the proposal as a working draft by SC5 on October 8, 1981 began the long climb through bureaucracy for GKS. More comments were received on the draft proposal, and three main changes were incorporated, no doubt due to the necessary politicking. Text alignment and a STROKE input primitive were accepted into the standard, and a compromise was made on the attribute bundling concept so that attributes could be set either individually or within a bundle. In March of 1983, the draft proposal matured into Draft International Standard (DIS) 7942.

Once the status of DIS had been achieved, GKS was not subject to further change. More work was completed to generate language bindings for the more popular languages including FORTRAN and Pascal. Then, finally, after another bout of politicking, GKS 7.2 was accepted as the international graphics standard, the combined creation resulting from the hard work of several determined nations.

PRESENT STATUS
Various Standards Available.

Currently, many different graphics standards exist, including descendants of GKS' competitors. The Core graphics standard is still in existence, as well as many new standards, including PHIGS and the VDI and VDM standards. As with all graphics standards, the utility of portability, universal design, and compatibility is the main objective. Each standard seeks to discover a better means of solving
the problems inherent in developing a graphics standard. These problems are many and varied.

The PHIGS (Programmer's Hierarchical Interactive Graphics Standard) is another computer graphics standard designed to fill a need remaining in the computer graphics community. PHIGS claims to perform the tasks of modification of data better than GKS. Also, PHIGS operates on geometrically related objects and facilitates rapid drawing of the objects. These attributes of PHIGS are seen as enhancements of GKS and not as competing factors. In fact, PHIGS has been approved an international standard with the same status as GKS. ISO is working on developing a 'family' concept in computer graphics standards with GKS as the founding member.[5, p.3] PHIGS is an addition to that family which is compatible with GKS and is intended to enhance the efficiency of computer graphics and graphics standards. The fields of study which can benefit from these improvements, and hence from PHIGS, are Computer Aided Engineering, Process Control (robotics), Simulations, and other precise fields such as architecture, chemistry, and so on. Individual users are encouraged to use GKS and supplement that system with PHIGS for the benefits mentioned above.

The VDI and VDM standards work together. The manual on iVDI 720, which is the application of the VDI standards for the Intel 310 minicomputer which the project portion was performed on, was based on both of these. Primarily VDM (or
the Virtual Device Metafile) was used for the output, description, and control commands. The Virtual Device Interface (VDI) was then used to supplement the VDM standards regarding inquiry commands and input for the string and locator type input devices. At the time the book was published in 1984, the VDI and VDM standards of October, 1983 were used, which were incomplete at that time.[11, p.1-1]

Discussion of several of these problems follows, and contained within such discussion will be examples of methods various standards have employed to solve or evade each problem as well as a case study in GKS.

The Case Study: Introduction.

Sherry is the Director of Data Processing for a medium-sized manufacturing company, Indy Aluminum, Inc. She has been assigned the task of acquiring a better inventory control system, since the current method is manual and does not fully meet their needs. After looking at several Database Management Systems on the market, she has concluded that none of those fully meet the needs of Indy Aluminum, either. Sherry has now determined that the data processing department, which is familiar with the industry, can create the desired package. It would mean pulling staff away from other projects and placing them on this one, but having a comprehensive, integrated, inventory control system is a high priority for the company and Sherry feels that
management will approve the idea. Now she needs to develop a proposal for the project. She attended a demonstration on GKS last week and was impressed by the capability of the graphics package. Another benefit was that the software, due to the standardized nature of the interface, was compatible with their current resources. Since some of her staff have had some experience with GKS, she feels the proposal should be programmed and presented in animated graphics, demonstrating the needs of the company and how well the data processing team can fill them. Even the financial data and requirements could be charted easily, clearly, and understandably using this system. With increasing confidence, Sherry begins to develop her proposal using the GKS graphics system.[1]

3D Graphics.

Whether to incorporate three-dimensional graphics in the standard and how to go about this is a major problem in developing a graphics standard. Naturally, the addition of three dimensional graphics increases the size of the standard and introduces some of its own unique problems. The Core has been a three dimensional graphics standard almost from the beginning, which was a factor placing it in competition with the then two-dimensional GKS. Since about 90% of all computer graphics applications only require

[1] This case is based on actual experiences of the author, but of course, the specific information has been changed.
2-dimensional images,[8, p.501] the absence of three-dimensional capability is not often a problem in a given application. However, in the development of a standard, three-dimensional methods should be covered.

After talking with Rick, her systems supervisor, Sherry determined that three-dimensional graphics might not be worth the effort for her proposal. Although the functions were easy to use in GKS, only one person on her staff was even vaguely familiar with them, and none had any actual experience with them. Still enthusiastic, Sherry then turned her attention to the other aspects of GKS.

Coordinate Systems.

Coordinate systems pose unique problems in designing a computer graphics standard. In applications, the operator of the standard may require one of several different coordinate systems, including the typical Cartesian plane system, polar coordinates, or a coordinate system uniquely scaled for the application. Because the Cartesian plane system is the most popular and lends itself easily to computer applications, it is the only system utilized inside the computer in calculating results and in plotting. Because of this conflict, the standard must make the adjustments from the coordinate system the user is familiar with and is using in his application to the Cartesian system. In most cases, the calculations can be performed easily enough, but the picture may be distorted.
Another problem which arises from the use of coordinate systems is that of scale. As in many graphics applications, the picture can be scaled, translated, or rotated to almost any position. If an operator has an application which is in polar coordinates and he desires it to be positioned in the upper right portion of the screen, the problem of calculating the proper screen coordinates is made even more difficult. For this reason several different coordinate systems are adopted in developing a graphics standard. These are the user coordinates, world coordinates, normalized coordinates, and screen coordinates.

User coordinates are plotted on a user coordinate scale, which is the coordinate system of the user's application. The total range of the coordinates in that scale provides boundaries which are used to determine limits on the user coordinate system. Put simply, the user coordinate system is the system the operator knows and utilizes in the application.

World coordinates are plotted in the Cartesian system calculated from the operators' system. The user coordinates are translated to the Cartesian system. The range of the coordinates in this system provides a limit which is further translated to other coordinate systems. These coordinates are named because the range of the coordinates defines the 'world' within which the user's picture exists.

Normalized coordinates is also a Cartesian system. These coordinates are calculated from the world coordinates
such that the lowest bound of the picture is set to 0 and the highest bound of the picture is set to 1. In this manner, all coordinates are 'normalized' or standardized to a value between 0 and 1 on each axis.

Screen coordinates are calculated from the normalized coordinates and are also expressed in a Cartesian plane system. Computer display screens are dimensioned in pixels, x-axis by y-axis. For example, if a display monitor is dimensioned 720 x 348 then the monitor is 720 pixels wide and 348 pixels high. The screen coordinates are then computed by multiplying the normalized coordinates by the number of pixels for that axis. In this manner, all significant points of the picture can be placed on the screen and the appropriate connections made to describe the picture on the screen. Notice again that the picture may be distorted because of the difference in the proportion of height to width between the picture and the screen. This proportion is called the aspect ratio and can give some indication of how distorted the final result may appear.

In graphics standards, handling coordinate systems is a problem because any number of different coordinate systems may be displayed on a myriad of hardware. To solve this problem, the standard makes all references to the device in normalized coordinates, allowing the device driver to compute and control the use of screen coordinates. Once again, we see the effect the graphics standard has on other areas, since the programmers of the device drivers and the
manufacturers of the hardware now need to develop methods of easily and quickly handling normalized coordinates.

Coordinate systems did not appear to be any problem for Sherry. Because she was developing the charts and supervising the overall appearance of the proposal, the size could be adjusted as needed. After checking with Karl, one of the programmers working on the programming aspect of her proposal, she arranged to draw the charts as she desired them on graph paper, then allow Karl to work off of those. Karl would use the graph paper to set up user and world coordinates, then create normalized and screen coordinates while working in GKS. Once the chart was completed on the screen, Sherry would then review the results.

Windows & Clipping.

This method of display makes several other graphics features possible. First, the operator defines some part of the picture as a window. These windows can then be mapped to a viewport specified on the device. In this case, the pixel dimensions of the viewports are used in multiplication by the normalized coordinates to compute the screen coordinates. Notice the aspect ratio of the viewport is used to compute the picture distortion.

Clipping is the removal of parts of the picture which extend beyond the bounds of the window, viewport, or screen. Any parts which extend beyond these boundaries are not displayed. For computer graphics standards, this means the
standard must have some means to convey to the device whether clipping is to take place or not and on what window. Again, once the problems of handling normalized coordinates are solved, the standard merely needs to develop a command to convey the window in normalized coordinates and to flag whether clipping is on or not.

"Windows are a nice concept," thought Sherry. She realized she could display one chart, then the other, and finally both charts side by side to demonstrate the relationships between the two. For example, one major element of the proposal was the need for a new, slightly larger computer. The benefits and capacities of the current system could be examined, then the benefits and capacities of the proposed system, and finally the two could be displayed simultaneously side by side and critically examined. "The presentation will look better and be more convincing with all the great-looking charts and examples," thought Sherry as she moved to the next concern.

Current Position.

At the time the ISO was comparing graphics standards, the American Core standard contained a concept called current position, which is a method of keeping track of the current position of the cursor on the screen. Being able to find a segment of the picture and calculate the current position was a large benefit in moving segments after having displayed them. Also, the cursor could then be used to
accept input from the operator. He could then enter a point to show which segment he desired to move, then enter another point to show where the indicated segment should be moved to. This concept was incorporated into GKS when the standard was being updated by the working group. In this manner, some parts of GKS could be programmed directly from the screen. GKS had developed the pen concept of attribute handling, which allowed for an easy method of altering or setting attributes of picture elements. These differences were eventually resolved and incorporated into the GKS standard, but are still a problem in the development of new graphics standards.[9, p.4]

During the discussions on the proposal, Karl had taken the time out to show the concept of current position to Sherry, although she would never need to use it. She just seemed so interested in everything GKS had to offer, whether it was of use in this proposal or not. Karl was sure they would be using GKS and graphics more than ever now that Sherry had gotten so excited and interested in GKS. Sherry left the method of programming up to Karl, who could program the motion just as easily in the graphics program, and so wouldn't need to use the current location concept in this instance. It looked like a handy part of GKS to keep in mind, though, just in case Sherry ever did want to get directly involved.
Picture Segments.

A concept often used in graphics applications is that of picture segmentation. Often, as the operator is creating his design, he will desire to adjust some part of the picture in some way, either scaling, rotating, or translating. In order to accomplish this, he first must have some method of distinguishing which element of the picture he desires to adjust. The concept of current position also comes into play here, since the current position of the cursor on the screen must be computed relative to the objects on the screen. The position of the cursor is determined, then the closest segment is determined. Usually the selected segment is then highlighted so that the operator can confirm the selection. For graphics standards, this requires defining some method of creating segments, deleting segments, highlighting segments, and manipulating segments.

Sherry had been well introduced to the concept of segmentation during the workshop she had attended earlier. She still needed to talk to Rick, though, to determine which elements of the picture should be segmented so that they could move together. Sherry wanted to demonstrate the anticipated growth in inventory control needs as well as personnel needs for the company through motion. She pointed out to Rick the idea she wanted and promised to indicate on the graphs she was drawing which elements should move and how. Rick then showed Karl the motion scheme and discussed
how they should implement the motion in GKS. After a short meeting, the segments were organized and one more step in the proposal was implemented.

Priorities.

The possibility of prioritizing segments is a major concept in computer graphics. Segments can be prioritized so that a given segment is selected first over another segment which may lie very near to it. This assures that it will be the first segment selected for manipulation. Also, segment prioritization can help to make a picture more realistic once the colors are filled in. For example, one segment can be drawn completely on top of another if its priority is higher. In essence, it is drawn on top of any segment with a lower priority. This allows the operator to move objects and simulate motion in his design through realistic scenes.

In some of the graphics applications, particularly the diagram emphasizing the importance of an integrated, on-line system, the segments needed to be prioritized so that each in turn could be drawn on top of the other while discussion referred to that portion of the system. To begin with, a small-scale drawing of the entire process, from ordering to billing was displayed on the screen. Then each segment, corresponding to each step in the manufacturing process for Indy, was drawn on top of the others and enlarged. Sherry could then discuss the needs, benefits, and effects of the proposed system on order entry. Each would then, in turn, be
displayed and discussed. Once again, Rick and Karl had a short meeting and determined the processing that needed to be performed. Sherry would also be able to leave the segment enlarged until discussion was complete on that segment, then push a button to signal that the next segment should be drawn. Sherry was thrilled with the results of that screen.

Input & Output.

Problems can occur related to device input and output, which is a very important concept when discussing computer graphics. Various methods of graphics input have existed for many years, long before the concept of graphics standards. It then became necessary for graphics standards to incorporate a method of handling the myriad of input devices available. It was no small problem to resolve the standard handling of light pens, graphics tablets, mice, keyboard input, and other devices. This is another change that was made to GKS in its development.

GKS defined six groups of input devices and the types of values these devices should provide as input data. These groups are the locator, valuator, pick, choice, string, and stroke. Typical devices for each of these, in order, include the terminal-displayed crosshair, potentiometers or dials, lightpen, button-box, keyboard, and graphics tablet. Six
separate commands, forms of the request command, are then used to accept the input data for each group.

The changes in this solution have had a great effect on the manufacturers of input devices, because the devices require some method of generating all the data which GKS needs for the input operation. Hence, any graphics standard needs to consider the hardware and manufacturers' limitations in determining how to successfully handle different input values from a myriad of input devices.

Output poses a similar, yet equally complex problem. Although some kind of graphics monitor is nearly always used, differing technologies incorporating Cathode Ray Tubes, Raster scan devices, as well as plotters and printers make the task challenging. The differing capabilities of these devices as well as the wide array of manufacturers increases the challenge in standardizing a method of output. During it's development, GKS was adapted to allow the use of several different output devices simultaneously. As with many innovations related to the computer, the complexity of the problems being solved is phenomenal.

The treatment of output is resolved differently in GKS. Instead of grouping the devices, GKS grouped the types of picture segments being output. These groups were defined primarily as lines, called polylines, points, called polymarkers, and text. Although other classifications do exist, these three cover the major portion of the output segments. These segments, then, are sent to the output
channel and the method of output is left once again to the device and the device driver.

Thrilled with the professional look of her proposal, Sherry wanted to do one more thing. She acquired a large screen television to use in the board room for her presentation. She would work from a terminal on a high stand in the front of the room. She could then stand and talk, changing the display as needed for the presentation. Any action on her monitor would also display on the large screen television. Although the resolution on the television would not be fantastic, it would be adequate and enable everyone to clearly see the display. Her terminal would also need several input devices, but she limited herself to a keyboard and a graphics tablet. With the keyboard, she would only need to hit return to advance the screens of the integrated on-line system. She could use the graphics tablet as a pointer, gesturing to portions of the display as needed by the locator function of the tablet.

Interaction.

Developing a program or standard which interacts with the operator is very critical. The ability to request input from the user on the spot while creating a design allows him to place each segment exactly as he desires. To create a program which does interact with the user, the concepts of windowing, input and output, and segment priority are crucial. Several approaches to this problem are available.
The application program could be placed on hold while the input process occurs. Or, the two processes could operate simultaneously. Finally, the input process could wait and upon receiving input suspend the graphics process and pass the data, then wait again. In a more complete graphics standard, all of these types will be made available so that every possibility is covered.

Interaction was a necessary element in the presentation because Sherry desired to control the pace of the motion. Interaction was needed to accept her input to the system and act upon it. Because the majority of the time would be spent in display with Sherry perhaps pointing to elements of the picture but not entering any data, Rick and Karl determined that simultaneous processing of the package and the input/output process would work best. This may slow the motion down, but Sherry desired control of the motion anyway. The event mode of input, as it was known in GKS, was then put into effect by Rick and Karl.

Another part of the proposal involved training and development for the workers on how to operate the new system once it was completed. Sherry anticipated having classroom style lectures on the reasons for the change and to answer questions. She then was proposing having tutorials set up based on graphics like the proposal to make learning the specifics of their new jobs easier and more enjoyable. Interaction plays a major role in such tutorials since the
program must prompt the user for answers, accept the response, and process the information.

Text.

The handling of text is a major problem with graphics standards for several reasons. First, the creation of the text can be difficult when using line segments or pixels as the basic graphical unit. The method of drawing the text as well as the resolution are concerns which need to be addressed. Also, items such as text height, proportion, direction, angle of rotation, and font need to be solved and handled. These problems must be resolved not only for the display screen, but also for the hardcopy devices. Once the problem has been solved for each of these, problems arise in the translation from one to the other. For example, because of the higher resolution possible, many angles of rotation, sizes, and fonts can be displayed on the typical graphics display. However, the various capabilities of the hardcopy devices may make similar quality impossible when that picture is sent for hardcopy. A font or drawing which looks great on the screen may appear jagged or unbalanced on the hardcopy device.

In response to this problem, several alternatives occur. In the system design, great care could be taken to eliminate the differences in capability and quality of the devices. This will definitely take a great amount of time in research and capital in procurement of the proper equipment.
A second alternative would be to determine which capabilities can be handled by both devices and limit the applications to these. This is a waste of resources, since tremendous capability of the display may lie unused, limited by the incompatible system. Another alternative would be to adjust the display so that it appears exactly as it would in hardcopy. If this were done, the designer could adjust the picture so that balance and form were achieved and could be duplicated. This may remove some of the portability of the standard, however, since the specific output devices then must be taken into account. Finally, the user could just live with the system as it is, realizing that the hardcopy may not reproduce the display to an accurate degree. This alternative is frustrating to the user and may result in duplication of effort since elements in the picture may need to be redone.

In fact, GKS has adapted to this problem by making adjustments within the standard. Text has several different indices, which correspond to differing qualities and capabilities of the text and the devices. Because producing higher quality text also takes more time, the index can be used to reduce the time as well as the quality where time is an important factor. The text is then under the control of the designer, who can either display high-quality text or lower quality text according to his own estimation and the requirements of the application.
Again, it is important to realize that a standardized, portable method of addressing these problems must be developed for any proposed graphics standard. Whether the solutions utilized in GKS are implemented in any given standard is not a requirement, yet some form of standardized handling must be made available to the user.

Rick made a special effort to discuss text with Sherry. Like most operators unacquainted with graphics displays, Sherry assumed text was relatively easy. She merely included the text she wanted on the graphs she gave to Rick. When he asked her how large the letters should be, what font, what color, and what angle, she was shocked. He explained about text to her then, and they decided on fine-quality, stroke precision text, black in color, not angled. Once again, Sherry was highly pleased with the results she got, but resolved to make more creative use in the future of the text available to her.

Workstations.

The concept of workstations is a secondary, but useful part of graphics standards. A workstation is defined as a group of devices capable of graphics input and output with no more than one graphics display device.[9, p.97] If an application or system is composed of two real display devices, the system can be described in several workstations. This concept prevents confusion concerning which input device is to be read and which output device the
output is to be plotted on. Three types of workstations are possible, those that do input only, those that do output only, and those capable of both within the limitation of only one graphics display device.

Because of the application Sherry was involved in Rick and Karl organized two workstations for her. One involved the terminal Sherry would be working on, including the monitor, keyboard, and graphics tablet. That workstation was defined as an 'outin' workstation because it would be performing both input and output. The other workstation was defined as the large screen television. It was defined as an output workstation because its only function was to perform output. These two stations needed to be defined individually because of the limitation of only one graphics display device per workstation. The same graphics could be displayed to both workstations, but needed to be programmed to display them on both workstations.

Language Bindings.

The use of language bindings specifies the names of the actual subroutines which should be called to perform a certain function. This is a further method of standardization so that programs can be transported from one GKS site to another and run without adaptation. The names of the subroutines and functions are prescribed by the ISO and can be found in a manual on GKS. The language bindings differ from language to language, so the proper language
binding for the application must be obtained before successful work on a system can be completed. A thorough graphics standard will supply language bindings for each widely-used language and make them available for installations to purchase.

Sherry notified Karl of the languages available which could work with GKS. After examining the choices, Karl selected FORTRAN, because it was available on their system, he was familiar with it, and function calls could be easily made in that language. In addition, Sherry could obtain the FORTRAN language binding from the sponsors of the workshop she had attended. After supplying the language binding to Karl, the function calls could be properly implemented and the proposal thoroughly tested.

Sherry finally had completed her presentation of the proposal. Several of the managers who had attended had complemented her on the style and information conveyed in it. She was thankful to Rick and Karl and happy that they could work so well using GKS.


d\textbf{PROJECTIONS}


Although no one can predict the future, one can safely state that computer graphics standards will be there. To date, in their short life, computer graphics standards have fulfilled the need of utility and portability in computer graphics. Changes have occurred in the development of
hardware to conform to needed standards, and software has recognized the need for a standard, universally accepted method of surmounting problems. Because computer graphics standards have met this need so well, they will remain.

As with any area related to the rapid-growth-and-change technology fields, computer graphics standards will continue to change. New realms of technology need to be implemented, solutions to new problems remain to be found, and new applications need to be accounted for. So, change, along standardized lines, will remain part of computer graphics standards. This fact was taken into account with GKS, since it was intended to be the father of a family of graphics standards which will continually evolve to meet changing needs.

The elements and definitions present today will continue as part of the computer graphics standards, and may become 'standard' features of all graphics standards. This idea is conveyed by the statement below:

Both users and industry are coming to the conclusion that if a standard exists that is shown to address the requirements of the industry, it should be used as the basis for further expansion.[6, p.65]

An example of such building is currently occurring in a project called Unix United, which is "...extending GKS to include high-level object-oriented facilities, which are more appropriate in applications where the graphical entities are modeling the physical world."[7, p.23] Later generations will build on this work and improve some of the
methods currently employed. Yet this work will remain important as the foundation of what has resulted.

Most likely one leader will emerge as the most accepted graphics standard. Because of its international status, GKS is a strong possibility, pending continuous updates and approval of these updates by the International Standards Organization. In the end, the future remains to be built and seen. That is the interesting element of the future. Be confident, though, that whatever happens, computer graphics standards will have a part in the presentation and recording of those events.

SUMMARY

Computer Graphics Standards.

Although computer graphics standards are a recent development, they have far-reaching effects on the computer field and on every other field concerned with the presentation of data. Nearly everyone can benefit from the timely and understandable reporting of information. The use of computers makes timely and accurate data possible, and the use of computer graphics aided in quality by graphics standards makes the presentation and reporting of data attractive, readable, and understandable. The applications in art and film careers should also not be overlooked.

Yes, computer graphics has a welcome place in this technological world. With the development of GKS as an international graphics standard, the world has come another
step in being able to communicate, work, and grow together. Although work already completed in computer graphics standards will become accepted as a 'standard' part of future systems, the field will continue to grow and develop in utility, dependability, and portability, helping us all to build on one another's work and progress into the future.

AN IMPLEMENTATION OF THE GKS GRAPHICS STANDARD
A Project In GKS.

The following materials are the result of the development of a tutorial on a currently developing GKS system. The system is being developed on an Intel 310 minicomputer with keyboard and graphics tablet input and an RGB graphics monitor output. The implementation is developed in the C computer language and addresses the major parts of a GKS graphics system. The books on C were of great aid in developing the programs for the tutorial. Also, the manual on the GKS language bindings for C gave guidance to what functions should be called. NOTICE: Copies of the ANSI GKS technical description can be acquired by sending $35.00, prepaid and with a self-addressed mailing label to:

X3 Secretariat/CBEMA
311 First Street, N. W.
Suite 500
Washington DC 20001
The Tutorial Project

Over two quarters, I have maintained an attempt to accomplish something difficult - To write a tutorial and make it understandable to the common user for a system which hasn’t been written yet. To compensate for this we are using the GKS standards, which helps very much. I have tried to conform to those standards in this tutorial so that the information presented in it will not be in error when the system finally is written. It is my hope that, in the end, the work so far will be completely usable for the tutorial.

The overall concept was to present a friendly and understandable environment for the user, giving him as much control over the pace and order of presentation as possible. To this end an introductory and a closing set of text files is printed during the tutorial. For each topic on the main menu there is a set of subtopics, which is observed by sending the users to a submenu. On each submenu, then, is the selection of subtopics related to the primary topic. As each of these subtopics is selected, an example is drawn on the graphics screen while explanatory text is presented on the terminal monitor. It was our hope from the beginning to then send the user to a practice area, where he can practice any commands recently learned. This has not been implemented to date, mainly because I have not written a parser before and ran into deep problems. I would like the opportunity to really try to write one sometime, but although I now understand much better the problems in writing one, I could not accomplish it in the time I had. Perhaps those who follow will have the inclination and the experience to add this step to the tutorial. I feel it would enhance the learning a great deal.

By looking at the source codes contained in the tutorial you will notice the *.c files are not documented. These are only intended to ensure the information was passing between the data files, as they will be rewritten when
the system is finally implemented. I'm sure the object and purpose of those files is readily obvious, and so did not include full documentation on those files that will (hopefully!) be soon thrown out.

You may also notice the interesting presence of getchar() function calls in the menu programs without any seeming purpose. They have been placed there because the display of the text files was thrown off. After many hours of searching, I discovered the following problem:

In order to get the menu to accept the selection (some integer) a return had to be entered to send the buffer contents to the CPU. The scanf function, however, only extracted the integer, leaving the return in the buffer. When the time came to display the text files, two screens were printing instead of one because this stray return was then extracted from the buffer, accepted as input that the user was ready to continue, and the display continued on. This use of the getchar() was then necessary to extract the stray return from the buffer so the proper display of the text files could be done.

Other problems include the time-consuming task of setting oriented to the system. To aid in future work, I have included a page of useful commands which I have encountered. Hopefully it will reduce the amount of time lost in setting oriented.
Up to 12/19/85:

Come in to try to get all previous code to work together, proper calls, etc. Began with many problems—couldn't find Vince's programs, compiler and linker don't work, and my code was faulty and now has syntax errors. I don't remember if it did before, but I thought I would take out the syntax errors! At end, fixed compiler, reduced linker to some obscure warning about unresolved symbols, and successfully compiled three of my old routines. I didn't get a chance to try them together, but I do understand the system a lot better (at least as good as last Spring), and got the compiler/linker working again. I hope to be able to spend more time on this per week after Christmas.

01/09/86:  in-7:05  out-8:35

Succeeded in loading Vince's programs after finding out from Dr. McGrew where they were. Erred last time I talked to Dr. McGrew, told him I knew where lib, lib was. It turns out it isn't anywhere, so I still can't link the programs. I'm leaving him a note and planning on coming back in tomorrow.

01/24/86:  in-1:00  out-3:00

Spent last week and a half trying to get output on the terminal. Dr. McGrew helped to resolve that today, apparently something was wrong with the link file. All syntax errors are gone on the files, and now that I can see output on the screen, hopefully I can begin new work very soon.

02/01/86:  in-1:50  out-5:20

Yay! All programs debugged again. Beginning merger of Vince's and my programs. It will take more work than I thought to do that. The format and logic isn't as close as I had assumed it would be. Changed his readfile routine to display one screen at a time, began merger. For some reason I can't get his readfile to work with my main program. The next step is to solve this dilemma. All in all, things are starting to come together—finally.

02/02/86:  in-5:10  out-7:00

Main and readfile modules are almost done. For some reason two enters are necessary for the first menu selection. Still trying to figure that one out.
02/06/86: in-9:25 out-11:30

Looking good. Still having to enter twice when returning to menu from readfile routine, but with Dr. McGrew’s help we don’t have to give the number of lines as a parameter to the readfile routine any more. Now working on second level of tree, almost complete. Will probably have all this code along with a practice area done by the end of the quarter...sure hope so, things seem to go so slow sometimes.

02/06/86: in-7:40 out-9:45

All second-level routines are now attached, began solving double-entry problem. Now the menus print fine, but it isn’t stopping after first screen on data files. Might be something with the flag I’m using. Oh, well, I’ll solve it one way or another, given time.

02/07/86: in-1:07 out-2:30

Still can’t figure out why I need to enter twice sometimes and not others. Right now I seem to have gone backwards! Quitting early in the lab, going to look the code over in my room.

02/25/86: in-4:00 out-4:30

Still working on the same problem after another week and more! Talked to Dr. McGrew, but no new ideas. I’m going to break this if it’s all I do!

02/25/86: in-7:30pm out-2:40am

EUREKA! silly of sillies! (Don’t you always feel that way when you FINALLY figure out what was wrong?) All this time I’ve been playing with the readfile function, and it was fine all along (almost had to be, I’ve changed every line of code in the module). Turns out that with the scanf function, the integer entered is removed from the buffer, but the return (which is required to dump the buffer to the CPU) is left in. Then, when the readfile function performs a getchar, this latent return is accepted as user input to go on, so it does. Repaired this problem by entering a getchar after each scanf in all the menu routines to remove the stray return. After fixing that problem I proceeded to complete all modules associated with the polyline. At this time, the main menu and all polyline modules are signed-off, complete, and function properly. Lookin good again, but oh I’m gonna crowd finishing the practice area now.

02/26/86: in-1:50 out-4:30

Have completed and printed all modules pertaining to the polymarker, now moving into text.
02/27/86: in-8:30am out-12:00pm

Completed text modules. Created a src directory to fit within the exetut.dir and to hold the source codes.

02/27/86: in-1:40pm out-4:20pm

All modules completed, proofread. Took 15 minutes to backup everything onto floppy! Ready to print final copies and to create final report.

02/27/86: in-6:40pm out-9:30pm

Printed final copies of everything. Created obj directory to hold object codes for tutorial, it also is a subdirectory to exetut.dir. Typed final reports into Intel and printed them off! Due to see Dr. McGrew tomorrow at 1:00 to hand it all in. It been real!

NOTE: This Journal is not all-inclusive and only contains the highlights of the quarter. Not all the time spent was accounted for in these pages. Therefore,

Total estimated time spent Winter 1985-86: 72 hours
Useful System Information

To `log in': turn on interface. After a few seconds a row of asterisks will begin printing on the screen. Immediately type shift-u to interrupt this process, and again immediately type ctrl-c to drop to rt-11 monitor mode (a period prompt).

Now enter .b /boot/vdisys and wait for the system to boot, ending in a dash prompt.

Super mode:

Because of the protection settings in the computer, you must now enter Super mode in order to accomplish very much. To do this, type super

It will return by asking for the password, which you should obtain from your mentor. When the process is properly completed, you will have a super prompt (super-).

Graphics Devices: The next step is to attach the graphics board and monitor. This is done by typing in

    super- adv il86vdi as :vdi: -

    (attachdevice devicename as :logical: physical)

Working Directory:

If you’ve made it this far, the hard parts over. Now you can begin work after entering your working directory. In order to access the work done on the tutorial, you must

    super- af ^/world/exetut.dir as :<logical>:

    exetut.dir is the name of the directory containing all the tutorial work. Inside you will find the `tutor' file, which is the tutorial. To run it merely type:

    super- tutor

    You will also find src and obj. These directories contain the source files and object files, respectively, for the tutorial. All data files and other files necessary during execution are kept in exetut.dir.

You may feel free to create your own directories as subdirectories to the world directory. Please stay within this limitation.

The actual pathname of the exetut.dir is /user/world/exetut.dir. The first slash denotes the root node, then the other nodes follow, separated by slashes.
C language:
You will notice within exetut.dir several .csd files. These are related to .com files on the VAX, and are executable. The files contained within exetut.dir are for use with the C language. CC.CSD is a file in the src directory which invokes the C86 compiler. To use this file, type:

```
super-submit cc(file)
```

NOTE: the file MUST have a .c extension.

Typing `super-submit tlink` while in the obj directory will link all the tutorial files together, producing an executable file called tutor in the exetut.dir directory which is the tutorial. If you wish to try creating a self-sufficient C program, you may use `super-submit tlink(file)` to link your program.
link86 /lib/cc86/ssmain.obj, &
main.obj, readfile.obj, polyline.obj, &
polymrkr.obj, text.obj, wsmenu.obj, &
plinex.obj, spolyl.obj, plinecolor.obj, gsetlcol.obj, &
plinetype.obj, gsetltype.obj, lindex.obj, gsetlindex.obj, &
prmrkrex.obj, spolym.obj, prmrkrcol.obj, gsetmcol.obj, &
prmrkrttype.obj, gsetmtype.obj, prmrkrindex.obj, gsetmindex.obj, &
textex.obj, gtext.obj, tefont.obj, gsetfntpec.obj, &
texcolor.obj, gsetxcol.obj, textelt.obj, gsetchelt.obj, &
txelcmt.obj, gsetxalign.obj, txindex.obj, gsetxindex.obj, &
sclearws.obj, &
/user/vdi/test/lib/vdapp1c.lib, &
/user/vdi/test/lib/vdi720c.lib, &
/lib/cc86/sclib.lib, &
/lib/cc86/87null.lib, &
/rmx86/lib/small.lib to ^/tutor bind map &
sessize(stack(+2048), memory(5000)) mempool(+6000h)
Enclosed on this page are the more common and useful commands I utilized on the Intel GKS project. Those who follow after will probably find them of much use, so I gathered them together here for that purpose. (I knew you all were wondering why I called you here!).

- copy file.ext
  displays file.ext on the screen
- copy file.ext to :lp:
  prints file.ext on printer
- copy file.ext over path/file.ext
  replaces old copy of file.ext in path with new version without a check inquiry.
- af path/dir as :logical:
  assign logical name :logical: to the given path.
- adv umfdx0 as :logical:
  attach the floppy drive with logical name :logical:
- detachdevice :logical:
  detaches device of logical name :logical: NOTE: the floppy drive must be detached before removing the floppy disk!

:-home: is the directory you boot into

:$/ is the current directory

You may assign logical names as you wish. To see the logical names already present in the system type -LOGICALNAMES


- tx file.ext
  enters the text editor to create or edit file.ext.
main()
{
    int int_response, int_sleep, int_i;
    
    fcn_readfile("greeting.dat");
    fcn_readfile("menu.dat");
    scanf("%d", &int_response);
    getchar();
    while (int_response != 7) {
        switch (int_response) {
            case 1: fcn_readfile("cursor.dat"); break;
            case 2: fcn_polylnmenu(); break;
            case 3: fcn_polylnmenu(); break;
            case 4: fcn_textmenu(); break;
            case 5: fcn_wsmenu(); break;
            case 6: fcn_readfile("command.sum"); break;
            default: printf("\nPlease enter a valid selection (1-7) \n");
                for (int_sleep = 0; int_sleep <= 800; int_sleep++) int_i += 1;
            }
            fcn_readfile("menu.dat");
            scanf("%d", &int_response);
            getchar();
        }
    fcn_readfile("goodbye.dat");
}
Hello, and welcome to the * * * * * * graphics package!

Of course, first we need to know what GKS is in order to know how good it is being here, so...

GKS stands for Graphics Kernel System, which is one of the first standardized graphics systems in existence. This tutorial is designed to introduce you to the commands you may use in this student-created GKS environment on the Intel Microcomputer. type <ret> to continue

Let's first talk about the tutorial itself.

To begin, the tutorial is menu driven for easy use.

This allows you to control the pace and to go back and look at something again. Also, you always know where you are, so you can exit the tutorial easily and find your place again when you come back.

<return> to continue

Second, a consistent pattern of explanation and demonstration is followed so that examples can be displayed on the graphics screen while explanations appear on your monitor.

This enables us to explain and demonstrate at the same time. Also, you will notice that as explanations fill the screen, the display will stop and wait for you to hit return. This allows you to control the pace and to take your time learning the separate lessons.
Now that you understand how to use the tutorial, let's start learning GKS!

Here we go......

<return> to continue
Please enter the number corresponding to your choice:

1) The Cursor
2) Polyline Menu
3) Polymarker Menu
4) Text Menu
5) Work Station Menu
6) Summary of Commands
7) Exit the tutorial

Your choice?
The cursor used in the tutorial is a typical underscore cursor '_'. This marks the place where the computer's attention is and usually is seen and noticed when the computer is expecting a response from the user (you). When you are entering commands or data, it marks where you currently are in the line, and which character will be deleted when the delete key is struck, in this case the character just ahead of the cursor will be deleted. In some applications of this GKS package, a crosshair cursor may be used on the graphics screen. This cursor looks like a huge plus '+' sign and denotes the location of activity on the screen.
That's all folks!

Hope you enjoyed learning GKS in this tutorial.

Have a GRReat DAY!!
**SUMMARY OF COMMANDS**

- `spolyline(n_points,points)`: int, `swpoint(x,y)`
- `ssetlinecolour(color)`: int
- `ssetlinetype(type)`: int
- `ssetlineindex(index)`: int

- `spolymarker(n_points,points)`: int, `swpoint(x,y)`
- `ssetmarkercolour(color)`: int
- `ssetmarkertype(type)`: int
- `ssetmarkerindex(index)`: int

- `settext(position,string)`: `swpoint(x,y)`, char array
- `ssettextfontprec(fp)`: `fp(font char; prec char)`
- `ssettextcolour(color)`: int
- `ssettextheight(height)`: float
- `ssettextindex(index)`: int
- `ssettextalign(align)`: align(hor,ver) enumerated:
  hor: TH_NORMAL, TH_LEFT, TH_CENTRE, TH_RIGHT
  ver: TV_NORMAL, TV_TOP, TV_CAP, TV_HALF, TV_BASE, TV_BOTTO

- `sopenws(ws_no)`: int
- `sclose(ws_no)`: int

<return> to continue
/* Name: READFILE FUNCTION */

/* AUTHORS: Ken Mikeworth & Vince Pedraza */

/* DATE WRITTEN: JANUARY 9, 1986 */

/* SIGN-OFF DATE: FEBRUARY 25, 1986 */

/* PURPOSE: The purpose of this function is to read any file */
/* of data containing each primitive. If the file */
/* does not exist, or if the file cannot be opened, */
/* then this function displays a message telling the */
/* user that the file was not opened. */

/* INPUTS: The input to this procedure is the name of the */
/* file to be opened. */

/* GENERAL DESCRIPTION: This function displays the lines that were */
/* read from the data file and prints them onto the */
/* screen. */

/* CALLING FUNCTIONS: Every module. */

/* MODULES CALLED: NONE. */

/* VDI FUNCTIONS CALLED: none. */

*************************************************************************

#include <stdio.h>
#include <ctype.h>
define BUFSIZE 81
define void int

void fcn_readfile(CHR_CHOICE)
char CHR_CHOICE[];
/* THE NAME OF THE FILE TO OPEN */
{
    int int_cnt; /* SCREEN COUNTER */
    int int_cont;
    char chr_inbuff[BUFSIZE];
    char chr_ch=' kir
    FILE *ptr_infile; /* A POINTER TO FILES */

    if ((ptr_infile = fopen(CHR_CHOICE,"r")) == 0)
        printf("CANNOT OPEN %. AS INPUT FILE.

    else
    {
        for (int_cnt=1; int_cnt<=5; int_cnt++) putchar("\n");
        while (chr_ch!=NULL)
        {
            for (int_cnt=1; int_cnt<=24; int_cnt++)
                chr_ch=fgets(chr_inbuff, BUFSIZE, ptr_infile); /*GET A LINE OF INPUT */
            fputs(chr_inbuff, stdout); /* AND DISPLAY IT */
        }
    }
}

if (chr_ch!=NULL) int_cont=getchar();
)
fclose(ptr_infile); /* CLOSE THE FILE BEFORE RETURNING */
)
**Name:** Polyl ine Menu Module

**Author:** Ken Mikeworth

**Date Written:** February 6, 1986

**Sign-Off Date:** February 25, 1986

**Purpose/Description:** To display a submenu of selections for the polyline. The student then selects which option is desired. An appropriate submodule is then called. Upon it's return the cycle is repeated as desired by the user.

**Calling modules:** Main module.

**Modules called:** fcn_readfile to display text, fcn_plinex as an overall example of a polyline, fcn_plinecolor to demonstrate and explain color, fcn_plinetype to explain different line types and fcn_plindex to explain polyline index control.

**VDI Functions called:** none.

```
#include <stdio.h>
#include <ctype.h>
#define void int
#define BUFSIZE 81

void fcn_polyline() {
    int int_response, int_sleep, int_i;

    fcn_readfile("linemenu.dat");
    scanf("%d", &int_response);
    getchar();
    while (int_response != 6) {
        switch (int_response) {
            case 1: fcn_readfile("linesen.dat"); break;
            case 2: fcn_plinex(); break;
            case 3: fcn_plinecolor(); break;
            case 4: fcn_plinetype(); break;
            case 5: fcn_plindex(); break;
            default: printf("\n\n Please enter a valid selection (1-6) \n");
                    for (int_sleep = 0; int_sleep <= 800; int_sleep++) int_i=1*1;
    }
    fcn_readfile("linemenu.dat");
    scanf("%d", &int_response);
    getchar();
}
```
Please enter the number corresponding to your choice:

1) General Description
2) Polyline Example
3) Polyline Color
4) Polyline Line Type
5) Setting Polyline Index
6) Exit this menu

Your choice?
To picture how a polyline looks, mentally draw a graph with several points spread out along it. Now connect those points successively with line segments, without picking up your ‘pencil’. What you are now picturing is a polyline.

A more precise description is — A set of line segments connecting successive points on a graph. When complete it will look like a jagged line as if drawing the outline of mountains. This is a benefit in graphing, because points can be found much more easily and quickly with polylines and one can step back and determine the overall trends related by the graph. With only points plotted, this is more difficult. These benefits increase comprehension of the graph and what it is telling you.

<return> to continue

In terms of graphics, you can enter several predetermined points say six or so, and have a drawing of a hexagon. With three you can form any shape of triangle, and so on.

Now, let's look at some examples of polylines to see how this works....
Name: Polyline Example ("plinx.c")

Author: Ken Mikeworth

Date Written: February 25, 1986

Sign-Off Date: February 25, 1986

Purpose/Description: To provide an example to the tutorial viewer of a
polyline and how to create one.

I/O Data: The number of points and the array of points to be graphed are
passed to the called GKS function.

General Description: The proper variables are initialized, then after
properly describing what is being done, the absolute values are
set up and the GKS function to draw the polyline to be shown is
called.

Calling Functions: fcn_polinmenu /* menu display/control */

Modules Called: sclearws /* clear the graphics monitor */
spolyline /* generate the polyline */

VDI Functions Called: None.

#include <stdio.h>
#include <ctype.h>
#include <gks.h>
/* GKS type definitions and constants */

#define void int
#define GMONITOR 1
#define ALWAYS 1
#define N_POINTS 3

void fcn_plinx()
{
    GwPoint points[N_POINTS]; /* array of vertices of polyline */
sclearws (GMONITOR, ALWAYS); /* clear graphics monitor */
    Points[0].x=0;
    Points[0].y=0;
    Points[1].x=100;
    Points[1].y=100;
    Points[2].x=150;
    Points[2].y=50;
    spolyline(N_POINTS, Points); /* plot polyline */
    fcn_readfile("plinx.dat"); /* display explanation on console */
#include <stdio.h>
#include <ctype.h>
typedef struct {
    int x;
    int y;
} sswPoint;

void spolyline(n_points, Points)
    int n_points;
    sswPoint Points[50];
{
    int int_cnt;

    printf("Polyline plotted to points (in order) ");
    for (int_cnt=1; int_cnt<=n_points; int_cnt++)
        printf(" %d,%d ",Points[int_cnt].x,Points[int_cnt].y);
    printf("\n");
}
Now we see an example of the polyline, with the color, line width, and line type at their default settings. Look at the polyline shown on the graphics screen. This line was drawn using the polyline command as follows:

GKS> spolyline(n_points,points)

where n_points is the number of points specified to plot and points is an array of coordinates containing the endpoints of the segments.

<return> to continue
* Name: Polyline Color Example
* Author: Ken Mikeworth
* Date Written: February 25, 1986
* Sign-Off Date: February 25, 1986
* Purpose/Description: To give an example of how to set the color for the
  polyline function.
* I/O Data: The color parameters are passed to the called routine,
  gsetlinecolour.
* General Description: After initializing the variables, the absolute
  data points to be used in the example are set, then the line
  is drawn, and finally an explanation to the student is
  displayed on the screen.
* Calling Modules: The Polyline menu module, fcn_polylmenu.
* Modules Called: The line color set module, gsetlinecolour and the Polyline
  module, gpolyline.
* VDI Functions Called: None.

#include <stdio.h>
#include <ctype.h>
define void int

void fcn_Plinecolor()
{
    int color,n_points;
    typedef struct {
        int x;
        int y;
    } gwpoint;

    gwPoint points[50];

    color=2;
    n_points=3;
    points[1].x=10;
    points[1].y=10;
    points[2].x=200;
    points[2].y=40;
    points[3].x=150;
    points[3].y=120;
    gsetlinecolour(color);
    gpolyline(n_points,points);
    color=3;
    n_points=3;
    points[1].x=10;
    points[1].y=40;
    points[2].x=200;
    points[2].y=150;
    points[3].x=120;
    points[3].y=150;
    gsetlinecolour(color);
    gpolyline(n_points,points);
    fcn_readfile("Plinecolor.dat");
}
#include <stdio.h>
#include <ctype.h>

void setlinecolour(color)
    int color;
    {
        printf("Polyline color set to color denoted by index number \%d\n",color);
    }