DEC GKS

User's Guide

Order Number: AA–PQP3A–TE

June 1992

This document is an instructional manual, supplementary to the DEC GKS™ binding manuals, and contains information for both the novice and the moderately experienced GKS programmer.

Revision/Update Information: This is a new manual.

Digital Equipment Corporation
Maynard, Massachusetts

First Printing June 1992

ZK5675

This manual is available on CDROM.

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

Tables

Preface

This document is instructional, is supplementary to the DEC GKS binding manuals, and contains information for both the novice and the moderately experienced DEC GKS programmer. Because the focus of this book is programming technique as opposed to complete product description, you may wish to review the introductory section of each chapter in the DEC GKS binding manuals as you read this manual.

Note

Before reading this manual, you should review the DEC GKS release notes on VMS™ systems by typing the following:

```
$ HELP GKS RELEASE_NOTES RETURN
```

To read the release notes on ULTRIX™ systems, type the following:

```
# more /usr/lib/GKS/gks_500.release_notes RETURN
```

Intended Audience

This manual is intended for both novice and moderately experienced application programmers who need information supplementary to the DEC GKS binding manuals. Readers should be familiar with one high-level language and the DIGITAL Command Language (DCL).

Document Structure

This manual contains the following components:

- Chapter 1, Introducing DEC GKS, provides a brief introduction to the GKS–3D standard, and a general overview of DEC GKS.
- Chapter 2, Getting Started with DEC GKS, introduces basic DEC GKS programming techniques.
- Chapter 3, Composing and Transforming Pictures, provides information concerning the DEC GKS coordinate systems, picture composition, and zooming in and out of a picture.
- Chapter 4, Customizing Output, provides information concerning DEC GKS output primitives, individual and bundled attributes, aspect source flags, segment formation, segment transformation, segment clipping, segment attributes, surface regeneration, and output deferral.
- Chapter 5, Data Input, provides information concerning the logical input device classes, normalization viewport priority, input data records, and synchronous input, as well as information concerning input process

documentation, simultaneously active input devices, and asynchronous input.

- Chapter 6, Writing Device-Independent and System-Independent Programs, introduces the method of using inquiry functions to write device-independent programs.
- Chapter 7, Storing the Picture in a Metafile, provides information about metafiles and how to work with them.
- Appendix A, Color Models, provides information about the color models supported by DEC GKS.
- The glossary provides definitions for DEC GKS terminology.

Associated Documents

You may find the following documents useful when using DEC GKS:

- *DEC GKS FORTRAN Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the FORTRAN binding.
- *DEC GKS GKS$ Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the GKS$ binding.
- *DEC GKS GKS3D$ Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the GKS3D$ binding.
- *DEC GKS C Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the C binding.
- *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*—For programmers who need information about specific devices.
- *Building a Device Handler System for DEC GKS and DEC PHIGS*—For programmers who need to provide support for a device unsupported by the DEC GKS graphics handlers.
- *Installing DEC GKS for VMS*—For system managers who install the DEC GKS software, including the run-time installation, on VMS operating systems.
- *Installing DEC GKS for ULTRIX on RISC Processors*—For system managers who install the DEC GKS software on ULTRIX operating systems running on RISC processors.

Conventions

The following conventions are used in this manual:

Convention	Meaning
$\boxed{\text{RETURN}}$	The symbol $\boxed{\text{RETURN}}$ represents a single stroke of the Return key on a terminal.
bold	Boldface type is used to introduce a new term. In interactive examples, user input appears in boldface type.
INTEGER X . . .	A vertical ellipsis indicates that not all of the text of a program or program output is illustrated. Only relevant material is shown in the example.
option, . . .	A horizontal ellipsis indicates that additional arguments, options, or values can be entered. A comma that precedes the ellipsis indicates that successive items must be separated by commas.
[output-source, . . .]	Square brackets, in function synopses and a few other contexts, indicate that a syntactic element is optional.
deferral mode	All names of the DEC GKS description table and state list entries, and of the workstation description table and state list entries, are italicized.

1 Introducing DEC GKS

DEC GKS Version 5.0 combines, and is a superset of, DEC GKS Version 4.2 and DEC GKS–3D™ Version 1.2. In this manual, DEC GKS refers to Version 5.0 or higher. DEC GKS is a run-time library (VMS) or an object library (ULTRIX) of graphic functions included in application programs. DEC GKS complies with the international standard for three-dimensional graphics (GKS–3D International Standard (ISO 8805(E) 1988)), and is an upwardly compatible extension to the ISO GKS standard (ISO 7942-1985). In this manual, references to the GKS standard apply to the GKS–3D ISO standard. DEC GKS is complementary to the PHIGS graphics system; GKS is suitable for unstructured, fixed images, while PHIGS is used for structured, editable graphics data.

This chapter provides a general overview of DEC GKS concepts. The remaining chapters show you how to apply the DEC GKS concepts presented in this chapter to actual application programs, including the following topics:

- Basic programming techniques
- Transformations and picture composition
- Generating output
- Accepting input
- Device-independent programming

This manual defines DEC GKS terminology as needed to describe examples and programming technique. Some descriptions contain only enough information to understand the topic. After reading a chapter in this manual, you may wish to review the corresponding information in your binding manual.

1.1 DEC GKS

DEC GKS is a set of functions that provides application programs with a standard method of producing graphics on a potentially large number of **physical devices** (such as workstations, terminal screens, pen plotters, or graphics printers). By using DEC GKS, you do not need to be concerned with the system-specific or device-specific requirements for producing graphic images.

1.1.1 The GKS–3D Standard

The GKS–3D standard specifies both a functional standard and a syntactical standard for three-dimensional GKS routines. A functional standard specifies the task that a function must perform, but does not impose a standard function name or syntax. A syntactical standard specifies the function name, syntax, and task.

DEC GKS implements the functional standards as a group of run-time functions. These functions perform tasks as required by the functional GKS standard. The functions are supplied in four bindings. The FORTRAN and C bindings are portable to non-Digital architectures. Use the GKS3D$ or GKS$ bindings if you plan to run your applications only on machines that have the Digital architecture.

DEC GKS also implements the syntactical standard (the FORTRAN binding) as a group of functions for use only in FORTRAN application programs. The FORTRAN and C bindings offer a complete set of GKS functions with standard identifier and parameter names for each of the binding functions. Using the FORTRAN and C binding functions, you can transport your programs from one GKS implementation to another. The FORTRAN binding functions all begin with the uppercase letter G. The C binding functions all begin with the lowercase letter g. For complete information concerning the FORTRAN binding, see the *DEC GKS FORTRAN Binding Reference Manual*. For complete information concerning the C binding, see the *DEC GKS C Binding Reference Manual*.

If you are programming using a DEC GKS supported language other than FORTRAN or C, you must use the GKS3D$ or GKS$ functions. See the *DEC GKS GKS3D$ Binding Reference Manual* and the *DEC GKS GKS$ Binding Reference Manual* for more information concerning these bindings.

1.1.2 DEC GKS Software Components

DEC GKS is made up of logical software layers separated by interfaces. The software components and the relationships between them are illustrated in Figure 1–1.

The **workstation-independent** layer includes all the GKS level 2c functionality and is independent of the hardware being used. (See Section 1.1.6 for more information on GKS levels.) It contains the following binding interfaces:

- C language binding
- FORTRAN language binding
- GKS$ binding
- GKS3D$ binding

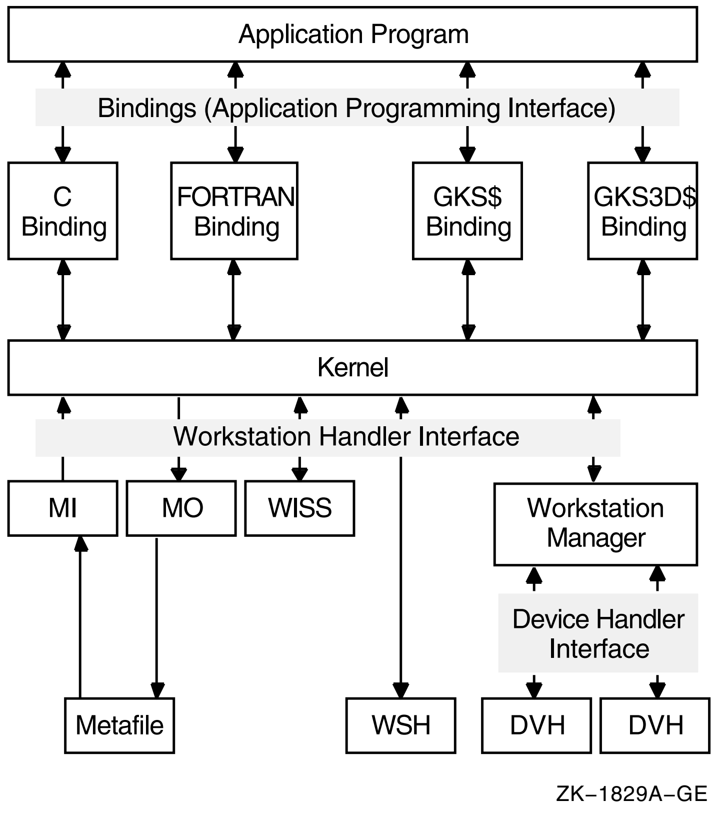
The C and FORTRAN bindings provide the application program with an implementation-independent GKS interface that allows application portability. The GKS$ binding and GKS3D$ binding are Digital proprietary bindings that are defined according to the VMS calling standard and support all VMS compilers. Both the GKS$ and GKS3D$ bindings support programs for which no language binding exists. Applications using the GKS$ and GKS3D$ bindings are not portable. See Table 6–2 for more information on binding portability.

The **kernel** is a set of workstation-independent routines that interpret the calls from the bindings. The kernel performs error checking, maintains data concerning the current state of the system, and calls the workstation modules for workstation-dependent and device-dependent requests.

The **workstation-dependent** layers of DEC GKS consist of the modules associated with the following logical workstation handlers:

- Metafile input (MI)
- Metafile output (MO)

Figure 1–1 DEC GKS Software Components



- Workstation independent segment storage (WISS)
- Workstation handlers (WSH) for three-dimensional devices
- Workstation manager for controlling the two-dimensional device handler

The **device-dependent** layer of DEC GKS consists of a set of **device handlers** (DVH). The device handlers direct the production of the output on a variety of devices. Some of the supported devices include:

- Motif® devices
- DECwindows™ workstations
- PostScript® devices
- Plotter devices

1.1.3 Data Structures

DEC GKS data structures consist of state lists and description tables. State lists contain modifiable information about GKS, segments, errors, or a workstation. Description tables contain read-only information about GKS or a workstation.

The kernel data structures are as follows:

- GKS operating state
- GKS description table
- GKS state list

- Segment state list
- Error state list

The workstation handler data structures are as follows:

- Workstation description table
- Workstation state list

Table 1–1 presents an overview of the DEC GKS data structures.

Table 1–1 DEC GKS Data Structures

Table or List	Description
GKS description table	This table contains information about the constants for the DEC GKS implementation you are using. It includes the level of GKS, the number and list of available workstation types, the maximum number of simultaneously open workstations, the maximum number of simultaneously active workstations, the maximum number of workstations activated with a segment, and the maximum normalization transformation number.
Workstation description table	This table contains information about the constants for one particular workstation, including the workstation type and category; device coordinate units; display space type and size; nominal, minumum, and maximum sizes for lines, markers, edges, text characters, and so on. Each graphics handler contains a workstation description table describing that particular device.
GKS operating state list	This list contains one of the following five operating states: GKCL (GKS closed), GKOP (GKS open), WSOP (workstation open), WSAC (workstation active), SGOP (segment open).
GKS state list	This list contains entries that specify the current DEC GKS values, such as the set of open workstations (if any); the set of active workstations; the current normalization transformation number; the current character height, line type, marker type, edge type, view index, segment names in use, and so on.
Workstation state list	For each workstation you open, DEC GKS creates a workstation state list. This list contains entries that specify whether output is deferred, whether the surface has to be redrawn to fulfill an output request, whether the workstation surface is “empty” as defined by GKS, whether the picture on the surface represents all of the requests for output made thus far by the application program, and so on. It also contains the tables for the defined line, marker, text, fill area, pattern, edge, and color bundles, as well as current input device information.

(continued on next page)

Table 1–1 (Cont.) DEC GKS Data Structures

Table or List	Description
Segment state list	The segment state list contains entries that specify the segment name, the set of associated workstations, the detectability, visibility, and highlighting of the segment, the segment transformation matrix, and the segment priority.
Error state list	This list contains information about the current error state and error file identifier, as well as identification of any logical input device that caused an input queue overflow.

Chapter 6 describes how to make use of the DEC GKS data structures for device-independent programming.

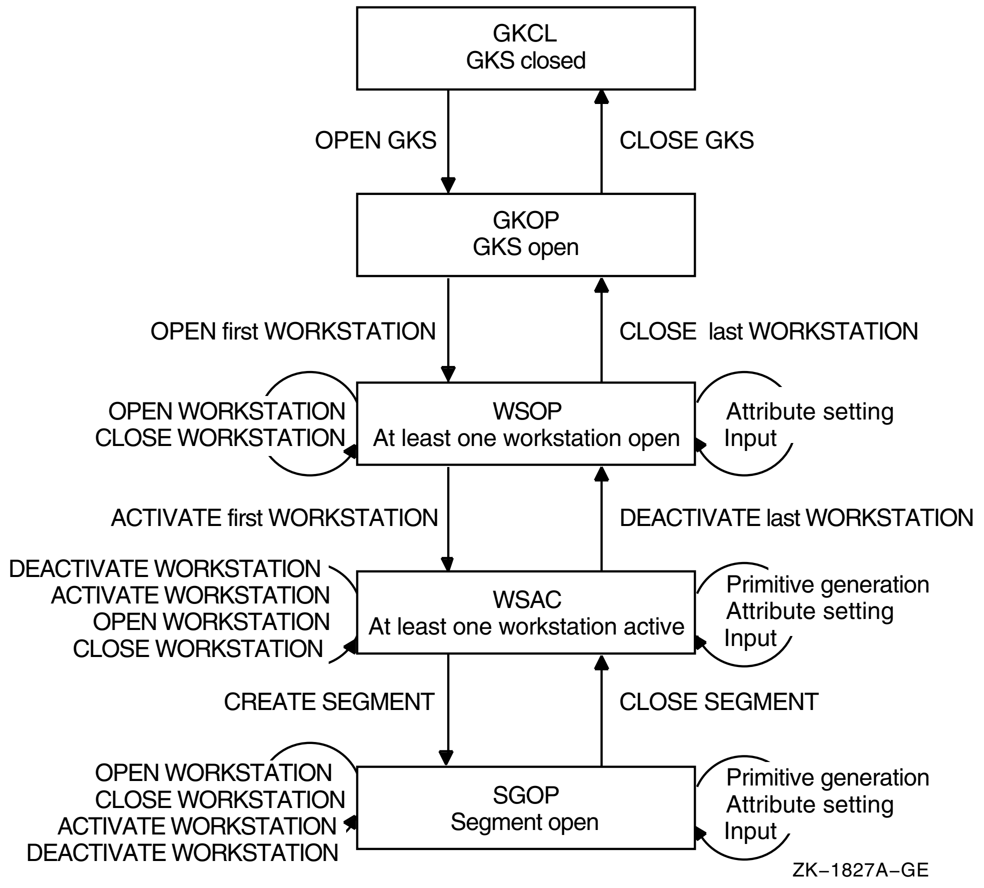
1.1.4 GKS Operating States

DEC GKS can be in any one of five **operating states**, which determine workstation activity. The operating states are listed in Table 1–2. The effects of the functions that control the operating state are summarized in Figure 1–2. The operating state is stored in the GKS state list (see Section 1.1.3).

Table 1–2 The GKS Operating States

Operating State	Description
GKCL	GKS is closed.
GKOP	GKS is open.
WSOP	One or more workstations are open.
WSAC	One or more workstations are active.
SGOP	A segment is open.

Figure 1–2 The GKS Operating States



Each function in the binding manuals lists the operating states in which the specific call can be made. For more information on the function categories, see Section 1.2.

1.1.5 DEC GKS Workstation Categories

A GKS workstation is a logical interface through which GKS supports an output device or file. The various capabilities of each physical device determine the **workstation category**. All workstations fall into one of the following categories:

Category	Type	Description
OUTPUT	Output only	Displays all primitives
INPUT	Input only	Supports at least one logical input device
OUTIN	Input and output	Combines characteristics of INPUT and OUTPUT workstations
WISS	Workstation independent segment storage	Stores segments and can copy them to other workstations

Category	Type	Description
MO	Metafile output	Stores image data in the form of a metafile
MI	Metafile input	Allows input and interpretation of metafile data

1.1.6 GKS Levels

The GKS American National Standard for Information Systems (ANSI) defines 12 levels of GKS implementation defined by input and output capabilities. The input and output levels are mutually independent. The lower output levels are subsets of the next highest output levels; likewise, the lower input levels are subsets of the next highest input levels.

Output levels are indicated in order of increasing capability by the characters m, 0, 1, and 2. The input levels are indicated in order of increasing capability by the characters a, b, and c.

The DEC GKS software is a level 2c implementation, incorporating all of the GKS output capabilities (level 2) and all input capabilities (level c). This manual uses the term DEC GKS when describing the 2c level DEC GKS product.

Figure 1–3 defines the 12 upwardly compatible levels of GKS, and outlines the functionality offered by DEC GKS.

The GKS input levels are determined by three types of input **operating modes**. The input operating modes are called **request**, **sample**, and **event mode**. In request mode, the application program waits for the user to enter input. Once the user signals the end of input, the application resumes. In sample and event modes, the application program and the input process operate asynchronously, so the user enters input as the application program continues to execute.

1.2 DEC GKS Function Categories

The DEC GKS functions are grouped into the following categories:

- Control
- Output
- Attribute
- Transformation
- Segment
- Input
- Metafile
- Inquiry
- Error-handling

The control functions open and close DEC GKS and control the activity of workstations, including the buffering of output and the regeneration of segments on the workstation surface.

Figure 1–3 Functionality by GKS Levels

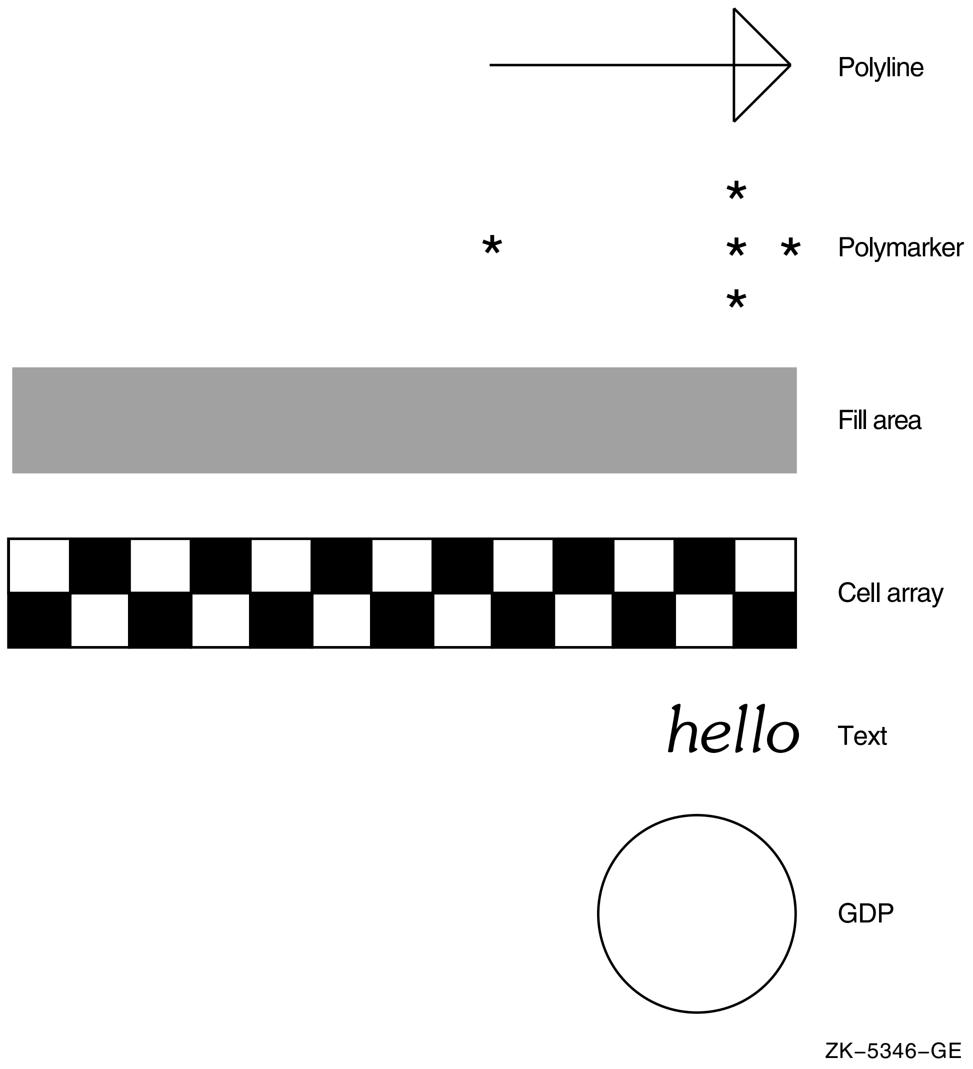
	Input Levels		
Output Levels	a	b	c
m	No input, minimal control, individual attributes, one settable normalization transformation, subset of output and attribute functions.	Request input, set operating mode and initialize functions for input devices, no pick input.	Sample and event input no pick.
0	Basic control, bundled attributes, multiple normalization transformations, all output and attribute functions, optional metafiles.	Set viewport input priority.	All of level mc, above.
1	Full output including settable bundles, multiple workstations, basic segmentation, no workstation independent segment storage, metafiles.	Request pick, set operating mode and initialize functions for pick input.	Sample and event input for pick.
2	Workstation independent segment storage	All of level 1b, above.	

ZK–5027–GE

The output functions produce seven different primitives: lines, markers or symbols, filled polygons, character strings, parallelograms of filled cells, and workstation-dependent images such as circles and ellipses.

Figure 1–4 illustrates possible output from some of the types of output primitives.

Figure 1–4 Possible DEC GKS Primitives



Attributes affect the appearance of a primitive. For example, by changing the line type attribute, you can produce solid, dashed, dotted, or dashed-dotted lines.

Transformations affect the composition of the graphic picture and the presentation of that picture. There are **normalization**, **view**, and **workstation transformations**. The normalization transformation allows you to use various coordinate ranges for different primitives within a single picture. In this way, you can use a coordinate range that suits each particular primitive in a large picture.

The view transformation allows you to tell DEC GKS from which direction you are looking at your picture, and what direction is up.

The workstation transformation controls the portion of the picture that you see on the workstation surface, and the portion of the surface used to display the picture. Using workstation transformations, you can pan across a picture, and zoom in or out of a picture.

The segment functions store and manipulate segments.

The input functions allow an application to accept input from a user.

The metafile functions allow you to store and recall an audit of calls to DEC GKS functions. Using metafiles, you can store a DEC GKS session so that another application can interpret that session, thus reproducing the picture created by the original application.

The inquiry functions return valuable default and current information about DEC GKS or about the device with which you are working. Chapter 6 describes the DEC GKS inquiry functions and how you use them to write device-independent programs.

The error-handling functions allow you to invoke a user-written error handler when a call to another DEC GKS function generates an error. For more information concerning error handling, see the DEC GKS binding manuals.

2 Getting Started with DEC GKS

This chapter provides an introduction to the following DEC GKS programming concepts:

- Using control and output functions
- Controlling workstation surface and surface regeneration
- Using segments

2.1 Using DEC GKS Control Functions

The control functions establish the DEC GKS and workstation environments, and control the workstation surface. You can start and end a GKS session by calling the following functions:

- OPEN GKS
- OPEN WORKSTATION
- ACTIVATE WORKSTATION
- DEACTIVATE WORKSTATION
- CLOSE WORKSTATION
- CLOSE GKS

You can control the workstation surface by calling the following functions:

- CLEAR WORKSTATION
- REDRAW ALL SEGMENTS ON WORKSTATION
- UPDATE WORKSTATION
- SET DEFERRAL STATE

The following sections describe how to use these control functions to start programming with DEC GKS.

2.1.1 Establishing a DEC GKS Session

The DEC GKS kernel and graphics handlers perform tasks according to values in DEC GKS internal data structures (see Table 1–1). The values in these data structures determine which DEC GKS functions you can call at a given point in your application. To perform input and output, almost all DEC GKS programs need to call a small set of DEC GKS control functions. See Example 2–1 for an example of the control functions used to establish a GKS session.

A call to OPEN GKS begins the session and establishes certain DEC GKS data structures necessary for all DEC GKS programming. The arguments to this function specify a pointer to the error log file and the number of memory units available for use. The default system error file will be used if the value 0 is passed to the first argument. The second argument is a dummy argument. It is included for compliance with the standard, but is not used by DEC GKS.

A call to OPEN WORKSTATION establishes certain DEC GKS data structures necessary for program input and output. The OPEN WORKSTATION call can be used in a device-independent manner, which permits greater program flexibility.

The first argument to OPEN WORKSTATION is a workstation identifier. You use the workstation identifier whenever you need to refer to a particular device.

In the second argument, you specify the connection identifier used to connect a device to the system. If you hard code the connection identifier, you create a device-dependent program. Use one of the following methods to avoid hard coding the connection identifier:

- Pass the value 0 as the DEC GKS connection identifier. If you use the default connection identifier constant, which is in the binding header file, DEC GKS reads either the ULTRIX environment variable (GKSconid) or the VMS logical name (GKS$CONID) to obtain the correct connection identifier for your particular device. You must set GKSconid or GKS$CONID before running your program. See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for more information on setting the environment variable or logical name for a specific device.
- Pass a string naming an ULTRIX environment variable or a VMS logical name as the connection identifier. Be sure to set the environment variable or the logical name before running your application. See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for more information on setting the environment variable or logical name of a specific device. The comments in Example 2–1 also explain how to set the environment variable or logical name for a specific device.

The third argument to OPEN WORKSTATION is the workstation type identifier. For example, DEC GKS predefines the constant 211 to specify the graphics handler that works with the DECwindows XUI handler. (For a list of the DEC GKS devices and their corresponding workstation type values, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.) To establish the workstation type in a device-independent manner, set the workstation type to the value 0, or to the default workstation type defined in the binding header file. DEC GKS reads either the ULTRIX environment variable (GKSwstype) or the VMS logical name (GKS$WSTYPE) to obtain the value of the workstation type identifier.

If the logical name, GKS$WSTYPE, or the environment variable, GKSwstype, is invalid, an error is generated.

You must call OPEN GKS before you call OPEN WORKSTATION. Depending on the needs of your application, you can open more than one workstation at a time. After the call to OPEN WORKSTATION, you can request input from the device, but you cannot generate output.

A call to ACTIVATE WORKSTATION alters values in the DEC GKS data structures necessary for a program that requires output generation. The argument passed to this function is the workstation identifier of an open workstation.

Once you call ACTIVATE WORKSTATION, DEC GKS produces any generated output on all workstations that are active at the time of output generation.

DEACTIVATE WORKSTATION, CLOSE WORKSTATION, and CLOSE GKS release the graphics handler data structures and DEC GKS. You must deactivate a workstation before you close it. You must close all active workstations before you close DEC GKS.

2.2 Controlling the Workstation Surface

Once you have generated a picture on the workstation surface, you may want to make some changes that require control of the workstation surface. To describe workstation surface control, it is necessary to first introduce the concept of **segments** and **primitives**. An output primitive is the image produced by a single DEC GKS output function. Individual primitives can be output directly to a display surface, or they can be grouped into segments and saved in workstation-dependent (or -independent) storage, as well as be output to all active workstations. Segments provide a means to save and manipulate groups of primitives, and they are crucial to controlling changes to a workstation surface. Segment manipulation is further described in Section 2.4.

When you control the workstation surface, you may work with individual primitives or primitives grouped in segments. The control functions may have different effects on individual primitives or segments. The following list presents the control functions that update the workstation surface and describes how the workstation surface is affected:

- The CLEAR WORKSTATION function clears the workstation surface of all primitive images, either inside or outside of segments, and deletes any segments from the workstation.
- The REDRAW ALL SEGMENTS ON WORKSTATION function clears the workstation surface of all images and then redisplays the primitives stored in segments.
- The UPDATE WORKSTATION function updates the workstation surface in one of two ways. If the update regeneration mode is set to POSTPONE, the UPDATE WORKSTATION function releases deferred output, but does not regenerate the display surface. If the regeneration mode is PERFORM and the display surface is out of date, the UPDATE WORKSTATION function forces a surface regeneration and all primitives not in segments are lost.
- The SET DEFERRAL STATE function establishes the current values for the **deferral mode** and **implicit regeneration mode**.

Deferral Mode

The deferral mode controls the time at which output functions have their visible effects. By delaying the output, the data sent to a device can be deferred to optimize data transfer. The four deferral modes, listed in increasing order, are as follows:

- ASAP (As soon as possible)

  The visual effect of each function is achieved on the workstation as soon as possible. GKS ensures that the actions necessary to achieve this visual effect are initiated before control is returned to the application program; however, actions are not necessarily completed before control is returned, if delays not under the influence of GKS occur.

- BNIG (Before next interaction globally)

  The visual effect of each function will be achieved on the workstation before the next interaction with a logical input device gets underway on any workstation. If an interaction on any workstation is already underway, the visual effect is achieved as soon as possible.

- BNIL (Before next interaction locally)

  The visual effect of each function will be achieved on the workstation before the next interaction with a logical input device gets underway on that workstation. If an interaction on that workstation is already underway, the visual effect will be achieved as soon as possible.

- ASTI (At some time)

  The visual effect of each function will be achieved on the workstation at some time.

Depending on its capabilities, the workstation can defer output at any level up to the level specified in the call to SET DEFERRAL STATE. For example, if you specify BNIL, the workstation can use the deferral mode ASAP, BNIG, or BNIL. Deferral applies to all functions that generate output:

- The 14 output functions that generate the GKS output primitives
- Four segment functions
  - INSERT SEGMENT (3)
  - ASSOCIATE SEGMENT WITH WORKSTATION
  - COPY SEGMENT TO WORKSTATION
- One metafile function
  - INTERPRET ITEM

Implicit Regeneration Mode

The implicit regeneration mode controls the time at which picture changes can occur. Picture changes generally refer to an alteration of an existing picture, and not an addition to the picture.

Each workstation description table has a *dynamic modification accepted* entry for picture changes initiated by changes to attribute bundle representations (see Chapter 4), the workstation and viewing transformations (see Chapter 3), and the hidden line hidden surface removal mode (HLHSR). The dynamic modification mode indicates which changes lead to an implicit regeneration (IRG) and which changes can be performed immediately (IMM). If changes can be performed immediately, primitives inside and outside of segments can be affected. If changes require an implicit regeneration, all primitives outside of segments are deleted from the display surface and the segments are redrawn. If surface regeneration is needed, you can delay the regeneration by specifying the appropriate value for the implicit regeneration mode. The implicit regeneration mode has two settings:

- SUPPRESSED—Implicit regeneration of the picture is suppressed until the picture is explicitly regenerated with the UPDATE WORKSTATION or REDRAW ALL SEGMENTS ON WORKSTATION function.
- ALLOWED—Implicit regeneration of the picture is allowed.

Usually, you do not want the workstation to regenerate the picture unless you request the updates. If many changes need to be made in immediate succession, make the changes and then request an update.

An implicit regeneration is made necessary if any of the following functions have a visible effect on the workstation surface and the given circumstances exist:

- The *dynamic modification accepted* entry in the workstation description table is set to IRG (implicit regeneration necessary) for the specified representation:
  - SET POLYLINE REPRESENTATION
  - SET POLYMARKER REPRESENTATION
  - SET TEXT REPRESENTATION
  - SET FILL AREA REPRESENTATION
  - SET EDGE REPRESENTATION
  - SET PATTERN REPRESENTATION
  - SET COLOUR REPRESENTATION
  - SET VIEW REPRESENTATION 3
- The *dynamic modification accepted* entry is set to IRG for the workstation transformation:
  - SET WORKSTATION WINDOW (3)
  - SET WORKSTATION VIEWPORT (3)
- The *dynamic modification accepted* entry is set to IRG for segment priority, the workstation supports segment priority, and any of the 14 output functions or INSERT SEGMENT (3) add primitives to an open segment overlapping a segment of higher priority.
- The *dynamic modification accepted* entry is set to IRG for segment priority, the workstation supports segment priority, and the complete execution of one of the following functions is affected by segment priority:
  - DELETE SEGMENT
  - DELETE SEGMENT FROM WORKSTATION
  - ASSOCIATE SEGMENT WITH WORKSTATION
  - SET SEGMENT TRANSFORMATION (3)
  - SET VISIBILITY
  - SET SEGMENT PRIORITY
- The *dynamic modification accepted* entry in the workstation description table is set to IRG for any of the following:
  - Segment transformation, controlled by the SET SEGMENT TRANSFORMATION(3) function
  - Visibility, controlled by the SET VISIBILITY function
  - Highlighting, controlled by the SET HIGHLIGHTING function
  - Delete segment, controlled by the DELETE SEGMENT and DELETE SEGMENT FROM WORKSTATION functions
  - HLHSR mode, controlled by the SET HLHSR MODE function

- HLHSR mode is set to perform HLHSR, and the operating state is SGOP for the 14 output functions and INSERT SEGMENT 3.

An implicit regeneration also is required if any of the situations in this list are the result of an INTERPRET ITEM function.

2.3 Generating Output

The DEC GKS output functions generate the basic components, or primitives, of all pictures. There are 14 function calls that specify the two-dimensional and three-dimensional versions of each of 7 primitives. For example, the POLYLINE and POLYLINE 3 functions generate a two- and three-dimensional line respectively. The DEC GKS output functions are:

- POLYLINE (3)—Draws connected lines between specified points sequence
- POLYMARKER (3)—Marks one or more locations with symbols
- TEXT (3)—Draws character strings
- FILL AREA (3)—Draws a closed polygon
- FILL AREA SET (3)—Draws a set of polygonal areas that may be hollow or filled with a color, a pattern, or a hatch style
- CELL ARRAY (3)—Colors cells of a specified parallelogram
- GENERALIZED DRAWING PRIMITIVE (3)—Draws a device-dependent primitive called a **generalized drawing primitive** (GDP)

When you generate DEC GKS primitives using the output functions, there are default attributes that affect the way the primitives appear on the workstation surface. For example, if you call POLYLINE 3 to output a line, the line is drawn solid (instead of dashed or dotted), in the **foreground color** and at the smallest width that the device handler supports (instead of a wider width). The foreground color is the color that your workstation uses, by default, to present text on the screen. The workstation uses the **background color** to fill the surface “behind” the text. Chapter 4 describes color and other output attributes in detail.

The line, marker, text, fill area, and fill area set primitives each have changeable output attributes associated with them. The output **attributes** describe the distinctive features of a primitive. For example, the line primitive has type (solid, dashed, dotted, dash-dotted), color, and width attributes. The generalized drawing primivites use the attributes of other primitives. There are function calls that let you change the attributes of each primitive individually or in groups. All attributes have default values if the program does not specify a value.

2.4 Working with Segments

Section 2.2 introduces the concept of segments and their importance in controlling the workstation surface. A segment is a group of output primitives that represents an object in a graphic image. Each segment has a unique name, which is specified by the application program, and a set of attributes, which are common to all the primitives in the segment. Segments have the following properties:

- GKS segments cannot be nested.
- An implicit regeneration causes individual primitives on a workstation to be lost, but primitives that are in segments can be redisplayed.

- Output primitives are stored in a segment after the normalization transformation has been applied to the primitives.
- Segments exist only on workstations.

A call to CREATE SEGMENT opens a segment on all active workstations. The CLOSE SEGMENT function closes the segment. Only one segment can be open at a time, and a closed segment cannot be reopened.

To add primitives to segment definitions, you call DEC GKS output functions when the segment is open. DEC GKS stores the current output attribute settings with the primitive when you call the output function. You can change a current individual primitive attribute setting while a segment is open, but the new value only affects subsequently generated primitives. To effectively change an attribute after segment creation, use bundled attributes. Bundled attributes are described in Chapter 4.

Manipulating segments is one of the most powerful features of a level 2c implementation of GKS. Using segments is the most effective way to control changes to the workstation surface. At times, you may need to redraw the picture on the surface of the workstation to reflect a change made by the application program. DEC GKS only redraws primitives contained in segments. Consequently, you should place retainable graphic data in segments. Otherwise, this graphic data is lost when DEC GKS performs certain surface control operations.

2.5 Program Example Used in This Chapter

Example 2–1 is located on the VMS kit in the following directory:

```
sys$common:[syshlp.examples.gks]
```

Example 2–1 is located on the ULTRIX kit in the following directory:

```
/usr/lib/GKS/examples
```

Example 2–1 uses C binding function calls to establishe a simple DEC GKS program that can be used for reference when developing other programs.

This example uses the default world coordinate (WC) range ([0,1] x [0,1]) to plot the picture. All text position coordinates are in the range [0,1]. DEC GKS transforms the WC points through four other coordinate systems and produces an image on the largest square of the display surface. Chapter 3 describes all the coordinate systems, and the functions and concepts needed to produce more sophisticated output.

Example 2–1 Beginning to Program with DEC GKS

```
/* This program illustrates a small skeleton of calls that are  */
/* needed to produce output with DEC GKS.  Three calls are      */
/* needed to open GKS and prepare the workstation for output,   */
/* OPEN GKS, OPEN WORKSTATION, and ACTIVATE WORKSTATION.  After */
/* these three calls are successfully completed, any of the     */
/* output functions can be called immediately or a segment can  */
/* be opened before the primitive calls are made.  After the    */
/* output is complete, three more calls are needed to clean up  */
/* the workstation environment(s) and to close GKS, DEACTIVATE  */
/* WORKSTATION, CLOSE WORKSTATION, and CLOSE GKS.  If a segment */
/* has been opened, you must close the segment before           */
/* deactivating the workstation.                        */
/*                                                              */
/*                                                              */
/****** ULTRIX RISC                                             */
/*                                                              */
/* To run this program on ULTRIX operating systems on RISC      */
/* processers, using any device handler EXCEPT DECwindows,      */
/* copy /usr/lib/GKS/gksconfig.c to your local directory if you */
/* wish to use the configuration file, and compile and link the */
/* program:                                                     */
/*                                                              */
/*  cc -I/usr/include/GKS -o user_manual_2_1 \                  */
/*    user_manual_2_1.c [gksconfig.c] -lGKScbnd -lGKS \         */
/*    /usr/lib/DXM/lib/Mrm/libMrm.a \                           */
/*    /usr/lib/DXM/lib/Xm/libXm.a /usr/lib/DSM/lib/Xt/libXt.a \ */
/*    -lddif -lcursesX -lc -lX11 -lm                            */
/*                                                              */
/*                                                              */
/* To run this program on ULTRIX operating systems on RISC      */
/* processors using the DECwindows device handler, copy file    */
/* /usr/lib/GKS/gks_decw_config.c to your local directory and   */
/* compile and link the program:                                */
/*                                                              */
/*  cc -I/usr/include/GKS -o user_manual_2_1 \                  */
/*    user_manual_2_1.c gks_decw_config.c -lGKScbnd -lGKS \     */
/*    -lddif -ldwt -lcursesX -lc -lX11 -lm                      */
/*                                                              */
/*                                                              */
/* Set the ULTRIX environment variables for the connection      */
/* identifier, GKSconid, the workstation type, GKSwstype, to    */
/* the desired values before running the executable. For        */
/* example, to run the program with the Motif device handler    */
/* use:                                                         */
/*                 setenv GKSconid mynode::0                    */
/*                 setenv GKSwstype 231                         */
/*                                                              */
/* To run the program with the DECwindows device handler use:   */
/*                                                              */
/*                 setenv GKSconid mynode::0                    */
/*                 setenv GKSwstype 211                         */
/*                                                              */
/* See the Device Specifics Reference Manual for DEC GKS and    */
/* DEC PHIGS for a list of available devices.                   */
/*                                                              */
/*                                                              */
```

(continued on next page)

Example 2–1 (Cont.) Beginning to Program with DEC GKS

```
    /****** OSF RISC                                              */
    /*                                                             */
    /* Because the information describing how to compile, link, and */
    /* run this program on an OSF system on RISC processors is not  */
    /* available at the time of this writing, see the gks manpage   */
    /* for the information.                                        */
    /*                                                             */
    /*                                                             */
    /****** VMS                                                    */
    /*                                                             */
    /* To run this program on VMS operating systems, change the     */
    /* "stderr" argument to "SYS$ERROR" in the OPEN GKS call,       */
    /* and compile and link the program:                           */
    /*                                                             */
    /*  CC user_manual_2_1.c                                       */
    /*  LINK user_manual_2_1, SYS$LIBRARY:GKS$CBND/LIB             */
    /*                                                             */
    /*                                                             */
    /* Set the VMS logical names for the connection identifier,     */
    /* GKS$CONID, and the workstation type, GKS$WSTYPE, to the      */
    /* desired values before running the executable. For example,   */
    /* to run the program with the Motif device handler use:       */
    /*                                                             */
    /*                 define GKS$CONID mynode::0.0                */
    /*                 define GKS$WSTYPE 231                       */
    /*                                                             */
    /* To run the program with the DECwindows device handler use:   */
    /*                                                             */
    /*                 define GKS$CONID mynode::0                  */
    /*                 define GKS$WSTYPE 211                       */
    /*                                                             */

/* Header files */

#include <stdio.h>                  /* standard C library I/O header file */
#include <gks.h>                    /* GKS C binding header file */

main ( )
    {
    Gconn    conn_id  = GWC_DEF;                 /* connection identifier */
    Glong    memory   = GDEFAULT_MEM_SIZE;       /* memory size */
    Gint     seg_id   = 1;                       /* segment name */

                                                 /* text string 1 */
    Gchar    *text_str = "This is a DEC GKS example program.";

    Gpoint   text_pos;                           /* text position */
    Gint     ws_id    = 1;                       /* workstation identifier */
    Gwstype  ws_type  = GWS_DEF;                 /* workstation type */

                                    /* event and timeout are used with the
                                       AWAIT EVENT call to pause program
                                       execution */
    Gevent  event;                  /* an input event */
    Gfloat  timeout = 8.00;         /* pause time in seconds */
```

(continued on next page)

Example 2–1 (Cont.) Beginning to Program with DEC GKS

```
/* Initialize GKS and prepare a workstation for output.         */
/* A pointer to a specified error file is passed in the call to */
/* OPEN GKS.  DEC GKS writes all errors to this file. In this   */
/* instance, DEC GKS uses the translation of the ULTRIX         */
/* environment variable, stderr, or the VMS logical name        */
/* SYS$ERROR. Argument, memory, is a dummy argument for         */
/* conformance with the GKS standard.                           */

gopengks (stderr, memory);
/*gopenks (SYS$ERROR, memory);*/
gopenws (ws_id, &conn_id, &ws_type);
gactivatews (ws_id);

/* Since at least one workstation is active, the output         */
/* primitives can be called.  In this instance, the output      */
/* primitives are saved in a segment. The workstation surface   */
/* is updated with the POSTPONE flag.  The update will release  */
/* any deferred output but will not force a surface             */
/* regeneration.  Since the flag is POSTPONE, the primitives    */
/* are visible after the update call even if they are not       */
/* stored in a segment.  The call to AWAIT EVENT pauses the     */
/* program execution to allow the user to view the picture.     */

gcreateseg (seg_id);
  gsetcharheight (0.03);
  text_pos.x = 0.1;
  text_pos.y = 0.6;
  gtext (&text_pos, text_str);
gcloseseg (seg_id);
gupdatews(ws_id, GPOSTPONE);
gawaitevent( timeout, &event );

/* To clean up the workstation environment(s), deactivate any   */
/* active workstations and close the open workstations. Lastly, */
/* close GKS.                                                   */

gdeactivatews (ws_id);
gclosews (ws_id);
gclosegks ( );

} /* End main */
```

3 Composing and Transforming Pictures

This chapter provides an overview of picture **transformation** that you need as a basis for learning additional details of output generation. A transformation is the mapping of primitives from the window of one coordinate system to the viewport of another coordinate system. This chapter describes the following concepts in detail:

- Transformation pipelines
- Normalization transformations
- View transformations
- Clipping of primitives
- Workstation transformations
- Transformations for particular effects

3.1 DEC GKS Coordinate Systems and Transformations

Example 6–1 plots the picture on the default world coordinate (WC) range, and DEC GKS draws the picture on the largest square that your workstation can produce, with the lower left corner of the WC range corresponding to the lower left corner of the workstation surface.

The DEC GKS coordinate systems offer greater flexibility than that provided by using default transformations. The DEC GKS coordinate systems address the following needs of graphics programming:

- Ability to plot portions of a picture on separate WC ranges (ranges that contain coordinate point values that are relevant to the data)
- Ability to construct a picture on a device-independent coordinate plane
- Ability to show any portion of the composed picture on any portion of any workstation surface

To meet the needs of graphics programming, DEC GKS supports both the two-dimensional and three-dimensional transformation pipelines. This section describes each pipeline in detail.

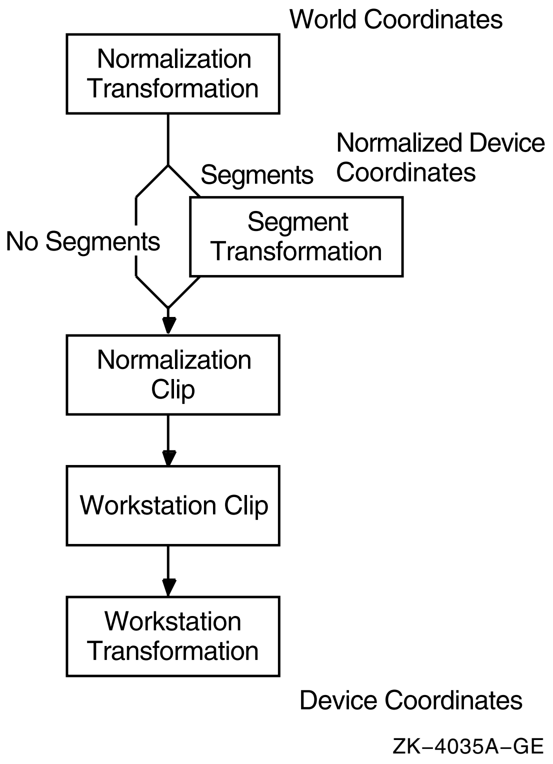
Two-Dimensional Transformation Pipeline

DEC GKS uses the following three distinct coordinate systems when producing any two-dimensional picture:

- World coordinate (WC) system
- Normalized device coordinate (NDC) system
- Device coordinate system

These three coordinate systems work as a transformation pipeline and ultimately display a two-dimensional object on your physical display device. You use portions of the WC system to plot the output primitives, a portion of the device-independent NDC plane to compose a complete picture, and a portion of the device coordinate plane to present all or part of your picture on all or part of the surface of the workstation. Figure 3–1 illustrates the DEC GKS two-dimensional transformation pipeline.

Figure 3–1 DEC GKS Two-Dimensional Transformation Pipeline



Three-Dimensional Transformation Pipeline

DEC GKS uses the following five distinct coordinate systems when producing any three-dimensional picture:

- WC system
- NDC system
- View reference coordinate (VRC) system
- Normalized projection coordinate (NPC) system
- Device coordinate system

These five coordinate systems work as a transformation pipeline and ultimately display a three-dimensional object on your physical display device. You use portions of the WC system to plot the output primitives, a portion of the device-independent NDC plane to compose a complete picture, portions of the VRC system to orient the picture, portions of the NPC system to determine projection volume, and a portion of the device coordinate plane to present all or part of your picture on all or part of the surface of the workstation.

The three-dimensional pipeline supports viewing, while the two-dimensional pipeline does not. The DEC GKS viewing pipeline is series of operations that let a three-dimensional object be displayed on the surface of a workstation. The viewing pipeline has three types of operations:

- Transformations

  The object undergoes several transformations between different coordinate systems.

- Clipping

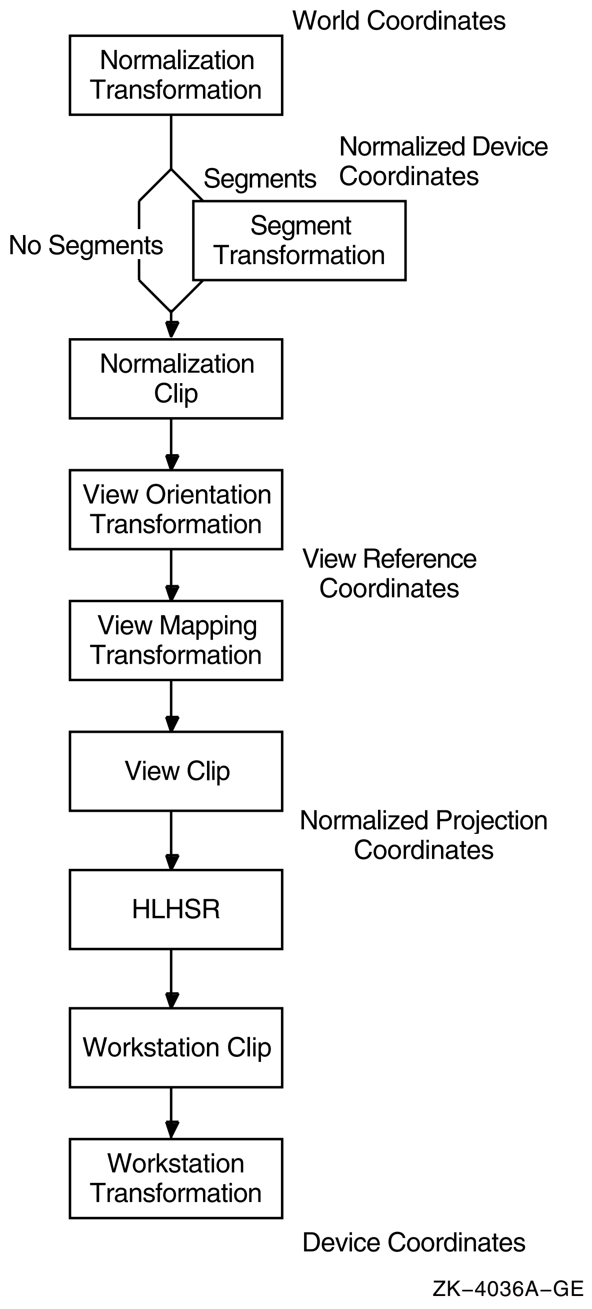
  The picture is **clipped** in the normalization, view, and workstation transformations according to the specified **clipping limits**.

- Hidden line hidden surface removal (HLHSR)

  HLHSR determines which parts of primitives are invisible as a result of being obscured by other primitives when the object is viewed along the direction of projection. HLHSR is done after the projection transformation. The HLHSR process is described in Section 3.1.3.

The DEC GKS three-dimensional viewing pipeline is depicted in Figure 3–2.

Figure 3–2 The DEC GKS Three-Dimensional Viewing Pipeline


World Coordinates
Normalization Transformation
Segments
Normalized Device Coordinates
No Segments
Segment Transformation
Normalization Clip
View Orientation Transformation
View Reference Coordinates
View Mapping Transformation
View Clip
Normalized Projection Coordinates
HLHSR
Workstation Clip
Workstation Transformation
Device Coordinates
ZK−4036A−GE

The following sections describe the DEC GKS coordinate systems and the three-dimensional viewing pipeline.

3.1.1 Normalization Transformation: WC Plane to NDC Plane

The WC system is the space in which objects are originally described. It is an imaginary coordinate system consisting of X, Y, and Z axes that extend infinitely in all six directions. (The Z axis pertains only to three-dimensional systems.) The system origin is the point (0.0, 0.0) for two-dimensional systems and (0.0, 0.0, 0.0) for three-dimensional systems. The infinite WC system gives you the flexibility to plot any output primitive according to whatever data is relevant. For example, if your data contains negative numbers, you can use a portion of the WC system that contains negative X, Y, and Z values. The NDC system is a workstation-independent abstract viewing system with a two-dimensional range of ([0,1] x [0,1]) or a three-dimensional range of ([0,1] x [0,1] x [0,1]).

For a two-dimensional system, the normalization transformation maps a specified rectangular area volume (**normalization window**) of the WC system on to a specified rectangular area volume (**normalization viewport**) of the NDC system. For a three-dimensional system, the normalization transformation maps a specified rectangular volume of the WC system on to a specified rectangular area volume of the NDC system. The normalized picture is stored and manipulated in segments and a metafile.

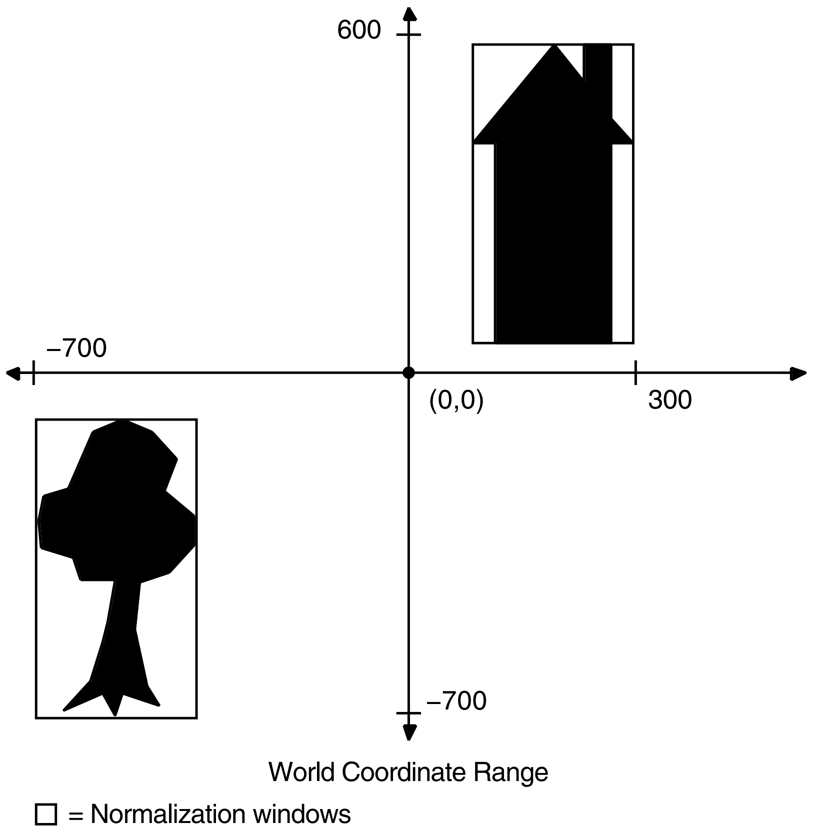
You can envision this process as cutting portions of the WC area or volume and pasting them in the NDC area or volume. The WC range is a dimensional scratch pad and the NDC range is your pasteboard. Figures 3–3 and 3–4 illustrate the mapping process from normalization windows to corresponding normalization viewports.

The normalization viewport of a particular normalization transformation is used to define a clipping volume. Clipping is enabled or disabled by a flag, the **clipping indicator**. The SET CLIPPING INDICATOR function turns normalization clipping on or off. Note that if the object is mapped to a subset of NDC space and clipping is OFF, the object can exceed the bounds of the NDC viewport. If clipping is ON, the parts of the object exceeding the bounds of the clipping volume are clipped. Clipping is explained further in Section 3.1.2.3.

Each normalization transformation is identified by a number in the GKS state list. Transformation number 0 always refers to the identity transformation. The zero normalization transformation for the two-dimensional pipeline maps ([0,1] x [0,1]) in WC points to ([0,1] x [0,1]) in NDC space. The normalization transformation for the three-dimensional pipeline maps ([0,1] x [0,1] x [0,1]) in WC points to ([0,1] x [0,1] x [0,1]) in NDC space. There is no scaling, rotation, or translation.

Example 2–1 uses the DEC GKS default transformations. By default, DEC GKS defines the two-dimensional WC range ([0,1] x [0,1]) to be the normalization window, and the two-dimensional NDC range ([0,1] x [0.1]) to be the normalization viewport. If you are using the three-dimensional transformation pipeline, the default WC range is ([0,1] x [0,1] x [0,1]), and the default NDC range is ([0,1] x [0,1] x [0,1]). Use the SET WINDOW or SET WINDOW 3 function to change the normalization window, the SET VIEWPORT or SET VIEWPORT 3 function to change the normalization viewport, and the SELECT NORMALIZATION TRANSFORMATION function to select the current normalization transformation.

Figure 3–3 Plotting Portions of a Picture in World Coordinates



ZK−5341−GE

3.1.1.1 Setting up the Normalization Transformation

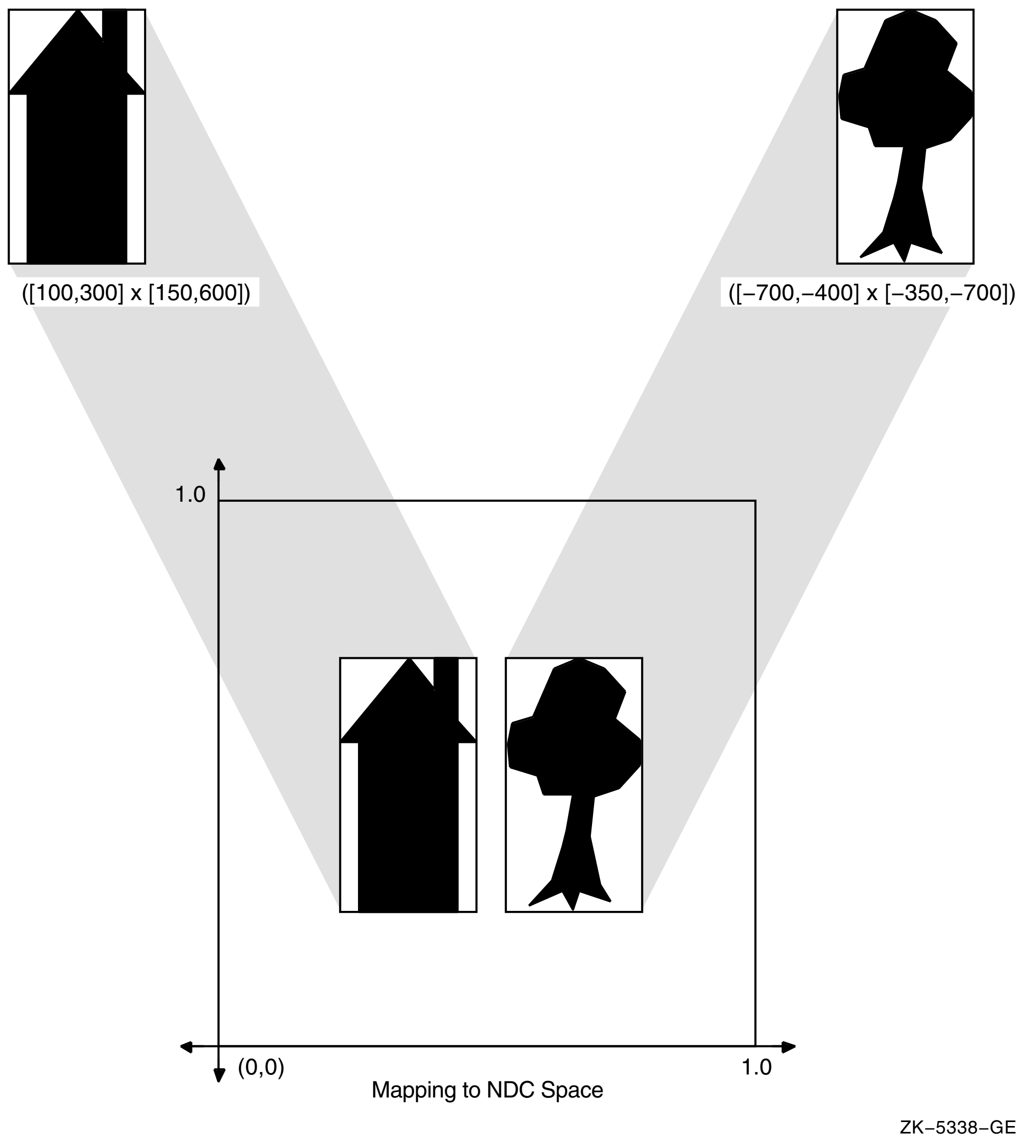
DEC GKS supports a range of normalization transformation numbers from 0 to 255. You can either choose a new normalization transformation number each time you wish to use a new window or viewport, or redefine the normalization window or normalization viewport associated with a single number.

Normalization transformation number 0 represents the DEC GKS default normalization transformation that maps primitives from the two-dimensional WC range ([0,1] x [0,1]) to the NDC range ([0,1] x [0,1]). If you are using the three-dimensional pipeline the default three-dimensional WC range ([0,1] x [0,1] x [0,1]) is mapped to the three-dimensional NDC range ([0,1] x [0,1] x [0,1]). The transformation associated with the normalization transformation number 0 is called the **unity transformation** and cannot be changed. Use numbers 1 to 255 to define new normalization transformations.

3.1.1.2 Normalization Clipping

In the two-dimensional transformation pipeline, the viewport of a particular normalization transformation is used to define a clipping rectangle, as well as, with the window, to specify the normalization transformation. When the viewport of the current normalization transformation is set or when a normalization transformation is selected, the clipping rectangle entry in the GKS state list is set to the resulting viewport of the current normalization transformation. Clipping to this clipping rectangle can be either enabled or disabled. The clipping indicator,

Figure 3–4 Composing a Picture in NDC Space



which is set with the SET CLIPPING INDICATOR function, enables and disables clipping.

Clipping does not take place when the normalization transformation is performed, but is delayed until the output primitives are displayed on the display surface. Output primitives stored in segments have their coordinates transformed to NDC points, and the associated clipping rectangle is stored with the primitives. The INSERT SEGMENT function allows the clipping rectangle in the GKS state list to replace the clipping rectangle that was stored with an output primitive when the segment was created. For more information on segments, see Chapter 4.

Clipping is an analagous activity for the three-dimensional pipeline; however, the two-dimensional clipping rectangle is replaced by the three-dimensional clipping volume. The INSERT SEGMENT 3 function allows the clipping rectangle in the GKS state list to replace the clipping rectangle that was stored with an output primitive when the segment was created.

3.1.2 View Transformation: NDC Space to NPC Space

The view transformation applies only to the three-dimensional pipeline. Before your three-dimensional picture can be generated, the NDC points must be mapped through the view transformation. The view transformation consists of three stages:

1. View orientation transformation

   The NDC image is transformed to VRC points according to the parameters in the **view orientation matrix**.

2. View mapping transformation

   The VRC image is projected to NPC points according to the parameters in the **view mapping matrix**.

3. View clip

   The image is clipped to the view volume in NPC space.

Before describing the steps of the view transformation, it is important to understand that each workstation stores a workstation-specific number of view entries in a **view table**, which is a part of the workstation state list. Every defined view is associated with an entry in the view table. The view entries are numbered consecutively, starting with 0. View 0 is predefined to the identity transformation and cannot be modified. Other entries are modified with the SET VIEW REPRESENTATION 3 function. This function associates a view with a number in the view table. The SET VIEW INDEX function specifies the current view, by number.

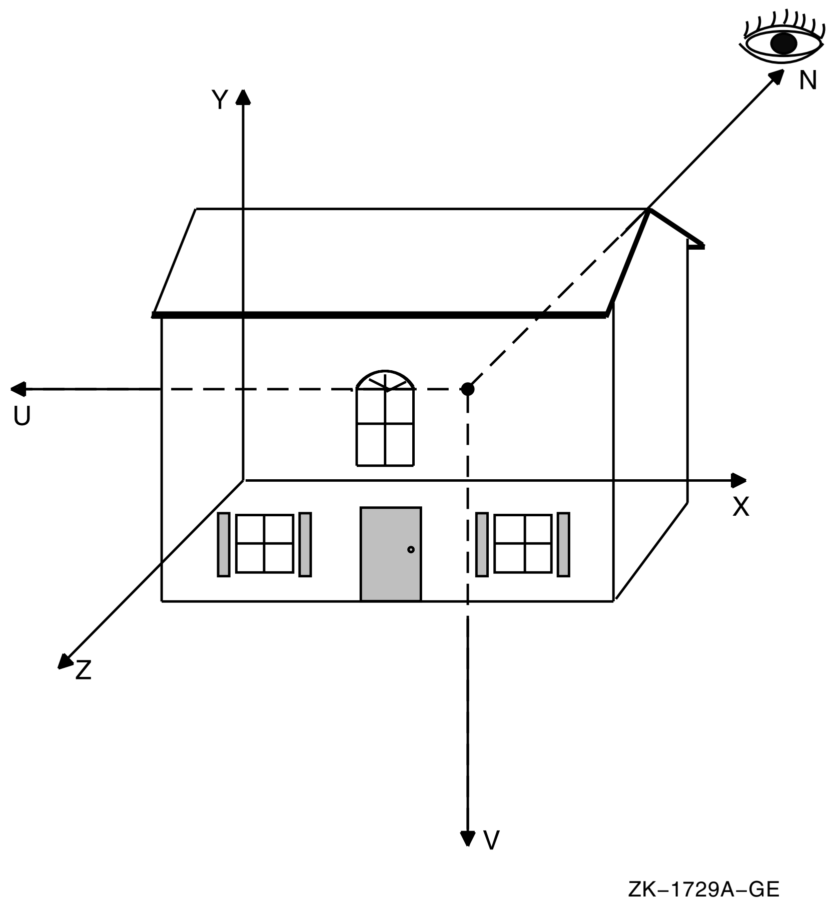
The view transformation employs the view orientation matrix to define what direction the picture is being viewed from, as well as the up direction. The view orientation matrix is used to transform each NDC point to an appropriate VRC point. The VRC axes are called U,V, and N.

3.1.2.1 View Orientation Transformation

The user changes the orientation of the picture by redefining the view orientation matrix. To define the transformation from NDC space to VRC space, the view orientation matrix uses the following information:

- The **view reference point** is the NDC point that is the origin in the VRC system. This point is usually on or near the object.
- The **view plane normal** is a vector relative to the view reference point and defines the N axis of the VRC system. The N axis is the third axis of the VRC system. The WC or NDC plane that contains the view reference point and that is perpendicular to the view plane normal is the **view reference plane**.
- The **view up vector** is a vector relative to the view reference point. The vector is projected onto the view reference plane via a projection parallel to the view plane normal. The projection of the view up vector into the view reference plane determines the V axis of the VRC system. The U axis is determined such that the UVN axes form a right-handed coordinate system. This is illustrated in Figure 3–5.

Figure 3–5 Establishing a View Reference Point



Change the orientation of the picture by redefining the view reference point (defining a new origin) and by declaring two vectors that specify the orientation of the output. The position of the view reference point and the orientation of the view plane are defined in the NDC system. Although the VRC system has the same units as the NDC system, the VRC system is effectively a shifted and rotated version of the NDC system, with the axes labeled U, V, and N. The view plane orientation is stored in the view orientation matrix.

DEC GKS supplies the utility function EVALUATE VIEW ORIENTATION MATRIX to calculate the matrix using the user-supplied data. It is best to use this function to compute the view orientation matrix, because the function returns an error if the orientation parameters are invalid. The matrix is computed from the view reference point and from the definitions of the two vectors. The calculated view orientation matrix is one of the arguments to the SET VIEW REPRESENTATION 3 function, which creates the view transformation.

3.1.2.2 View Mapping Transformation

The view mapping transformation maps a specified volume, the **view volume**, of the VRC system onto a specified volume, the **projection viewport**, of the NPC system. The view volume is defined by the following:

- Front plane
- Back plane
- View plane
- View window
- Projection reference point (eye point)

The user must choose between a **parallel** or **perspective projection**. Parallel projections maintain object size regardless of distance, so that the back side of a cube is the same size as the front side. Figure 3–6 illustrates a parallel projection.

In parallel projection, the view volume consists of the space between the front plane, back plane, and the parallel rays between the view plane, front plane, and back plane.

Perspective projections show an object that recedes with distance, so that the back side of a cube appears smaller than the front side. Figure 3–7 illustrates a perspective projection.

In perspective projection, the view volume consists of the space between the front plane, back plane, and intersecting rays emanating from the projection reference point and through the corners of the view window.

The view volume in the VRC system is mapped to a volume in the NPC system, which is defined by the front plane, back plane, and view plane. These planes are parallel to the UV plane and are defined by their distances from it: front plane distance, back plane distance, and view plane distance. The front plane distance must not be less than the back plane distance, as the front plane cannot be behind the back plane.

The data supplied to calculate the view mapping matrix is as follows:

- View window limits

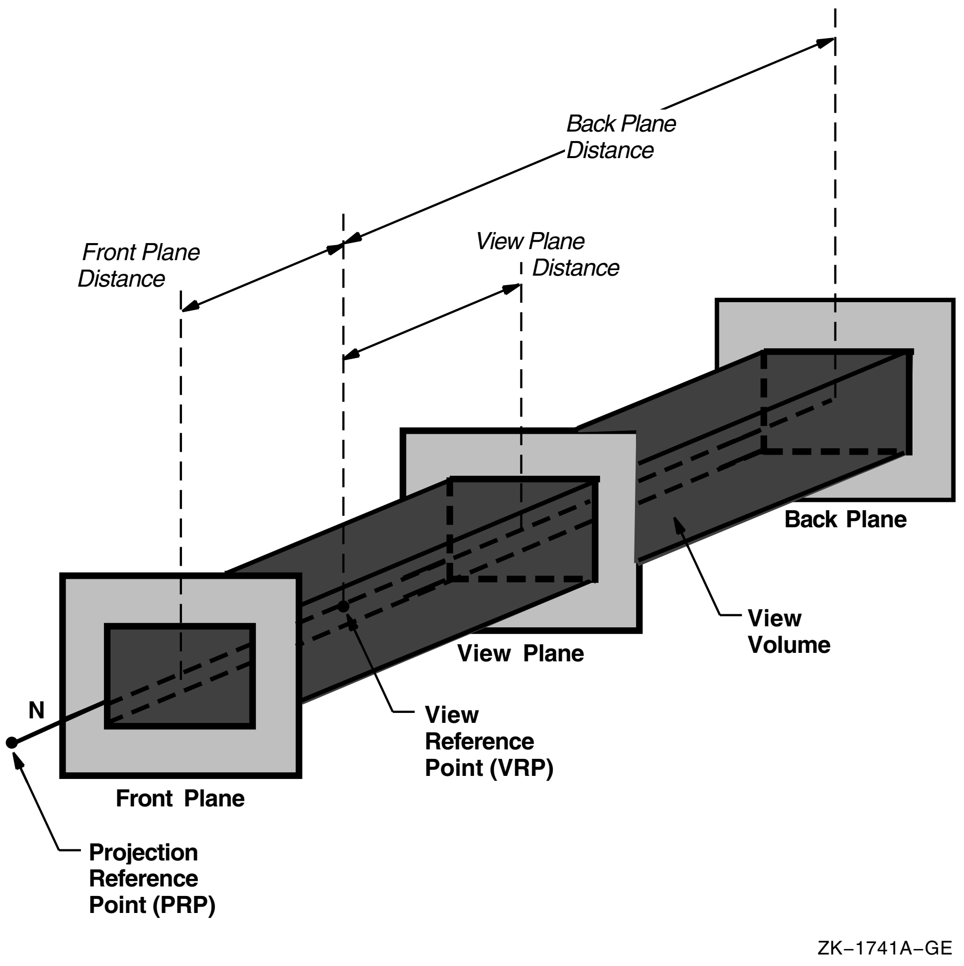
  The **view window** defines a rectangular region in the view plane.
- Viewport limits

  The **projection viewport** limits define a rectangular parallelepiped in the NPC system with edges parallel to the NPC axes.
- Projection type

  This is a parallel or perspective projection.
- Projection reference point

  The **projection reference point** defines the eye position in the VRC system. It is usually defined in front of the view plane, and cannot be between the front and back planes.

Figure 3–6 Parallel Projection



- View plane distance

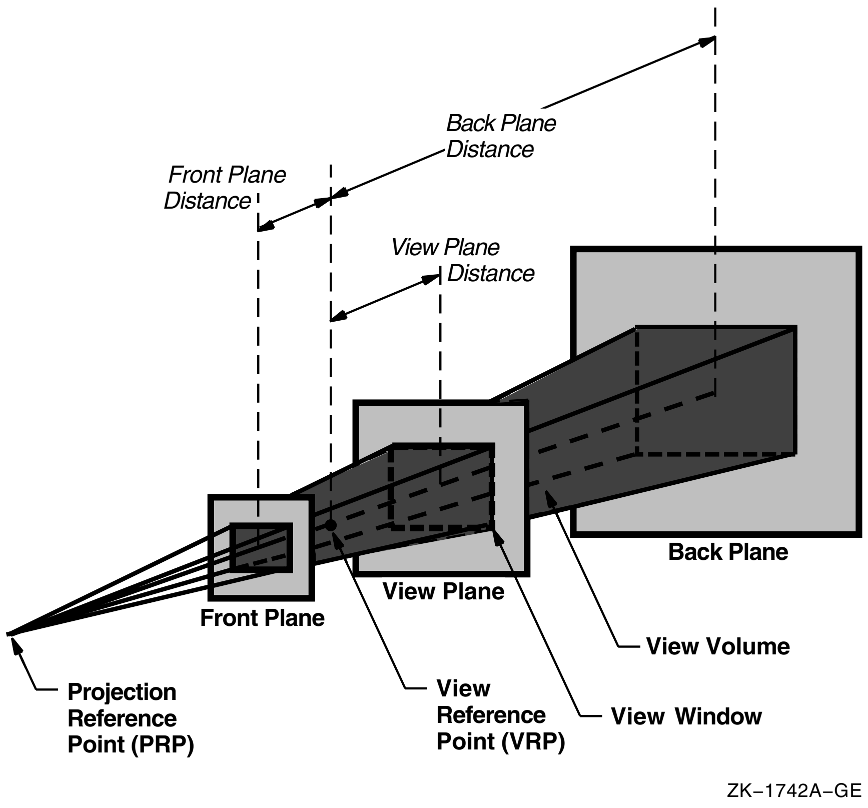
  The **view plane distance** is the distance of the view plane from the view reference point in the VRC system.

- Front and back plane distances

  The **front** and **back planes** are parallel to the view plane. They are usually defined as being, respectively, in front of and behind the object. If either plane intersects the object, the object is clipped.

DEC GKS uses this data to compute the view mapping matrix using the EVALUATE VIEW MAPPING MATRIX function. The use of this function to compute the mapping matrix is recommended, because it returns an error if the orientation parameters are invalid. The SET VIEW REPRESENTATION 3 function then uses the view mapping matrix to tell DEC GKS how to transform VRC data into NPC data.

Figure 3–7 Perspective Projection



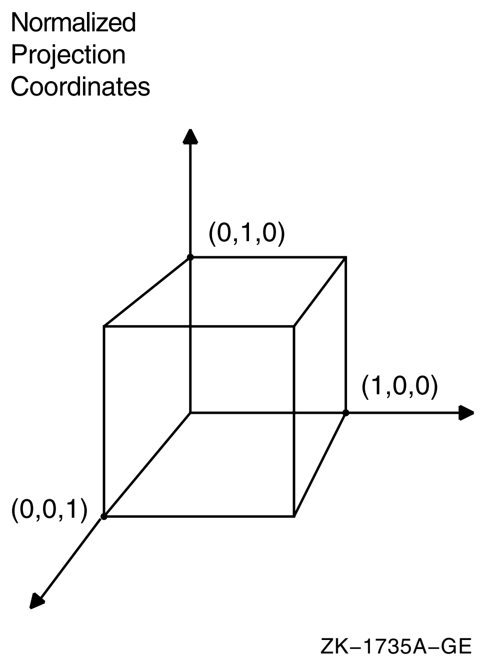
3.1.2.3 View Clip

After the view orientation and view mapping transformation have been applied, the image is clipped in NPC space according to defined view clipping limits. The six NPC clipping coordinates are set by the transformation function SET VIEW REPRESENTATION 3. The view clipping limits must be located inside the unit cube ([0,1] x [0,1] x [0,1]). The clipping limits always define a parallelepiped volume.

When an object is mapped from NDC points to NPC points, the **unit cube** is the default clipping volume. This means that if you do not specify a clipping volume, DEC GKS uses the unit cube as the clipping volume. You can specify your own clipping volume (a subvolume of the NPC unit cube), or turn view clipping on and off, with the SET VIEW REPRESENTATION 3 function. Figure 3–8 illustrates the unit cube in NPC points.

The SET VIEW REPRESENTATION 3 function specifies the clipping indicators, which control whether the planes defined by the clipping limits are active or inactive. If a clipping indicator is turned on, NPC data is clipped at the corresponding plane defined by the clipping limits. If a clipping indicator is off, the NPC data is not clipped at the plane. Instead, it is allowed to extend through the clip plane until it is conceptually clipped at the NPC system boundary.

Figure 3–8 Unit Cube



Consider the following analogy. To understand the concept of mapping against the clipping volume, suppose you have built a house that measures 5 inches high, 4 inches wide, and 4 inches deep. You want to put the house inside a box that is a 4 inch cube. (The box is analogous to the view clipping volume.) Because the house is too tall to fit into the box, you must remove the roof to make it fit. In essence, the roof is "clipped"—the entire house will not fit in the box, so the roof is discarded.

This is similar to what happens when a three-dimensional object is mapped to NPC points. If the object does not fall entirely in the clipping volume, the parts of the object that are outside the clipping volume may be clipped. This means that *only* the parts of the object that fall within the clipping volume are mapped to NPC points.

3.1.3 Hidden Line Hidden Surface Removal

Hidden line hidden surface removal (HLHSR) is optional. HLHSR is the process of determining the parts of primitives that are invisible because they are hidden by other primitives when they are observed from the projection reference point (perspective projection) or along the direction of the projection (parallel projection).

You can control the HLHSR processing with the HLHSR identifier and HLHSR mode. The HLHSR identifier is a global primitive attribute and controls HLHSR processing on a primitive-by-primitive basis. At primitive creation, the current value of the HLHSR identifier is bound to the primitive. The value of the identifier indicates no HLHSR processing or HLHSR processing with the **painter's algorithm**. The HLHSR mode controls HLHSR processing on a workstation-by-workstation basis. The value of the mode also indicates no HLHSR processing or HLHSR processing with the painter's algorithm. The current values for the HLHSR identifier and mode are set by the SET HLHSR

IDENTIFIER and SET HLHSR MODE functions, respectively. Both the identifier and the mode need to be set for HLHSR processing to occur.

HLHSR has the following relations with primitives and segments:

- Adding a primitive may cause an implicit regeneration (see Section 2.2) of the picture if HLHSR is being performed.
- Primitives that are not in segments do not take part in HLHSR calculations.
- Primitives in segments whose visibility is set to INVISIBLE cannot obscure other primitives.
- Primitives that are not visible as a result of HLHSR are not detectable.
- The HLHSR identifier bound to a primitive is replaced with the current HLHSR identifier by the functions INSERT SEGMENT and INSERT SEGMENT 3.
- HLHSR takes precedence over segment priority.

3.1.4 Workstation Clipping

Before proceeding with the workstation transformation, DEC GKS clips the picture at the workstation window boundaries, which are defined with the SET WINDOW or SET WINDOW 3 function. This clipping cannot be disabled. Even though DEC GKS stores primitives outside the default NPC range ([0,1] x [0,1]) in a two-dimensional system and the default NPC range ([0,1] x [0,1] x [0,1]) in a three-dimensional system, you cannot define a workstation window larger than the default range. If you attempt to do so, DEC GKS generates an error. You cannot map a primitive located outside of the default NPC two-dimensional or three-dimensional ranges to the physical device surface. DEC GKS clips all primitives at the workstation window.

3.1.5 Workstation Transformation

Two-Dimensional Workstation Transformation: NDC Space to DC Space

The two-dimensional workstation transformation is a uniform mapping from NDC space to device coordinate space (surface of the physical device). The two-dimensional workstation transformation is specified by defining the limits of an area in the NDC system (workstation window) which is mapped to a specified area of device coordinate space (workstation viewport). The workstation window and the workstation viewport limits specify rectangles parallel to the coordinate axes in the NDC and device coordinate systems.

Three-Dimensional Workstation Transformation: NPC Space to DC Space

The three-dimensional workstation transformation maps NPC space to the device coordinate space (the surface of the physical device). As with normalization transformations, DEC GKS uses a window and viewport to perform this mapping. DEC GKS maps the picture from the workstation window (located in NPC space) to the workstation viewport (located in device coordinate space). The range of the device coordinate space is completely device dependent. When DEC GKS maps images from the NPC space to the surface of the physical device, the largest possible portion of the NPC space that can be mapped is ([0,1] x [0,1] x [0,1]). Consequently, only images mapped within this volume of the NPC space can be mapped onto the physical device surface.

General Workstation Transformation Information

The workstation transformation is a uniform mapping from NPC space, or NDC space, to device coordinate space for X and Y, and thus performs translation and equal scaling with a positive scale factor for these two axes. Thus, picture composition can be achieved using the normalization transformations, but the workstation transformation allows different aspects of the composed picture to be viewed on different workstations. For example, a drawing could be output to a plotter at the correct scale and simultaneously some part of the drawing could be displayed on the full display space of an interactive terminal.

Unlike the normalization transformations, you have limited control over the workstation windows and viewports because you must work with limitations placed on both ranges. To make the distinction between the two types of transformations, you can think of the normalization transformations as being device-independent picture **composition**, and the workstation transformations as being device-dependent picture **presentation**. There can be many normalization transformations used to create a single picture, but there is only one current workstation transformation, with one window and viewport used to present a picture.

When working with the workstation viewport, you cannot define a viewport that is larger than the display surface. If you attempt to do so, DEC GKS generates an error. When redefining the workstation viewport, you should use the INQUIRE DISPLAY SPACE SIZE (3) function to determine the limits of the device's coordinate space.

You need to keep in mind that DEC GKS always maintains the X and Y aspect ratio of the picture when mapping to the workstation viewport. This means that DEC GKS may not use the entire defined viewport; DEC GKS uses the largest rectangle *within the current workstation viewport* that is proportionately equivalent to the current workstation window. So, if the current workstation window is square, DEC GKS maps the contents of the square workstation window to the largest square space within the current workstation viewport, beginning at the lower left corner. If the workstation window and the workstation viewport have different aspect ratios for X, Y, and Z, the specified scaling would be different on each axis if the workstation window is mapped onto the workstation viewport in its entirety. To ensure optimum results on the display space, the workstation transformation maps the workstation window onto the largest rectangular parallelepiped that can fit within the workstation viewport, such that:

- The aspect ratio in X and Y are preserved (two- and three-dimensional).
- The lower left corner of the workstation window closest to 0 in Z is mapped to the lower left corner of the workstation viewport furthest from the observer (three-dimensional only).
- The Z extent of the workstation window is mapped to the entire Z extent of the workstation viewport (three-dimensional only).

Thus, unused space is left at the top or right side of the workstation viewport if the aspect ratios of the workstation window and workstation viewport are different.

Another consideration when working with the workstation transformations is whether or not the screen is out of date. Using most of the DEC GKS supported devices, if you make a change to the workstation window or viewport after you generate output, you need to regenerate the surface of the workstation (if the workstation does not perform this action by default) to implement the changes. In making such a change, you cause all primitives not contained in segments to be cleared from the workstation surface. Changing the workstation transformation may cause an implicit regeneration of the surface.

Even though your control over the workstation transformation is limited, the effects on the representation of your picture can be quite impressive. You can pan across a picture, and zoom in and out of a picture. The remaining sections in this chapter illustrate workstation transformations to create particular effects.

3.2 Transformations to Produce Particular Effects

Once you compose your picture, you must decide how you wish to present the picture on the workstation surface. For example, you need to decide whether you wish to show the entire picture, whether you want to zoom in on a particular object, whether you want to pan across a picture, and how much of the device surface you need to display the portion of the picture on a given workstation.

3.2.1 Unused Display Space

The normalization transformation maps the picture from the WC system to the NDC system. This mapping is defined by specifying a normalization window in the WC system, that is to be mapped to a specified normalization viewport in the NDC system. If the window and the viewport do not have the same aspect ratios, mapping is not uniform.

To thoroughly understand this concept, consider the three-dimensional transformation process with respect to the unit cube. The picture is defined within the unit cube in WC points. If no normalization window is defined, the unit cube is the default. The picture in the default normalization window is mapped to NDC space, which has a (1 x 1 x 1) **aspect ratio** (the proportionate shape of primitives contained in a window) in the normalization viewport. The normalization viewport volume has the same ratios as the unit cube. Through the view transformation, the picture is mapped to the workstation window in NPC space. The workstation window also has a (1 x 1 x 1) aspect ratio, so again the ratios are the same as those of the unit cube. The default display space size (in device coordinates) does not have a (1 x 1 x 1) aspect ratio. The default display space size has a rectangular volume. When the picture in NPC space is mapped to device coordinate space, only a square volume in the device coordinate space is used. The workstation transformation maintains the workstation window aspect ratio; therefore, the entire display space size is not used. Section 3.2.3 provides more information on setting the display space size.

3.2.2 Aspect Ratio and Transformations

The aspect ratio is the ratio of the length along the principle axes of an area or volume. If the normalization window and normalization viewport have the same aspect ratio, objects will map undistorted from the normalization window to the normalization viewport.

To alter the shape or size of an object in a normalization window, alter the dimensions of the normalization viewport. If the aspect ratios of the normalization viewport and the normalization window remain the same, but the size of the normalization viewport varies, the image will change size uniformly, maintaining the original proportions of the image. If the aspect ratios of the normalization window and the normalization viewport change with respect to each other, the image in the normalization viewport will be distorted with respect to the original image in the normalization window. For example, if the normalization window is square and the normalization viewport has a length equal to twice the height, the normalization viewport primitives will appear wider than the primitives in the normalization window.

3.2.3 Mapping a Picture to the Entire Workstation Surface

To map a picture to the entire display, you must set the workstation window and the workstation viewport appropriately. To do this, you must first determine the display size in device coordinates using the INQUIRE DISPLAY SPACE SIZE (3) function. Use this information to determine the X and Y values for the workstation window dimensions. Assign the value 1.0 to the workstation window dimension corresponding to the larger display dimension. Assign the ratio (value of the smaller display dimension divided by the value of the larger display dimension) to the workstation window dimension corresponding to the smaller display dimension. Use the SET WORKSTATION WINDOW (3) function to define the new workstation window. The minimum value is 0.0 and the maximum value is the value determined from the display information.

Use the SET WORKSTATION VIEWPORT (3) function to define the workstation viewport. The minimum value is 0.0 and the maximum value is the display size value returned from the INQUIRE DISPLAY SPACE SIZE (3) call. Having defined the workstation window and the workstation viewport to the same proportions as the maximum display size assures that the entire display is used. Example 3–1 demonstrates this process.

Additionally, it is useful to set the normalization viewport values to the same values as the workstation window. This prevents a possible mapping into the upper portion of NPC space. As mentioned in Section 3.2.2, the normalization viewport and normalization window must have the same aspect ratio to avoid object distortion.

3.2.4 Zooming In and Out of a Picture

One of the most useful graphic effects that you can achieve using the workstation transformations is the zooming effect. By zooming in and out of a picture, you can give the effect of movement and distance. To zoom in on a picture, you need to reduce the dimensions of the workstation window while keeping the workstation viewport the same size. Subroutine ZOOM_PICTURE in Example 3–1 demonstrates how to code the zoom effect.

3.2.5 Panning Across a Picture

Panning across the picture does not require altering the size of the workstation window. You simply move the workstation window from position to position on the NPC space. Panning across a picture is useful when you do not want to show the user the entire picture at one time. Subroutine PAN_PICTURE in Example 3–1 demonstrates how to code for a panning effect.

3.2.6 Using a Smaller Portion of the Workstation Surface

To shrink a picture, reduce the size of the workstation viewport while maintaining the size of the workstation window. Keep the viewport proportions constant as you reduce the size. Subroutine SHRINK_PICTURE in Example 3–1 demonstrates the code associated with this process.

3.3 Two-Dimensional Pipeline Optimization

DEC GKS uses a two-dimensional pipeline if your program uses two-dimensional primitives and the current viewing index is set to 0.

If DEC GKS uses the two-dimensional pipeline, optimization includes the following features:

- Concatenation of normalization, view, and workstation clipping
- Concatenation of normalization, view, and workstation transformations
- Computation of *only* the X and Y coordinates

Two-dimensional GKS applications achieve the performance level of DEC GKS Version 4.2. Two-dimensional points are converted to three-dimensional points.

3.4 Program Example Used in This Chapter

Example 3–1 is located on the VMS kit in the following directory:

```
sys$common:[syshlp.examples.gks]
```

Example 3–1 is located on the ULTRIX kit in the following directory:

```
/usr/lib/GKS/examples
```

Example 3–1 illustrates how to work with transformations to achieve varied effects on the workstation surface.

Example 3–1 Using the DEC GKS Transformations

```
/* Header files */

#include <stdio.h>                  /* standard C library I/O header file */
#include <gks.h>                    /* GKS C binding header file */

                                    /* Constant definitions */
#define     NUM_SIDE_COLORS  6
#define     NUM_ROAD_COLORS  2
#define     NUM_STAR_PTS     6
#define     NUM_MOON_PTS     2
#define     NUM_TREE_PTS    29
#define     NUM_FRONT_PTS   10
#define     NUM_LROOF_PTS    4
#define     NUM_RROOF_PTS    4
#define     NUM_BACK_PTS     6
#define     NUM_RIGHT_PTS    5
#define     NUM_LEFT_PTS     5
#define     NUM_LAND_PTS    15
#define     NUM_SIDE_PTS     4
#define     NUM_ROAD_PTS     4
#define     BACK_IND         0
#define     RED_IND          1
#define     GREEN_IND        2
#define     BLUE_IND         3

static Gfile  *error_file;          /* error file */

                                    /* event and timeout are used with the
                                       AWAIT EVENT call to pause program
                                       execution */
Gevent  event;                      /* an input event */
Gfloat  timeout = 4.00;             /* pause time in seconds */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/*******************************************************************/
/*                                                                 */
/*  MAIN ()                                                        */
/*                                                                 */
/*******************************************************************/

main ( )

{
    void    set_up( );
    void    draw_picture( );
    void    clean_up( );
    Gint    ws_id   = 1;        /* workstation identifier */
    Gint    title   = 1,        /* segment identifier for the title */
            stars   = 2,        /* segment identifier for the stars and moon */
            tree    = 3,        /* segment identifier for the tree */
            side    = 4,        /* segment identifier for the sidewalk */
            road    = 5,        /* segment identifier for the road */
            land    = 6,        /* segment identifier for the land */
            house   = 7;        /* segment identifier for the house */
    Gwstype ws_type = GWS_DEF;  /* workstation type */
    Gconn   conid   = GWC_DEF;  /* connection identifier */

    set_up (ws_id, &ws_type, conid);

    draw_picture (ws_type, ws_id, title, stars, tree, side, road,
                  house, land);

    clean_up (ws_id);

} /* end main */

/*******************************************************************/
/*  SET_UP ()                                                      */
/*                                                                 */
/*  This subroutine sets up the DEC GKS and workstation            */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void set_up (ws_id, ws_type, conid)
Gint       ws_id;    /* workstation identifier */
Gwstype   *ws_type;  /* workstation type */
Gconn      conid;    /* connection identifier */

{
    Glong      memory = GDEFAULT_MEM_SIZE;  /* memory size */
    Gint       buf_size;                    /* buffer size */
    Gwscat     category;                    /* workstation category */
    Gchar      conid_buff[80];              /* buffer for workstation
                                               connection identifier */
    Gwsct      ct;                          /* workstation connection
                                               identifier and type */
    Gdefmode   def_mode;                    /* deferral mode */
    Girgmode   irg_mode;                    /* implicit regeneration mode */
    Gint       ret_size;                    /* returned size */
    Gint       status;                      /* returned status */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
   /* Initialize GKS.                                            */
   /* A pointer to a specified error file is passed in the call to */
   /* OPEN GKS.  DEC GKS writes all errors to this file.          */

   error_file = fopen ("example_3_1.error", "w");
   status = gopengks (error_file, memory);
   if (status != NO_ERROR)
      {
       gemergencyclosegks ( );
       fprintf (error_file, "Error opening GKS.\n");
       exit (0);
      }

   /* Open and activate a workstation.                           */
   /* Both the workstation type and the connection have been      */
   /* initialized to 0 so DEC GKS reads the VMS logical names,    */
   /* GKS$WSTYPE and GKS$CONID, or the ULTRIX enviroment variables,*/
   /* GKSwstype and GKSconid, for the values.                     */

   status = gopenws (ws_id, &conid, ws_type);
   if (status != NO_ERROR)
      {
       gemergencyclosegks ( );
       fprintf (error_file, "Error opening workstation.\n");
       exit (0);
      }
   gactivatews (ws_id);

   /* Defer output as long as possible and suppress implicit      */
   /* regeneration.                                               */
   /* The deferral mode specifies when calls to the output        */
   /* functions have their effect.  The implicit regeneration mode */
   /* specifies when changes to the existing picture are seen.    */
   /* This application chooses to control the time when attribute  */
   /* changes are seen on the display so it suppresses the        */
   /* implicit regeneration mode in case it is IMMEDIATE for the   */
   /* workstation.                                                */

   def_mode = GASTI;
   irg_mode = GSUPPRESSED;
   gsetdeferst (ws_id, def_mode, irg_mode);

   /* Determine the workstation connection and type.              */
   /* The application specifies the default workstation and type,  */
   /* which are defined by environment options.  This inquiry      */
   /* returns the connection and type to the application program,  */
   /* so it has the information.                                  */

   buf_size = sizeof(conid_buff);
   ct.conn = conid_buff;
   ct.type = ws_type;
   ginqwsconntype (ws_id, buf_size, &ret_size, &ct, &status);
   if (status)
      {
       gemergencyclosegks ( );
       fprintf (error_file,
 "Error: Inquire workstation connection and type.\n");
       fprintf (error_file, "Error status: %d\n", status);
       exit (0);
      }
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
    /* Inquire about the category of the workstation. The inquiry   */
    /* function verifies the workstation has a workstation type     */
    /* that can perform output.                                     */

    ginqwscategory (ws_type, &category, &status);
    if ( (status) || ((category != GOUTIN) && (category != GOUTPUT)) )
       {
        gemergencyclosegks ( );
        fprintf (error_file, "The workstation category is invalid.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

    /* If the workstation is DECWINDOWS(XUI) or MOTIF output, turn  */
    /* on hidden line hidden surface removal capabilities that      */
    /* eliminate hidden portions of objects from the picture.       */

    if ((*ws_type == GWS_DECWO) || (*ws_type == GWS_DECW) ||
 (*ws_type == GWS_MOTIFO) || (*ws_type == GWS_MOTIF))
       {
        gsethlhsrmode (ws_id, GHLHSR_MODE_PAINTER);
        gsethlhsrid (GHLHSR_ID_PAINTER);
        gupdatews (ws_id, GPERFORM);
       }

} /* end set_up */

/*********************************************************************/
/*  DRAW_PICTURE ()                                                  */
/*                                                                   */
/*  This subroutine draws the picture and places each primitive in  */
/*  a segment.  Subroutine draw_house is called to draw the house.  */
/*  All objects, except the house, are defined within the default   */
/*  world coordinate  range ([0, 1] x [0, 1] x [0, 1]). If the      */
/*  workstation has sufficient color capabilities, the objects are  */
/*  drawn in four colors which are defined with the SET COLOUR      */
/*  REPRESENTATION function.  This code assumes an RGB color model. */
/*  Device-independent code should inquire about the available      */
/*  color models and set the color model explicitly.  If the        */
/*  workstation is monochrome, the objects are drawn in the         */
/*  default foreground color.                                        */
/*                                                                   */
/*********************************************************************/

void draw_picture (ws_type, ws_id, title, stars, tree, side, road, house, land)

Gwstype ws_type;  /* workstation type */
Gint    ws_id,    /* workstation identifier */
        title,    /* segment identifier for the title */
        stars,    /* segment identifier for the stars and moon */
        tree,     /* segment identifier for the tree */
        side,     /* segment identifier for the sidewalk */
        road,     /* segment identifier for the road */
        house,    /* segment identifier for the house */
        land;     /* segment identifier for the land */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
{
    void     draw_house( );
    void     zoom_picture( );
    void     shrink_picture( );
    void     view_picture( );
                                /* array of the sidewalk colors for the
                                   cellarray call */
    static Gint  side_colors[NUM_SIDE_COLORS] = {2, 3, 1, 2, 3, 1},
                                /* array of the road colors for the
                                   cellarray call */
                 road_colors[NUM_ROAD_COLORS]  = {1, 3};
    Gint     status,            /* returned status */
             bufsize,           /* buffer size */
             colour,            /* flag to indicate if the workstation
                                   is color or monochrome */
             fac_size;          /* number of facilities */
    Gfloat   char_height = 0.04,    /* character height */
             line_width  = 3.0;     /* line width */
    Glnfac   line_fac;          /* line facility */
    Gintlist line_types_st;     /* integer list stucture of line types */
    Gint     line_types[20];    /* list of line types */
    Gcofac   col_fac;           /* color facility */
    Gchar *title_str = "Starry Night";  /* title string */
                                /* 3 sidewalk "corner" points for the
                                   cellarray call */
    static Grect3 side_rectangle_coordinates = { {0.2,  0.0, 0.8},
                                                 {0.25, 0.0, 0.8},
                                                 {0.2,  0.3, 0.7} };

                                /* 3 road "corner" points for the
                                   cellarray call */
    static Grect3 road_rectangle_coordinates = { {0.0, 0.0, 1.0},
                                                 {0.0, 0.0, 0.8},
                                                 {1.0, 0.0, 1.0} };
                                /* dimensions of the array with the
                                   with the sidewalk colors */
    static Gidim side_rectangle_dim = {1, 6},
                                /* dimensions of the array with the
                                   with the road colors */
                 road_rectangle_dim = {2, 1};

                                /* color definitions with RGB values */
    static Gcobundl black = {0.0, 0.0, 0.0};
    static Gcobundl red   = {1.0, 0.0, 0.0};
    static Gcobundl green = {0.0, 1.0, 0.0};
    static Gcobundl blue  = {0.0, 0.0, 1.0};

    Ggdprec  moon_data_rec;  /* GDP data record for the moon */
                                /* moon point values for the GDP */
    static Gpoint   moon_pts[NUM_MOON_PTS] = { {0.9,0.9}, {0.9,0.84} };

                                /* text font and precision */
    static Gtxfp text_font_prec = {1, GP_STROKE};
                                 /* text position */
    static Gpoint3  title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                    text_vec1 = {1.0, 0.0, 0.0},
                    text_vec2 = {0.0, 1.0,-1.0},

                                /* star coordinates */
                    stars_pts[NUM_STAR_PTS] =
                      { {0.05,0.70,0.0},  {0.06,0.86,0.0},   {0.36,0.81,0.0},
                        {0.66,0.86,0.0},  {0.835,0.7,0.0},   {0.92,0.82,0.0} },
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
                                /* land coordinates */
                land_pts[NUM_LAND_PTS] =
                  { {0.0,0.35,0.0},   {0.04,0.375,0.0},  {0.055,0.376,0.0},
                    {0.08,0.36,0.0},  {0.1,0.365,0.0},   {0.3,0.366,0.0},
                    {0.375,0.38,0.0}, {0.44,0.385,0.0},  {0.49,0.375,0.0},
                    {0.56,0.36,0.0},  {0.68,0.38,0.0},   {0.8,0.35,0.0},
                    {0.9,0.359,0.0},  {0.95,0.375,0.0},  {1.0,0.385,0.0} },

                                /* tree coordinates */
                tree_pts[NUM_TREE_PTS] =
                  { {0.425,0.28,0.0}, {0.5,0.3,0.0},     {0.52,0.26,0.0},
                    {0.54,0.3,0.0},   {0.6,0.28,0.0},    {0.575,0.33,0.0},
                    {0.56,0.42,0.0},  {0.559,0.49,0.0},  {0.64,0.53,0.0},
                    {0.69,0.57,0.0},  {0.689,0.61,0.0},  {0.66,0.64,0.0},
                    {0.63,0.66,0.0},  {0.645,0.71,0.0},  {0.59,0.76,0.0},
                    {0.53,0.78,0.0},  {0.48,0.75,0.0},   {0.45,0.71,0.0},
                    {0.42,0.65,0.0},  {0.375,0.645,0.0}, {0.35,0.6,0.0},
                    {0.375,0.55,0.0}, {0.44,0.54,0.0},   {0.45,0.5,0.0},
                    {0.515,0.5,0.0},  {0.51,0.425,0.0},  {0.495,0.38,0.0},
                    {0.475,0.33,0.0}, {0.425,0.28,0.0} },

                                /* sidewalk coordinates */
                side_pts[NUM_SIDE_PTS] =
                  { {0.2,0.0,0.8},   {0.25,0.0,0.8},
                    {0.25,0.3,0.7},  {0.2,0.3,0.7} },

                                /* road coordinates */
                road_pts[NUM_ROAD_PTS] =
                  { {0.0,0.0,1.0},    {1.0,0.0,1.0},
                    {1.0,0.0,0.8},    {0.0,0.0,0.8} };

Gdspsize  display_size;         /* display size */
Glimit3   window;               /* normalization window */
Glimit3   viewport;             /* normalization viewport */
Gfloat    ratio_x;              /* X dimension display ratio */
Gfloat    ratio_y;              /* Y dimension display ratio */
Glimit3   ws_window;            /* workstation window */
Glimit    ws_viewport;          /* workstation viewport */
Gint      largest_viewport = 1; /* normalization transformation indexes */
Gint      h_norm_left  = 2;
Gint      h_norm_back  = 3;
Gint      h_norm_front = 4;
Gmodws3   ws_dyn;               /* regeneration flag for workstation
                                   transformations */
Gint      view_index   = 1;     /* view index */

/* Set up normalization transformation.                         */
/* It is useful to set the normalization viewport values to the */
/* same values as the workstation window.  This prevents a      */
/* possible mapping into the upper portion of NPC space. The    */
/* following code sets up the normalization and workstation     */
/* transformations so that the entire display is used.  First,  */
/* inquire about the display size and use the information to    */
/* determine the two ratios of the maximum X and Y dimensions   */
/* to the larger of the two dimensions. In this manner, one     */
/* ratio is 1.0, which is the largest NPC value that can be     */
/* used as a dimension of the workstation  window. The other    */
/* ratio is less than 1.0.  With these ratios, define a         */
/* portion of the default NPC space ([0,1] x [0,1] x [0,1])     */
/* that is proportional to the device coordinate space used.    */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
ginqdisplaysize (&ws_type, &display_size, &status);
if (status != NO_ERROR)
   {
    gemergencyclosegks ( );
    fprintf (error_file, "Error inquiring workstation display size.\n");
    exit (0);
   }
if (display_size.device.x > display_size.device.y)
   {
    ratio_x = 1.0;
    ratio_y = display_size.device.y / display_size.device.x;
   }
else
   {
    ratio_x = display_size.device.x / display_size.device.y;
    ratio_y = 1.0;
   }

/* Set the normalization viewport dimensions so they are        */
/* proportional to the device display size and select the newly */
/* defined  normalization transformation to be the current      */
/* normalization transformation. The default normalization      */
/* window ([0,1] x [0,1] x [0,1]) is used since the objects     */
/* were defined within the default world coordinate range.      */

viewport.xmin = 0.0;
viewport.xmax = ratio_x;
viewport.ymin = 0.0;
viewport.ymax = ratio_y;
viewport.zmin = 0.0;
viewport.zmax = 1.0;
gsetviewport3 (largest_viewport, &viewport);
gselntran (largest_viewport);

/* Set the workstation window dimensions so they are            */
/* proportional to the device display.  The workstation window  */
/* establishes the portion of the composed picture, in NDC      */
/* space, that is mapped to the current workstation viewport.   */

ws_window.xmin = 0.0;
ws_window.xmax = ratio_x;
ws_window.ymin = 0.0;
ws_window.ymax = ratio_y;
ws_window.zmin = 0.0;
ws_window.zmax = 1.0;
gsetwswindow3 (ws_id, &ws_window);

/* Set the workstation viewport to the entire display.          */
/* Defining the workstation window and workstation viewport to  */
/* the same proportions as the maximum display size assures the */
/* entire display is used.                                      */

ws_viewport.xmin = 0.0;
ws_viewport.xmax = display_size.device.x;
ws_viewport.ymin = 0.0;
ws_viewport.ymax = display_size.device.y;
gsetwsviewport (ws_id, &ws_viewport);
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/* Check if the initialized line width is too wide.           */
/* The value of line_width is initialized to 3.00.  To avoid   */
/* requesting a line width that is wider than the workstation's */
/* widest line, call INQUIRE LINE FACILITIES.  If the line is   */
/* too wide, set it to the widest available width.             */

bufsize = sizeof(line_types);
line_types_st.integers = line_types;
line_fac.types = &line_types_st;
ginqlinefacil (&ws_type, bufsize, &fac_size, &line_fac, &status);
if (line_width * line_fac.nom_width > line_fac.max_width)
    line_width = line_fac.max_width / line_fac.nom_width;

/* Check if you are working with a color workstation.         */
/* If 4 colors are available, set each of the 4 color         */
/* representation indices for the specified workstation to the */
/* desired colors.  This type of coding is useful since       */
/* different workstations may have different default color     */
/* representations.                                            */

ginqcolourfacil (&ws_type, bufsize, &fac_size, &col_fac, &status);
colour = (col_fac.coavail == GCOLOUR) && (col_fac.predefined >= 4);
if (colour)
   {
    gsetcolourrep (ws_id, BACK_IND, &black);
    gsetcolourrep (ws_id, RED_IND, &red);
    gsetcolourrep (ws_id, GREEN_IND, &green);
    gsetcolourrep (ws_id, BLUE_IND, &blue);
   }

/* Draw the picture using view 1.                              */

gsetviewindex (view_index);


/* Create a segment for the title.                             */
/* If the workstation is color, make the text red. Set the     */
/* character height, and text precision and font.  All values  */
/* are initialized above. Use TEXT 3 to generate the string.   */
/* The position of the text string, the text vectors, and the  */
/* text string are initialized above.                          */

gcreateseg (title);
   if (colour)
       gsettextcolourind (RED_IND);
   gsetcharheight (char_height);
   gsettextfontprec (&text_font_prec);
   gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
gcloseseg ( );
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/* Create a segment for the stars and moon.                  */
/* If the workstation is color, make the stars blue. Make the  */
/* star shape a '+' and set the marker size scale factor. Use  */
/* POLYMARKER 3 to generate the star markers. The macro        */
/* NUM_STAR_PTS specifies the number of points. The list of    */
/* positions, stars_pts, is initialized above.                 */
/* Use a generalized drawing primitive (GDP) for the moon.     */
/* The moon object uses GDP_FCCP (-333), which is a filled     */
/* circle described by the center and a point on the           */
/* circumference.  The description in the Device Specifics     */
/* Reference Manual indicates that the data record is null so  */
/* the numbers of integers, floats, and strings in the         */
/* data record are set to 0.  The number of points, 2, is      */
/* defined by the macro NUM_MOON_PTS, and the center and       */
/* circumference point are passed in the list moon_pts.        */
/* GDP_FCCP is a macro defining the GDP number.  It is in the  */
/* C binding include file.                                     */

gcreateseg (stars);
   if (colour)
       gsetmarkercolourind (BLUE_IND);
   gsetmarkertype (GMK_PLUS);
   gsetmarkersize (2.0);
   gpolymarker3 (NUM_STAR_PTS, stars_pts);
   gsetfillintstyle (GSOLID);
   moon_data_rec.gdp_datarec.number_integer = 0;
   moon_data_rec.gdp_datarec.number_float = 0;
   moon_data_rec.gdp_datarec.number_strings = 0;
   ggdp (NUM_MOON_PTS, moon_pts, GDP_FCCP, &moon_data_rec);
gcloseseg ( );

/* Create a segment for the tree.                              */
/* If the workstation is color, make the tree green. Set the   */
/* fill area interior style to solid so the tree is a solid    */
/* shape. Draw the fill area described by the number of points */
/* defined in the NUM_TREE_PTS macro and the list of points in */
/* tree_pts.  The points are initialized above.                */

gcreateseg (tree);
   if (colour)
       gsetfillcolourind (GREEN_IND);
   gsetfillintstyle (GSOLID);
   gfillarea3 (NUM_TREE_PTS, tree_pts);
gcloseseg ( );

/* Create a segment for the sidewalk.                          */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the sidewalk.  If it is monochrome, use FILL AREA 3.        */
/* The CELL ARRAY 3 function divides a parallelogram into      */
/* cells and displays each cell in a specified color.  The     */
/* call requires 3 points on the parallelogram (lower left     */
/* front corner, upper right front corner and upper right back */
/* corner), the number of rows and columns into which the      */
/* parallelogram will be divided, and a 2-dimensional array    */
/* containing the color index values of the cells.  The        */
/* dimensions of the color index array correspond to the       */
/* dimension (number of rows and columns) of the parallelogram */
/* for the C binding.                                          */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
gcreateseg(side);
    if (colour)
       gcellarray3 (&side_rectangle_coordinates, &side_rectangle_dim,
                   side_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_SIDE_PTS, side_pts);
       }
gcloseseg ( );

/* Create a segment for the road.                              */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the road.  If it is monochrome, use FILL AREA 3.  All       */
/* values are initialized above.                               */

gcreateseg(road);
    if (colour)
       gcellarray3 (&road_rectangle_coordinates, &road_rectangle_dim,
                   road_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_ROAD_PTS, road_pts);
       }
gcloseseg ( );

/* Create a segment for the land.                              */
/* If the workstation is color, make the line outlining the    */
/* land green. Set the line width, make the line dashed, and   */
/* use the POLYLINE 3 function to draw the land horizon.       */

gcreateseg (land);
   if (colour)
       gsetlinecolourind (GREEN_IND);
   gsetlinewidth (line_width);
   gsetlinetype (GLN_DASHED);
   gpolyline3 (NUM_LAND_PTS, land_pts);
gcloseseg ( );

/* Create a segment for the house and draw it in three          */
/* different sizes.  Use different normalization                */
/* transformations to change the shape and size of the house.   */
/* The normalization window is set to values which allow a 10   */
/* unit border around the house in all three dimensions. By     */
/* varying the size of the normalization viewport, the size and */
/* proportions of the house are changed when the house is       */
/* mapped onto the normalization viewport.  The proportions of  */
/* the house change if the window and viewport are not          */
/* proportional.                                                */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
gcreateseg (house);
   window.xmin =  90.0;
   window.xmax = 310.0;
   window.ymin = 290.0;
   window.ymax = 760.0;
   window.zmin = 290.0;
   window.zmax = 710.0;
   gsetwindow3 (h_norm_left, &window);
   viewport.xmin = 0.090*ratio_x;
   viewport.xmax = 0.310*ratio_x;
   viewport.ymin = 0.290*ratio_y;
   viewport.ymax = 0.760*ratio_y;
   viewport.zmin = 0.290;
   viewport.zmax = 0.710;
   gsetviewport3 (h_norm_left, &viewport);
   gselntran (h_norm_left);

   draw_house (colour);

   window.xmin =  90.0;
   window.xmax = 310.0;
   window.ymin = 290.0;
   window.ymax = 760.0;
   window.zmin = 290.0;
   window.zmax = 710.0;
   gsetwindow3 (h_norm_back, &window);

 /* Mapping the house to the normalization viewport gives the   */
 /* effect of mapping a small house in the distance.            */

   viewport.xmin = 0.320*ratio_x;
   viewport.xmax = 0.465*ratio_x;
   viewport.ymin = 0.345*ratio_y;
   viewport.ymax = 0.467*ratio_y;
   viewport.zmin = 0.400;
   viewport.zmax = 0.600;
   gsetviewport3 (h_norm_back, &viewport);
   gselntran (h_norm_back);

   draw_house (colour);

   window.xmin =  90.0;
   window.xmax = 310.0;
   window.ymin = 290.0;
   window.ymax = 760.0;
   window.zmin = 290.0;
   window.zmax = 710.0;
   gsetwindow3 (h_norm_front, &window);

 /* Mapping the house to the normalization viewport gives the   */
 /* effect of mapping a tall house to the forefront of the      */
 /* picture.  This house occludes the tree.                     */

   viewport.xmin = 0.600*ratio_x;
   viewport.xmax = 0.800*ratio_x;
   viewport.ymin = 0.150*ratio_y;
   viewport.ymax = 0.900*ratio_y;
   viewport.zmin = 0.400;
   viewport.zmax = 0.800;
   gsetviewport3 (h_norm_front, &viewport);
   gselntran (h_norm_front);

   draw_house (colour);

gcloseseg ( );
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
    /* Inquire if the workstation transformations require an update */
    /* to take effect or if the change is made immediately.         */

    ginqmodwsattr3 (&ws_type, &ws_dyn, &status);

    /* Perform the update if it is needed and pause program         */
    /* execution to view the picture.                               */

    if (ws_dyn.wstran == GIRG)
       gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );

    /* Call routines to zoom into the picture and to shrink it.     */

    zoom_picture (ws_id, ws_dyn.wstran, ratio_x, ratio_y);

    shrink_picture (ws_id, ws_dyn.wstran, display_size.device.x,
                    display_size.device.y);

    /* Reset the workstation window so that it is again             */
    /* proportional to the display and perform an update if         */
    /* the workstation transformation requires it.  Pause the       */
    /* program execution to allow the user to view the picture.     */

    gsetwswindow3 (ws_id, &ws_window);
    gsetwsviewport (ws_id, &ws_viewport);
    if (ws_dyn.wstran == GIRG)
       gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );

    /* Examine the picture from different viewpoints.               */

    view_picture (ws_id, view_index, ws_dyn.view);

} /* end draw_picture */

/*******************************************************************/
/*  DRAW_HOUSE ()                                                  */
/*                                                                 */
/*  This subroutine draws a house.                                 */
/*  If the workstation is color make the back, left side, and right */
/*  side of the house red, and the left and right sides of the roof */
/*  blue.  Use FILL AREA 3 to draw the back, left side, and right   */
/*  side of the house.  Use FILL AREA SET 3 to draw the front of    */
/*  the house.  FILL AREA SET 3 draws the door and the front of the */
/*  house.  The area for the door is hollow and is contained        */
/*  within the house front area.  The green from the back of the    */
/*  house shows through.                                           */
/*                                                                 */
/*******************************************************************/

void draw_house (colour)

Gint colour;  /* flag to indicate if the the workstation is color
                 or monochrome. */
{
    Gfloat   edge_width = 2.0;      /* edge width */

                                    /* number of points in each of the fill
                                       areas comprising the fill area set */
    static Gint     front_sizes[2] = {6, 4};
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
    static Gpoint3
                                    /* house front coordinates */
           front_pts[NUM_FRONT_PTS] =
             { {100.0,300.0,700.0}, {300.0,300.0,700.0}, {300.0,600.0,700.0},
               {200.0,750.0,700.0}, {100.0,600.0,700.0}, {100.0,300.0,700.0},
               {200.0,300.0,700.0}, {250.0,300.0,700.0}, {250.0,400.0,700.0},
               {200.0,400.0,700.0} },

                                    /* house back coordinates */
           back_pts[NUM_BACK_PTS] =
             { {100.0,300.0,300.0}, {100.0,600.0,300.0}, {200.0,750.0,300.0},
               {300.0,600.0,300.0}, {300.0,300.0,300.0}, {100.0,300.0,300.0} },

                                    /* house right side coordinates */
           right_pts[NUM_RIGHT_PTS] =
             { {300.0,300.0,700.0}, {300.0,300.0,300.0}, {300.0,600.0,300.0},
               {300.0,600.0,700.0}, {300.0,300.0,700.0} },

                                    /* house left side coordinates */
           left_pts[NUM_LEFT_PTS] =
             { {100.0,300.0,300.0}, {100.0,300.0,700.0}, {100.0,600.0,700.0},
               {100.0,600.0,300.0}, {100.0,300.0,300.0} },

                                    /* roof left side coordinates */
           lroof_pts[NUM_LROOF_PTS] =
             { {100.0,600.0,700.0}, {200.0,750.0,700.0},
               {200.0,750.0,300.0}, {100.0,600.0,300.0} },

                                    /* roof right side coordinates */
           rroof_pts[NUM_RROOF_PTS] =
             { {300.0,600.0,700.0}, {300.0,600.0,300.0},
               {200.0,750.0,300.0}, {200.0,750.0,700.0} };

    gsetfillintstyle (GSOLID);
    if (colour)
        gsetfillcolourind (GREEN_IND);
    gfillarea3 (NUM_BACK_PTS, back_pts);
    if (colour)
        gsetfillcolourind (RED_IND);
    gfillarea3 (NUM_LEFT_PTS, left_pts);
    gfillarea3 (NUM_RIGHT_PTS, right_pts);
    if (colour)
        gsetfillcolourind (BLUE_IND);
    gfillarea3 (NUM_LROOF_PTS, lroof_pts);
    gfillarea3 (NUM_RROOF_PTS, rroof_pts);
    if (colour)
        gsetfillcolourind (BLUE_IND);
    gsetedgeflag (GEDGE_ON);
    gsetedgewidthscfac (edge_width);
    gfillareaset3 (2, front_sizes, front_pts);
} /* end draw_house */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/*******************************************************************/
/*  ZOOM_PICTURE ()                                                */
/*                                                                 */
/*  This subroutine produces the effect of zooming into a picture.  */
/*  To produce this effect, reduce the size of the workstation     */
/*  window while keeping the workstation viewport constant.        */
/*                                                                 */
/*******************************************************************/

void zoom_picture (ws_id, ws_xforms, max_x, max_y)

Gint     ws_id;      /* workstation identifier */
Gint     ws_xforms;  /* regeneration flag for workstation
                        transformations */
Gfloat   max_x;      /* maximum X NPC value of the current
                       workstation window */
Gfloat   max_y;      /* maximum Y NPC value of the current
                        workstation window */
{
    void      pan_picture( );
    Gfloat    start_x = 0.0;   /* minimum x workstation window value */
    Gfloat    start_y = 0.0;   /* minimum y workstation window value */
    Glimit3   ws_window;       /* workstation window value */
    int       k;               /* loop control constant */

    /* Zoom into the picture 3 consecutive times.                  */

    for (k = 0; k < 3; k++)
       {

    /* Reduce the workstation window by 12% for each zoom into the  */
    /* the picture.  To map the workstation window onto the entire  */
    /* display surface, keep the workstation window proportional to */
    /* the display surface.  The values passed to this routine      */
    /* are already proportional, so if the maximum value is         */
    /* decreased by 12%, increase the start value by 12% to         */
    /* preserve the proportions.                                    */

        max_x -= max_x*0.12;
        max_y -= max_y*0.12;
        start_x += max_x*0.12;
        start_y += max_y*0.12;

    /* Create the new workstation window and perform an update if   */
    /* the workstation transformation requires it.  Pause the       */
    /* program execution to allow the user to view the picture.     */

        ws_window.xmin = start_x;
        ws_window.xmax = max_x;
        ws_window.ymin = start_y;
        ws_window.ymax = max_y;
        ws_window.zmin = 0.0;
        ws_window.zmax = 1.0;
        gsetwswindow3 (ws_id, &ws_window);
        if (ws_xforms == GIRG)
           gupdatews (ws_id, GPERFORM);
        gawaitevent( timeout, &event );
       }

    /* Now that the current workstation window is a small portion   */
    /* of the entire picture, pan across the picture.               */

    pan_picture (ws_id, ws_xforms, start_x, max_x, start_y, max_y);

} /* end zoom_picture */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/*******************************************************************/
/*  PAN_PICTURE ()                                                 */
/*                                                                 */
/*  This subroutine creates the effect of panning across the       */
/*  picture in a horizontal direction, first to the left and then   */
/*  to the right.                                                  */
/*                                                                 */
/*******************************************************************/

void pan_picture (ws_id, ws_xforms, start_x, max_x, start_y, max_y)

Gint     ws_id;       /* workstation identifier */
Gint     ws_xforms;   /* regeneration flag for workstation
                         transformations */
Gfloat   start_x;     /* minimum x workstation window value */
Gfloat   max_x;       /* maximum x workstation window value */
Gfloat   start_y;     /* minimum y workstation window value */
Gfloat   max_y;       /* maximum y workstation window value */

{
    Gint     n;                 /* loop control variable */
    Gint     k;                 /* loop control variable */
    Gfloat   pan_step = 0.075;  /* size of the change for the panning */
    Glimit3  ws_window;         /* workstation window */
    /* First pan to the left in 3 frames and then pan to the right  */
    /* in 3 frames.                                                 */

    for (n = 0; n < 2; n++)
       {
        for (k = 0; k < 3; k++)
           {

    /* Set the workstation window for the next frame of the         */
    /* X-direction panning and perform regeneration of the image if */
    /* needed.  The change defined by pan_step shifts the           */
    /* workstation window to the left or to the right depending on  */
    /* the sign of the value of pan_step.  Pause the program        */
    /* execution to allow the user to view the picture.             */

            max_x -= pan_step;
            start_x -= pan_step;
            ws_window.xmin = start_x;
            ws_window.xmax = max_x;
            ws_window.ymin = start_y;
            ws_window.ymax = max_y;
            ws_window.zmin = 0.0;
            ws_window.zmax = 1.0;
            gsetwswindow3 (ws_id, &ws_window);
            if (ws_xforms == GIRG)
               gupdatews (ws_id, GPERFORM);
            gawaitevent( timeout, &event );
           }  /* end for k */

    /* Now pan in the opposite direction for the last 3 frames.     */
        pan_step = -pan_step;

       }  /* end for n */

} /* end pan_picture */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/*******************************************************************/
/*  SHRINK_PICTURE ()                                              */
/*                                                                 */
/*  This subroutine shrinks the picture.  To shrink a picture,     */
/*  reduce the size of the workstation viewport while maintaining   */
/*  the size of the workstation window.  Keep the workstation      */
/*  viewport proportions constant while reducing the workstation   */
/*  viewport size.                                                 */
/*                                                                 */
/*******************************************************************/

void shrink_picture (ws_id, ws_xforms, display_x, display_y)

Gint     ws_id;      /* workstation identifier */
Gint     ws_xforms;  /* regeneration flag for workstation
                        transformations */
Gfloat   display_x;  /* maximum X display size in DC */
Gfloat   display_y;  /* maximum Y display size in DC */

{
    Gfloat   start_x = 0.0;     /* minimum x workstation viewport value */
    Gfloat   max_x;             /* maximum x workstation viewport value */
    Gfloat   start_y = 0.0;     /* minimum y workstation viewport value */
    Gfloat   max_y;             /* maximum y workstation viewport value */
    Glimit   ws_viewport;       /* workstation viewport */

    max_x = display_x;
    max_y = display_y;

   /* Set the workstation viewport to a reduced size and perform an */
   /* update if the workstation transformation requires it.  Pause  */
   /* the program execution to allow the user to view the picture.  */

    max_x -= max_x * 0.3;
    max_y -= max_y * 0.3;
    start_x += max_x * 0.3;
    start_y += max_y * 0.3;
    ws_viewport.xmin = start_x;
    ws_viewport.xmax = max_x;
    ws_viewport.ymin = start_y;
    ws_viewport.ymax = max_y;
    gsetwsviewport (ws_id, &ws_viewport);
    if (ws_xforms == GIRG)
       gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );

    /* Set the workstation viewport to the entire device coordinate */
    /* space and perform an update if the workstation               */
    /* transformation requires it.  Pause the program execution to  */
    /* allow the user to view the picture.                          */

    ws_viewport.xmin = 0.0;
    ws_viewport.xmax = display_x;
    ws_viewport.ymin = 0.0;
    ws_viewport.ymax = display_y;
    gsetwsviewport (ws_id, &ws_viewport);
    if (ws_xforms == GIRG)
       gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );

} /* end shrink_picture */
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
/*****************************************************************/
/*  VIEW_PICTURE ()                                              */
/*                                                               */
/*  This subroutine shows five different views of the scene.     */
/*                                                               */
/*****************************************************************/

void view_picture (ws_id, view_index, view_xforms)

Gint   ws_id;         /* workstation identifier */
Gint   view_index;    /* view index */
Gint   view_xforms;   /* regeneration flag for viewing
                         transformations */

{
#define  NUM_VPN   4  /* largest index in array of view
                         plane normals */

    static Gpoint3   vrp = {0.5, 0.5, 0.5};   /* view reference point */

                                              /* view plane normals */
    static Gpoint3   vpn[NUM_VPN] =
                          { {0.9, 0.8, 1.0},  {0.9, 0.8,-1.0},
                            {-0.9, 0.8,-1.0}, {-0.9, 0.8, 1.0} };

    static Gpoint3   vuv = {0.0, 1.0, 0.0};   /* view up vector */
           Gfloat    orientation[4][4];       /* view orientation matrix */

                                              /* view window */
    static Glimit    view_window   = {-1.0, 1.0, -1.0, 1.0};

                                              /* projection viewport */
    static Glimit3   proj_viewport = {0.0, 1.0, 0.0, 1.0, 0.0, 1.0};

    static Gpoint3   prp = {0.0, 0.0, 100.0};  /* projection reference point */
           Gfloat    vpd =  0.0;              /* view plane distance */
           Gfloat    fpd =  1.0;              /* front plane distance */
           Gfloat    bpd = -1.0;              /* back plane distance */
           Gfloat    mapping[4][4];           /* view mapping matrix */
           Glimit3   clip_limits;             /* view clipping limits */
           Gint      status;                  /* returned status */
           Gint      k;                       /* loop control variable */

    for (k = 0; k < NUM_VPN; k++)
       {

    /* Calculate the view orientation matrix which specifies the    */
    /* new orientation for the output of the picture. The view      */
    /* orientation is defined by the view reference point, the view */
    /* plane normal, and the view up vector.                        */

        gevalvieworienttran3 (&vrp, &vpn[k], &vuv, GNDC, orientation, &status);
           if (status != NO_ERROR)
             {
              gemergencyclosegks ( );
              fprintf (error_file,
                       "Error evaluate view orientation transformation.\n");
              exit (0);
             }
```

(continued on next page)

Example 3–1 (Cont.) Using the DEC GKS Transformations

```
    /* Calculate the view mapping matrix which is used to transform */
    /* the picture in the VRC system into the NPC system. The view  */
    /* mapping matrix is defined by the view window, the projection */
    /* viewport, the type of projection, the projection reference   */
    /* point, the view plane distance, and the front and back plane */
    /* distances.                                                   */

        gevalviewmaptran3 (&view_window, &proj_viewport, GPERSPECTIVE, &prp,
                           vpd, fpd, bpd, mapping, &status);
          if (status != NO_ERROR)
             {
              gemergencyclosegks ( );
              fprintf (error_file,
                       "Error evaluate view mapping transformation.\n");
              exit (0);
             }

    /* Set the clipping limits in NPC for the X, Y, and Z axes.     */

        clip_limits.xmin = clip_limits.ymin = clip_limits.zmin = 0.0;
        clip_limits.xmax = clip_limits.ymax = clip_limits.zmax = 1.0;
    /* Set the new view and perform an update if it is needed.  The */
    /* new view is defined by the orientation and mapping matrices. */
    /* Clip_limits specifies the view volume clipping limits.  The  */
    /* last three arguments, the clipping indicators, specify       */
    /* whether each of the planes of the clipping limits is active  */
    /* or not active.  Pause the program execution to allow the     */
    /* user to view the picture.                                    */

        gsetviewrep3 (ws_id, view_index, orientation, mapping, &clip_limits,
                      GNOCLIP, GNOCLIP, GNOCLIP);
        if (view_xforms == GIRG)
           gupdatews (ws_id, GPERFORM);
        gawaitevent( timeout, &event );

       } /* end for k */

}  /* end view_picture */

/********************************************************************/
/*  CLEAN_UP ()                                                     */
/*                                                                  */
/*  This subroutine cleans up the DEC GKS and workstation           */
/*  environments.                                                   */
/*                                                                  */
/********************************************************************/

void clean_up (ws_id)

Gint ws_id;  /* workstation identifier */

{
    /* Deactivate all active workstations and close all open        */
    /* workstations before closing GKS.                             */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ( );

} /* end clean_up */
```

4 Customizing Output

Chapter 2 introduces DEC GKS output generation with descriptions of output functions and segments. Attributes exist for each primitive, for any defined segments. This chapter provides a description of attributes and their use in customizing output. For example, you can generate the same polyline in the same position several times during program execution, but the line can appear differently for each generation, depending on the associated attributes.

This chapter describes the following concepts:

- Geometric and nongeometric output attributes
- Individual and bundled output attributes
- Aspect source flags
- Text attributes
- Segment attributes
- Workstation independent segment storage

4.1 Output Attributes

An output attribute is an aspect of an output primitive that determines how the primitive appears on the surface of the workstation. For example, lines and markers each have color, type, and size attributes. The current attribute settings determine how DEC GKS represents an output primitive.

DEC GKS stores the default output attribute values for a given workstation in the workstation description table, and stores the current values in the GKS and workstation state lists. You can use the inquiry functions to obtain information about the default or current attribute settings.

Each output primitive potentially has four types of attributes:

- Geometric
- Nongeometric
- Viewing
- Identification

Geometric, **nongeometric**, and **viewing attributes** determine the appearance of output primitives. The identification attribute is used in connection with input. The values of all four types of attributes are set modally and are recorded in the GKS state list. A function is provided for each primitive attribute that lets the application program specify the value of an attribute. When one of these output primitive functions is invoked, the attribute values are bound to the primitive and cannot be changed.

Geometric Attributes

Geometric attributes control the shape and size of the whole primitive. Fill area, fill area set, and text are the only output primitives that have changeable geometric attributes. The geometric attributes are device independent. If they represent coordinate data (points), they are stored in WC points. The attributes are subject to the same transformations as the primitive.

Nongeometric Attributes

Nongeometric attributes affect the style and the pattern of the output primitives or the shape and size of its component parts. Because many of the nongeometric attributes involve scale factors and **nominal sizes**, the effects of these attributes are device dependent.

Nominal sizes are the default sizes of markers and line widths as defined by a workstation. In most cases, the nominal size is also the smallest size that a workstation can produce, but not always. To reset a marker size or polyline width, DEC GKS multiplies the scale factor values by the nominal size. The default value for a scale factor is 1.0 (the nominal size multiplied by the value 1.0, producing no change in the default size).

When you alter the nongeometric attributes, you might specify a scale factor creating a size that is larger than the largest size of the workstation. If this happens, the graphics handler ignores the scale specification and uses the largest size defined by the handler. You can obtain the smallest, largest, and nominal sizes by calling one of the INQUIRE ... FACILITIES functions.

Nongeometric attributes may be specified in one of two ways:

- **Bundled** aspects are controlled by a **bundle index** into a **bundle table**. Each entry in the bundle table contains nongeometric aspects of a primitive. When nongeometric attributes are specified through a bundle, they are workstation dependent. Each workstation has its own set of bundle tables, which are stored in the workstation state list. The values in a particular bundle (or entry in the bundle table) may be different for different workstations.
- Individual aspects of a primitive are each specified by a separate attribute. These attributes are workstation independent and are stored in the workstation state list.

For more information on bundled attributes, see Section 4.2.

View Attributes

View attribute is a label that identifies an attribute associated with viewing. There are two view attributes. They are view index and HLHSR identifier. The attribute view index is a pointer to a view table entry at a workstation. There is one view table per workstation. The attribute HLHSR identifier supplies hidden line hidden surface removal information about the primitive to the workstation.

Identification Attributes

There is only one identification attribute. It is pick identifier. This is used to identify a primitive, or a group of primitives, in a segment when that segment is picked.

4.2 Individual and Bundled Attributes

The program in Example 6–1 changes individual attribute settings. When you need to change one attribute value, you call a single output attribute function that alters that setting. The GKS state list stores the current individual attribute setting for each attribute.

You can only alter the geometric output attributes of a primitive individually. When altering the nongeometric attributes of a primitive, you have the option to change the values individually, or change them in a bundle.

Bundles are groupings of nongeometric attribute settings. Each workstation predefines a bundle table for each primitive that has nongeometric attributes. Each table contains all the defined bundle groups for its primitive. To access a given bundle, you must specify an integer index value that refers to one group of bundled settings within the table.

For example, a workstation can predefine the index value 1 to represent a solid green line with a width scale factor of 1.0 (the nominal width). A portion of a default polyline bundle table can be:

Index	Line Type	Line Width	Color Index	Description
1	1	1.0	1	Solid green
2	1	1.0	2	Solid red
3	1	1.0	3	Solid blue
	.			
	.			
	.			
26	4	1.0	5	Dashed-dotted magenta
27	4	1.0	6	Dashed-dotted yellow
28	4	1.0	7	Dashed-dotted black

With all bundle tables, you have the option to use the predefined representations of the index values, to define representations for additional index values, or to redefine an existing representation of index values.

If you define or change a bundle index value used by a primitive already generated on the workstation surface, you can cause an implicit regeneration of the surface, under certain circumstances. See the section on Implicit Regeneration Mode in Section 2.2 for more information on restrictive circumstances. If an implicit regeneration occurs, you lose all primitives not contained in segments. Section 4.2.3 describes representation changes in detail.

DEC GKS stores the current bundle table representations in the workstation state list. Because each graphics handler can predefine different bundle tables with a different number of index values, the bundled attributes are device dependent.

Using bundled attributes saves you time, because you do not have to set the nongeometric attributes individually with separate function calls. With a single function call, you can set a group of attributes.

DEC GKS binds either the individual or the bundled attributes to a primitive at the time of output generation. If you specify the attributes of a primitive individually, you cannot alter its appearance in subsequent portions of the program. If you specified the attributes of a primitive using a bundle index, and if the primitive is in a segment (or if your workstation supports dynamic attribute changes), you can alter the primitive by redefining the representation of its bundle index. You change the representation of a bundle index by calling one of the SET ... REPRESENTATION functions.

Before output generation, DEC GKS must determine whether to use individual or bundled attributes for a specified primitive. To determine which type of attribute to use, DEC GKS checks the **aspect source flag** (ASF) of the attribute. Section 4.2.2 describes aspect source flags in detail.

4.2.1 Binding Attributes to Primitives

The attributes that control the appearance of a GKS picture are stored in three state lists:

- GKS state list

  The GKS state list stores the geometric attributes that control the shape and size of the primitive, and the global (workstation-independent) nongeometric attributes of a primitive. These are set by the output attribute functions described in the GKS binding manuals.

- Segment state list

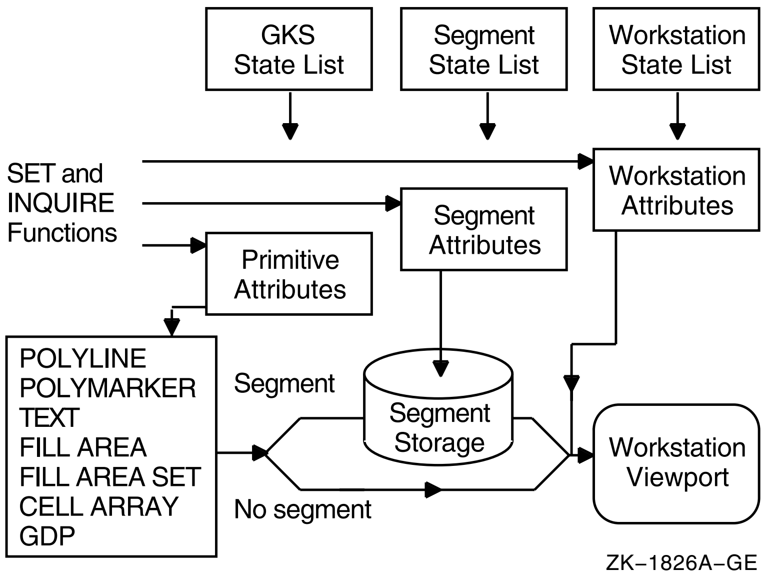
  Primitives and their attributes are grouped together in a segment. In addition to the primitive attributes, there are five segment attributes (transformation, visibility, highlighting, priority, and detectability). Segment attributes apply to all the primitives in a segment. They are set by the GKS segment functions and are described in the GKS binding manuals.

- Workstation state list

  The bundled (workstation-dependent) nongeometric attributes are stored in the workstation state list.

Attributes are bound to a primitive at any of the three stages in the output of a GKS picture as shown in Figure 4–1.

Figure 4–1 Binding of Attributes



The types of attributes bound to primitives are listed in Table 4–1.

Table 4–1 Primitives and Relevant Attributes

Primitive Attribute Type	Attribute Type	Attribute Name	Values
POLYLINE	Nongeometric	Polyline index	Integer
		Line type	Solid, dashed, dotted, dashed-dotted
		Line width scale factor	Real
		Polyline color index	Integer
		Line type ASF	Bundled, individual
		Line width scale factor ASF	Bundled, individual
		Polyline color index ASF	Bundled, individual
	View	View index	Integer
		HLHSR identifier	On, off
	Identification	Pick identifier	On, off
POLYMARKER	Nongeometric	Polymarker index	Integer
		Marker type	Dot (.), plus (+), asterisk (*), circle (O), cross (X)
		Marker size scale factor	Real
		Polymarker color index	Integer
		Marker type ASF	Bundled, individual
		Marker size scale factor ASF	Bundled, individual

(continued on next page)

Table 4–1 (Cont.) Primitives and Relevant Attributes

Primitive Attribute Type	Attribute Type	Attribute Name	Values
		Polymarker color index ASF	Bundled, individual
	View	View index	Integer
		HLHSR identifier	On, off
	Identification	Pick identifier	On, off
TEXT	Nongeometric	Text index	Integer
		Text font and precision	Integer
		Character expansion factor	Real
		Character spacing	Real
		Text color index	Integer
		Text font and precision ASF	Bundled, individual
		Character expansion factor ASF	Bundled, individual
		Character spacing ASF	Bundled, individual
		Text color index ASF	Bundled, individual
	Geometric	Character height	Real
		Character up vector	Real
		Text path	Left, right, up, down
		Text alignment	Horizontal, vertical
	View	HLHSR identifier	Integer
	Identification	Pick identifier	Integer
FILL AREA	Nongeometric	Fill area index	Integer
		Fill area interior style	Hollow, solid, pattern, hatch
		Fill area style index	Integer
		Fill area color index	Integer
		Fill area interior style ASF	Bundled, individual
		Fill area style index ASF	Bundled, individual
		Fill area color index ASF	Bundled, individual
	Geometric	Pattern size	Real
		Pattern reference point and vectors	Real
	View	View index	Integer
		HLHSR identifier	Integer
	Identification	Pick identifier	Integer
FILL AREA SET	Nongeometric	Fill area index	Integer
		Fill area interior style	Hollow, solid, pattern, hatch

(continued on next page)

Table 4–1 (Cont.) Primitives and Relevant Attributes

Primitive Attribute Type	Attribute Type	Attribute Name	Values
		Fill area style index	Integer
		Fill area color index	Integer
		Fill area interior style ASF	Bundled, individual
		Fill area style index ASF	Bundled, individual
		Fill area color index ASF	Bundled, individual
		Edge index	Integer
		Edge flag	Off, on
		Edge type	Integer
		Edge width scale factor	Real
		Edge color index	Integer
		Edge flag ASF	Bundled, individual
		Edge type ASF	Bundled, individual
		Edge width scale factor ASF	Bundled, individual
		Edge color index ASF	Bundled, individual
	Geometric	Pattern size	Real
		Pattern reference point and vectors	Real
	View	View index	Integer
		HLHSR identifier	Integer
	Identification	Pick identifier	Integer
CELL ARRAY	View	View index	Integer
		HLHSR identifier	Integer
	Identification	Pick identifier	Integer
GENERALIZED DRAWING PRIMITIVE	View	View index	Integer
		HLHSR identifier	Integer
	Identification	Pick identifier	Integer

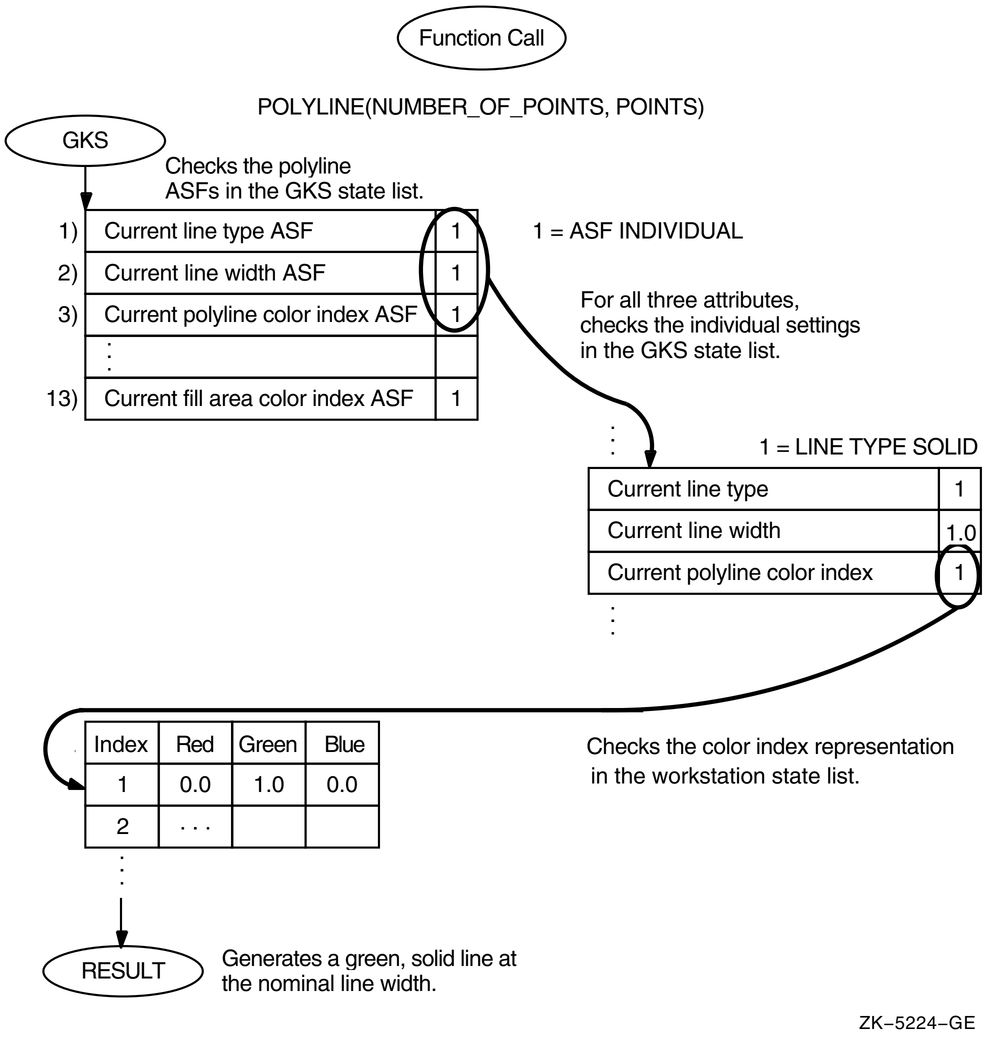
4.2.2 Aspect Source Flags

When you call an output function, DEC GKS must determine whether to use the current individual attributes associated with a primitive, all the attribute values associated with the current bundle index, or some individual settings and some bundled settings.

To determine which setting to use for which nongeometric attribute, DEC GKS checks the current value of the aspect source flag (ASF). The ASFs are elements of the GKS state list that contain either the value BUNDLED (0) or the value INDIVIDUAL (1). If the ASF contains INDIVIDUAL (the default), DEC GKS uses the individual setting for that particular attribute. If the ASF contains BUNDLED, DEC GKS determines the current bundle index, accesses the appropriate bundle table, obtains the value for the particular setting, and uses that setting during output generation.

Figure 4–2 illustrates action taken by DEC GKS for the default polyline ASF settings.

Figure 4–2 Default Polyline Aspect Source Flag Settings

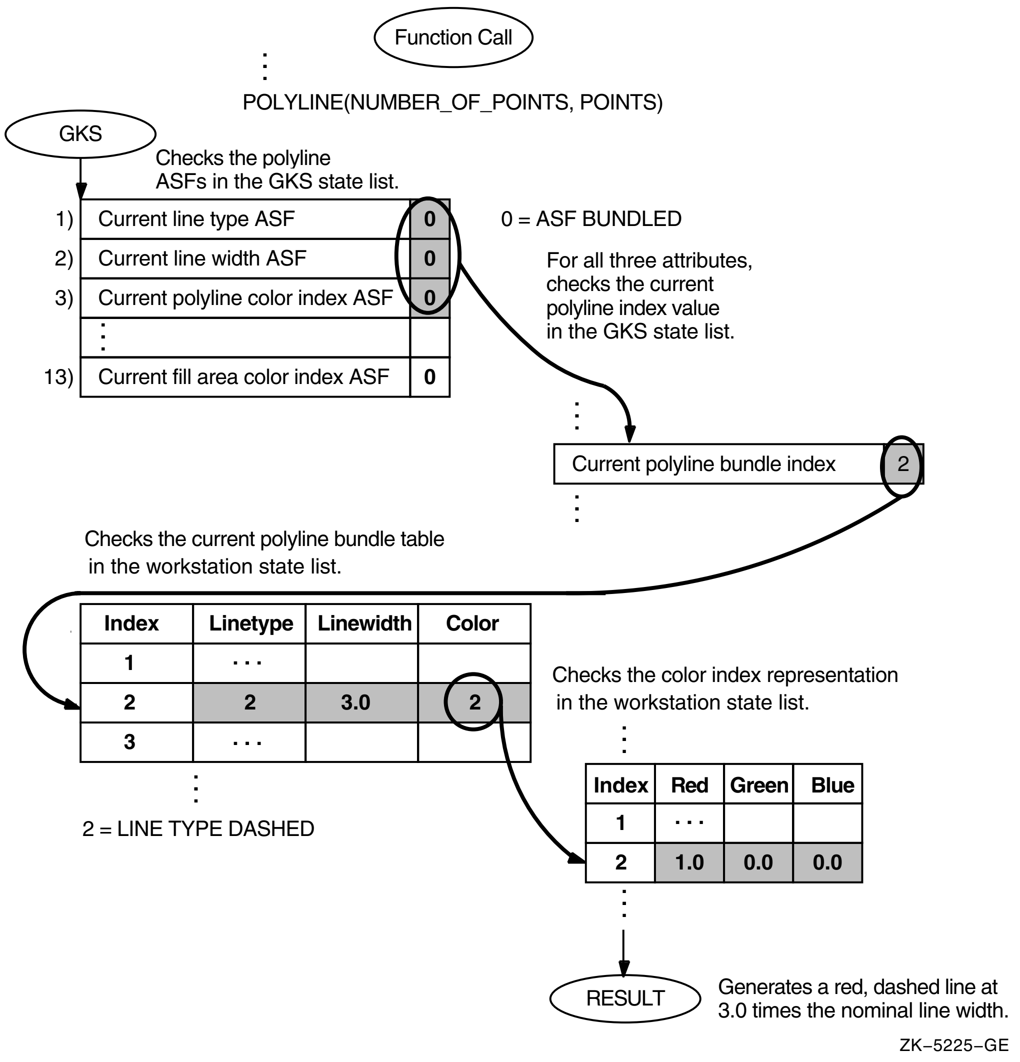

Function Call
POLYLINE(NUMBER_OF_POINTS, POINTS)
GKS
Checks the polyline ASFs in the GKS state list.
1) Current line type ASF 1
2) Current line width ASF 1
3) Current polyline color index ASF 1
13) Current fill area color index ASF 1
1 = ASF INDIVIDUAL
For all three attributes, checks the individual settings in the GKS state list.
1 = LINE TYPE SOLID
Current line type 1
Current line width 1.0
Current polyline color index 1
Index Red Green Blue
1 0.0 1.0 0.0
2 . . .
Checks the color index representation in the workstation state list.
RESULT
Generates a green, solid line at the nominal line width.
ZK–5224–GE

There are a total of 17 nongeometric attributes. Each attribute has an ASF associated with it. Depending on the binding you use, you will need to use an array or a structure to set the flags using the SET ASPECT SOURCE FLAGS (3) function. See the DEC GKS binding manuals for more information. Example 4–1 at the end of this chapter also illustrates the use of these functions.

When working with bundled attribute values, you can either allow the workstation to use the default index value or specify a new index value. To specify a new bundle index value, use a SET ... INDEX function after defining the index with a SET ... REPRESENTATION function.

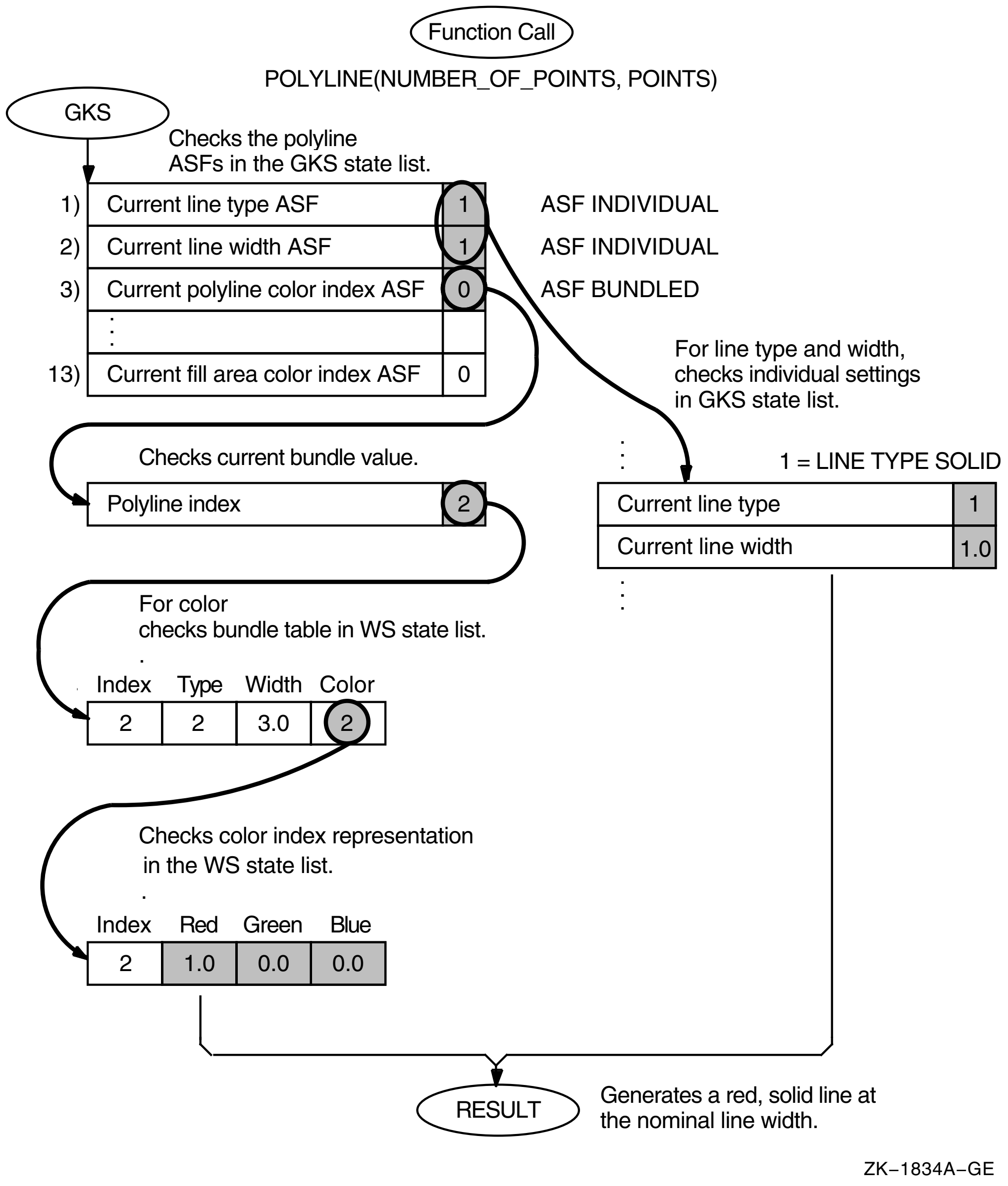
Figure 4–3 illustrates what happens if you pass BUNDLED in the first three elements of the integer array (which correspond to line type, line width, and line color).

Figure 4–3 Specifying Bundled Aspect Source Flag Settings


Function Call
POLYLINE(NUMBER_OF_POINTS, POINTS)
GKS
Checks the polyline ASFs in the GKS state list.
1) Current line type ASF 0
2) Current line width ASF 0
3) Current polyline color index ASF 0
13) Current fill area color index ASF 0
0 = ASF BUNDLED
For all three attributes, checks the current polyline index value in the GKS state list.
Current polyline bundle index 2
Checks the current polyline bundle table in the workstation state list.
Index Linetype Linewidth Color
1 . . .
2 2 3.0 2
3 . . .
2 = LINE TYPE DASHED
Checks the color index representation in the workstation state list.
Index Red Green Blue
1 . . .
2 1.0 0.0 0.0
RESULT
Generates a red, dashed line at 3.0 times the nominal line width.
ZK−5225−GE

DEC GKS treats each nongeometric attribute value separately according to its current ASF. Depending on the ASF value, DEC GKS can use either individual settings or bundled settings for the generation of an output primitive. Figure 4–4 illustrates what happens if you define BUNDLED for some polyline ASF values and INDIVIDUAL for others. If an ASF is set to BUNDLED and you have not set a bundle index, DEC GKS uses bundle index 1 by default.

Figure 4–4 Specifying Bundled and Individual Aspect Source Flags


Function Call
POLYLINE(NUMBER_OF_POINTS, POINTS)
GKS
Checks the polyline ASFs in the GKS state list.
1) Current line type ASF 1
2) Current line width ASF 1
3) Current polyline color index ASF 0
13) Current fill area color index ASF 0
ASF INDIVIDUAL
ASF INDIVIDUAL
ASF BUNDLED
For line type and width, checks individual settings in GKS state list.
Checks current bundle value.
Polyline index 2
1 = LINE TYPE SOLID
Current line type 1
Current line width 1.0
For color checks bundle table in WS state list.
Index Type Width Color
2 2 3.0 2
Checks color index representation in the WS state list.
Index Red Green Blue
2 1.0 0.0 0.0
RESULT
Generates a red, solid line at the nominal line width.
ZK−1834A−GE

4.2.3 Bundle Index Representations

The separate nongeometric attribute settings that comprise an attribute bundle are the representation of the bundle index. For example, the predefined representation of polyline bundle index 3 for a DECwindows workstation includes a solid line type, a nominal line width, and a color index of 3. The predefined color representation for the DECwindows workstation for index number 3 is the predefined color green. The following table illustrates the representation of bundle index 3:

Index	Line Type	Line Width	Color Index	Description
	.			
	.			
	.			
3	1	1.0	3	Thin, green solid
	.			
	.			
	.			

A workstation supports a given number of bundle representations. Of that maximum number of bundle indexes, the workstation can predefine any number of them.

Bundle representations are not static. You can change the attribute settings associated with a predefined bundle index, or you can establish a new representation for a supported index value that was not predefined. To establish or change a representation associated with a given index value, you use the SET ... REPRESENTATION functions (SET POLYLINE REPRESENTATION, SET POLYMARKER REPRESENTATION, and so on).

When you change individual attribute settings, the change does not take place until a subsequent generation of the appropriate output primitive. This is not true for bundle representation changes. A change to a bundle representation affects previously generated primitives whose attributes are bound to that bundle representation. When you call the SET ... REPRESENTATION functions, DEC GKS can either make the change immediately or require an implicit regeneration of the workstation surface to make the change. Chapter 2 describes surface regeneration and the use of the functions SET DEFERRAL STATE and UPDATE WORKSTATION to control the workstation surface. Example 4–1 at the end of this chapter demonstrates the use of these functions.

4.2.4 Text Attributes

When you work with normalization transformation changes, you may find that you need to alter the text attributes to maintain the size and proportion of the character string.

For example, if you map a normalization window onto a small portion of the default NDC space ([0,1] x [0,1] x [0,1]), the text maintains its relative position, alignment, color, font, precision, path, and direction. However, the text may then appear small, crowded, and possibly skewed when mapped to the workstation viewport.

DEC GKS calculates the character spacing and width according to the current text height (default is 0.01). Consequently, you can adjust all three attributes by adjusting the character height to the current normalization transformation proportions.

If you use normalization transformations whose viewport boundaries are disproportionate to the window boundaries (for example, mapping a square to a wide rectangle), you may need to adjust the character expansion factor so that the text width is adequate. For more information concerning character expansion, see the chapter on attribute functions in your DEC GKS binding manual.

4.3 Using Segments

Segments, collections of primitives treated as single entities, are introduced in Chapter 2. Both Chapter 2 and Section 4.1 describe the pictorial influence of the output attributes of primitives contained in a segment. Chapter 2 also describes the importance of storing primitives in a segment to prevent image deletion during surface regenerations. This section describes the segment attributes applicable to the segment as a whole and introduces the concept of **workstation independent segment storage** (WISS), which allows further segment and image manipulations.

4.3.1 Segment Attributes

A segment has five attributes: visibility, highlighting, priority, detectability, and a segment transformation matrix.

Visibility indicates whether a particular segment is displayed or not displayed. **Highlighting** indicates whether a visible segment is displayed in a normal or highlighted manner. If parts of segments overlap, the **priority** determines which segment is in front of another. **Detectability** determines whether the segment can be selected by a pick input device. See Chapter 5 for more information on picking.

The **segment transformation** attribute is a matrix designating values for scaling, rotation, and translation of the segment. To build a segment transformation matrix, use the EVALUATE TRANSFORMATION MATRIX (3) or ACCUMULATE TRANSFORMATION MATRIX (3) function. Set the transformation matrix attribute with the SET SEGMENT TRANSFORMATION (3) function. The default segment transformation matrix is the identity transformation matrix, which makes no changes to segment primitives. When generating a picture containing a segment, DEC GKS performs tasks in the following order:

1. Applies the current normalization transformation to the primitives in the segment
2. Applies the current segment transformation to the primitives in the segment
3. Clips each of the segment primitives according to the clipping volume and the clipping indicator associated with the primitive when it was put into the segment
4. If clipping is enabled, clips the entire picture at the current clipping volume (normalization viewport in NDC space)

4.3.2 Workstation Independent Segment Storage

Segment storage can be workstation dependent or workstation independent. Up to this point, we have described workstation dependent segment storage (WDSS). Three reasons for putting primitives in segments include:

- To keep the primitives from being deleted upon surface regeneration
- To take advantage of having output attributes bound to the primitives at the time of generation
- To take advantage of the segment attributes that allow visual manipulation of a group of primitives as a single entity

Using workstation independent segment storage (WISS), which is a device-independent storage structure, permits transportation of output primitives from WISS to other workstations. WISS is a data structure that stores information pertinent to the primitives contained in a segment. DEC GKS treats WISS as a workstation. To open and activate WISS use the OPEN WORKSTATION and ACTIVATE WORKSTATION functions. The workstation type is the binding-dependent constant representing the WISS workstation type.

Once you store all desired segments on WISS, you can deactivate the workstation using the DEACTIVATE WORKSTATION function. However, you cannot close WISS until you have completely finished all WISS manipulations. In other words, you cannot copy segments from WISS to other open workstations unless WISS is open. When you are finished using WISS, you can close WISS with the CLOSE WORKSTATION function. Once you close WISS, DEC GKS deletes all stored segments in WISS.

There are three ways to transport a segment, or its primitives, from WISS to other open workstations:

1. Associate the segment with a specified open workstation.
2. Copy the segment's primitives to a specified open workstation.
3. Insert the segment's primitives into the viewing pipeline of a specific open workstation.

The ASSOCIATE SEGMENT WITH WORKSTATION function copies a segment to the WDSS of a specified workstation in much the same way that a segment is created when the workstation is active. Clipping volumes and clipping indicators are copied unchanged.

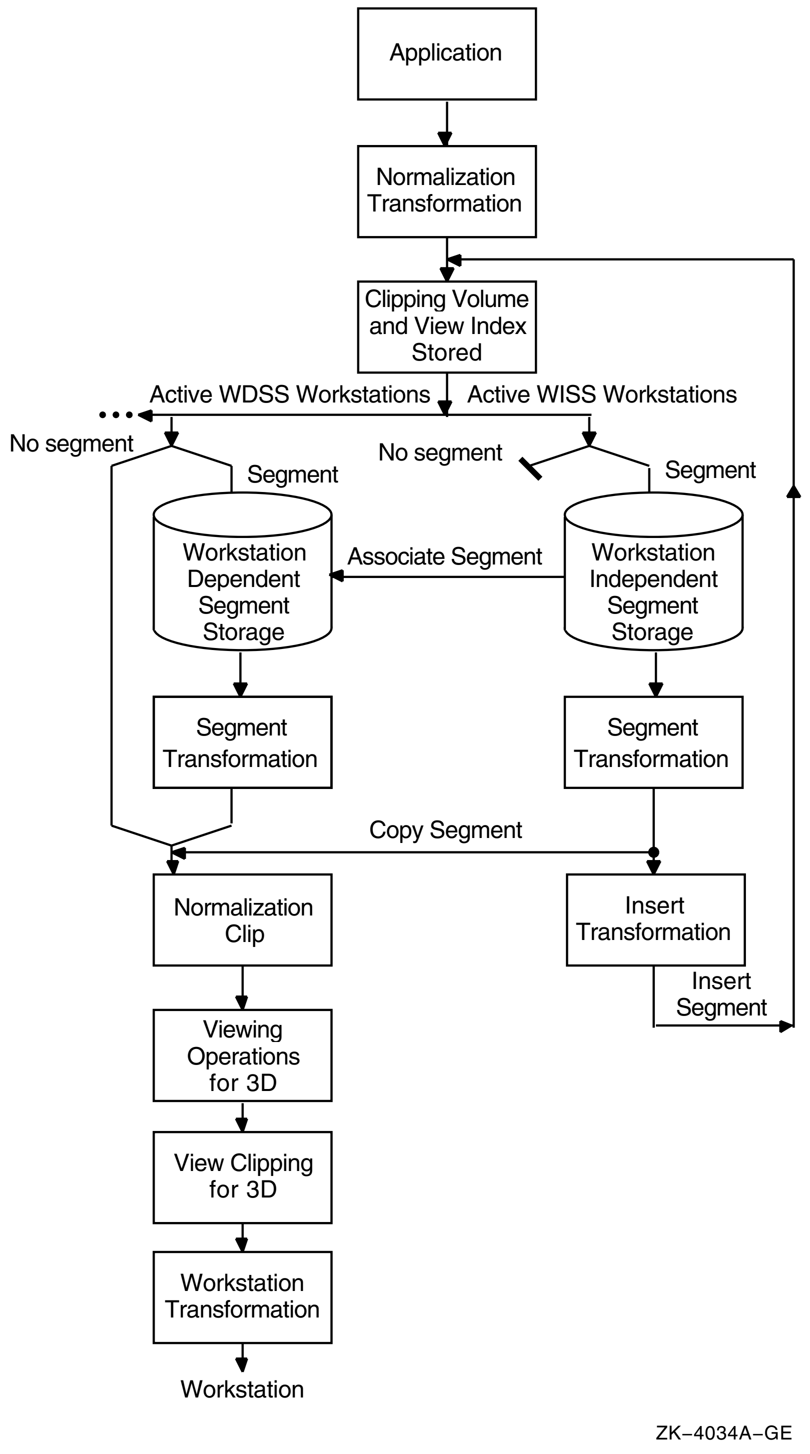
The COPY SEGMENT TO WORKSTATION function copies primitives from a segment in WISS to be output on a specified workstation. The function takes a copy of each primitive, and its associated clipping volume and clipping indicator, from a segment in WISS. The function then transforms the primitives by the segment transformation and puts the clipping volumes and the transformed primitives in the viewing pipeline at the place equivalent to where the information was removed from.

The INSERT SEGMENT (3) function allows previously stored primitives (in segments in WISS) to be transformed and again placed into the stream of output primitives. INSERT SEGMENT (3) reads the primitives from a segment in WISS, applies the segment transformation followed by the **insert transformation**, and then inserts the primitives into the viewing pipeline at the point before data is distributed to the workstations. All clipping volumes, clipping indicators, view indexes, and HLHSR identifiers in the inserted segment are ignored. Each processed primitive is assigned a new clipping volume, clipping indicator, view index, and HLHSR identifier from the GKS state list. In other words, inserted primitives are assigned clipping volumes, clipping indicators, view indexes, and HLHSR identifiers in the same manner as directly created primitives. Thus, all primitives processed by a single invocation of INSERT SEGMENT (3) receive the same clipping volume, clipping indicator, view index, and HLHSR identifier. Inserted information may be reentered in the WISS, if the WISS is active and a segment is open.

The difference between copying and inserting a segment is that you can insert a segment's primitives into an open segment, but you cannot copy a segment's primitives into an open segment. The receiving workstation does not treat the inserted set of segment primitives as a segment, but does add those transformed primitives to the segment being created. If you insert a segment when there is no segment open, segment insertion transforms the segment and then copies the primitives to the workstation surface. Another difference between copying and inserting segments is the existence of the insert transformation.

During segment insertion, DEC GKS allows you to specify an additional segment transformation matrix to apply to the inserted segment primitives. A segment transformation allows you to scale, rotate, and shift (or translate) all of the primitives in the segment. Figure 4–5 illustrates the WDSS and WISS pipeline.

Figure 4–5 The Segment Pipeline



4.4 Program Example Used in this Chapter

Example 4–1 is located on the VMS kit in the following directory:

```
sys$common:[syshlp.examples.gks]
```

Example 4–1 is located on the ULTRIX kit in the following directory:

```
/usr/lib/GKS/examples
```

Example 4–1 presents an extension to Example 3–1 to include the functionality described in this chapter.

Example 4–1 Using DEC GKS Output Functions

```
/* Header files */

#include <stdio.h>                  /* standard C library I/O header file */
#include <gks.h>                    /* GKS C binding header file */

                                    /* Constant definitions */
#define     NUM_SIDE_COLORS  6
#define     NUM_ROAD_COLORS  2
#define     NUM_STAR_PTS     6
#define     NUM_MOON_PTS     2
#define     NUM_TREE_PTS    29
#define     NUM_LROOF_PTS    4
#define     NUM_RROOF_PTS    4
#define     NUM_FRONT_PTS   10
#define     NUM_BACK_PTS     6
#define     NUM_RIGHT_PTS    5
#define     NUM_LEFT_PTS     5
#define     NUM_LAND_PTS    15
#define     NUM_SIDE_PTS     4
#define     NUM_ROAD_PTS     4
#define     BACK_IND         0
#define     RED_IND          1
#define     GREEN_IND        2
#define     BLUE_IND         3

static Gfile  *error_file;          /* error file */

                                    /* event and timeout are used with the
                                       AWAIT EVENT call to pause program
                                       execution */
Gevent  event;                      /* an input event */
Gfloat  timeout = 3.00;             /* pause time in seconds */
/*******************************************************************/
/*                                                                 */
/*  MAIN ()                                                        */
/*                                                                 */
/*******************************************************************/

main ( )

{
    void    set_up( );
    void    draw_picture( );
    void    title_attrib( );
    void    seg_attrib( );
    void    clean_up( );
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    Gint    ws_id   = 1;         /* workstation identifier */
    Gint    wiss    = 2;         /* workstation independent segment
                                         storage identifier */
    Gint    title   = 1,         /* segment identifier for the title */
            stars   = 2,         /* segment identifier for the stars
                                    and moon */
            tree    = 3,         /* segment identifier for the tree */
            side    = 4,         /* segment identifier for the sidewalk */
            road    = 5,         /* segment identifier for the road */
            land    = 6,         /* segment identifier for the land */
            house   = 7;         /* segment identifier for the house */

    Gwstype ws_type = GWS_DEF;        /* workstation type */
    Gconn   conid   = GWC_DEF;        /* connection identifier */
    Gwstype wiss_ws_type = GWS_WISS;  /* WISS workstation type */

    set_up (ws_id, &ws_type, conid, wiss, wiss_ws_type);

    draw_picture (ws_type, ws_id, wiss, title, stars, tree, side, road,
                  house, land);

    /* Update the picture and pause the program execution to allow  */
    /* the user to view the picture.                                */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

    title_attrib (ws_id, title, stars, tree, side, road, house, land);

    seg_attrib (ws_id, ws_type, title, stars, tree, side, road, house, land);

    clean_up (ws_id, wiss);

} /* end main */

/*********************************************************************/
/*  SET_UP ()                                                        */
/*                                                                   */
/*  This subroutine sets up the DEC GKS and workstation              */
/*  environments.                                                    */
/*                                                                   */
/*********************************************************************/

void set_up (ws_id, ws_type, conid, wiss, wiss_ws_type)

Gint       ws_id;        /* workstation identifier */
Gwstype   *ws_type;      /* workstation type */
Gconn      conid;        /* connection identifier */
Gint       wiss;         /* WISS workstation identifier */
Gwstype    wiss_ws_type; /* WISS workstation type */
{
    Glong      memory = GDEFAULT_MEM_SIZE;  /* memory size */
    Gint       buf_size;                    /* buffer size */
    Gwscat     category;                    /* workstation category */
    Gchar      conid_buff[80];              /* buffer for workstation
                                               connection identifier */
    Gwsct      ct;                          /* workstation connection
                                               identifier and type */
    Gdefmode   def_mode;                    /* deferral mode */
    Girgmode   irg_mode;                    /* implicit regeneration mode */
    Gint       ret_size;                    /* returned size */
    Gint       status;                      /* returned status */
    Glevel     gks_level;                   /* level of GKS implementation */
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Initialize GKS.                                             */
/* A pointer to a specified error file is passed in the call to */
/* OPEN GKS.  DEC GKS writes all errors to this file.          */

error_file = fopen ("example_4_1.error", "w");
status = gopengks (error_file, memory);
if (status != NO_ERROR)
   {
    gemergencyclosegks ( );
    fprintf (error_file, "Error opening GKS.\n");
    exit (0);
   }

/* Open and activate a workstation.                            */
/* Both the workstation type and the connection have been      */
/* initialized to 0 so DEC GKS reads the VMS logical names,    */
/* GKS$WSTYPE and GKS$CONID, or the ULTRIX enviroment variables,*/
/* GKSwstype and GKSconid, for the values.                      */

status = gopenws (ws_id, &conid, ws_type);
if (status != NO_ERROR)
   {
    gemergencyclosegks ( );
    fprintf (error_file, "Error opening workstation.\n");
    exit (0);
   }
gactivatews (ws_id);

/* Check that WISS is supported by this implementation of GKS.  */

ginqlevelgks (&gks_level, &status);
if ((status != NO_ERROR) || (gks_level < GL2A))
   {
    gemergencyclosegks ( );
    fprintf (error_file, "Error - level of GKS does not support WISS.\n");
    exit (0);
   }

/* Open a WISS workstation.                                    */

status = gopenws (wiss, &conid, &wiss_ws_type);
if (status != NO_ERROR)
   {
    gemergencyclosegks ( );
    fprintf (error_file, "Error opening WISS workstation.\n");
    exit (0);
   }

/* Defer output as long as possible and suppress implicit      */
/* regeneration.                                               */
/* The deferral mode specifies when calls to the output        */
/* functions have their effect.  The implicit regeneration mode */
/* specifies when changes to the existing picture are seen.    */
/* This application chooses to control the time when attribute  */
/* changes are seen on the display so it suppresses the        */
/* implicit regeneration mode in case it is IMMEDIATE for the   */
/* workstation.                                                */

def_mode = GASTI;
irg_mode = GSUPPRESSED;
gsetdeferst (ws_id, def_mode, irg_mode);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    /* Determine the workstation connection and type.              */
    /* The application specifies the default workstation and type,  */
    /* which are defined by environment options.  This inquiry      */
    /* returns the connection and type to the application program   */
    /* so it has the information.                                   */

    buf_size = sizeof(conid_buff);
    ct.conn = conid_buff;
    ct.type = ws_type;
    ginqwsconntype (ws_id, buf_size, &ret_size, &ct, &status);
    if (status)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error inquire workstation connection/type.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

    /* Inquire about the category of the workstation. The inquiry   */
    /* function verifies the workstation has a workstation type     */
    /* that indicates it can perform output.                        */

    ginqwscategory (ws_type, &category, &status);
    if ( (status) || ((category != GOUTIN) && (category != GOUTPUT)) )
       {
        gemergencyclosegks ( );
        fprintf (error_file, "The workstation category is invalid.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

} /* end set_up */

/*******************************************************************/
/*  DRAW_PICTURE ()                                                 */
/*                                                                  */
/*  This subroutine draws the picture and places each primitive in  */
/*  a segment.  Subroutine draw_house is called to draw the house.  */
/*  All objects, except the house, are defined within the default   */
/*  world coordinate  range ([0, 1] x [0, 1] x [0, 1]).  If the     */
/*  workstation has sufficient color capabilities, the objects are  */
/*  drawn in four colors which are defined with the SET COLOUR      */
/*  REPRESENTATION function.  This code assumes an RGB color model. */
/*  Device-independent code should inquire about the available      */
/*  color models and set the color model explicitly.  If the        */
/*  workstation is monochrome, the objects are drawn in the         */
/*  default foreground color.                                       */
/*                                                                  */
/*******************************************************************/

void draw_picture (ws_type, ws_id, wiss, title, stars, tree, side,
                   road, house, land)
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
Gwstype ws_type;  /* workstation type */
Gint    ws_id,    /* workstation identifier */
        wiss,     /* WISS workstation identifier */
        title,    /* segment identifier for the title */
        stars,    /* segment identifier for the stars */
        tree,     /* segment identifier for the tree */
        side,     /* segment identifier for the sidewalk */
        road,     /* segment identifier for the road */
        house,    /* segment identifier for the house */
        land;     /* segment identifier for the land */

{
    void     fill_attrib( );
    Gint     unity = 0;            /* normalization transformation index */
    Gint     house_norm = 1;       /* normalization transformation index */
    Glimit3  viewport;             /* normalization viewport */
    Glimit3  window;               /* normalization window */

                                   /* array of the sidewalk colors for the
                                      cellarray call */
    static Gint  side_colors[NUM_SIDE_COLORS] = {1, 2, 3, 1, 2, 3},
                                   /* array of the road colors for the
                                      cellarray call */
             road_colors[NUM_ROAD_COLORS] = {1, 3};
    Gint     status,               /* returned status */
             bufsize,              /* buffer size */
             colour,               /* flag to indicate if the workstation
                                      is color or monochrome */
             fac_size;             /* number of facilities */
    Gfloat   char_height = 0.04,   /* character height */
             line_width  = 3.0,    /* line width */
             edge_width = 2.0;     /* edge width */
    Glnfac   line_fac;             /* line facility */
    Gintlist line_types_st;        /* integer list stucture of line types */
    Gint     line_types[20];       /* list of line types */
    Gcofac   col_fac;              /* color facility */
    static Gchar *title_str = "Starry Night";  /* title string */

                                   /* 3 sidewalk "corner" points for the
                                      cellarray call */
    static Grect3 side_rectangle_coordinates = {{0.2,  0.0, 0.8},
                                                {0.25, 0.0, 0.8},
                                                {0.2,  0.3, 0.7}};

                                   /* 3 road "corner" points for the
                                      cellarray call */
    static Grect3 road_rectangle_coordinates = {{0.0, 0.0, 1.0},
                                            {0.0, 0.0, 0.8},
                                            {1.0, 0.0, 1.0}};

                                   /* dimensions of the array with the
                                      with the sidewalk colors */
    static Gidim side_rectangle_dim = {1, 6},
                                   /* dimensions of the array with the
                                      with the road colors */
             road_rectangle_dim = {2, 1};

                                   /* color definitions with RGB values */
    static Gcobundl black = {0.0, 0.0, 0.0};
    static Gcobundl red   = {1.0, 0.0, 0.0};
    static Gcobundl green = {0.0, 1.0, 0.0};
    static Gcobundl blue  = {0.0, 0.0, 1.0};
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
Ggdprec  moon_data_rec;  /* GDP data record for the moon */
                                /* moon point values for the GDP */
static Gpoint   moon_pts[NUM_MOON_PTS] = { {0.9,0.9}, {0.9,0.84} };

                                /* number of points in each of the fill
                                   areas comprising the fill area set */
static Gint     front_sizes[2] = {6, 4};

                                /* text font and precision */
static Gtxfp text_font_prec = {1, GP_STROKE};
                                 /* text position */
static Gpoint3  title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                text_vec1 = {1.0, 0.0, 0.0},
                text_vec2 = {0.0, 1.0,-1.0},
                                /* star coordinates */
                stars_pts[NUM_STAR_PTS] =
                  { {0.05,0.70,0.0},  {0.06,0.86,0.0},   {0.36,0.81,0.0},
                    {0.66,0.86,0.0},  {0.835,0.7,0.0},   {0.92,0.82,0.0} },

                                /* land coordinates */
                land_pts[NUM_LAND_PTS] =
                  { {0.0,0.35,0.0},   {0.04,0.375,0.0},  {0.055,0.376,0.0},
                    {0.08,0.36,0.0},  {0.1,0.365,0.0},   {0.3,0.366,0.0},
                    {0.375,0.38,0.0}, {0.44,0.385,0.0},  {0.49,0.375,0.0},
                    {0.56,0.36,0.0},  {0.68,0.38,0.0},   {0.8,0.35,0.0},
                    {0.9,0.359,0.0},  {0.95,0.375,0.0},  {1.0,0.385,0.0} },

                                /* tree coordinates */
                tree_pts[NUM_TREE_PTS] =
                  { {0.425,0.28,0.0}, {0.5,0.3,0.0},     {0.52,0.26,0.0},
                    {0.54,0.3,0.0},   {0.6,0.28,0.0},    {0.575,0.33,0.0},
                    {0.56,0.42,0.0},  {0.559,0.49,0.0},  {0.64,0.53,0.0},
                    {0.69,0.57,0.0},  {0.689,0.61,0.0},  {0.66,0.64,0.0},
                    {0.63,0.66,0.0},  {0.645,0.71,0.0},  {0.59,0.76,0.0},
                    {0.53,0.78,0.0},  {0.48,0.75,0.0},   {0.45,0.71,0.0},
                    {0.42,0.65,0.0},  {0.375,0.645,0.0}, {0.35,0.6,0.0},
                    {0.375,0.55,0.0}, {0.44,0.54,0.0},   {0.45,0.5,0.0},
                    {0.515,0.5,0.0},  {0.51,0.425,0.0},  {0.495,0.38,0.0},
                    {0.475,0.33,0.0}, {0.425,0.28,0.0} },
                                /* house front coordinates */
                front_pts[NUM_FRONT_PTS] =
                  { {100.0,300.0,700.0}, {300.0,300.0,700.0},
                    {300.0,600.0,700.0}, {200.0,750.0,700.0},
                    {100.0,600.0,700.0}, {100.0,300.0,700.0},
                    {200.0,300.0,700.0}, {250.0,300.0,700.0},
                    {250.0,400.0,700.0}, {200.0,400.0,700.0} },

                                /* house back coordinates */
                back_pts[NUM_BACK_PTS] =
                  { {100.0,300.0,300.0}, {100.0,600.0,300.0},
                    {200.0,750.0,300.0}, {300.0,600.0,300.0},
                    {300.0,300.0,300.0}, {100.0,300.0,300.0} },

                                /* house right side coordinates */
                right_pts[NUM_RIGHT_PTS] =
                  { {300.0,300.0,700.0}, {300.0,300.0,300.0},
                    {300.0,600.0,300.0}, {300.0,600.0,700.0},
                    {300.0,300.0,700.0} },
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
                                    /* house left side coordinates */
                    left_pts[NUM_LEFT_PTS] =
                      { {100.0,300.0,300.0}, {100.0,300.0,700.0},
                        {100.0,600.0,700.0}, {100.0,600.0,300.0},
                        {100.0,300.0,300.0} },

                                    /* roof left side coordinates */
                    lroof_pts[NUM_LROOF_PTS] =
                      { {100.0,600.0,700.0}, {200.0,750.0,700.0},
                        {200.0,750.0,300.0}, {100.0,600.0,300.0} },

                                    /* roof right side coodinates */
                    rroof_pts[NUM_RROOF_PTS] =
                      { {300.0,600.0,700.0}, {300.0,600.0,300.0},
                        {200.0,750.0,300.0}, {200.0,750.0,700.0} },

                                    /* sidewalk coordinates */
                    side_pts[NUM_SIDE_PTS] =
                      { {0.2,0.0,0.8},   {0.25,0.0,0.8},
                        {0.25,0.3,0.7},  {0.2,0.3,0.7} },

                                    /* road coordinates */
                    road_pts[NUM_ROAD_PTS] =
                      { {0.0,0.0,1.0},    {1.0,0.0,1.0},
                        {1.0,0.0,0.8},    {0.0,0.0,0.8} };

/* Check if the initialized line width is too wide.           */
/* The value of line_width is initialized to 3.00.  To avoid    */
/* requesting a line width that is wider than the workstation's */
/* widest line, call INQUIRE LINE FACILITIES.  If the line is   */
/* too wide, set it to the widest available width.            */

bufsize = sizeof(line_types);
line_types_st.integers = line_types;
line_fac.types = &line_types_st;
ginqlinefacil (&ws_type, bufsize, &fac_size, &line_fac, &status);
if (line_width * line_fac.nom_width > line_fac.max_width)
    line_width = line_fac.max_width / line_fac.nom_width;

/* Check if you are working with a color workstation.          */
/* If 4 colors are available, set each of the 4 color          */
/* representation indices for the specified workstation to the */
/* desired colors.  This type of coding is useful since        */
/* different workstations may have different default color     */
/* representations.                                            */

ginqcolourfacil (&ws_type, bufsize, &fac_size, &col_fac, &status);
colour = (col_fac.coavail == GCOLOUR) && (col_fac.predefined >= 4);
if (colour)
   {
    gsetcolourrep (ws_id, BACK_IND, &black);
    gsetcolourrep (ws_id, RED_IND, &red);
    gsetcolourrep (ws_id, GREEN_IND, &green);
    gsetcolourrep (ws_id, BLUE_IND, &blue);
   }


/* Activate the WISS workstation so the segments are stored in */
/* workstation independent segment storage.                    */

gactivatews (wiss);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Call subroutine fill_attrib to set the bundled attributes   */
/* for fill areas.  After this call, all fill areas will use a */
/* fill sytle of "hatch" for the rest of the program.          */

fill_attrib (ws_id, ws_type);


/* Create a segment for the title.                             */
/* If the workstation is color, make the text red. Set the     */
/* character height, and text precision and font.  All values  */
/* are initialized above. Use TEXT 3 to generate the string.   */
/* The position of the text string, the text vectors, and the  */
/* text string are initialized above.                          */

gcreateseg (title);
   if (colour)
       gsettextcolourind (RED_IND);
   gsetcharheight (char_height);
   gsettextfontprec (&text_font_prec);
   gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
gcloseseg ( );

/* Create a segment for the stars and moon.                    */
/* If the workstation is color, make the stars blue. Make the  */
/* star shape a '+' and set the marker size scale factor. Use  */
/* POLYMARKER 3 to generate the star markers. The macro        */
/* NUM_STAR_PTS specifies the number of points. The list of    */
/* positions, stars_pts, is initialized above.                 */
/* Use a generalized drawing primitive (GDP) for the moon.     */
/* The moon object uses GDP_FCCP (-333), which is a filled     */
/* circle described by the center and a point on the           */
/* circumference.  The description in the Device Specifics     */
/* Reference Manual indicates that the data record is null so  */
/* the numbers of integers, floats, and strings in the data    */
/* record are set to 0.  The number of points, 2, is defined   */
/* by the macro NUM_MOON_PTS, and the center and circumference */
/* point are passed in the list moon_pts.  GDP_FCCP is a macro */
/* defining the GDP number.  It is in the C binding include    */
/* file.                                                       */

gcreateseg (stars);
   if (colour)
       gsetmarkercolourind (BLUE_IND);
   gsetmarkertype (GMK_PLUS);
   gsetmarkersize (2.0);
   gpolymarker3 (NUM_STAR_PTS, stars_pts);
   gsetfillintstyle (GSOLID);
   moon_data_rec.gdp_datarec.number_integer = 0;
   moon_data_rec.gdp_datarec.number_float = 0;
   moon_data_rec.gdp_datarec.number_strings = 0;
   ggdp (NUM_MOON_PTS, moon_pts, GDP_FCCP, &moon_data_rec);
gcloseseg ( );
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Create a segment for the tree.                              */
/* If the workstation is color, make the tree green. Set the   */
/* fill area interior style to solid so the tree is a solid   */
/* shape. Draw the fill area described by the number of points */
/* defined in the NUM_TREE_PTS macro and the list of points in */
/* tree_pts.  The points are initialized above.                */

gcreateseg (tree);
   if (colour)
       gsetfillcolourind (GREEN_IND);
   gsetfillintstyle (GSOLID);
   gfillarea3 (NUM_TREE_PTS, tree_pts);
gcloseseg ( );

/* Create a segment for the sidewalk.                          */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the sidewalk.  If it is monochrome, use FILL AREA 3.        */
/* The CELL ARRAY 3 function divides a parallelogram into      */
/* cells and displays each cell in a specified color.  The     */
/* call requires 3 points on the parallelogram (lower left     */
/* front corner, upper right front corner and upper right back */
/* corner), the number of rows and columns into which the      */
/* parallelogram will be divided, and a 2-dimensional array    */
/* containing the color index values of the cells.  The        */
/* dimensions of the color index array correspond to the       */
/* dimension (number of rows and columns) of the parallelogram */
/* for the C binding.                                          */

gcreateseg(side);
    if (colour)
       gcellarray3 (&side_rectangle_coordinates, &side_rectangle_dim,
                    side_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_SIDE_PTS, side_pts);
       }
gcloseseg ( );

/* Create a segment for the road.                              */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the road.  If it is monochrome, use FILL AREA 3.  All       */
/* values are initialized above.                               */

gcreateseg(road);
    if (colour)
       gcellarray3 (&road_rectangle_coordinates, &road_rectangle_dim,
                    road_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_ROAD_PTS, road_pts);
       }
gcloseseg ( );
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Create a segment for the land.                            */
/* If the workstation is color, make the line outlining the   */
/* land green. Set the line width, make the line dashed, and   */
/* use the POLYLINE 3 function to draw the land horizon.      */

gcreateseg (land);
   if (colour)
       gsetlinecolourind (GREEN_IND);
   gsetlinewidth (line_width);
   gsetlinetype (GLN_DASHED);
   gpolyline3 (NUM_LAND_PTS, land_pts);
gcloseseg ( );

/* Set the normalization transformation so the house maps onto */
/* a small portion of NDC space.                               */

window.xmin =  90.0;
window.xmax = 310.0;
window.ymin = 290.0;
window.ymax = 760.0;
window.zmin = 290.0;
window.zmax = 710.0;
gsetwindow3 (house_norm, &window);
viewport.xmin = 0.090;
viewport.xmax = 0.310;
viewport.ymin = 0.290;
viewport.ymax = 0.760;
viewport.zmin = 0.290;
viewport.zmax = 0.710;
gsetviewport3 (house_norm, &viewport);
gselntran (house_norm);

/* Create a segment for the house and draw it.                */
/* If the workstation is color, make the back of the house     */
/* green, the front of the house blue, the left side and right */
/* side of the house red, and the left side and right side of  */
/* the roof blue.  Use FILL AREA 3 to draw the back, left      */
/* side, and right side of the house, and the left and right   */
/* sides of the roof.  Use FILL AREA SET 3 to draw the front   */
/* of the house.  FILL AREA SET 3 draws the door and the front */
/* of the house.  The area for the door is hollow and is       */
/* contained within the house front area.                      */

gcreateseg (house);
   gsetfillintstyle (GSOLID);
   if (colour)
       gsetfillcolourind (GREEN_IND);
   gfillarea3 (NUM_BACK_PTS, back_pts);
   if (colour)
       gsetfillcolourind (RED_IND);
   gfillarea3 (NUM_LEFT_PTS, left_pts);
   gfillarea3 (NUM_RIGHT_PTS, right_pts);
   if (colour)
       gsetfillcolourind (BLUE_IND);
   gfillarea3 (NUM_LROOF_PTS, lroof_pts);
   gfillarea3 (NUM_RROOF_PTS, rroof_pts);
   if (colour)
       gsetfillcolourind (BLUE_IND);
   gsetedgeflag (GEDGE_ON);
   gsetedgewidthscfac (edge_width);
   gfillareaset3 (2, front_sizes, front_pts);
gcloseseg ( );
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    /* Reset the normalization transformation to the default.       */

    gselntran (unity);

    /* Stop storing segments in WISS.                               */

    gdeactivatews (wiss);

} /* end draw_picture */

/*********************************************************************/
/*  FILL_ATTRIB ()                                                   */
/*                                                                   */
/*  This subroutine defines a fill area representation and sets the */
/*  the aspect source flags for fill area attributes to bundled.    */
/*                                                                   */
/*********************************************************************/

void fill_attrib (ws_id, ws_type)
Gint      ws_id;
Gwstype   ws_type;
{
    Gint     bufsize;              /* buffer size */
    Gint     num_hatch_styles;     /* number of hatch styles */
    Gflfac   fill_fac;             /* fill area facility */
    Gint     status;               /* returned status */
    Gflinter fill_int_style[4];    /* list of fill interior styles */
    Gintlist fill_hatch_list;      /* integer list stucture of hatch
                                      values */
    Gint     fill_hatch[10];       /* list of hatch values */

                                   /* aspect source flags */
    static Gasfs3   asf = {GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL,
                           GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL,
                           GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL,
                           GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL, GINDIVIDUAL,
                           GINDIVIDUAL};

    Gint     fill_index = 1;       /* fill area index */
    Gflbundl fill_rep;             /* fill area representation */

    /* Inquire about the fill area facilities for the workstation   */
    /* to determine if hatch styles are available.                  */

    bufsize = 10;
    fill_fac.interiors = fill_int_style;
    fill_fac.hatches = &fill_hatch_list;
    fill_hatch_list.integers = fill_hatch;
    ginqfillfacil (&ws_type, bufsize, &num_hatch_styles, &fill_fac, &status);
    if (status != NO_ERROR)
      {
       gemergencyclosegks ( );
       fprintf (error_file, "Error inquire fill area facilities.");
       exit (0);
      }
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    /* If a hatch style is available, set the fill area           */
    /* representation to specify a hatch interior style, a fill     */
    /* style corresponding to the first hatch style in the list of  */
    /* available hatch styles, and fill color to red.  Set each of  */
    /* the aspect source flags for the nongeometric fill area       */
    /* attributes to bundled.  SET FILL AREA REPRESENTATION         */
    /* associates the new fill values with the fill index,          */
    /* SET FILL AREA INDEX makes the fill index the current fill    */
    /* index, and  SET ASPECT SOURCE FLAGS 3 sets the newly defined */
    /* aspect source flags so that all fill areas use the sttribute */
    /* values defined in the bundle.                                */

    if (num_hatch_styles > 0)
       {
        fill_rep.inter = GHATCH;
        asf.fl_inter = GBUNDLED;
        fill_rep.style = fill_hatch[0];
        asf.fl_style = GBUNDLED;
        fill_rep.colour = RED_IND;
        asf.fl_colour = GBUNDLED;
        gsetfillrep (ws_id, fill_index, &fill_rep);
        gsetfillind (fill_index);
       }

    gsetasf3 (&asf);

} /* end fill_attrib */

/*******************************************************************/
/*  TITLE_ATTRIB ()                                                */
/*                                                                 */
/*  This subroutine demonstrates how the text attributes of the    */
/*  title string are affected by changes to the normalization      */
/*  transformation.  Since the workstation is cleared of all       */
/*  segments initially, this subroutine also demonstrates how the  */
/*  function ASSOCIATE SEGMENT WITH WORKSTATION is used to         */
/*  repopulate the workstation segments from the segments in WISS. */
/*                                                                 */
/*******************************************************************/

void title_attrib (ws_id, title, stars, tree, side, road, house, land)

Gint  ws_id,  /* workstation identifier */
      title,  /* segment identifier for the title */
      stars,  /* segment identifier for the stars and moon */
      tree,   /* segment identifier for the tree */
      side,   /* segment identifier for the sidewalk */
      road,   /* segment identifier for the road */
      house,  /* segment identifier for the house */
      land;   /* segment identifier for the land */
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
{
                                /* text position */
    static Gpoint3 title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                   text_vec1 = {1.0, 0.0, 0.0},
                   text_vec2 = {0.0, 1.0,-1.0};
    static Gchar *title_str = "Starry Night";  /* text string */
    Gfloat    larger = 0.04;      /* larger character height */
    Glimit3   window;             /* normalization window */
    Glimit3   viewport;           /* normalization viewport */
    Gint      unity = 0;          /* normalization transformation index */
    Gint      upper_quarter = 1;  /* normalization transformation index */
    Gfloat    height_change;      /* change in viewport height */

    /* Call CLEAR WORKSTATION to clear the workstation surface and  */
    /* to delete all the segments from the workstation with         */
    /* identifier ws_id. Once deleted, the segments cannot be       */
    /* recalled.                                                    */

    gclearws (ws_id, GALWAYS);

    /* Set the text height and draw the title text.  Call UPDATE    */
    /* WORKSTATION to display the text on the screen, pause, and    */
    /* clear the workstation surface again.                         */

    gsetcharheight (larger);
    gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);
    gclearws (ws_id, GALWAYS);

    /* Map the default normalization window to the lower left       */
    /* quarter of NDC space ([0.0, 0.0, 0.0] x [0.5, 0.5, 0.5]).    */
    /* Since the text height is specified in WC units, the new      */
    /* normalization transformation reduces the height              */
    /* proportionately with the viewport reduction.  Also,          */
    /* since the character spacing and width are dependent on       */
    /* character height, those attribute values are reduced.        */

    window.xmin = 0.0;
    window.xmax = 1.0;
    window.ymin = 0.0;
    window.ymax = 1.0;
    window.zmin = 0.0;
    window.zmax = 1.0;
    gsetwindow3 (upper_quarter, &window);

    viewport.xmin = 0.0;
    viewport.xmax = 0.5;
    viewport.ymin = 0.0;
    viewport.ymax = 0.5;
    viewport.zmin = 0.0;
    viewport.zmax = 0.5;
    gsetviewport3 (upper_quarter, &viewport);
    gselntran (upper_quarter);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    /* Generate the reduced text using the new normalization       */
    /* transformation. Display the text on the screen, pause, and   */
    /* clear the workstation surface.                               */

    gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);
    gclearws (ws_id, GALWAYS);

    /* Determine the change to the Y dimension resulting from the   */
    /* normalization transformation change.  The change in the Y    */
    /* dimension affects text height, whereas the changes to the X  */
    /* and Z dimensions do not.                                     */

    height_change = viewport.ymax - viewport.ymin;

    /* Since the change in text height within the world coordinate  */
    /* space may cause the text to exceed the defined normalization */
    /* window, turn off clipping so DEC GKS maps all of the text to */
    /* NDC space.  Use the calculated Y dimension change to adjust  */
    /* the text height.  Adjusting the text height causes an       */
    /* automatic adjustment to the character spacing and width.     */
    /* Display the text on the screen, pause, and clear the         */
    /* workstation surface.                                         */

    gsetclip (GNOCLIP);
    gsetcharheight (larger * (1.0 + height_change));
    gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

    /* Reset the normalization transformation to the default.       */

    gselntran (unity);
    gsetclip (GNOCLIP);

    /* Restore the picture to its original form.                    */
    /* By associating WISS segments with the workstation, which has */
    /* the workstation identifier ws_id, the workstation stores all */
    /* the associated segments as its own segments.                 */

    gassocsegws (ws_id, title);
    gassocsegws (ws_id, stars);
    gassocsegws (ws_id, tree);
    gassocsegws (ws_id, side);
    gassocsegws (ws_id, road);
    gassocsegws (ws_id, land);
    gassocsegws (ws_id, house);


    /* Redraw the segments unconditionally.  A call to UPDATE       */
    /* WORKSTATION with the PERFORM argument will perform the same */
    /* same action as function REDRAW ALL SEGMENTS ON WORKSTATION   */
    /* if the workstation is out-of-date.  Any primitives not in a  */
    /* segment are deleted. Pause the program execution to allow    */
    /* the user to view the picture.                                */

    gredrawsegws (ws_id);
    gawaitevent (timeout, &event);

}  /* end title_attrib */
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/*******************************************************************/
/*  SEG_ATTRIB ()                                                  */
/*                                                                 */
/*  This subroutine demonstrates how to set segment attributes and */
/*  shows the effects of changing segment transformations, priority */
/*  changes, visibility changes, and highlighting changes.         */
/*                                                                 */
/*******************************************************************/

void seg_attrib (ws_id, ws_type, title, stars, tree, side, road, house, land)

Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */
Gint      title;    /* segment identifier for the title */
Gint      stars;    /* segment identifier for the stars and moon */
Gint      tree;     /* segment identifier for the tree */
Gint      side;     /* segment identifier for the sidewalk */
Gint      road;     /* segment identifier for the road */
Gint      house;    /* segment identifier for the house */
Gint      land;     /* segment identifier for the land */

{
#define    PI   3.1415926

    Gint             status;   /* returned status */
    Gmodseg          seg_dyn;  /* flags for dynamic segment attribute
                                  modification */

                                         /* fixed point for rotation
                                            and scaling */
    static Gpoint3   fixed_pt = {0.0, 0.0, 0.0};
                                         /* identity translation vector */
    static Gpoint3   translate_vec = {0.0, 0.0, 0.0};

                                         /* identity rotation vector */
    static Gangle3   rotate_vec = {0.0, 0.0, 0.0};

                                         /* identity scaling vector */
    static Gscale3   scale_vec = {1.0, 1.0, 1.0};

                                  /* identity transformation matrix */
    Gfloat           identity_xform_matrix[3][4];

                                  /* fixed point for house rotation and
                                     scaling */
    static Gpoint3   house_fixed_pt      = {0.200, 0.525, 0.500};

                                  /* identity translation vector for the
                                     house */
    static Gpoint3   house_translate_vec = {0.0, 0.0, 0.0};

                                  /* a translation vector for the house
                                     with an X direction translation
                                     component */
    static Gpoint3   house_translate_vec_2 = {0.1, 0.0, 0.0};

                                  /* a rotation vector for the house
                                     with a rotation of PI radians
                                     (180 degrees) about the Z axis */
    static Gangle3   house_rotate_vec = {0.0, 0.0, PI};

                                  /* an identity scaling vector for the
                                     house */
    static Gscale3   house_scale_vec = {1.0, 1.0, 1.0};
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
                                /* house segment transformation matrix */
Gfloat           house_xform_matrix[3][4];
                                /* fixed point for tree rotation and
                                   scaling */
static Gpoint3   tree_fixed_pt = {0.52, 0.35, 0.0};

                                /* identity translation vector for
                                   the tree */
static Gpoint3   tree_translate_vec = {0.0, 0.0, 0.0};

                                /* an identity rotation vector for
                                   the tree */
static Gangle3   tree_rotate_vec = {0.0, 0.0, 0.0};

                                /* scaling vectors for the tree with
                                   X and Y direction components and
                                   identity Z components */
static Gscale3   tree_scale_vec = {0.25, 0.25, 1.0};
static Gscale3   tree_scale_vec_2 = {1.2, 1.2, 1.0};

                                /* tree segment transformation matrix */
Gfloat           tree_xform_matrix[3][4];

                                /* fixed point for title rotation */
static Gpoint3   title_fixed_pt = {0.0, 0.0, 0.0};

                                /* translation vector for the title */
static Gpoint3   title_translate_vec = {-0.25, 0.8, -0.6};

                                /* identity rotation vector for the
                                   title */
static Gangle3   title_rotate_vec = {0.0, 0.0, 0.0};

                                /* identity scaling vector for the
                                   title */
static Gscale3   title_scale_vec = {1.0, 1.0, 1.0};

                                /* title segment transformation matrix */
Gfloat           title_xform_matrix[3][4];

Gint             k;             /* loop control variable */

/* Check the ability of the workstation to dynamically generate */
/* segment transformations, visibility changes, highlighting    */
/* changes, priority changes, primitive additions, and segment  */
/* deletions.  If implicit regenerations are required for       */
/* particular actions, it will be necessary to call UPDATE      */
/* WORKSTATION to see the effects of the action.                */

ginqmodsegattr (&ws_type, &seg_dyn, &status);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Create an identity segment transformation using translation, */
/* rotation, and scaling vectors that produce no change.       */
/* Components of a translation vector are added to current     */
/* segment coordinates to alter the position of the segment so  */
/* [0.0, 0.0, 0.0] is the identity translation vector that     */
/* produces no change.  Components of a rotation vector        */
/* indicate the amount of rotation about the X, Y, and Z axes   */
/* respectively. The axes pass through the fixed point.        */
/* [0.0, 0.0, 0.0] is the identity rotation vector. Components  */
/* of the scaling vector are scale factors that are multiplied  */
/* by the distance between the segment's fixed point and the    */
/* points in the segment's primitives, shrinking or expanding   */
/* the segment.  [1.0, 1.0, 1.0] is the identity scaling       */
/* vector. The NDC argument specifies that the fixed point and  */
/* tranlation vector components are expressed in NDC points.    */
/* If the values are expressed in WC points, DEC GKS uses the   */
/* current normalization transformation to transform the points */
/* to the NDC space.  If the current transformation is         */
/* different than the one used during segment creation, there   */
/* can be unexpected results. The call to EVALUATE             */
/* TRANSFORMATION MATRIX 3 only creates a matrix and does NOT   */
/* put it into effect.                                         */

gevaltran3 (&fixed_pt, &translate_vec, &rotate_vec, &scale_vec, GNDC,
            identity_xform_matrix);

/* Rotate the house, 180 degrees about the Z axis.             */
/* Call EVALUATE TRANSFORMATION MATRIX 3 to create the matrix,  */
/* and call SET SEGMENT TRANSFORMATION 3 to accomplish the      */
/* transformation.                                             */

gevaltran3 (&house_fixed_pt, &house_translate_vec, &house_rotate_vec,
            &house_scale_vec, GNDC, house_xform_matrix);
gsetsegtran3 (house, house_xform_matrix);


/* Shrink the tree.                                            */
/* Call EVALUATE TRANSFORMATION MATRIX 3 to create the matrix,  */
/* and call SET SEGMENT TRANSFORMATION 3 to accomplish the      */
/* transformation.                                             */

gevaltran3 (&tree_fixed_pt, &tree_translate_vec, &tree_rotate_vec,
            &tree_scale_vec, GNDC, tree_xform_matrix);
gsetsegtran3 (tree, tree_xform_matrix);

/* Move the title.                                             */
/* Call EVALUATE TRANSFORMATION MATRIX 3 to create the matrix   */
/* and call SET SEGMENT TRANSFORMATION 3 to accomplish the      */
/* transformation.                                             */

gevaltran3 (&title_fixed_pt, &title_translate_vec, &title_rotate_vec,
            &title_scale_vec, GNDC, title_xform_matrix);
gsetsegtran3 (title, title_xform_matrix);


/* Perform an update of the workstation surface if a segment    */
/* transformation requires an implicit regeneration.  Pause     */
/* the program execution to allow the user to view the picture. */

if (seg_dyn.transform == GIRG)
   gupdatews (ws_id, GPERFORM);
gawaitevent (timeout, &event);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
/* Modify the existing tree segment's transformation matrix.    */
/* To create a matrix that simulates a cumulative effect of     */
/* several segment transformations, use the function ACCUMULATE */
/* TRANSFORMATION MATRIX 3.  The first argument is the          */
/* existing transformation matrix to which the translation,     */
/* rotation, and scaling transformations described in the next  */
/* four arguments are added.  The resulting new matrix is       */
/* written to the last argument.  After setting the new segment */
/* transformation, call UPDATE WORKSTATION if the implicit      */
/* regeneration mode is set to implicit regeneration (IRG)      */
/* required. Pause the program execution to allow the user to   */
/* view the picture.                                            */

for (k = 0; k < 6; k++)
   {
    gaccumtran3 (tree_xform_matrix, &tree_fixed_pt, &tree_translate_vec,
            &tree_rotate_vec, &tree_scale_vec_2, GNDC, tree_xform_matrix);
    gsetsegtran3 (tree, tree_xform_matrix);
    if (seg_dyn.transform == GIRG)
       gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);
   }

/* Return the tree segment to its original size, orientation,   */
/* and position, and perform regeneration if needed.  Pause     */
/* the program execution to allow the user to view the picture. */

gsetsegtran3 (tree, identity_xform_matrix);
if (seg_dyn.transform == GIRG)
    gupdatews (ws_id, GPERFORM);
gawaitevent (timeout, &event);


/* Shift the house beyond its normalization viewport boundary   */
/* to show that segments are clipped.  After translating the    */
/* house, the primitive is not contained in the normalization   */
/* viewport stored with the segment as its clipping volume.     */
/* The normalization viewport is stored as the clipping volume  */
/* for the primitives at the time of segment creation and       */
/* cannot be changed. The only way to prevent the clipping of   */
/* segments is to disable clipping at the time of segment       */
/* creation.                                                    */

gaccumtran3 (house_xform_matrix, &house_fixed_pt,
            &house_translate_vec_2, &house_rotate_vec,
            &house_scale_vec, GNDC, house_xform_matrix);
gsetsegtran3 (house, house_xform_matrix);

/* Perform regeneration if needed and pause the program         */
/* execution to allow the user to view the picture.             */

if (seg_dyn.transform == GIRG)
    gupdatews (ws_id, GPERFORM);
gawaitevent (timeout, &event);
```

(continued on next page)

Example 4–1 (Cont.) Using DEC GKS Output Functions

```
    /* Give the land a higher display priority than the house. DEC  */
    /* GKS implements segment priority on a scale of real numbers   */
    /* from 0.0 (lowest priority) to 1.0 (highest priority).  Do an */
    /* update of the workstation surface if the priority change     */
    /* requires an implicit regeneration.  Pause the program        */
    /* execution to allow the user to view the picture.             */

    gsetsegpri (house, 0.0);
    gsetsegpri (land, 1.0);
    if (seg_dyn.priority == GIRG)
        gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);


    /* Change the land segment's visibility attribute to INVISIBLE  */
    /* and update the workstation surface if a visibility change    */
    /* requires an implicit regeneration.  Pause the program        */
    /* execution to allow the user to view the picture.             */

    gsetvis (land, GINVISIBLE);
    if (seg_dyn.disappear == GIRG)
        gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

    /* Change the tree segment's highlight attribute to HIGHLIGHTED */
    /* and update the workstation surface if a highlighting change  */
    /* requires an implicit regeneration.  Pause the program        */
    /* execution to allow the user to view the picture.             */

    gsethighlight (tree, GHIGHLIGHTED);
    if (seg_dyn.highlight == GIRG)
        gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

} /* end seg_attrib */

/********************************************************************/
/*  CLEAN_UP ()                                                     */
/*                                                                  */
/*  This subroutine cleans up the DEC GKS and workstation           */
/*  environments.                                                   */
/*                                                                  */
/********************************************************************/

void clean_up (ws_id, wiss)

Gint ws_id;  /* workstation identifier */
Gint wiss;   /* WISS workstation identifier */

{
    /* Deactivate all active workstations and close all open        */
    /* workstations before closing GKS.                             */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosews (wiss);
    gclosegks ( );

} /* end clean_up */
```

5 Data Input

DEC GKS accepts input from a variety of input devices, such as a keyboard, mouse, or graphics tablet. The differences between these **physical input devices** are hidden from the application program by **logical input devices**. A GKS application obtains graphic input from the user by controlling one or more logical input devices, which allow the application to ignore the characteristics of the physical device used.

This chapter describes the following concepts:

- Logical input devices
- Operating modes
- Input viewport priority
- Prompt and echo types
- Concurrent use of several logical input devices
- Documenting your application with respect to logical input devices

Logical input devices are used synchronously or asynchronously with the application program. There are three DEC GKS input modes:

- Request mode—Data is input synchronously.
- Sample mode—Data is input asynchronously.
- Event mode—Data is input asynchronously.

The **input class** determines the type of logical input that the logical input device delivers. The six logical input classes are as follows:

- LOCATOR
- STROKE
- VALUATOR
- CHOICE
- PICK
- STRING

5.1 Logical Input Devices

A physical input device provides input to an application. A single application can use input from many physical input devices. You can translate the information obtained from physical input devices in several ways. For example, a letter typed on the keyboard can represent a decimal ASCII value, a character value, a character string containing one letter, or many other values depending on how the application stores and then interprets the input.

DEC GKS accepts three types of input:

- Real numbers (as real values, coordinate points, or a series of coordinate points)
- Integers (as choice numbers, segment names, or pick identifiers)
- Character strings

Because many kinds of physical input devices exist, DEC GKS maintains device independence by using logical input devices. A logical input device acts as an intermediary between a physical input device and the application. That is, the user inputs data from the physical input device to the application via the logical input device. GKS lets a single open workstation have zero or more logical input devices active at the same time.

5.1.1 Characteristics of Logical Input Devices

A logical input device is a piece of software that accepts input of one data type from one open workstation using a combination of physical input devices. A logical input device is defined by three identifiers:

- Workstation identifier

  The **workstation identifier** identifies an open workstation that belongs to the category INPUT or OUTIN. The logical input device is part of the workstation. How DEC GKS implements the logical input device depends on what physical input devices the workstation is using.

- Input class

  The input class determines the type of input the logical input device returns to the application. The six classes of GKS input are described in Section 5.1.2.

- Logical device number

  The **logical device number** distinguishes one logical input device from another of the same input class on the same workstation. It lets you use more than one logical input device of the same class on a single workstation. For example, you can use a mouse as one choice-class logical input device and the keys on a keyboard as another choice-class logical input device. If you do, though, you must tell the user which physical input device controls which logical input device.

  The logical device number determines what mechanism **triggers** the logical input device. (See Section 5.1.5 for information about triggering a logical input device.) DEC GKS defines at least four device numbers for each input class. For example, a choice 1 device number requires the user to press mouse button 1 to trigger the logical input device. (Without a mouse, the user must press Return.) A choice 2 device number requires the user to press the arrow keys or the keys on the numeric keypad.

  The logical device number also determines the format in which DEC GKS returns data. For example, a string 1 device number returns a Digital multinational text string, while a string 3 device number returns an ASCII value.

Each logical input device has associated features that determine the behavior of the device and the input data that it returns. These features can be divided into two groups: workstation components, which are parts of the workstation containing the logical input device, and application components, which can be supplied by the application program, but default values do exist. The attributes are as follows:

- Workstation components
  - Measure

    The **measure** of a logical input device is the value that the logical input device returns to the application. This might be a position or a choice, for example.
  - Trigger

    The trigger is the process that causes DEC GKS to accept an input measure in request or event mode. It is usually invoked (fired) by an action such as pressing a mouse button.

    If a request is pending when the trigger fires, the measure of the device is returned. If one or more devices are in event mode when the device fires, the device identifiers and measure values are passed to the input queue as separate event reports. In either case, GKS outputs a device-dependent logical input acknowledgment.
  - Break

    Break is the process by which the user forces DEC GKS to abandon input. DEC GKS returns a status to the application that indicates the user did not want the measure returned to the application.
- Application components

  Each logical input device is optionally initialized by the user. The device must be in request mode (Section 5.2) before it is initialized. Use one of the INITIALIZE ... or INITIALIZE ... 3 functions to set the following values:
  - Initial value—A value appropriate to the input class.
  - Prompt and echo type (PET)—A value that selects the prompting technique. If echoing is set to ON, the value also selects the echoing technique.
  - Echo volume or area—This is supplied in the form of three three-dimensional or two two-dimensional points in device coordinates.
  - Data record of prompt and echo type—Some input classes and PETs must have control values in the **input data record** for that input class. The control values determine the appearance of a particular PET on the workstation surface, such as the thickness of cross-hairs.

5.1.2 Classes of Logical Input Device

The input class determines the type of input the logical input device accepts. There are six GKS input classes:

Locator Input Class

A locator-class device returns a position on the display surface selected by the user. It lets the user position a prompt such as a "+" on the workstation surface to define a point. The user can then trigger a request for input. It provides the following logical input values to the application:

- The position in WC points
- The normalization transformation number
- The view index (three-dimensional devices only)

Stroke Input Class

A stroke-class device returns a sequence of point positions that the user draws on the display surface. Your application must specify a distance vector and a length of time in the initialization data record. A stroke-class device provides the following logical input values to the application:

- The sequence of WC points
- The normalization transformation number
- The view index (three-dimensional devices only)

Valuator Input Class

A valuator-class device displays a numerical scale on the display surface that allows the user to select a real number with a pointer. The logical input value returned is a real number representing the current position of the pointer on the scale.

Choice Input Class

A choice-class device lists a series of choices on the display surface. The user selects a choice, and DEC GKS returns an integer value representing the choice. It provides the following logical input values to the application:

- The choice status
- The integer representing the choice made

Pick Input Class

A pick-class device returns the identity of a segment selected by the user and a **pick identifier**. A pick identifier is a name that is attached to an output primitive within a segment. DEC GKS assigns the current pick identifier to a primitive when the primitive is being generated. To assign a different pick identifier during segment generation, use the SET PICK IDENTIFIER function.

When pick input is requested, DEC GKS provides a cursor on the screen. When the cursor is in contact with a segment to be picked and the device is triggered, the cursor provides the following logical input values:

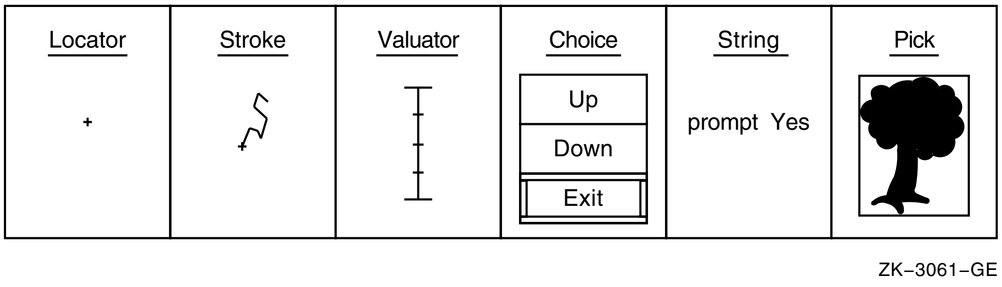
- The pick status
- The segment name
- The pick identifier

String Input Class

A string-class device returns a character string entered by the user. A string-class device creates an area on the screen where the user enters a character string. If you provide an initial string as a prompt, DEC GKS appends the input string to the initial string. The logical input value it provides is a character string.

Figure 5–1 illustrates the visual prompts that a workstation may use to implement the DEC GKS logical input classes. For each logical input class, you choose from the workstation's available prompt and echo types. Each workstation can support a different number of prompt and echo types for a given input class. Section 5.4 describes prompt and echo types in detail.

Figure 5–1 Possible Prompts for DEC GKS Logical Input Classes



5.1.3 Initializing a Logical Input Device

Each workstation has its own default values that a logical input device can use. You also can set your own values for the logical input device. To set your own values, you must initialize the logical input device using an INITIALIZE ... function. The logical input device must be in request mode to be initialized. For example, to initialize a locator-class logical input device, you would put it in request mode by calling SET LOCATOR MODE. Then you would call INITIALIZE LOCATOR and supply the values you want.

If you do not initialize the logical input device, it uses the default values.

For more information concerning the SET ... MODE functions, see Section 5.2.3.1, and the chapter on input functions in your DEC GKS binding manual.

5.1.4 Activating a Logical Input Device

To activate a logical input device in request mode, you simply call a REQUEST ... function. DEC GKS activates the logical input device and displays the input prompt (if echoing is enabled) when you call the function. For example, to activate a locator-class logical input device in request mode, you would call REQUEST LOCATOR and supply the appropriate values to the arguments. DEC GKS deactivates the logical input device when the REQUEST function completes. The SAMPLE ... functions activate a logical input device in sample mode, and the GET ... functions activate a logical input device in event mode. (See the chapter on input functions in your DEC GKS binding manual for information on the REQUEST, SAMPLE, and GET functions.)

You can enable or disable echoing by setting the echo flag to ECHO or NOECHO. Disabling echoing is useful when the DEC GKS echo types are inadequate and you must echo the input in an application-specific manner. When you enable echoing, DEC GKS echoes the input prompt and the input values.

For more information concerning prompt and echo types, see Section 5.4. For more information concerning input modes, see Section 5.2.

5.1.5 Obtaining Measures from a Logical Input Device

A logical input device returns a value to the application. The value it returns is called the measure of the device. Two input modes, request and event, require the user to perform an action on a physical input device to return the measure. The action is called the input trigger. When the user performs the action, the user triggers the logical input device, which then returns its measure. The physical input device determines what kind of action the user must perform to trigger the logical input device. For example, when using a keyboard, the user triggers the logical input device by pressing a key on the keyboard. When using a mouse, the user triggers the logical input device by clicking a mouse button.

Sample mode does not require the user to trigger a logical input device. For example, SAMPLE LOCATOR gets the current value of a locator-class logical input device without a trigger action from the user.

5.2 Operating Modes of Logical Input Devices

There are three DEC GKS input operating modes: request, sample, and event modes. An input mode is **synchronous** (request mode) or **asynchronous** (sample and event modes).

5.2.1 Deciding Which Mode to Use

The first decision that you must make when choosing an input operating mode is whether you need to use a synchronous or an asynchronous input operating mode. A synchronous input operating mode synchronizes the process so that the application pauses for user input and continues when the user triggers the input device. An asynchronous input operating mode allows the application to continue to execute while the user enters input on the logical input devices. Consequently, if your application cannot perform any work until it receives input from the user, then you should use the DEC GKS synchronous input operating mode (request mode). If your application should continue to process as the user enters input values, then you should use one of the DEC GKS asynchronous input operating modes (sample or event mode).

If you decide to use sample or event mode, you need to decide who, according to the needs of your application, should have more control over entering input values: the application program or the user.

During sample mode, the user can change the current measure of a given input device by altering the position of the prompt, but the user cannot trigger the device. At any time determined by the application program, the application samples (takes) the current measure of the device. The user specifies possible input values, but the application controls when values are actually accepted. The application chooses when to end the sampling input process.

During event mode, the user can change the measure of the device and trigger a desired value in the same manner as in request input mode. However, in event mode, the input prompt does not disappear when the user triggers the device. Every time the user triggers the device, the action generates an **event report**. The DEC GKS input process places these reports on a time-ordered queue (first in, first out). When the application chooses, it removes the reports and processes the input. Also, the application chooses when to end the event input process.

Consequently, if you want the application to control the timing of the acceptance of input values, use sample mode. If you want the user to control entering input values, use event mode.

5.2.2 How to Select an Input Mode

The mode of the logical input device is changed using one of the SET ... MODE input functions. The following criteria must be satisfied in the selection process of each mode:

- Use synchronous input (request mode) if the application cannot do any work until it receives input. In request mode, the application pauses until the user triggers the input device or performs a break action.

A device must be in request mode before it is initialized.

- Use asynchronous input if the application must continue to process while the user enters data. In asynchronous mode (sample or event input), the application continues to execute while the user enters input.
  - Use sample input if the application is to control the timing of input value acceptance. Sample mode lets the user change the current measure of the device by altering the position of the prompt, but the user cannot trigger the device. The application samples the current measure of the device at any time determined by the application. The user specifies possible input values, but the application decides when values are accepted.
  - Use event input if the user is to control entering input values. Event mode lets the user change the measure of a device and trigger a desired value as in request mode. When the user triggers the device, the input prompt remains visible; an event report is generated and queued in a first in, first out (FIFO) queue. The application removes the report, processes the input, or ends the event input process.

5.2.3 Input Operating Mode Differences

Before you choose one of the three available input operating modes, it is helpful to see the differences when all three modes are used in a similar application.

Imagine a program that uses locator input. The program prompts the user to move the cursor to the top of the surface (a maximum Y value).

Using request mode, you can only determine the position of the locator prompt once. As soon as the user moves the locator prompt and then triggers the device, the input process ends.

Using sample and event mode, the program can loop to continually check for a change in the input value. Using sample mode, the application can continually monitor the position of the prompt without a trigger from the user. Using event mode, the application is not aware of a change in the position of the locator prompt until the user triggers the input device (placing a report on the queue), and until the application removes the report from the queue and processes the input.

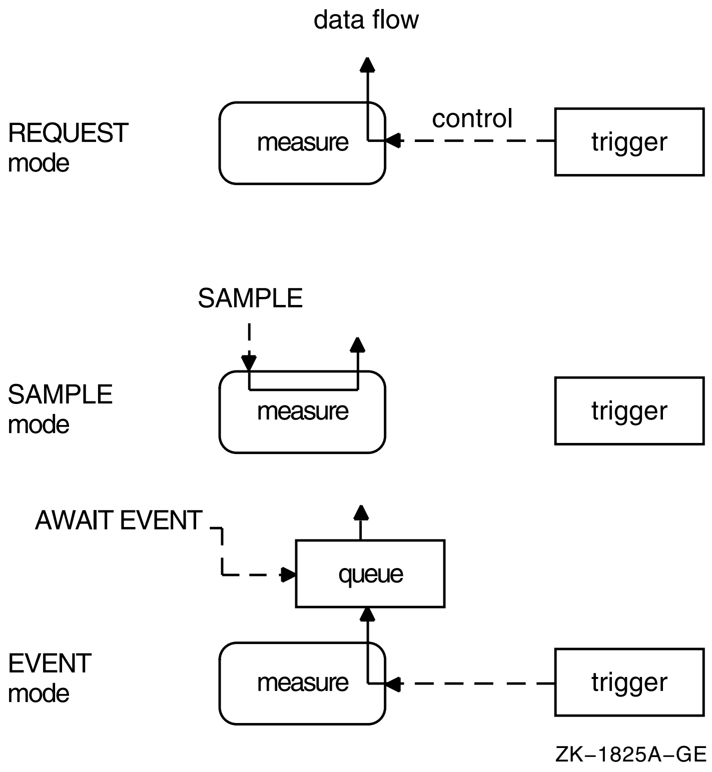
Clearly, such an application favors sample input, as the application can easily and continually monitor the position of the locator prompt. Using this example, you can compare the differences in the ways in which the operating modes perform. In this way, you can judge which input operating mode best serves a particular type of application.

The differences between the three modes are shown in Figure 5–2.

5.2.3.1 Setting the Mode

You must establish the mode of the logical input device with the SET ... MODE functions. This function passes the workstation identifier, device number, operating mode (request, sample, or event), and the echo switch (ECHO or NOECHO). The echo switch determines whether GKS echoes the prompt and input values on the workstation surface.

Figure 5–2 Relationship Between Measure and Trigger for Different Input Modes



5.2.3.2 Request Mode

A logical input device is initially in request mode. Select this mode if the application requires further input from the user before it continues.

5.2.3.3 Sample Mode

To use sample mode, you initialize the logical input device (if the device is in request mode—the DEC GKS default mode) and set the input operating mode of the input device to sample. After you call one of the SET ... MODE functions, the workstation activates the specified logical input device and the prompt appears on the workstation surface.

After the prompt appears on the workstation surface, the user can alter the current measure of the device. For example, if you place a pick device in sample mode, the pick aperture appears on the surface of the workstation. At this time, the user can move the aperture, highlighting visible segments.

After the application program samples the pick device, the program can perform tasks depending on the sampled values. After the application samples the pick device, it does not matter if the user moves the prompt to a different segment. The application program controls when an input device measure is accepted; the user can only supply possible measures.

If you want to remove the pick prompt from the workstation surface, place the logical input device in request mode by using the SET PICK MODE function.

After you remove the pick prompt from the workstation surface, you can call one of the INITIALIZE ... functions to reinitialize the input device, if you choose. You can initialize only devices that are in request mode. If you need to sample the pick device in some subsequent portion of your program, simply reset the input mode to sample by calling SET PICK MODE again. At this point, the workstation places the prompt on the workstation surface again.

5.2.3.4 Event Mode

To use event mode, you initialize the logical input device (if the device is in request mode—the DEC GKS default mode) and set the input operating mode of the device to event. Once you call one of the SET ... MODE functions, the workstation activates the specified logical input device and the prompt appears on the workstation surface.

Once the prompt appears on the workstation surface, the user can alter the measure and trigger the device as often as desired. For example, if you place a choice device in event mode, the prompt (a selection of choices) appears on the workstation surface as soon as you call SET CHOICE MODE.

Every time the user triggers the input device, the workstation places a report on the event input queue located in the GKS state list. Each report includes the following information:

- The workstation identifier
- The input class of the logical input device
- The logical input device number
- Input data (varies according to input class)

If you want to remove the choice prompt from the workstation surface, place the logical input device into request mode with the SET CHOICE MODE function.

Once you set an input device to request mode, the workstation removes the device prompt from the workstation surface. At this point, you can call one of the INITIALIZE ... functions to reinitialize the device, if you choose. You can only initialize devices that are in request mode. If you need to have the user generate events using the choice device in some subsequent portion of your program, simply reset the input mode to event by calling SET CHOICE MODE again. At this point, the workstation places the prompt on the workstation surface again.

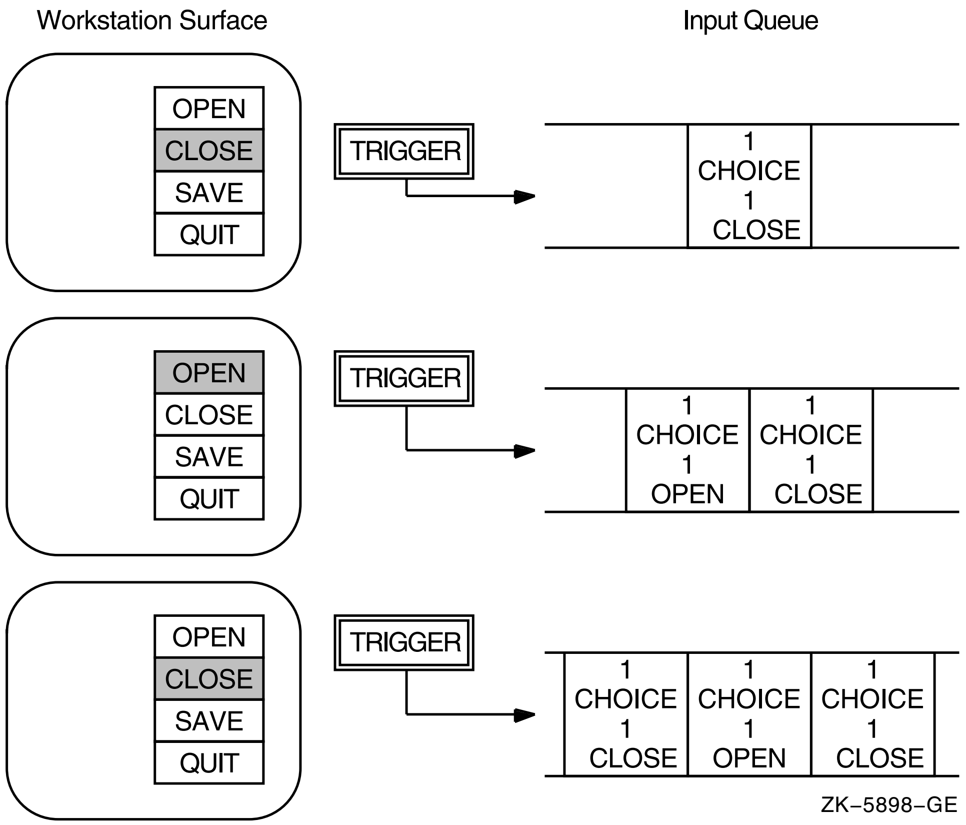
5.2.3.5 The Event Input Report Queue

When the input device in event mode is triggered, the device handler places a report on the event input queue in the GKS state list. Each report includes the following data:

- Workstation identifier
- Input class of the logical input device
- Logical input device number
- Input data

Generation of event input reports is summarized in Figure 5–3. This example illustrates a choice menu, with the options open, close, save, and quit. If close is selected and the trigger is activated, close is stored in the input queue as the first event.

Figure 5–3 Generating Event Input Reports with CHOICE



5.2.3.6 Removing Reports from the Queue

The functions AWAIT EVENT, FLUSH DEVICE EVENTS, and CLOSE WORKSTATION all remove event input reports from the queue. The application removes reports at any time with a call to AWAIT EVENT. AWAIT EVENT has the following arguments:

- Time-out period (seconds)
- Workstation identifier
- Input class
- Logical input device number

The AWAIT EVENT function performs the following sequence of actions:

1. Suspends execution of the application for the specified time-out period, or until there is a report in the event queue (whichever occurs first). If there is initially a report in the event queue, execution is not suspended.
2. Removes the oldest report (if there is one) from the queue.
3. Copies the logical input value (if there is one) to the *current event report* entry in the GKS state list.
4. Writes the workstation identifier, input class, and logical input device number into its argument list (if there is a report in the queue).
5. Returns to the application.

When using event mode, you must call AWAIT EVENT to obtain the workstation identifier, device number, and input class of the event prior to calling a GET ... function. Check the class of the event returned by the AWAIT EVENT call, and then call the corresponding GET ... function to obtain the measure of the device.

5.2.3.7 Generating Simultaneous Events

Two or more events are generated simultaneously if two or more active devices in event mode recognize the same trigger. The inquiry function INQUIRE MORE SIMULTANEOUS EVENTS checks if an event taken from the queue is the last in a series of simultaneous events. This lets the user ensure that the order of event reporting is precise.

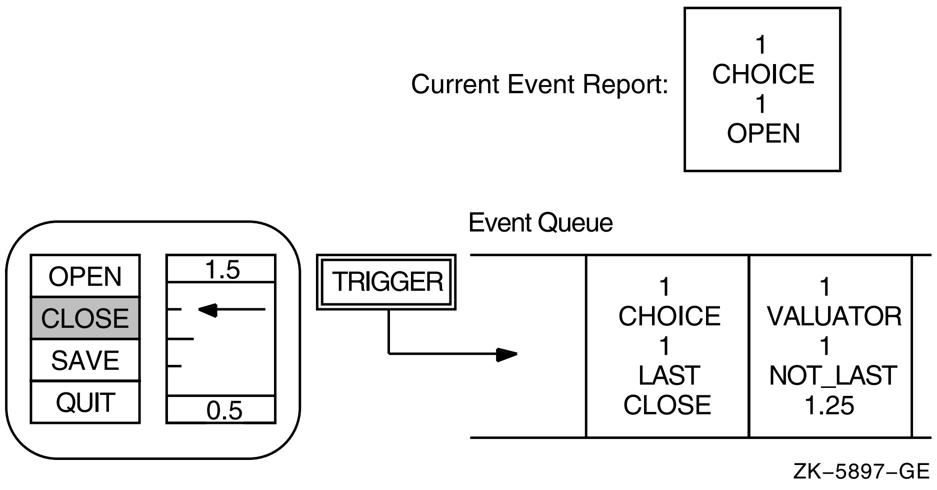
Code that checks for simultaneous events does the following:

1. Removes a report from the queue
2. Calls INQUIRE MORE SIMULTANEOUS EVENTS
3. Checks the *events_flag* argument
   - If *events_flag* equals MORE_EVENTS, removes another event from the queue and calls INQUIRE MORE SIMULTANEOUS EVENTS again
   - If *events_flag* equals NOMORE_EVENTS, all the simultaneously generated reports have been removed

The application then decides the order in which to process the input.

Figure 5–4 illustrates generating simultaneous event reports with choice and valuator modes.

Figure 5–4 Generating Simultaneous Event Reports with Choice and Valuator



5.2.3.8 Overflow of the Input Queue

The input queue overflows if many devices are rapidly triggered. The device handler does not accept additional reports until the queue is cleared. Check for an input queue overflow after a call to AWAIT EVENT by calling INQUIRE INPUT QUEUE OVERFLOW.

An error status of 0 indicates that the queue has overflowed. If the queue has overflowed, proceed as follows:

1. Deactivate all devices currently in event mode by putting them in request mode. No more additional reports are generated.
2. Call AWAIT EVENT and remove the reports one by one. Processing of the input continues. When the queue is empty, put the device back into event mode.

The function FLUSH DEVICE EVENTS is used to remove all the reports generated by a device of a specified input class. To clear the queue, call FLUSH DEVICE EVENTS once for each possible input class.

The function AWAIT EVENT resets queue overflow if the queue is empty.

5.3 Input Viewport Priority

During locator and stroke input, the user positions the prompt on the workstation surface and returns one point or a series of points in device coordinates. DEC GKS translates the device coordinates to NDC points, and then uses the viewport input priority to determine which normalization transformation to use when translating to WC points.

To decide which normalization viewport has a higher input priority, DEC GKS maintains a priority list. By default, DEC GKS assigns the highest priority to the transformation (0). The viewports of all remaining transformations decrease in priority as their transformation numbers increase (viewport 0 higher than viewport 1, 1 higher than 2, 2 higher than 3, and so on).

When using a locator-class device, DEC GKS uses the normalization and view transformations of the highest input priority that contains the input point. When using stroke input, DEC GKS uses the normalization transformation of the highest priority that contains *all* of the points in the stroke. Because a locator or stroke input device can return device coordinate points that could fall outside of the default two-dimensional normalization viewport ([0,1] [0,1]) or the three-dimensional normalization viewport ([0,1] x [0,1] x [0,1]), you can always use the unity transformation to transform stroke input data.

5.3.1 Restricting Movement of Locator, Stroke, and Pick Prompts

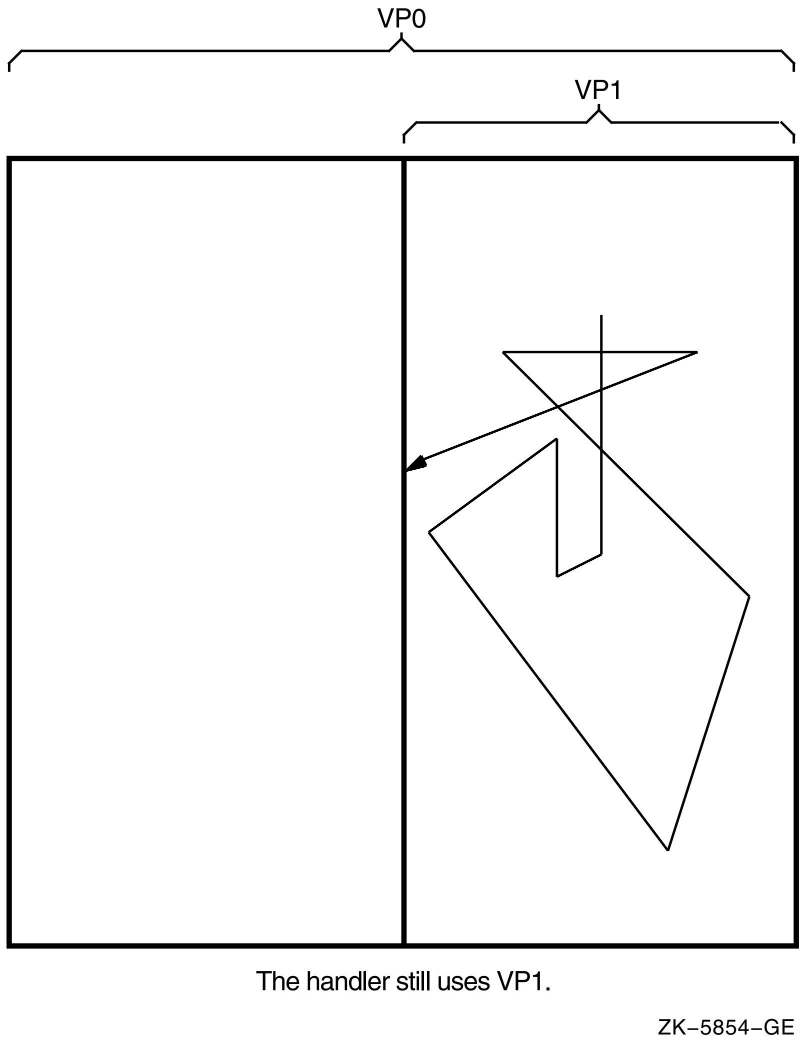
In some applications, you may wish to limit the range of the locator, stroke, or pick prompt so that you have complete control over the input values returned by the input device. Using stroke and locator input with a picture composed of overlapping viewports, you can restrict the input prompt to a single portion of the workstation surface. By restricting the movement of the prompt, you limit the number of normalization transformation numbers DEC GKS can use to translate a given input point or series of input points. Using a pick device, you can use the echo area—along with the visibility and detectability attributes of each segment—to control which segments the user can and cannot pick.

Assuming the normalization transformations established in Example 5–1, you have two possible normalization transformations (0 and 1) used to transform an input point. If you want the user to enter a point or a series of points to be transformed using normalization transformation number 1, you need to make sure that the input device cannot return a point that falls outside of normalization viewport 1.

To accomplish this, you adjust the input echo area. Adjusting the echo area restricts the amount of the workstation surface that the user can use to move the input prompt.

Figure 5–5 illustrates the surface of the workstation when the user attempts to move the prompt past the currently defined echo area.

Figure 5–5 Restricting the Echo Area



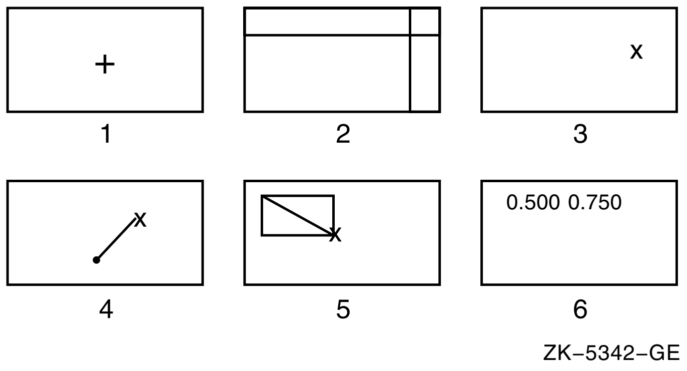
5.4 Prompt and Echo Types

Many times, you may find that the default prompt and echo type does not suit your application. For example, you may wish to have the prompt for the choice logical input device look like a menu; you may want control of the number of items in a menu; and, you may want to label the menu items so that the labels apply to your application. One logical input device can have a number of prompt and echo types. For example, DEC GKS supports six different prompt and echo types for a locator input device. A workstation may implement any of the following locator prompts:

1. A tracking plus sign (+) (implementation dependent)
2. A cross hair (the lines of the cross hair extend to the boundaries of the echo area)
3. A tracking cross (X)
4. A line from an initial position to the current locator position
5. A rectangle whose diagonal connects the initial and current positions
6. A numeric representation of the current locator position

Figure 5–6 illustrates possible implementations of the six locator prompts.

Figure 5–6 Possible Locator Prompts



Of the prompt and echo types defined by the GKS standard, a workstation may implement any number of them. If you are unaware of the capabilities of your device, or if you write an application that runs on several devices of varying prompt and echo type support, you need to inquire about the supported prompt and echo types of each device. You can choose from the prompt and echo types available on a particular workstation.

You must use an **input data record** to pass information about a specific prompt and echo type on a given logical input device. You can use either the default input data record or an application-specified input data record. The application-specified input data record is an argument to the INITIALIZE ... input functions.

For a complete description of the GKS standard prompt and echo types for each class of logical input device, see the chapter on input functions in your DEC GKS binding manual.

5.4.1 Data Records

Because the workstations use DEC GKS primitives such as lines, markers, and fill areas to construct input prompts, the workstations optionally use additional information that determines how the prompt and echoed input appears on the surface. For example, a workstation may use a polyline output attribute that would affect the appearance of cross hairs on the surface. The requirements depend on the needs of the different prompt and echo types on different physical devices.

To pass information to meet the requirements of a certain prompt and echo type on a given logical input device, you adjust components of the input data record. The input data record is a series of components that specify additional information needed to implement a certain prompt interface (according to a prompt and echo type value).

A specific prompt and echo type needs specific information for a particular logical input class. You must pass an input data record if you call one of the INITIALIZE ... functions. See your binding manual for the data information needed for the different prompt and echo types.

5.4.2 Inquiry Functions

When you decide to change the default input values by calling one of the INITIALIZE ... input functions, you need to pass a valid data record. To obtain a valid data record, you can either construct the record according to the GKS standard data record specifications for your chosen prompt and echo type, or you can call an inquiry function to obtain the default (or currently specified) data record.

To obtain input data records and other values, you can either inquire from the workstation description table (default values) or from the workstation state list (current values). To obtain default values, you call the INQUIRE DEFAULT ... DEVICE DATA (3) functions. To obtain the current values, you call the INQUIRE ... DEVICE STATE (3) functions. See your DEC GKS binding manual for more information on inquiry functions.

5.4.3 Controlling the Prompt Interface: Input Data Record

The input data record is a series of data components that describe the prompt interface specified by a particular prompt and echo type value. The input data record forms part of the workstation description table. It is accessed when input devices are being initialized. The components are of different data types and the organization of the data is binding specific. See the appropriate binding manual for the necessary information.

To change the default input data record with an INITIALIZE ... input function, pass a valid data record. This is done either by using the standard specification or by using an inquiry function to obtain the default or current record, changing the returned data record, and calling an INITIALIZE ... function with the adjusted information.

5.5 Using Several Logical Input Devices Concurrently

When using only request mode, the workstation does not place the logical input device prompt on the workstation surface until you call one of the REQUEST ... functions. Because the application pauses until one request for input is complete, you can have, at most, one logical input device prompt present on the workstation surface at any one time. If using only request mode, you need only one logical input device (numbered 1) for each of the input classes (choice, locator, pick, string, stroke, and valuator).

When you use sample and event input modes, the prompt appears on the workstation surface as soon as you call the appropriate SET ... MODE function. Because you can call the SET ... MODE function for several devices, possibly of the same class, setting each to any of the three modes, DEC GKS provides you with more than one logical input device number for each class. (To review the

DEC GKS supported input devices, see the chapter on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.)

For example, depending on the needs of your application, you can place two choice devices in any of the following combination of input operating modes:

- Both in sample mode, by calling SET CHOICE MODE for each device.
- Both in event mode, by calling SET CHOICE MODE for each device.
- One in sample and one in event mode, by calling SET CHOICE MODE for each device.
- One device in either sample or event mode, and the other device in request mode. To do this, you call SET CHOICE MODE for each device, followed by a call to REQUEST CHOICE to activate the other device's prompt. (The user can view only the prompt of a single device in request mode at any given time.)

The application in Example 5–1 requires the use of a choice input device to obtain input from the user that affects subsequent execution of the program. Using the Continue/Exit choice menu, the application cannot continue to process until it knows whether the user wants to continue scaling or to exit from the program.

Suppose two choice input devices were required in this application. If your application should require two logical input devices of the same class, you must choose one of two solutions to this problem:

- Use two distinct logical input device numbers for each device.
- Use the same device number, and reinitialize the device each time you need a change.

If you use two distinct logical input device numbers, you may have two devices that look the same to the user, but measure and trigger differently. For example, choice device number 1 might require that the user press the arrow keys to change the measure and the Return key to trigger. Choice device number 2 may require that the user press either the arrow keys or one of the numeric keypad keys to both change the measure and trigger the device (the user does not need to press Return). Consequently, if you use two distinct logical input device numbers for devices of the same class, you need to be sure that the user knows the differences in the ways in which the two devices are measured and triggered.

The second option for using two logical input devices of the same class is to reinitialize the device when it needs a different choice menu. When reinitializing a single logical input device, you accomplish this by performing the following tasks:

- By storing the current values of the device obtained by using one of the INQUIRE ... DEVICE STATE (3) functions
- By reinitializing the device using different values

Once you are finished using one device, you can reinitialize the device using the values that you had stored in buffers, thus reestablishing the values of the old device.

5.6 Documenting How to Use Logical Input Devices

When using only a single logical input device of each class in request mode, the user needs minimal documentation to understand how to operate the device. By default, most of the devices require the use of arrow keys or a mouse (to alter the measure of the device), the Return key or the mouse buttons (to trigger the device), and the keyboard (to type a string).

When using more than one active logical input device and when specifying more than one input operating mode, you need to provide the user with more documentation. For example, the user needs to know the following:

- What type of input information each device accepts, and how the application uses this information
- How to change the measure and trigger each device
- How to cycle through devices
- How to end the input process

The following list presents several ways to provide the user with adequate documentation:

- Input instructions located in a distinct window. (This method is illustrated in the create_help subroutine in Example 5–2.)
- Input instructions listed at the beginning of the input process and cleared from the surface once input begins.
- A help screen using segment visibility to control its presence on the workstation surface.
- Written documentation.
- Labels placed on the top of the input device echo areas. (For more information, see the chapter on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.)

Although you can choose any method to document your application, the remainder of this section shows you how to use segment visibility to hide and to present a help screen.

When you use segment visibility to hide a help screen from the user (until the user requests help), you need to find a method of creating the segment without having the text appear on the active workstation's surface. One way to accomplish this is to perform the following tasks:

- Deactivate the OUTIN workstation so that the help text does not appear on its surface.
- Activate WISS to store the help screen segment.
- Create the help screen segment.
- Set the visibility of the help screen segment to INVISIBLE (you do not want the help text to be visible until the user asks for help).
- Associate the help screen segment to the OUTIN workstation.
- Deactivate WISS (no other segments need be stored in WISS).
- Activate the OUTIN workstation to enable further output.

When you finish performing these tasks, the OUTIN workstation contains an invisible segment. When requiring help, the user signals the application. The application makes the segments in the current picture invisible, deactivates the input devices, and makes only the help text visible. When the user chooses, the application resets visibility attributes, and the user can once again view the picture and enter valid input.

Example 5–2 uses a help screen to inform the user about logical input devices. The code in Example 5–2 defines a subroutine that controls the input process that is used to determine if the help screen is needed or not. If the help screen is needed, another subroutine creates a segment containing the text of a help screen, and accomplishes this task so that the user does not view the help screen unless needed.

5.7 Program Examples Used in This Chapter

Example 5–1 and Example 5–2 are located on the VMS kit in the following directory:

```
sys$common:[syshlp.examples.gks]
```

Example 5–1 and Example 5–2 are located on the ULTRIX kit in the following directory:

```
/usr/lib/GKS/examples
```

Example 5–1 illustrates the use of data that is input synchronously (request mode). This program allows the user to do the following:

- Pick a segment.
- Scale more than one segment.
- Reevaluate scaled segments.
- Change the viewport input priority and request locator input.

Example 5–1 Synchronous Input Example

```
/* Header files */

#include <stdio.h>                   /* standard C library I/O header file */
#include <gks.h>                     /* GKS C binding header file */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
                                  /* Constant definitions */
#define     NUM_SIDE_COLORS  6
#define     NUM_ROAD_COLORS  2
#define     NUM_STAR_PTS     6
#define     NUM_MOON_PTS     2
#define     NUM_TREE_PTS    29
#define     NUM_FRONT_PTS   10
#define     NUM_LROOF_PTS    4
#define     NUM_RROOF_PTS    4
#define     NUM_BACK_PTS     6
#define     NUM_RIGHT_PTS    5
#define     NUM_LEFT_PTS     5
#define     NUM_LAND_PTS    15
#define     NUM_SIDE_PTS     4
#define     NUM_ROAD_PTS     4
#define     BACK_IND         0
#define     RED_IND          1
#define     GREEN_IND        2
#define     BLUE_IND         3
static Gfile  *error_file;          /* error file */

                                  /* event and timeout are used with the
                                     AWAIT EVENT call to pause program
                                     execution */
Gevent  event;                    /* an input event */
Gfloat  timeout = 5.00;           /* pause time in seconds */

                   /* number of points in each of the fill
                                     areas comprising the fill area set */
static Gint     front_sizes[2] = {6, 4};

                                  /* house front coodinates */
static Gpoint3  front_pts[NUM_FRONT_PTS] =
                  { {0.1,0.3,0.7},  {0.3,0.3,0.7},  {0.3,0.6,0.7},
                    {0.2,0.75,0.7}, {0.1,0.6,0.7},  {0.1,0.3,0.7},
                    {0.2,0.3,0.7},  {0.25,0.3,0.7}, {0.25,0.4,0.7},
                    {0.2,0.4,0.7} },

                                  /* house back coodinates */
                back_pts[NUM_BACK_PTS] =
                  { {0.1,0.3,0.3},  {0.1,0.6,0.3},  {0.2,0.75,0.3},
                    {0.3,0.6,0.3},  {0.3,0.3,0.3},  {0.1,0.3,0.3} },

                                  /* house right side coodinates */
                right_pts[NUM_RIGHT_PTS] =
                  { {0.3,0.3,0.7},  {0.3,0.3,0.3},  {0.3,0.6,0.3},
                    {0.3,0.6,0.7},  {0.3,0.3,0.7} },

                                  /* house left side coodinates */
                left_pts[NUM_LEFT_PTS] =
                  { {0.1,0.3,0.3},  {0.1,0.3,0.7},  {0.1,0.6,0.7},
                    {0.1,0.6,0.3},  {0.1,0.3,0.3} },

                                  /* roof left side coodinates */
                lroof_pts[NUM_LROOF_PTS] =
                  { {0.1,0.6,0.7},  {0.2,0.75,0.7},
                    {0.2,0.75,0.3}, {0.1,0.6,0.3} },

                                  /* roof right side coodinates */
                rroof_pts[NUM_RROOF_PTS] =
                  { {0.3,0.6,0.7},  {0.3,0.6,0.3},
                    {0.2,0.75,0.3}, {0.2,0.75,0.7} };
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
                                       /* list of text positions for
                                          messages to the user */
static Gpoint   text_pos[13] = { {0.05, 0.95}, {0.05, 0.9},
                                 {0.05, 0.85}, {0.05, 0.8},
                                 {0.05, 0.75}, {0.05, 0.7},
                                 {0.05, 0.25}, {0.05, 0.22},
                                 {0.05, 0.19}, {0.05, 0.16},
                                 {0.05, 0.13}, {0.05, 0.10},
                                 {0.05, 0.07} };

/*******************************************************************/
/*                                                                 */
/*  MAIN ()                                                        */
/*                                                                 */
/*******************************************************************/

main ( )
{
    void    set_up( );
    void    draw_picture( );
    void    clean_up( );

    Gint    ws_id   = 1;        /* workstation identifier */
    Gint    title   = 1,        /* segment identifier for the title */
            stars   = 2,        /* segment identifier for the stars
                                   and moon */
            tree    = 3,        /* segment identifier for the tree */
            side    = 4,        /* segment identifier for the sidewalk */
            road    = 5,        /* segment identifier for the road */
            land    = 6,        /* segment identifier for the land */
            house   = 7;        /* segment identifier for the house */

    Gwstype ws_type = GWS_DEF;  /* workstation type */
    Gconn   conid   = GWC_DEF;  /* connection identifier */

    set_up (ws_id, &ws_type, conid);

    draw_picture (ws_type, ws_id, title, stars, tree, side, road,
                  house, land);

    clean_up (ws_id);

} /* end main */

/*******************************************************************/
/*  SET_UP ()                                                      */
/*                                                                 */
/*  This subroutine sets up the DEC GKS and workstation            */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void set_up (ws_id, ws_type, conid)

Gint       ws_id;         /* workstation identifier */
Gwstype   *ws_type;       /* workstation type */
Gconn      conid;         /* connection identifier */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
{
    Glong      memory = GDEFAULT_MEM_SIZE;  /* memory size */
    Gint       buf_size;                    /* buffer size */
    Gwscat     category;                    /* workstation category */
    Gchar      conid_buff[80];              /* buffer for workstation
                   connection identifier */
    Gwsct      ct;                          /* workstation connection
                   identifier and type */
    Gdefmode   def_mode;                    /* deferral mode */
    Girgmode   irg_mode;                    /* implicit regeneration mode */
    Gint       ret_size;                    /* returned size */
    Gint       status;                      /* returned status */

    /* Initialize GKS.                                            */
    /* A pointer to a specified error file is passed in the call to */
    /* OPEN GKS.  DEC GKS writes all errors to this file.          */

    error_file = fopen ("example_5_1.error", "w");
    status = gopengks (error_file, memory);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error opening GKS.\n");
        exit (0);
       }


    /* Open and activate a workstation.                           */
    /* Both the workstation type and the connection have been      */
    /* initialized to 0 so DEC GKS reads the VMS logical names     */
    /* GKS$WSTYPE and GKS$CONID or the ULTRIX environment variables  */
    /* GKSwstype and GKSconid for the values.                      */

    status = gopenws (ws_id, &conid, ws_type);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error opening workstation.\n");
        exit (0);
       }
    gactivatews (ws_id);

    /* Defer output as long as possible and suppress implicit      */
    /* regeneration.                                               */
    /* The deferral mode specifies when calls to the output        */
    /* functions have their effect.  The implicit regeneration mode */
    /* specifies when changes to the existing picture are seen.    */
    /* This application chooses to control the time when attribute  */
    /* changes are seen on the display so it suppresses the        */
    /* implicit regeneration mode in case it is IMMEDIATE for the   */
    /* workstation.

    def_mode = GASTI;
    irg_mode = GSUPPRESSED;
    gsetdeferst (ws_id, def_mode, irg_mode);
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Determine the workstation connection and type.             */
    /* The application specifies the default workstation and type,  */
    /* which are defined by environment options.  This inquiry      */
    /* returns the connection and type to the application program   */
    /* so it has the information.                                   */

    buf_size = sizeof(conid_buff);
    ct.conn = conid_buff;
    ct.type = ws_type;
    ginqwsconntype (ws_id, buf_size, &ret_size, &ct, &status);
    if (status)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error inquire workstation connection/type.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

    /* Inquire about the category of the workstation. The inquiry   */
    /* function verifies the workstation has a workstation type     */
    /* that indicates it can perform both input and output.         */

    ginqwscategory (ws_type, &category, &status);
    if ((status) || (category != GOUTIN))
       {
        gemergencyclosegks ( );
        fprintf (error_file, "The workstation category is not OUTIN.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

} /* end set_up */

/*******************************************************************/
/*  DRAW_PICTURE ()                                                */
/*                                                                 */
/*  This subroutine draws the picture and places each primitive in  */
/*  a segment.  All objects are defined within the default world    */
/*  coordinate range ([0, 1] x [0, 1] x [0, 1]).   If the           */
/*  workstation has sufficient color capabilities, the objects are  */
/*  drawn in four colors which are defined with the SET COLOUR      */
/*  REPRESENTATION function.  This code assumes an RGB color model. */
/*  Device-independent code should inquire about the available      */
/*  color models and set the color model explicitly.  If the        */
/*  workstation is monochrome, the objects are drawn in the         */
/*  default foreground color.                                       */
/*                                                                 */
/*******************************************************************/

void draw_picture (ws_type, ws_id, title, stars, tree, side, road, house, land)

Gwstype  ws_type;  /* workstation type */
Gint     ws_id,    /* workstation identifier */
         title,    /* segment identifier for the title */
         stars,    /* segment identifier for the stars */
         tree,     /* segment identifier for the tree */
         side,     /* segment identifier for the sidewalk */
         road,     /* segment identifier for the road */
         house,    /* segment identifier for the house */
         land;     /* segment identifier for the land */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
{
    void  go_for_input( );

                                    /* array of the sidewalk colors for
                                       the cell array call */
    static Gint  side_colors[NUM_SIDE_COLORS] = {1, 2, 3, 1, 2, 3},
                                    /* array of the road colors for
                                       the cell array call */
                 road_colors[NUM_ROAD_COLORS] = {1, 3};

    Gint     status,                /* returned status */
             bufsize,               /* buffer size */
             colour,                /* flag to indicate if the workstation
               is color or monochrome */
             fac_size;              /* number of facilities */

    Gfloat   char_height = 0.04,    /* character height */
             line_width  = 3.0,     /* line width */
             edge_width  = 2.0;     /* edge width */
    Glnfac   line_fac;              /* line facility */
    Gintlist line_types_st;         /* integer list stucture of line
                                      types */
    Gint     line_types[20];        /* list of line types */
    Gcofac   col_fac;               /* color facility */
    static Gchar *title_str = "Starry Night";  /* title string */

                                /* 3 sidewalk "corner" points for the
                                   cellarray call */
    static Grect3 side_rectangle_coordinates = {{0.2,  0.0, 0.8},
                                                {0.25, 0.0, 0.8},
                                                {0.2,  0.3, 0.7}};

                                /* 3 road "corner" points for the
                                   cellarray call */
    static Grect3 road_rectangle_coordinates = {{0.0, 0.0, 1.0},
                                                {0.0, 0.0, 0.8},
                                                {1.0, 0.0, 1.0}};
                                /* dimensions of the array with the
                         with the sidewalk colors */
    static Gidim side_rectangle_dim = {1, 6},
                                /* dimensions of the array with the
                         with the road colors */
                 road_rectangle_dim = {2, 1};

                                /* color definitions with RGB values */
    static Gcobundl black = {0.0, 0.0, 0.0};
    static Gcobundl red   = {1.0, 0.0, 0.0};
    static Gcobundl green = {0.0, 1.0, 0.0};
    static Gcobundl blue  = {0.0, 0.0, 1.0};

    Ggdprec  moon_data_rec;  /* GDP data record for the moon */
                                /* moon point values for the GDP */
    static Gpoint   moon_pts[NUM_MOON_PTS] = { {0.9,0.9}, {0.9,0.84} };

                                /* text font and precision */
    static Gtxfp    text_font_prec = {1, GP_STROKE};
                                /* text position */
    static Gpoint3  title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                    text_vec1 = {1.0, 0.0, 0.0},
                    text_vec2 = {0.0, 1.0,-1.0},
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
                            /* star coordinates */
              stars_pts[NUM_STAR_PTS] =
                { {0.05,0.70,0.0}, {0.06,0.86,0.0}, {0.36,0.81,0.0},
                  {0.66,0.86,0.0}, {0.835,0.7,0.0}, {0.92,0.82,0.0} },

                            /* land coordinates */
              land_pts[NUM_LAND_PTS] =
                { {0.0,0.35,0.0}, {0.04,0.375,0.0}, {0.055,0.376,0.0},
                  {0.08,0.36,0.0},  {0.1,0.365,0.0},  {0.3,0.366,0.0},
                  {0.375,0.38,0.0}, {0.44,0.385,0.0}, {0.49,0.375,0.0},
                  {0.56,0.36,0.0},  {0.68,0.38,0.0},  {0.8,0.35,0.0},
                  {0.9,0.359,0.0},  {0.95,0.375,0.0}, {1.0,0.385,0.0} },

                            /* tree coordinates */
              tree_pts[NUM_TREE_PTS] =
                { {0.425,0.28,0.0}, {0.5,0.3,0.0},     {0.52,0.26,0.0},
                  {0.54,0.3,0.0},   {0.6,0.28,0.0},    {0.575,0.33,0.0},
                  {0.56,0.42,0.0},  {0.559,0.49,0.0},  {0.64,0.53,0.0},
                  {0.69,0.57,0.0},  {0.689,0.61,0.0},  {0.66,0.64,0.0},
                  {0.63,0.66,0.0},  {0.645,0.71,0.0},  {0.59,0.76,0.0},
                  {0.53,0.78,0.0},  {0.48,0.75,0.0},   {0.45,0.71,0.0},
                  {0.42,0.65,0.0},  {0.375,0.645,0.0}, {0.35,0.6,0.0},
                  {0.375,0.55,0.0}, {0.44,0.54,0.0},   {0.45,0.5,0.0},
                  {0.515,0.5,0.0},  {0.51,0.425,0.0},  {0.495,0.38,0.0},
                  {0.475,0.33,0.0}, {0.425,0.28,0.0} },

                            /* sidewalk coordinates */
              side_pts[NUM_SIDE_PTS] =
                { {0.2,0.0,0.8},   {0.25,0.0,0.8},
{0.25,0.3,0.7},  {0.2,0.3,0.7} },

                            /* road coordinates */
              road_pts[NUM_ROAD_PTS] =
                { {0.0,0.0,1.0},    {1.0,0.0,1.0},
{1.0,0.0,0.8},   {0.0,0.0,0.8} };

 /* Check if the initialized line width is too wide.            */
 /* The value of line_width is initialized to 3.00.  To avoid    */
 /* requesting a line width that is wider than the workstation's */
 /* widest line, call INQUIRE LINE FACILITIES.  If the line is   */
 /* too wide, set it to the widest available width.              */

 bufsize = sizeof(line_types);
 line_types_st.integers = line_types;
 line_fac.types = &line_types_st;
 ginqlinefacil (&ws_type, bufsize, &fac_size, &line_fac, &status);
 if (line_width * line_fac.nom_width > line_fac.max_width)
     line_width = line_fac.max_width / line_fac.nom_width;
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
/* Check if you are working with a color workstation.          */
/* If 4 colors are available, set each of the 4 color          */
/* representation indices for the specified workstation to the */
/* desired colors.  This type of coding is useful since        */
/* different workstations may have different default color     */
/* representations.                                            */

ginqcolourfacil (&ws_type, bufsize, &fac_size, &col_fac, &status);
colour = (col_fac.coavail == GCOLOUR) && (col_fac.predefined >= 4);
if (colour)
   {
    gsetcolourrep (ws_id, BACK_IND, &black);
    gsetcolourrep (ws_id, RED_IND, &red);
    gsetcolourrep (ws_id, GREEN_IND, &green);
    gsetcolourrep (ws_id, BLUE_IND, &blue);
   }


/* Create a segment for the title.                             */
/* If the workstation is color, make the text red. Set the     */
/* character height and text precision and font.  All values   */
/* are initialized above. Use TEXT 3 to generate the string.   */
/* The position of the text string, the text vectors, and the  */
/* text string are initialized above.                          */

gcreateseg (title);
   if (colour)
       gsettextcolourind (RED_IND);
   gsetcharheight (char_height);
   gsettextfontprec (&text_font_prec);
   gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
gcloseseg ( );


/* Create a segment for the stars and moon.                    */
/* If the workstation is color, make the stars blue. Make the  */
/* star shape a '+' and set the marker size scale factor. Use  */
/* POLYMARKER3 to generate the star markers. The macro         */
/* NUM_STAR_PTS specifies the number of points. The list of    */
/* positions, stars_pts, is initialized above.                 */
/* Use ageneralized drawing primitive (GDP) for the moon.      */
/* The moon object uses GDP GDP_FCCP (-333), which is a filled */
/* circle described by the center and a point on the           */
/* circumference.  The description in the Device Specifics     */
/* Manual indicates that the data record is null so the        */
/* numbers of integers, floats, and strings in the the data    */
/* record are set to 0.  The number of points, 2, is defined   */
/* by the macro NUM_MOON_PTS, and the center and circumference */
/* point are passed in the list moon_pts.  GDP_FCCP is a macro */
/* defining the GDP number.  It is in the C binding include    */
/* file.                                                       */

gcreateseg (stars);
   if (colour)
       gsetmarkercolourind (BLUE_IND);
   gsetmarkertype (GMK_PLUS);
   gsetmarkersize (2.0);
   gpolymarker3 (NUM_STAR_PTS, stars_pts);
   gsetfillintstyle (GSOLID);
   moon_data_rec.gdp_datarec.number_integer = 0;
   moon_data_rec.gdp_datarec.number_float = 0;
   moon_data_rec.gdp_datarec.number_strings = 0;
   ggdp (NUM_MOON_PTS, moon_pts, GDP_FCCP, &moon_data_rec);
gcloseseg ( );
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
/* Create a segment for the tree.                             */
/* If the workstation is color, make the tree green. Set the   */
/* fill area interior style to solid so  the tree is a solid   */
/* shape. Draw the fill area described by the number of points */
/* defined in the NUM_TREE_PTS macro and the list of points in */
/* tree_pts.  The points are initialized above.                */

gcreateseg (tree);
   if (colour)
       gsetfillcolourind (GREEN_IND);
   gsetfillintstyle (GSOLID);
   gfillarea3 (NUM_TREE_PTS, tree_pts);
gcloseseg ( );

/* Create a segment for the sidewalk.                          */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the sidewalk.  If it is monochrome, use FILL AREA 3.        */
/* The CELL ARRAY 3 function divides a parallelogram into      */
/* cells and displays each cell in a specified color.  The     */
/* call requires 3 points on the parallelogram (lower left     */
/* front corner, upper right front corner and upper right back */
/* corner), the number of rows and columns into which the      */
/* parallelogram will be divided, and a 2-dimensional array    */
/* containing the color index values of the cells.  The        */
/* dimensions of the color index array correspond to the       */
/* dimension (number of rows and columns) of the parallelogram */
/* for the C binding.                                          */

gcreateseg(side);
    if (colour)
       gcellarray3 (&side_rectangle_coordinates, &side_rectangle_dim,
                    side_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_SIDE_PTS, side_pts);
       }
gcloseseg ( );

/* Create a segment for the road.                              */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the road.  If it is monochrome, use FILL AREA 3.  All       */
/* values are initialized above.                               */

gcreateseg(road);
    if (colour)
       gcellarray3 (&road_rectangle_coordinates, &road_rectangle_dim,
                    road_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_ROAD_PTS, road_pts);
       }
gcloseseg ( );
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Create a segment for the land.                           */
    /* If the workstation is color, make the line outlining the    */
    /* land green.  Set the line width and make the line dashed.   */
    /* Set the line width, make the line dashed, and use the      */
    /* POLYLINE3 function to draw the land horizon.               */

    gcreateseg (land);
       if (colour)
           gsetlinecolourind (GREEN_IND);
       gsetlinewidth (line_width);
       gsetlinetype (GLN_DASHED);
       gpolyline3 (NUM_LAND_PTS, land_pts);
    gcloseseg ( );

    /* Create a segment for the house and draw it.                */
    /* If the workstation is color make the back, left side, and   */
    /* right side of the house red, and the left and right sides   */
    /* of the roof blue.  Use FILL AREA 3 to draw the back, left   */
    /* side, and right side of the house.  Use FILL AREA SET 3     */
    /* to draw the front of the house.  FILL AREA SET 3 draws      */
    /* the door and the front of the house.  The area for          */
    /* the door is hollow and is contained within the house front  */
    /* area.  The green from the back of the house shows through.  */

    gcreateseg (house);
       gsetfillintstyle (GSOLID);
       if (colour)
           gsetfillcolourind (GREEN_IND);
       gfillarea3 (NUM_BACK_PTS, back_pts);
       if (colour)
           gsetfillcolourind (RED_IND);
       gfillarea3 (NUM_LEFT_PTS, left_pts);
       gfillarea3 (NUM_RIGHT_PTS, right_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gfillarea3 (NUM_LROOF_PTS, lroof_pts);
       gfillarea3 (NUM_RROOF_PTS, rroof_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gsetedgeflag (GEDGE_ON);
       gsetedgewidthscfac (edge_width);
       gfillareaset3 (2, front_sizes, front_pts);
    gcloseseg ( );

    /* Ask the user for input.                                     */

    go_for_input (ws_id, ws_type, title, stars, tree, side, road, house, land);

} /* end draw_picture */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
/*******************************************************************/
/*  GO_FOR_INPUT ()                                                */
/*                                                                 */
/*  This subroutine coordinates the user input.                   */
/*                                                                 */
/*******************************************************************/

void go_for_input (ws_id, ws_type, title, stars, tree, side, road, house, land)

Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */
Gint      title,    /* segment identifier for the title */
          stars,    /* segment identifier for the stars */
          tree,     /* segment identifier for the tree */
          side,     /* segment identifier for the sidewalk */
          road,     /* segment identifier for the road */
          house,    /* segment identifier for the house */
          land;     /* segment identifier for the land */

{
    void      pick_it( );
    void      specify_value( );
    void      view_priority( );

    Gint      unity = 0;           /* default normalization transformation
                                      identifier */
    Gint      picked_segment;      /* identifier for the picked segment */
    Gwscat    category;            /* workstation category */
    Gint      status;              /* returned status */

                                   /* event and timeout are used with the
                                      AWAIT EVENT call to pause program
                                      execution */
    Gevent  event;                 /* an input event */
    Gfloat  timeout = 5.00;        /* amount of time to wait for an event */

    /* Select the unity normalization transformation.              */

    gselntran (unity);

    /* Tell the user to pick input by pressing mouse button 1.     */
    /* Set the character height, the text color index, generate the */
    /* text, and update the workstation surface.                   */

    gsetcharheight (0.028);
    gsettextcolourind (1);
    gtext (&text_pos[0], "Move the cursor to a segment and");
    gtext (&text_pos[1], "trigger input by pressing MB1.");
    gupdatews (ws_id, GPERFORM);

    /* Make all the segments "pickable" so the user can pick any of */
    /* them by setting the detectability of all segments to        */
    /* DETECTABLE.                                                 */

    gsetdet (title, GDETECTABLE);
    gsetdet (stars, GDETECTABLE);
    gsetdet (tree, GDETECTABLE);
    gsetdet (side, GDETECTABLE);
    gsetdet (road, GDETECTABLE);
    gsetdet (house, GDETECTABLE);
    gsetdet (land, GDETECTABLE);

    /* Initialize and request the pick input.                      */

    pick_it (ws_id, ws_type, &picked_segment);
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Let the user scale the segment.                          */

    specify_value (ws_id, ws_type, picked_segment,
                   title, stars, tree, side, road, house, land);

    /* Show the final picture and use AWAIT EVENT to pause the      */
    /* program execution to allow the user to view the picture.     */

    gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );


    /* Make all segments invisible by setting the segment           */
    /* visibility to INVISIBLE and redrawing the segments.          */

    gsetvis (title, GINVISIBLE);
    gsetvis (stars, GINVISIBLE);
    gsetvis (tree, GINVISIBLE);
    gsetvis (side, GINVISIBLE);
    gsetvis (road, GINVISIBLE);
    gsetvis (house, GINVISIBLE);
    gsetvis (land, GINVISIBLE);
    gredrawsegws (ws_id);
    gawaitevent( timeout, &event );


    /* Change the viewport input priority and request locator input. */
    /* Show the picture and pause the program execution to allow    */
    /* the user to view the picture.                                */

    view_priority (ws_id, ws_type);
    gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );

} /* end go_for_input */

/*********************************************************************/
/*  PICK_IT ()                                                       */
/*                                                                   */
/*  This subroutine initializes the pick input and requests pick    */
/*  input from the user for a segment.                               */
/*                                                                   */
/*********************************************************************/

void pick_it (ws_id, ws_type, picked_segment)

Gint      ws_id;           /* workstation identifier */
Gwstype   ws_type;         /* workstation type */
Gint     *picked_segment;  /* segment identifier of the
                              picked segment */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
{
   Gint      dev_num  = 1;         /* pick input device number */
   Gqpick    pick;                 /* pick request structure */
   Gnumdev   num_input_dev;        /* available number of each input
                                      device */
   Gint      status;               /* returned status */
   Gint      pick_buff_size = 80;  /* allocated pick buffer size (be
                                      sure it is large enough) */
   Gchar     pick_buff[80];        /* pick buffer */
   Gint      act_pick_buff_size;   /* required pick buffer size */
   Gpickst   pick_state;           /* pick state data structure */
   Gpick     init_pick;            /* pick data */
   Gint      pick_pet;             /* pick prompt and echo type */
   Glimit    pick_echo_area;       /* pick echo area */
   Gpickrec  pick_data_rec;        /* pick data arecord */

   /* Obtain the number of pick logical input devices supported by */
   /* the workstation.                                              */

   ginqnumavailinput (&ws_type, &num_input_dev, &status);
   if ((num_input_dev.pick == 0) || (status))
      {
       gemergencyclosegks ( );
       fprintf (error_file, "The workstation does not support pick input.\n");
       fprintf (error_file, "Error status: %d\n", status);
       exit (0);
      }

   /* Inquire about and initialize a pick logical input device.    */
   /* The call to INQUIRE PICK DEVICE STATE obtains the current    */
   /* pick input values which, in this instance, are the default   */
   /* pick values since the first pick input initialization has    */
   /* not occurred yet.  Not all the current pick values are       */
   /* suitable for the application, so INITIALIZE PICK is used to  */
   /* change them.  The pick status is set to OK which indicates   */
   /* that cursor motion is not required before triggering the     */
   /* the input device.  This makes the program easier to use. The */
   /* prompt and echo type, with a default value of 1 (pick by     */
   /* primitive), is set to value 3 (pick by segment) since all    */
   /* objects in the scene are in segments and it is easier to     */
   /* pick by segment.  Also, the aperture size is multiplied by   */
   /* 4 to make the aperture easier to see.  The call to           */
   /* INITIALIZE PICK establishes the new values.                  */

   pick_state.record.pickpet1_datarec.data = pick_buff;
   ginqpickst (ws_id, dev_num, ws_type, pick_buff_size,
               &act_pick_buff_size, &pick_state, &status);
   init_pick.status = GP_OK;
   init_pick.seg = 1;
   init_pick.pickid = 0;
   pick_pet = 3;
   pick_echo_area.xmin = pick_state.e_area.xmin;
   pick_echo_area.xmax = pick_state.e_area.xmax;
   pick_echo_area.ymin = pick_state.e_area.ymin;
   pick_echo_area.ymax = pick_state.e_area.ymax;
   pick_data_rec.pickpet3_datarec.aperture =
                  pick_state.record.pickpet3_datarec.aperture * 4.0;
   pick_data_rec.pickpet3_datarec.data = pick_buff;
   ginitpick (ws_id, dev_num, &init_pick, pick_pet,
              &pick_echo_area, &pick_data_rec);
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Loop until the user picks a valid segment.  REQUEST PICK is  */
    /* the request for pick input and the user must select a       */
    /* segment using the mouse.  If the returned pick status is OK, */
    /* a segment has been picked successfully and its identifier is */
    /* available.  To perform a pick, the user must trigger input   */
    /* while the segment is highlighted.   If the user does not     */
    /* select a valid segment (pick status is NOPICK or NONE), the  */
    /* segments are redrawn and the user is prompted to pick a      */
    /* segment.                                                     */

    *picked_segment = 0;
    while (*picked_segment == 0)
       {
        greqpick (ws_id, dev_num, &pick);
        if (pick.status == GP_OK)
            *picked_segment = pick.seg;
        else
           {
            gredrawsegws (ws_id);
            gtext (&text_pos[0], "You must pick a segment!");
            gtext (&text_pos[1], "Move the cursor to a segment and");
            gtext (&text_pos[2], "trigger input by pressing MB1.");
            gupdatews (ws_id, GPERFORM);
           }
       } /* end while (*picked_segment == 0) */

    gredrawsegws (ws_id);

} /* end pick_it */

/*******************************************************************/
/*  SPECIFY_VALUE ()                                               */
/*                                                                 */
/*  This subroutine uses a valuator logical input device to obtain */
/*  the scaling factor for a picked segment from the user.         */
/*                                                                 */
/*******************************************************************/

void specify_value (ws_id, ws_type, picked_segment,
                    title, stars, tree, side, road, house, land)

Gint      ws_id;            /* workstation identifier */
Gwstype   ws_type;          /* workstation type */
Gint      picked_segment;   /* picked segment identifier */
Gint      title,            /* segment identifier for the title */
          stars,            /* segment identifier for the stars */
          tree,             /* segment identifier for the tree */
          side,             /* segment identifier for the sidewalk */
          road,             /* segment identifier for the road */
          house,            /* segment identifier for the house */
          land;             /* segment identifier for the land */

{
    void      satisfied_choice( );
    Gint      dev_num = 1;          /* valuator device number */
    Gqval     val;
                                    /* identity transformation vectors
                                       used to create the identity
                                       transformation matrix */
    static Gpoint3   fixed_pt = {0.0, 0.0, 0.0};
    static Gpoint3   translate_vec = {0.0, 0.0, 0.0};
    static Gangle3   rotate_vec = {0.0, 0.0, 0.0};
    static Gscale3   scale_vec = {1.0, 1.0, 1.0};
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
Gfloat    xform_matrix[3][4];   /* segment transformation matrix */
Gint      scale;
Gnumdev   num_input_dev;        /* available number of each input
                                   device */
Gint      status;               /* returned status */
Gint      buff_size = 80;       /* allocated valuator buffer size */
Gchar     val_buff[80];         /* valuator buffer */
Gint      state_size;           /* required valuator buffer size */
Gvalst    val_state;            /* valuator state data structure */
Gint      prompt_echo_type;     /* valuator prompt and echo type */
Glimit    echo_area;            /* valuator echo area */
Gvalrec   val_data_rec;         /* valuator data record */
Gfloat    val_init_val;         /* initial value for the valuator */

/* Obtain the number of valuator logical input devices          */
/* supported by the workstation.                                */

ginqnumavailinput (&ws_type, &num_input_dev, &status);
if ((num_input_dev.valuator == 0) || (status))
   {
    gemergencyclosegks ( );
    fprintf (error_file,
             "The workstation does not support valuator input.\n");
    fprintf (error_file, "Error status: %d\n", status);
    exit (0);
   }

/* Create the identity matrix which will be used as the segment */
/* transformation matrix.  The identity segment transformation  */
/* matrix will cause no change to the segment stored in the NDC */
/* space.                                                       */

gevaltran3 (&fixed_pt, &translate_vec, &rotate_vec,
             &scale_vec, GNDC, xform_matrix);

/* Since all devices are initially in request mode by default,  */
/* initialize the valuator logical input device.  INQUIRE       */
/* VALUATOR DEVICE STATE returns the current values of the      */
/* logical input device. Some of the values are adjusted to fit */
/* the application and INITIALIZE VALUATOR establishes the new  */
/* values.  The echo area and range values are adjusted.  The   */
/* valuator displays the initial value 1.  A message is output  */
/* to the user telling him/her to use the valuator to indicate  */
/* the scaling factor for the selected segment.                 */

val_state.record.valpet1_datarec.title_string = val_buff;
ginqvalst (ws_id, dev_num, buff_size, &state_size, &val_state, &status);
prompt_echo_type = 1;
echo_area.xmin = val_state.e_area.xmax * 0.75;
echo_area.xmax = val_state.e_area.xmax;
echo_area.ymin = val_state.e_area.ymin;
echo_area.ymax = val_state.e_area.ymax;
val_data_rec.valpet1_datarec.low  = 0.5;
val_init_val = 1.0;
val_data_rec.valpet1_datarec.high = 1.5;
val_data_rec.valpet1_datarec.title_string = (char *)0;
ginitval (ws_id, dev_num, val_init_val, prompt_echo_type,
          &echo_area, &val_data_rec);

gtext (&text_pos[0], "Scale the selected object using the");
gtext (&text_pos[1], "valuator input device.");
gupdatews (ws_id, GPERFORM);
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Let the user scale the object until he/she is done.         */
    /* Call REQUEST VALUATOR to obtain the scaling factor from the  */
    /* user.  Use the fixed point value, which is best for the      */
    /* segment, for scaling.  Set the scaling factor of each        */
    /* dimension to the selected value.  Call ACCUMULATE            */
    /* TRANSFORMATION MATRIX 3 to calculate the new segment         */
    /* transformation matrix based on the old one. Call SET SEGMENT */
    /* TRANSFORMATION 3 to apply the new transformation matrix to   */
    /* the segment and redraw all the segments on the workstation.  */
    /* At the bottom of the loop verify the user is done.           */

    scale = 1;
    while (scale)
       {
        /* get amount to scale segment */
        greqval (ws_id, dev_num, &val);

        switch (picked_segment)
           {
            case 1: fixed_pt.x = 0.5;
                    fixed_pt.y = 0.1;
                    fixed_pt.z = 0.8;
                    break;
            case 2: fixed_pt.x = 0.5;
                    fixed_pt.y = 0.8;
                    fixed_pt.z = 0.0;
                    break;
            case 3: fixed_pt.x = 0.52;
                    fixed_pt.y = 0.35;
                    fixed_pt.z = 0.0;
                    break;
            case 4: fixed_pt.x = 0.225;
                    fixed_pt.y = 0.225;
                    fixed_pt.z = 0.15;
                    break;
            case 5: fixed_pt.x = 0.5;
                    fixed_pt.y = 0.075;
                    fixed_pt.z = 0.0;
                    break;
            case 6: fixed_pt.x = 0.05;
                    fixed_pt.y = 0.37;
                    fixed_pt.z = 0.0;
                    break;
            case 7: fixed_pt.x = 0.200;
                    fixed_pt.y = 0.525;
                    fixed_pt.z = 0.500;
                    break;
           } /* end switch (picked_segment) */

        scale_vec.x_scale = scale_vec.y_scale = scale_vec.z_scale = val.val;
        gaccumtran3 (xform_matrix, &fixed_pt, &translate_vec, &rotate_vec,
                     &scale_vec, GNDC, xform_matrix);
        gsetsegtran3 (picked_segment, xform_matrix);
        gredrawsegws (ws_id);
        satisfied_choice (ws_id, ws_type, &scale);

       } /* end while (scale) */
}  /* end specify_value */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
/*******************************************************************/
/*  SATISFIED_CHOICE ()                                            */
/*                                                                 */
/*  This subroutine uses the choice logical input device to check  */
/*  whether the user wants to continue scaling the segment.        */
/*                                                                 */
/*******************************************************************/

void satisfied_choice (ws_id, ws_type, scale)

Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */
Gint     *scale;    /* flag indicating continuation or end of
                       scaling */

{
    Gint          dev_num = 1;      /* logical input device number */
    Gchoice       init_choice;      /* choice data */
    Gint          pet = 1;          /* choice prompt and echo type */
    Gdspsize      dsp_siz;          /* display size */
    static Glimit echo_area;        /* echo area */
    Gchoicerec    choice_rec;       /* choice data record */
    Gqchoice      choice;           /* choice request response */

                                    /* list of string lengths for the
                                       choice device */
    static Gint str_lengths[2] = {3, 2};
                                    /* list of strings for the choice
                                       device */
    static Gchar *strings[] = {"yes", "no"};
                                    /* choice device title */
    static Gchar data_rec[] = "Scale";

    Gnumdev      num_input_dev;     /* available number of each input
                                       device */
    Gint         status;            /* returned status */


    /* Inquire the size of the display.  This information is needed */
    /* to set the echo area in a device independent manner.        */

    ginqdisplaysize (&ws_type, &dsp_siz, &status);
    if (status)
       {
        gemergencyclosegks ( );
        fprintf (stderr, "Inquire display size.\n");
        fprintf (stderr, "Error status: %d\n", status);
        exit (0);
       }


    /* Obtain the number of choice logical input devices supported  */
    /* by the workstation.                                          */

    ginqnumavailinput (&ws_type, &num_input_dev, &status);
    if ((num_input_dev.choice == 0) || (status))
       {
        gemergencyclosegks ( );
        fprintf (error_file,
                 "The workstation does not support choice input.\n");
        fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Since all devices are initially in request mode by default,  */
    /* initialize the choice logical input device.  INQUIRE  CHOICE */
    /* DEVICE STATE returns the current values of the logical       */
    /* input device. Some of the values are adjusted to fit the     */
    /* application, and INITIALIZE CHOICE establishes new string    */
    /* and title values. REQUEST CHOICE gets the user's selection.  */

    init_choice.status = GC_OK;
    init_choice.choice = 1;
    echo_area.xmin = 0.0;
    echo_area.xmax = dsp_siz.device.x * 0.3;
    echo_area.ymin = 0.0;
    echo_area.ymax = dsp_siz.device.y * 0.3;
    choice_rec.choicepet1_datarec.number = 2;
    choice_rec.choicepet1_datarec.lengths = str_lengths;
    choice_rec.choicepet1_datarec.strings = strings;
    choice_rec.choicepet1_datarec.title_string = data_rec;
    ginitchoice (ws_id, dev_num, &init_choice, pet, &echo_area, &choice_rec);

    greqchoice (ws_id, dev_num, &choice);

    if ((choice.status == GC_OK) && (choice.choice == 1))
        *scale = 1;   /* scale segment again */
    else
        *scale = 0;   /* finished scaling segment */

} /* end satisfied_choice */

/*******************************************************************/
/*  VIEW_PRIORITY ()                                               */
/*                                                                 */
/*  This subroutine changes the viewport input priority and        */
/*  requests locator input.                                        */
/*                                                                 */
/*******************************************************************/

void view_priority (ws_id, ws_type)
Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */

{
    Gint      num_points = 5;       /* number of points in each outline */

                                    /* line outlining the larger world
                                       coordinate space */
    static Gpoint3  points_a[5] = {0.0,0.0,0.0, 1.0,0.0,0.0, 1.0,1.0,0.0,
                                   0.0,1.0,0.0, 0.0,0.0,0.0};

                                    /* line outlining the smaller world
                                       coordinate space */
    static Gpoint3  points_b[5] = {0.0,0.0,0.0, 0.5,0.0,0.0, 0.5,1.0,0.0,
                                   0.0,1.0,0.0, 0.0,0.0,0.0};
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
Glimit3      window,                    /* normalization window */
             viewport;                  /* normalization viewport */
Gint         house_norm = 1;            /* normalization transformation
                                           index for the house */
Gint         dev_num = 1;               /* locator device number */
Gqloc        loc_resp;                  /* locator response to a request */
Gnumdev      num_input_dev;             /* available number of
                                           each input device */
Gint         status;                    /* returned status */
Gchar        locator_msg[80];           /* message buffer */
static Gloc  init_loc = {0, 0.0, 0.0};  /* locator data */
Gint         loc_pet = 1;               /* locator prompt and echo type */
Glimit       loc_echo_area;             /* locator echo area */
Glocrec      loc_rec;                   /* locator data record */
Gdspsize     display_size;              /* workstation display size */

/* Obtain the number of valuator logical input devices          */
/* supported by the workstation.                                */

ginqnumavailinput (&ws_type, &num_input_dev, &status);
if ((num_input_dev.locator == 0) || (status))
   {
    gemergencyclosegks ( );
    fprintf (error_file,
             "The workstation does not support locator input.\n");
    fprintf (error_file, "Error status: %d\n", status);
    exit (0);
   }

/* Draw the house and output the text directing the user to     */
/* select a point inside viewport 0. Viewport 0 is the viewport */
/* associated with normalization transformation 0 which is the  */
/* default transformation.                                      */

gselntran (0);
gsetfillintstyle (GSOLID);
gfillarea3 (NUM_BACK_PTS, back_pts);
gfillarea3 (NUM_LEFT_PTS, left_pts);
gfillarea3 (NUM_RIGHT_PTS, right_pts);
gfillarea3 (NUM_LROOF_PTS, lroof_pts);
gfillarea3 (NUM_RROOF_PTS, rroof_pts);
gsetedgeflag (GEDGE_ON);
gfillareaset3 (2, front_sizes, front_pts);
gsettextcolourind (1);
gsetcharheight (0.02);
gtext (&text_pos[0],
       "Viewport zero (0), represented by the square outline,");
gtext (&text_pos[1],
       "has a higher input priority than viewport one (1).");
gtext (&text_pos[2],
       "Use MB1 to select a point inside viewport zero (0).");
gtext (&text_pos[6],
       "NOTE: The DECwindows and Motif workstation types will not grab the");
gtext (&text_pos[7],
       "      cursor. To see the effect of different viewport priorities,");
gtext (&text_pos[8],
       "      select a position in the right half of the viewport and press");
gtext (&text_pos[9],
       "      MB1.  Do not move your cursor. In the next frame, press MB1");
gtext (&text_pos[10],
       "      again. This will show you the different coordinates for the");
gtext (&text_pos[11],
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
           "      same location that result from the different priorities of");
    gtext (&text_pos[12],
           "      the viewports.");
    gupdatews (ws_id, GPERFORM);

    /* Outline the default world coordinate space.                 */

    gsetlinetype (GLN_SOLID);
    gpolyline3 (num_points, points_a);

    /* Inquire the size of the display.                            */

    ginqdisplaysize (&ws_type, &display_size, &status);


    /* Initialize the echo area of the locator device to the full   */
    /* display surface and request the user to trigger an input on  */
    /* the device.  The initial locator position argument specifies */
    /* where the locator should be placed initially.  Neither       */
    /* DECwindows(XUI) or Motif will grab the cursor so the locator */
    /* echo is not positioned at the designated value for these     */
    /* devices.                                                     */

    loc_echo_area.xmin = 0.0;
    loc_echo_area.xmax = 1.0 * display_size.device.y;
    loc_echo_area.ymin = 0.0;
    loc_echo_area.ymax = 1.0 * display_size.device.y;
    ginitloc (ws_id, dev_num, &init_loc, loc_pet, &loc_echo_area, &loc_rec);
    greqloc (ws_id, dev_num, &loc_resp);
    gclearws (ws_id, GALWAYS);

    /* Map the house onto a small portion of NDC space.            */

    window.xmin = 0.0;
    window.xmax = 0.5;
    window.ymin = 0.0;
    window.ymax = 1.0;
    window.zmin = 0.0;
    window.zmax = 1.0;
    gsetwindow3 (house_norm, &window);
    viewport.xmin = 0.5;
    viewport.xmax = 1.0;
    viewport.ymin = 0.0;
    viewport.ymax = 1.0;
    viewport.zmin = 0.0;
    viewport.zmax = 1.0;
    gsetviewport3 (house_norm, &viewport);
    gselntran (house_norm);
    gsetfillintstyle (GSOLID);
    gfillarea3 (NUM_BACK_PTS, back_pts);
    gfillarea3 (NUM_LEFT_PTS, left_pts);
    gfillarea3 (NUM_RIGHT_PTS, right_pts);
    gfillarea3 (NUM_LROOF_PTS, lroof_pts);
    gfillarea3 (NUM_RROOF_PTS, rroof_pts);
    gsetedgeflag (GEDGE_ON);
    gfillareaset3 (2, front_sizes, front_pts);
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
    /* Select the identity transformation and output the text       */
    /* messages to the user.                                         */

    gselntran (0);
    sprintf (locator_msg,
             "Normalization xform: %2d   World Coordinate: (%f, %f)",
             loc_resp.loc.transform,
             loc_resp.loc.position.x, loc_resp.loc.position.y);
    gtext (&text_pos[0], locator_msg);
    gtext (&text_pos[1],
           "Viewport one (1), represented by the rectangular outline,");
    gtext (&text_pos[2],
           "has higher input priority than viewport zero (0).");
    gtext (&text_pos[3],
           "Use MB1 to select a point inside viewport one (1).");
    gupdatews (ws_id, GPERFORM);

    /* Select the second normalization transformation which has the */
    /* smaller window and viewport, draw the outline of the         */
    /* viewport.                                                     */

    gselntran (house_norm);
    gpolyline3 (num_points, points_b);


    /* Set the input priority of the smaller viewport higher than   */
    /* the input priority of the larger viewport.                   */

    gsetviewportinputpri (1, 0, GHIGHER);

    /* Initialize the echo area to represent the right half of the  */
    /* default display and request the user to trigger the locator  */
    /* input device.  The position is transformed back to WC using  */
    /* the current normalization transformation information.        */
    /* If the cursor is in the same position for both triggers (and */
    /* therefore in the right half of NDC space), it is easy to see */
    /* the different point coordinates resulting from the different */
    /* viewport priorities.                                          */

    loc_echo_area.xmin = 0.5 * display_size.device.y;
    loc_echo_area.xmax = 1.0 * display_size.device.y;
    loc_echo_area.ymin = 0.0;
    loc_echo_area.ymax = 1.0 * display_size.device.y;
    ginitloc (ws_id, dev_num, &init_loc, loc_pet, &loc_echo_area, &loc_rec);
    greqloc (ws_id, dev_num, &loc_resp);

    /* Output the position information in WC for the user. Use the   */
    /* default normalization transformation for the message.         */

    sprintf (locator_msg,
             "Normalization xform: %2d   World Coordinate: (%f, %f)",
             loc_resp.loc.transform,
             loc_resp.loc.position.x, loc_resp.loc.position.y);
    gselntran (0);
    gtext (&text_pos[4], locator_msg);
    gupdatews (ws_id, GPERFORM);

}  /* end view_priority */
```

(continued on next page)

Example 5–1 (Cont.) Synchronous Input Example

```
/*******************************************************************/
/*  CLEAN_UP ()                                                    */
/*                                                                 */
/*  This subroutine cleans up the DEC GKS and workstation          */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void clean_up (ws_id)

Gint ws_id;  /* workstation identifier */

{
    /* Deactivate all active workstations and close all open       */
    /* workstations before closing GKS.                            */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ( );

} /* end clean_up */
```

Example 5–2 allows the user to enter input using pick, choice, and valuator devices that are active concurrently. In essence, this example performs the same task as the example in Example 5–1, allowing the user to scale a segment. However, Example 5–2 also allows the user to do the following:

- Independently scale all the segments in the Starry Night picture
- Scale segments more than once without having to be prompted by the application
- Use a help screen interactively
- Stop subsequent scaling of a segment

Example 5–2 Asynchronous Input Example

```
/* Header files */

#include <stdio.h>                  /* standard C library I/O header file */
#include <gks.h>                    /* GKS C binding header file */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
                                    /* Constant definitions */
#define     NUM_SIDE_COLORS  6
#define     NUM_ROAD_COLORS  2
#define     NUM_STAR_PTS     6
#define     NUM_MOON_PTS     2
#define     NUM_TREE_PTS    29
#define     NUM_FRONT_PTS   10
#define     NUM_LROOF_PTS    4
#define     NUM_RROOF_PTS    4
#define     NUM_BACK_PTS     6
#define     NUM_RIGHT_PTS    5
#define     NUM_LEFT_PTS     5
#define     NUM_LAND_PTS    15
#define     NUM_SIDE_PTS     4
#define     NUM_ROAD_PTS     4
#define     BACK_IND         0
#define     RED_IND          1
#define     GREEN_IND        2
#define     BLUE_IND         3
/*******************************************************************/
/*                                                                 */
/*  MAIN ()                                                        */
/*                                                                 */
/*******************************************************************/

main ( )

{
    void    set_up( );
    void    draw_picture( );
    void    clean_up( );

    Gint    ws_id   = 1;          /* workstation identifier */
    Gint    wiss    = 2;          /* workstation independent segment
                                          storage identifier */
    Gint    title   = 1,          /* segment identifier for the title */
            stars   = 2,          /* segment identifier for the stars
                                     and moon */
            tree    = 3,          /* segment identifier for the tree */
            side    = 4,          /* segment identifier for the sidewalk */
            road    = 5,          /* segment identifier for the road */
            land    = 6,          /* segment identifier for the house */
            house   = 7;          /* segment identifier for the land */

    Gwstype ws_type = GWS_DEF;   /* workstation type */
    Gconn   conid   = GWC_DEF;   /* connection identifier */

    set_up (ws_id, &ws_type, conid, wiss);

    draw_picture (ws_type, ws_id, wiss, title, stars, tree, side, road,
                  house, land);

    clean_up (ws_id, wiss);

} /* end main */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
/*******************************************************************/
/*  SET_UP ()                                                      */
/*                                                                 */
/*  This subroutine sets up the DEC GKS and workstation            */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void set_up (ws_id, ws_type, conid, wiss)

Gint       ws_id;         /* workstation identifier */
Gwstype   *ws_type;       /* workstation type */
Gconn      conid;         /* connection identifier */
Gint       wiss;          /* WISS workstation identifier */

{
    Glong      memory = GDEFAULT_MEM_SIZE;  /* memory size */
    Gint       buf_size;                    /* buffer size */
    Gwscat     category;                    /* workstation category */
    Gchar      conid_buff[80];              /* buffer for workstation
                                               connection identifier */
    Gwsct      ct;                          /* workstation connection
                                               identifier and type */
    Gdefmode   def_mode;                    /* deferral mode */
    Girgmode   irg_mode;                    /* implicit regeneration mode */
    Gint       ret_size;                    /* returned size */
    Gint       status;                      /* returned status */
    Glevel     gks_level;                   /* level of GKS implementation */
    Gwstype    wiss_ws_type = GWS_WISS;     /* WISS workstation type */

    /* Initialize GKS.                                              */
    /* A pointer to a specified error file is passed in the call to */
    /* OPEN GKS.  DEC GKS writes all errors to this file.           */

    status = gopengks (stderr, memory);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (stderr, "Error opening GKS.\n");
        exit (0);
       }

    /* Open and activate a workstation.                               */
    /* Both the workstation type and the connection have been         */
    /* initialized to 0 so DEC GKS reads the VMS logical names,       */
    /* GKS$WSTYPE and GKS$CONID, or the ULTRIX environment variables, */
    /* GKSwstype and GKSconid, for the values.                        */

    status = gopenws (ws_id, &conid, ws_type);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (stderr, "Error opening workstation.\n");
        exit (0);
       }
    gactivatews (ws_id);
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
/* Defer output as long as possible and suppress implicit       */
/* regeneration.                                                 */
/* The deferral mode specifies when calls to the output         */
/* functions have their effect.  The implicit regeneration mode */
/* specifies when changes to the existing picture are seen.     */
/* This application chooses to control the time when attribute  */
/* changes are seen on the display so it suppresses the         */
/* implicit regeneration mode in case it is IMMEDIATE for the   */
/* workstation.                                                  */

def_mode = GASTI;
irg_mode = GSUPPRESSED;
gsetdeferst (ws_id, def_mode, irg_mode);

/* Determine the workstation connection and type.               */
/* The application specifies the default workstation and type,  */
/* which are defined by environment options.  This inquiry      */
/* returns the connection and type to the application program,  */
/* so it has the information.                                   */

buf_size = sizeof(conid_buff);
ct.conn = conid_buff;
ct.type = ws_type;
ginqwsconntype (ws_id, buf_size, &ret_size, &ct, &status);
if (status)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "Error inquire workstation connection/type.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }

/* Inquire about the category of the workstation. The inquiry   */
/* function verifies the workstation has a workstation type     */
/* that can perform both input and output.                      */

ginqwscategory (ws_type, &category, &status);
if ((status) || (category != GOUTIN))
   {
    gemergencyclosegks ( );
    fprintf (stderr, "The workstation category is invalid.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }

/* Inquire about the level of the GKS implementation to verify  */
/* that WISS, input, and output are fully supported.            */

ginqlevelgks (&gks_level, &status);
if (status || (gks_level != GL2C))
   {
    gemergencyclosegks ( );
    fprintf (stderr,
             "This level of GKS does not support WISS and full input.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    /* Open WISS to store the help information segment.             */

    gopenws (wiss, &conid, &wiss_ws_type);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (stderr, "Error opening WISS workstation.\n");
        exit (0);
       }

} /* end set_up */

/*******************************************************************/
/*  DRAW_PICTURE ()                                                */
/*                                                                 */
/*  This subroutine draws the picture and places each primitive in */
/*  a segment.  All objects are defined within the default world   */
/*  coordinate range ([0, 1] x [0, 1] x [0, 1]).   If the          */
/*  workstation has sufficient color capabilities, the objects are */
/*  drawn in four colors which are defined with the SET COLOUR     */
/*  REPRESENTATION function.  This code assumes an RGB color model. */
/*  Device-independent code should inquire about the available     */
/*  color models and set the color model explicitly.  If the       */
/*  workstation is monochrome, the objects are drawn in the        */
/*  default foreground color.                                      */
/*                                                                 */
/*******************************************************************/

void draw_picture (ws_type, ws_id, wiss, title, stars, tree, side,
                   road, house, land)

Gwstype ws_type;  /* workstation type */
Gint    ws_id,    /* workstation identifier */
        wiss,     /* WISS workstation identifier */
        title,    /* segment identifier for the title */
        stars,    /* segment identifier for the stars */
        tree,     /* segment identifier for the tree */
        side,     /* segment identifier for the sidewalk */
        road,     /* segment identifier for the road */
        house,    /* segment identifier for the house */
        land;     /* segment identifier for the land */
{
    void     go_for_input( );

                                   /* array of the sidewalk colors for the
                                      cellarray call */
    static Gint  side_colors[NUM_SIDE_COLORS] = {2, 3, 1, 2, 3, 1},
                                   /* array of the road colors for the
                                      cellarray call */
                 road_colors[NUM_ROAD_COLORS]  = {1, 3};
    Gint     status,               /* returned status */
             bufsize,              /* buffer size */
             colour,               /* flag to indicate if the workstation
                                      is color or monochrome */
             fac_size;             /* number of facilities */
    Gfloat   char_height = 0.04,   /* character height */
             line_width  = 3.0,    /* line width */
             edge_width = 2.0;     /* edge width */
    Glnfac   line_fac;             /* line facility */
    Gintlist line_types_st;        /* integer list stucture of line types */
    Gint     line_types[20];       /* list of line types */
    Gcofac   col_fac;              /* color facility */
    static Gchar *title_str = "Starry Night";  /* title string */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
                                /* 3 sidewalk "corner" points for the
                                   cellarray call */
static Grect3 side_rectangle_coordinates = { {0.2,  0.0, 0.8},
                                             {0.25, 0.0, 0.8},
                                             {0.2,  0.3, 0.7} };

                                /* 3 road "corner" points for the
                                   cellarray call */
static Grect3 road_rectangle_coordinates = { {0.0, 0.0, 1.0},
                                             {0.0, 0.0, 0.8},
                                             {1.0, 0.0, 1.0} };
                                /* dimensions of the array with the
                                   with the sidewalk colors */
static Gidim side_rectangle_dim = {1, 6},
                                /* dimensions of the array with the
                                   with the road colors */
             road_rectangle_dim = {2, 1};
                                /* color definitions with RGB values */
static Gcobundl black = {0.0, 0.0, 0.0};
static Gcobundl red   = {1.0, 0.0, 0.0};
static Gcobundl green = {0.0, 1.0, 0.0};
static Gcobundl blue  = {0.0, 0.0, 1.0};

Ggdprec  moon_data_rec;  /* GDP data record for the moon */

                                /* moon point values for the GDP */
static Gpoint   moon_pts[NUM_MOON_PTS] = { {0.9,0.9}, {0.9,0.84} };

static Gint     front_sizes[2] = {6, 4};

                                /* text font and precision */
static Gtxfp text_font_prec = {1, GP_STROKE};
                                 /* text position */
static Gpoint3  title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                text_vec1 = {1.0, 0.0, 0.0},
                text_vec2 = {0.0, 1.0,-1.0},
                                /* star coordinates */
                stars_pts[NUM_STAR_PTS] =
                  { {0.05,0.70,0.0}, {0.06,0.86,0.0}, {0.36,0.81,0.0},
                    {0.66,0.86,0.0}, {0.835,0.7,0.0}, {0.92,0.82,0.0} },

                                /* land coordinates */
                land_pts[NUM_LAND_PTS] =
                  { {0.0,0.35,0.0}, {0.04,0.375,0.0}, {0.055,0.376,0.0},
                    {0.08,0.36,0.0},  {0.1,0.365,0.0},  {0.3,0.366,0.0},
                    {0.375,0.38,0.0}, {0.44,0.385,0.0}, {0.49,0.375,0.0},
                    {0.56,0.36,0.0},  {0.68,0.38,0.0},  {0.8,0.35,0.0},
                    {0.9,0.359,0.0},  {0.95,0.375,0.0}, {1.0,0.385,0.0} },

                                /* tree coordinates */
                tree_pts[NUM_TREE_PTS] =
                  { {0.425,0.28,0.0}, {0.5,0.3,0.0},     {0.52,0.26,0.0},
                    {0.54,0.3,0.0},   {0.6,0.28,0.0},    {0.575,0.33,0.0},
                    {0.56,0.42,0.0},  {0.559,0.49,0.0},  {0.64,0.53,0.0},
                    {0.69,0.57,0.0},  {0.689,0.61,0.0},  {0.66,0.64,0.0},
                    {0.63,0.66,0.0},  {0.645,0.71,0.0},  {0.59,0.76,0.0},
                    {0.53,0.78,0.0},  {0.48,0.75,0.0},   {0.45,0.71,0.0},
                    {0.42,0.65,0.0},  {0.375,0.645,0.0}, {0.35,0.6,0.0},
                    {0.375,0.55,0.0}, {0.44,0.54,0.0},   {0.45,0.5,0.0},
                    {0.515,0.5,0.0},  {0.51,0.425,0.0},  {0.495,0.38,0.0},
                    {0.475,0.33,0.0}, {0.425,0.28,0.0} },
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
                        /* house front coordinates */
        front_pts[NUM_FRONT_PTS] =
          { {0.1,0.3,0.7},  {0.3,0.3,0.7},  {0.3,0.6,0.7},
            {0.2,0.75,0.7}, {0.1,0.6,0.7},  {0.1,0.3,0.7},
            {0.2,0.3,0.7},  {0.25,0.3,0.7}, {0.25,0.4,0.7},
            {0.2,0.4,0.7} },

                        /* house back coordinates */
        back_pts[NUM_BACK_PTS] =
          { {0.1,0.3,0.3},  {0.1,0.6,0.3}, {0.2,0.75,0.3},
            {0.3,0.6,0.3},  {0.3,0.3,0.3}, {0.1,0.3,0.3} },

                        /* house right side coordinates */
        right_pts[NUM_RIGHT_PTS] =
          { {0.3,0.3,0.7},  {0.3,0.3,0.3}, {0.3,0.6,0.3},
            {0.3,0.6,0.7},  {0.3,0.3,0.7} },

                        /* house left side coordinates */
        left_pts[NUM_LEFT_PTS] =
          { {0.1,0.3,0.3},  {0.1,0.3,0.7}, {0.1,0.6,0.7},
            {0.1,0.6,0.3},  {0.1,0.3,0.3} },

                        /* roof left side coordinates */
        lroof_pts[NUM_LROOF_PTS] =
          { {0.1,0.6,0.7},  {0.2,0.75,0.7},
            {0.2,0.75,0.3}, {0.1,0.6,0.3} },

                        /* roof right side coordinates */
        rroof_pts[NUM_RROOF_PTS] =
          { {0.3,0.6,0.7},  {0.3,0.6,0.3},
            {0.2,0.75,0.3}, {0.2,0.75,0.7} },

                        /* sidewalk coordinates */
        side_pts[NUM_SIDE_PTS] =
          { {0.2,0.0,0.8},   {0.25,0.0,0.8},
            {0.25,0.3,0.7},  {0.2,0.3,0.7} },

                        /* road coordinates */
        road_pts[NUM_ROAD_PTS] =
          { {0.0,0.0,1.0},    {1.0,0.0,1.0},
            {1.0,0.0,0.8},    {0.0,0.0,0.8} };

/* Check if the initialized line width is too wide.            */
/* The value of line_width is initialized to 3.00.  To avoid    */
/* requesting a line width that is wider than the workstation's */
/* widest line, call INQUIRE LINE FACILITIES.  If the line is   */
/* too wide, set it to the widest available width.             */

bufsize = sizeof(line_types);
line_types_st.integers = line_types;
line_fac.types = &line_types_st;
ginqlinefacil (&ws_type, bufsize, &fac_size, &line_fac, &status);
if (line_width * line_fac.nom_width > line_fac.max_width)
    line_width = line_fac.max_width / line_fac.nom_width;


/* Check if you are working with a color workstation.          */
/* If 4 colors are available, set each of the 4 color          */
/* representation indices for the specified workstation to the */
/* desired colors.  This type of coding is useful since        */
/* different workstations may have different default color     */
/* representations.                                            */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
ginqcolourfacil (&ws_type, bufsize, &fac_size, &col_fac, &status);
colour = (col_fac.coavail == GCOLOUR) && (col_fac.predefined >= 4);
if (colour)
   {
    gsetcolourrep (ws_id, BACK_IND, &black);
    gsetcolourrep (ws_id, RED_IND, &red);
    gsetcolourrep (ws_id, GREEN_IND, &green);
    gsetcolourrep (ws_id, BLUE_IND, &blue);
   }

/* Create a segment for the title.                            */
/* If the workstation is color, make the text red. Set the    */
/* character height, and text precision and font.  All values  */
/* are initialized above. Use TEXT 3 to generate the string.   */
/* The position of the text string, the text vectors, and the  */
/* text string are initialized above.                         */

gcreateseg (title);
   if (colour)
       gsettextcolourind (RED_IND);
   gsetcharheight (char_height);
   gsettextfontprec (&text_font_prec);
   gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
gcloseseg ( );

/* Create a segment for the stars and moon.                   */
/* If the workstation is color, make the stars blue. Make the  */
/* star shape a '+' and set the marker size scale factor. Use  */
/* POLYMARKER 3 to generate the star markers. The macro        */
/* NUM_STAR_PTS specifies the number of points. The list of    */
/* positions, stars_pts, is initialized above.                */
/* Use a generalized drawing primitive (GDP) for the moon.     */
/* The moon object uses GDP_FCCP (-333), which is a filled     */
/* circle described by the center and a point on the           */
/* circumference.  The description in the Device Specifics     */
/* Reference Manual indicates that the data record is null so  */
/* the numbers of integers, floats, and strings in the data    */
/* record are set to 0.  The number of points, 2, is defined   */
/* by the macro NUM_MOON_PTS, and the center and circumference */
/* point are passed in the list moon_pts.  GDP_FCCP is a macro */
/* defining the GDP number.  It is in the C binding include    */
/* file.                                                      */

gcreateseg (stars);
   if (colour)
       gsetmarkercolourind (BLUE_IND);
   gsetmarkertype (GMK_PLUS);
   gsetmarkersize (2.0);
   gpolymarker3 (NUM_STAR_PTS, stars_pts);
   gsetfillintstyle (GSOLID);
   moon_data_rec.gdp_datarec.number_integer = 0;
   moon_data_rec.gdp_datarec.number_float = 0;
   moon_data_rec.gdp_datarec.number_strings = 0;
   ggdp (NUM_MOON_PTS, moon_pts, GDP_FCCP, &moon_data_rec);
gcloseseg ( );
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
/* Create a segment for the tree.                          */
/* If the workstation is color, make the tree green. Set the   */
/* fill area interior style to solid so the tree is a solid    */
/* shape. Draw the fill area described by the number of points */
/* defined in the NUM_TREE_PTS macro and the list of points in */
/* tree_pts.  The points are initialized above.              */

gcreateseg (tree);
   if (colour)
       gsetfillcolourind (GREEN_IND);
   gsetfillintstyle (GSOLID);
   gfillarea3 (NUM_TREE_PTS, tree_pts);
gcloseseg ( );

/* Create a segment for the sidewalk.                        */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the sidewalk.  If it is monochrome, use FILL AREA 3.        */
/* The CELL ARRAY 3 function divides a parallelogram into      */
/* cells and displays each cell in a specified color.  The     */
/* call requires 3 points on the parallelogram (lower left     */
/* front corner, upper right front corner and upper right back */
/* corner), the number of rows and columns into which the      */
/* parallelogram will be divided, and a 2-dimensional array    */
/* containing the color index values of the cells.  The        */
/* dimensions of the color index array correspond to the       */
/* dimension (number of rows and columns) of the parallelogram */
/* for the C binding.                                        */

gcreateseg(side);
    if (colour)
       gcellarray3 (&side_rectangle_coordinates, &side_rectangle_dim,
                   side_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_SIDE_PTS, side_pts);
       }
gcloseseg ( );

/* Create a segment for the road.                            */
/* If the workstation is color, use CELL ARRAY3 to describe    */
/* the road.  If it is monochrome, use FILL AREA 3.  All       */
/* values are initialized above.                             */

gcreateseg(road);
    if (colour)
       gcellarray3 (&road_rectangle_coordinates, &road_rectangle_dim,
                   road_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_ROAD_PTS, road_pts);
       }
gcloseseg ( );
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    /* Create a segment for the land.                              */
    /* If the workstation is color, make the line outlining the    */
    /* land green. Set the line width, make the line dashed, and   */
    /* use the POLYLINE 3 function to draw the land horizon.       */

    gcreateseg (land);
       if (colour)
           gsetlinecolourind (GREEN_IND);
       gsetlinewidth (line_width);
       gsetlinetype (GLN_DASHED);
       gpolyline3 (NUM_LAND_PTS, land_pts);
    gcloseseg ( );

    /* Create a segment for the house and draw it.                 */
    /* If the workstation is color make the back, left side, and   */
    /* right side of the house red, and the left and right sides   */
    /* of the roof blue.  Use FILL AREA 3 to draw the back, left   */
    /* side, and right side of the house.  Use FILL AREA SET 3     */
    /* to draw the front of the house.  FILL AREA SET 3 draws      */
    /* the door and the front of the house.  The area for          */
    /* the door is hollow and is contained within the house front  */
    /* area.  The green from the back of the house shows through.  */

    gcreateseg (house);
       gsetfillintstyle (GSOLID);
       if (colour)
           gsetfillcolourind (GREEN_IND);
       gfillarea3 (NUM_BACK_PTS, back_pts);
       if (colour)
           gsetfillcolourind (RED_IND);
       gfillarea3 (NUM_LEFT_PTS, left_pts);
       gfillarea3 (NUM_RIGHT_PTS, right_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gfillarea3 (NUM_LROOF_PTS, lroof_pts);
       gfillarea3 (NUM_RROOF_PTS, rroof_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gsetedgeflag (GEDGE_ON);
       gsetedgewidthscfac (edge_width);
       gfillareaset3 (2, front_sizes, front_pts);
    gcloseseg ( );

    /* Ask the user for input.                                      */

    go_for_input (ws_id, ws_type, wiss,
                  title, stars, tree, side, road, house, land);

} /* end draw_picture */

/*******************************************************************/
/*  GO_FOR_INPUT ()                                                */
/*                                                                 */
/*  This subroutine coordinates the user input.                    */
/*                                                                 */
/*******************************************************************/

void go_for_input (ws_id, ws_type, wiss, title, stars, tree, side,
                   road, house, land)
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */
Gint      wiss,     /* WISS workstation identifier */
          title,    /* segment identifier for the title */
          stars,    /* segment identifier for the stars */
          tree,     /* segment identifier for the tree */
          side,     /* segment identifier for the sidewalk */
          road,     /* segment identifier for the road */
          house,    /* segment identifier for the house */
          land;     /* segment identifier for the land */
{
    void      create_help ( );

    Gint  help = 8;      /* segment identifier for the help screen */
    Gint  help_box = 9;  /* segment identifier for help box */

                         /* text position for HELP/EXIT label */
    static  Gpoint  text_pos = {0.65, 0.9};

    /* Select the default normalization transformation and create   */
    /* the help and help box segments. Help is the segment          */
    /* containing the help screen.  Help box is the segment         */
    /* representing the small rectangular area on the workstation   */
    /* which is labeled "HELP/EXIT".  The user needs to pick the    */
    /* help box segment to see the help screen.                     */

    gselntran (0);
    create_help (ws_id, wiss, help);
    gsetcharheight (0.02);
    gcreateseg (help_box);
    gtext (&text_pos, "HELP/EXIT");
    gcloseseg ( );

    /* Initialize all the logical input devices.                    */

    init_devices (ws_id, ws_type);

    /* Make the segments detectable, or "pickable".               */

    gsetdet (title, GDETECTABLE);
    gsetdet (stars, GDETECTABLE);
    gsetdet (tree, GDETECTABLE);
    gsetdet (side, GDETECTABLE);
    gsetdet (road, GDETECTABLE);
    gsetdet (house, GDETECTABLE);
    gsetdet (land, GDETECTABLE);
    gsetdet (help, GUNDETECTABLE);
    gsetdet (help_box, GDETECTABLE);
    gupdatews (ws_id, GPOSTPONE);

    /* Ask the user for input.                                      */

    get_values (ws_id, title, stars, tree, side, road, house, land, help,
                help_box, ws_type);

} /* End go_for_input */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
/*******************************************************************/
/*  CREATE_HELP ()                                                 */
/*                                                                 */
/*  This subroutine creates the help segment on the WISS           */
/*  workstation only and then assoicates the segment with the      */
/*  OUTIN workstation.                                             */
/*                                                                 */
/*******************************************************************/

void create_help (ws_id, wiss, help)

Gint   ws_id;  /* workstation identifier */
Gint   wiss;   /* WISS workstation identifier */
Gint   help;   /* segment identifier for the help screen */

{
    int    n;
    static Gpoint  text_pos = {0.1, 0.9};
    static Gchar   *text[27] =
                  {"DEVICES: One device chooses a picture item and one",
                   "    changes the current scaling value affecting the",
                   "    chosen picture item.",
                   "",
                   "CYCLING INPUT DEVICES:  If you have a numeric keypad,",
                   "    you can use the two keys, in the upper left corner,",
                   "    to turn devices on and off.  Otherwise, all devices",
                   "    move synchronously.  Pressing the upper leftmost",
                   "    key will let you cycle through the devices.  Only",
                   "    the prompt of a single device will be available to",
                   "    you at a given time.  Press the key repeatedly until",
                   "    you have the prompt you want.  Pressing the key to ",
                   "    the right of the leftmost key ends the cycling    ",
                   "    and activates the prompts of all the logical input",
                   "    devices present on the workstation.  Cycling through",
                   "    the input devices is useful if the device has a",
                   "    numeric keypad but does not have a mouse.",
                   "",
                   "MOVING THE PROMPTS:  Use whatever your device normally",
                   "    uses to move a cursor on the surface.  This can",
                   "    include arrow keys, a mouse, a puck, or a tablet.",
                   "",
                   "ENTERING VALUES:  To enter values, use whatever your",
                   "    device normally uses, such as the RETURN key,",
                   "    mouse button, or puck button. To pick an element,",
                   "    you must align the rectangular prompt with the",
                   "    element you choose."};

    /* Deactivate the OUTIN workstation and activate the WISS       */
    /* workstation. Deactivating the OUTIN workstation identified   */
    /* by ws_id prevents segments from being stored on the         */
    /* workstation, and prevents output from appearing on the      */
    /* workstation’s surface. Activating the WISS workstation      */
    /* causes the help segment to be stored on WISS.               */

    gdeactivatews (ws_id);
    gactivatews (wiss);

    /* Create the help segment and make it invisible. Making the    */
    /* segment invisible prevents the help screen from appearing    */
    /* on the workstation surface when the help segment is          */
    /* associated with the workstation identified by ws_id.         */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    gcreateseg (help);
    gsetcharheight (0.018);
    for (n = 0; n < 27; n++)
       {
        gtext (&text_pos, text[n]);
        text_pos.y -= 0.03;
       }
    gcloseseg ( );
    gsetvis (help, GINVISIBLE);

    /* Associate the help segment with the OUTIN workstation       */
    /* identified by ws_id, deactivate the WISS workstation, and    */
    /* activate the OUTIN workstation.  Once the WISS workstation   */
    /* is deactived, no more segments can be stored on WISS.  The   */
    /* OUTIN workstation is activated again to allow further       */
    /* generation of output on the workstation surface.            */

    gassocsegws (ws_id, help);
    gdeactivatews (wiss);
    gactivatews (ws_id);

} /* End create_help */

/*********************************************************************/
/*  INIT_DEVICES ()                                                  */
/*                                                                   */
/*  This subroutine initializes the logical input devices that are  */
/*  used in this program.                                            */
/*                                                                   */
/*********************************************************************/

init_devices (ws_id, ws_type)

Gint      ws_id;    /* workstation identifier */
Gwstype   ws_type;  /* workstation type */

{
    Gint          dev_num  = 1;          /* input device number */
    Gnumdev       num_input_dev;         /* available number of each input
                                            device */
    Gint          status;                /* returned status */
    Gchar         pick_buff[80];         /* pick buffer */
    Gpick         init_pick;             /* pick data */
    Gint          pick_pet;              /* pick prompt and echo type */
    Glimit        pick_echo_area;        /* pick echo area */
    Gpickrec      pick_data_rec;         /* pick data record */
    Gchoice       init_choice;           /* choice data */
    Gchar         choice_buff[80];       /* choice buffer */
    Gint          choice_pet = 1;        /* choice prompt and echo type */
    Glimit        choice_echo_area;      /* echo area */
    Gchoicerec    choice_rec;            /* choice data record */

                                         /* list of string lengths for the
                                            choice device */
    static Gint   choice_str_len[] = {8, 4};

                                         /* list of strings for the choice
                                            device */
    static Gchar  *choice_str[] = {"Continue", "Exit"};

                                         /* choice device title */
    static Gchar  choice_data[] = "Program Control";
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
Gchar         val_buff[80];          /* valuator buffer */
Gint          val_pet;               /* valuator prompt and echo type */
Glimit        val_echo_area;         /* valuator echo area */
Gvalrec       val_data_rec;          /* valuator data record */
Gfloat        val_init_val;          /* initial value for the valuator */
static Gchar  val_data[] = "Scale";  /* valuator title */
Gdspsize      dsp_siz;               /* display size */
int           k;                     /* loop control variable */

/* Inquire about the size of the display.                      */

ginqdisplaysize (&ws_type, &dsp_siz, &status);
if (status)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "Inquire display size.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }


/* Obtain the number of devices of each logical input device    */
/* class supported by the workstation.                          */

ginqnumavailinput (&ws_type, &num_input_dev, &status);
if (status)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "Inquire number of available input devices.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }

/* Verify the workstation supports at least one pick logical    */
/* input device.                                                */

if (num_input_dev.pick == 0)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "The workstation does not support pick input.\n");
    exit (0);
   }

/* Initialize a pick input device that is in request mode by    */
/* default. The pick status is set to OK which indicates that   */
/* cursor motion is not required before triggering the input    */
/* device.  This makes the program easier to use. The prompt    */
/* and echo type, with a default value of 1 (pick by            */
/* primitive), is set to value 3 (pick by segment) since all    */
/* objects in the scene are in segments, and it is easier to    */
/* pick by segment.  Setting the maximum X value of the echo    */
/* area to dsp_siz.device.y assumes that the Y dimension of     */
/* the display size is less than the X dimension.               */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
init_pick.status = GP_OK;
init_pick.seg = 1;
init_pick.pickid = 0;
pick_pet = 3;
pick_echo_area.xmin = 0.0;
pick_echo_area.xmax = dsp_siz.device.y;
pick_echo_area.ymin = 0.0;
pick_echo_area.ymax = dsp_siz.device.y;
pick_data_rec.pickpet3_datarec.aperture = dsp_siz.device.y / 20.0;
pick_data_rec.pickpet3_datarec.data = pick_buff;
ginitpick (ws_id, dev_num, &init_pick, pick_pet,
           &pick_echo_area, &pick_data_rec);

/* Verify the workstation supports at least one choice logical  */
/* input device.                                                 */

if (num_input_dev.choice == 0)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "The workstation does not support choice input.\n");
    fprintf (stderr, "Error status: %d\n", status);
    exit (0);
   }

/* Initialize a choice input device that is in request mode by  */
/* default. The minimum X value of the choice echo area is set  */
/* larger than the maximum X value of the pick echo area to     */
/* avoid overlapping the two echo areas. The choice device echo */
/* area is positioned to the right of the pick echo area       */
/* avoiding any overlap between the 2 echo areas.              */

init_choice.status = GC_OK;
init_choice.choice = 1;
choice_echo_area.xmin = dsp_siz.device.y * 1.05;
choice_echo_area.xmax = dsp_siz.device.x;
choice_echo_area.ymin = dsp_siz.device.y * 0.55;
choice_echo_area.ymax = dsp_siz.device.y;
choice_rec.choicepet1_datarec.number = 2;
choice_rec.choicepet1_datarec.lengths = choice_str_len;
choice_rec.choicepet1_datarec.strings = choice_str;
choice_rec.choicepet1_datarec.title_string = choice_data;
ginitchoice (ws_id, dev_num, &init_choice, choice_pet,
            &choice_echo_area, &choice_rec);

/* Verify the workstation supports at least one valuator        */
/* logical input device.                                         */

if (num_input_dev.valuator == 0)
   {
    gemergencyclosegks ( );
    fprintf (stderr, "The workstation does not support valuator input.\n");
    exit (0);
   }
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    /* Initialize a valuator input device that is in request mode   */
    /* by default. The minimum X value of the valuator echo area is */
    /* set larger than the maximum X value of the pick echo area,   */
    /* and the maximum Y value of the echo area is set smaller than */
    /* the minimum Y echo area of the choice device.  The valuator  */
    /* device echo area is positioned to the right of the pick echo */
    /* and below the choice echo area, thus preventing overlapping  */
    /* among the 3 device echo areas.                               */

    val_pet = 1;
    val_echo_area.xmin = dsp_siz.device.y * 1.05;
    val_echo_area.xmax = dsp_siz.device.x;
    val_echo_area.ymin = 0.0;
    val_echo_area.ymax = dsp_siz.device.y * 0.45;
    val_data_rec.valpet1_datarec.low  = 0.5;
    val_init_val = 1.0;
    val_data_rec.valpet1_datarec.high = 1.5;
    val_data_rec.valpet1_datarec.title_string = val_data;
    ginitval (ws_id, dev_num, val_init_val, val_pet,
              &val_echo_area, &val_data_rec);

}  /* end init_devices */

/*******************************************************************/
/*  GET_VALUES ()                                                  */
/*                                                                 */
/*  This subroutine reads the input from the devices and responds   */
/*  to the input information.  It scales the selected segment by    */
/*  the valuator value or displays the HELP/EXIT screen.            */
/*                                                                 */
/*******************************************************************/

get_values (ws_id, title, stars, tree, side, road, house, land, help,
            help_box, ws_type)

Gint     ws_id,     /* workstation identifier */
         title,     /* segment identifier for the title */
         stars,     /* segment identifier for the stars */
         tree,      /* segment identifier for the tree */
         side,      /* segment identifier for the sidewalk */
         road,      /* segment identifier for the road */
         house,     /* segment identifier for the house */
         land,      /* segment identifier for the land */
         help,      /* segment identifier for the help screen */
         help_box;  /* segment identifier for the help box */
Gwstype  ws_type;   /* workstation type */

{
    Gint  pick_dev_num = 1,    /* pick device number */
          choice_dev_num = 1,  /* choice device number */
          val_dev_num = 1;     /* valuator device number */

                                    /* identity transformation vectors
                                       used to create an identity
                                       segment transformation matrix */
    static Gpoint3  fixed_pt = {0.0, 0.0, 0.0},
                    shift_vec = {0.0, 0.0, 0.0};
    static Gangle3  angle_vec = {0.0, 0.0, 0.0};
    static Gscale3  scale_vec = {1.0, 1.0, 1.0};
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
Gfloat  identity_matrix[3][4],  /* identify matrix */
        title_matrix[3][4],     /* segment transformation matrix for
                                   for title */
        stars_matrix[3][4],     /* segment transformation matrix for
                                   for stars */
        house_matrix[3][4],     /* segment transformation matrix for
                                   for house */
        tree_matrix[3][4],      /* segment transformation matrix for
                                   for tree */
        side_matrix[3][4],      /* segment transformation matrix for
                                   for sidewalk */
        road_matrix[3][4],      /* segment transformation matrix for
                                   for road */
        land_matrix[3][4];      /* segment transformation matrix for
                                   for land */
Gint    finished_flag = 0;      /* loop controlling flag */
Gpick   pick_resp;              /* pick data */
Gevent  event;                  /* an event */
Gfloat  scale_value;            /* scale value */

/* Set the pick device input mode to EVENT and the valuator     */
/* device input mode to SAMPLE.  The choice device input mode   */
/* is set to REQUEST, by default.                               */

gsetpickmode (ws_id, pick_dev_num, GEVENT, GECHO);
gsetvalmode (ws_id, val_dev_num, GSAMPLE, GECHO);

/* Create an identity matrix and the initial segment            */
/* transformation matrices.                                     */

gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            title_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            stars_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            house_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            tree_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            side_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            road_matrix);
gevaltran3 (&fixed_pt, &shift_vec, &angle_vec, &scale_vec, GNDC,
            land_matrix);

/* Set the picked segment to 0, which indicates that no         */
/* segments have been picked yet, and enter the loop to collect */
/* input.  The finished_flag can only be set by the EXIT option */
/* in the help menu. AWAIT EVENT searches the input queue for   */
/* an event.  If the queue is empty, then a NONE value is       */
/* returned immediately for the input class since the timeout   */
/* is 0.0.  If the queue contains a report, which means a       */
/* segment was picked, the segment identifier is returned. If   */
/* the help box was chosen, the help screen is displayed.  If   */
/* there is no pick event in the queue, then the valuator       */
/* device is sampled and its setting is used to scale the       */
/* currently selected segment.  Each segment has a fixed point  */
/* which is at the center of the particular segment.            */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
pick_resp.seg = 0;
while (!finished_flag)
   {
    gawaitevent (0.0, &event);
    if (event.class == GPICK)
       {
        ggetpick (&pick_resp);
        if (pick_resp.seg == help_box)
           get_help (ws_id, ws_type, title, stars, tree, side, road,
                     house, land, help, help_box, &finished_flag);
       }
    else
       {
        gsampleval (ws_id, val_dev_num, &scale_value);
           scale_vec.x_scale = scale_value;
           scale_vec.y_scale = scale_value;
           if (pick_resp.seg == title)
              {
               fixed_pt.x = 0.5;
               fixed_pt.y = 0.1;
               fixed_pt.z = 0.8;
               gevaltran3 (&fixed_pt, &shift_vec,
                           &angle_vec, &scale_vec, GNDC, title_matrix);
               gsetsegtran3 (title, title_matrix);
              }
           else if (pick_resp.seg == stars)
              {
               fixed_pt.x = 0.5;
               fixed_pt.y = 0.8;
               fixed_pt.z = 0.0;
               gevaltran3 (&fixed_pt, &shift_vec,
                           &angle_vec, &scale_vec, GNDC, stars_matrix);
               gsetsegtran3 (stars, stars_matrix);
              }
           else if (pick_resp.seg == tree)
              {
               fixed_pt.x = 0.52;
               fixed_pt.y = 0.35;
               fixed_pt.z = 0.0;
               gevaltran3 (&fixed_pt, &shift_vec,
                           &angle_vec, &scale_vec, GNDC, tree_matrix);
               gsetsegtran3 (tree, tree_matrix);
              }
           else if (pick_resp.seg == side)
              {
               fixed_pt.x = 0.225;
               fixed_pt.y = 0.225;
               fixed_pt.z = 0.15;
               gevaltran3 (&fixed_pt, &shift_vec,
                           &angle_vec, &scale_vec, GNDC, side_matrix);
               gsetsegtran3 (side, side_matrix);
              }
           else if (pick_resp.seg == road)
              {
               fixed_pt.x = 0.5;
               fixed_pt.y = 0.075;
               fixed_pt.z = 0.9;
               gevaltran3 (&fixed_pt, &shift_vec,
                           &angle_vec, &scale_vec, GNDC, road_matrix);
               gsetsegtran3 (road, road_matrix);
              }
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
               else if (pick_resp.seg == land)
                  {
                   fixed_pt.x = 0.05;
                   fixed_pt.y = 0.37;
                   fixed_pt.z = 0.0;
                   gevaltran3 (&fixed_pt, &shift_vec,
                               &angle_vec, &scale_vec, GNDC, land_matrix);
                   gsetsegtran3 (land, land_matrix);
                  }
               else if (pick_resp.seg == house)
                  {
                   fixed_pt.x = 0.200;
                   fixed_pt.y = 0.525;
                   fixed_pt.z = 0.500;
                   gevaltran3 (&fixed_pt, &shift_vec,
                               &angle_vec, &scale_vec, GNDC, house_matrix);
                   gsetsegtran3 (house, house_matrix);
                  }
               gupdatews (ws_id, GPERFORM);
          }

      }  /* end while */

}  /* end get_values */

/*********************************************************************/
/*  GET_HELP ()                                                      */
/*                                                                   */
/*  This subroutine displays the help screen. When the help screen  */
/*  is visible to the user, the subroutine uses a choice device, in */
/*  request mode, to ask the user if he/she wishes to continue      */
/*  scaling or to exit from the program.                            */
/*                                                                   */
/*********************************************************************/

get_help (ws_id, ws_type, title, stars, tree, side, road, house,
          land, help, help_box, finished_flag)

Gint     ws_id;     /* workstation identifier */
Gwstype  ws_type;   /* workstation type */
Gint     title,     /* segment identifier for the title */
         stars,     /* segment identifier for the stars */
         tree,      /* segment identifier for the tree */
         side,      /* segment identifier for the sidewalk */
         road,      /* segment identifier for the road */
         house,     /* segment identifier for the house */
         land,      /* segment identifier for the land */
         help,      /* segment identifier for the help screen */
         help_box;  /* segment identifier for the help box */

Gint     *finished_flag;  /* flag indicating the user wants to
                             exit the program. */

{
    Gint     choice_dev_num = 1;  /* choice device number */
    Gint     pick_dev_num = 1;    /* pick device number */
    Gint     val_dev_num = 1;     /* valuator device number */
    Gqchoice choice_resp;         /* choice request response */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    /* Display the help screen by making all the segments, except   */
    /* help screen, invisible. Set the pick and valuator modes to   */
    /* REQUEST to remove their prompts from the screen.             */

    gsetvis (title, GINVISIBLE);
    gsetvis (stars, GINVISIBLE);
    gsetvis (tree, GINVISIBLE);
    gsetvis (side, GINVISIBLE);
    gsetvis (road, GINVISIBLE);
    gsetvis (house, GINVISIBLE);
    gsetvis (land, GINVISIBLE);
    gsetvis (help_box, GINVISIBLE);
    gsetvis (help, GVISIBLE);
    gsetpickmode (ws_id, pick_dev_num, GREQUEST, GECHO);
    gsetvalmode (ws_id, val_dev_num, GREQUEST, GECHO);
    gupdatews (ws_id, GPERFORM);

    /* Place the choice menu on the workstation surface and wait    */
    /* for the user to respond.                                     */

    greqchoice (ws_id, choice_dev_num, &choice_resp);
    if ((choice_resp.status == GC_OK) && (choice_resp.choice == 2))
       *finished_flag = 1;
    else
       *finished_flag = 0;

    /* Make the help screen invisible and all other segments        */
    /* visible.  Set the pick mode back to EVENT and the valuator   */
    /* mode back to SAMPLE.                                         */

    gsetvis (title, GVISIBLE);
    gsetvis (stars, GVISIBLE);
    gsetvis (tree, GVISIBLE);
    gsetvis (side, GVISIBLE);
    gsetvis (road, GVISIBLE);
    gsetvis (house, GVISIBLE);
    gsetvis (land, GVISIBLE);
    gsetvis (help_box, GVISIBLE);
    gsetvis (help, GINVISIBLE);
    gupdatews (ws_id, GPERFORM);
    gsetpickmode (ws_id, pick_dev_num, GEVENT, GECHO);
    gsetvalmode (ws_id, val_dev_num, GSAMPLE, GECHO);

}

/*******************************************************************/
/*  CLEAN_UP ()                                                    */
/*                                                                 */
/*  This subroutine cleans up the DEC GKS and workstation          */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void clean_up (ws_id, wiss)

Gint ws_id;  /* workstation identifier */
Gint wiss;   /* WISS workstation identifier */

{
    /* Deactivate all active workstations and close all open        */
    /* workstations before closing GKS.                             */
```

(continued on next page)

Example 5–2 (Cont.) Asynchronous Input Example

```
    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosews (wiss);
    gclosegks ( );

} /* end clean_up */
```

6 Writing Device-Independent and System-Independent Programs

This chapter provides an introduction to writing device-independent programs. Establishing device independence includes but is not limited to:

- Setting the device connection and type without hard coding the information into the program
- Examining or setting the deferral mode and implicit regneration modes, and examining the dynamic modification of the attributes, so that setting updates of the display surface is controlled
- Setting workstation-dependent output attributes within the bounds of a particular workstation
- Examining the color capabilities of a particular device and adjusting the code accordingly

To access information stored in the DEC GKS data structures, use the DEC GKS inquiry functions. These structures contain information that affects the operation of DEC GKS. You can use this information to make valuable programming decisions.

The inquiry functions access the description tables and state lists described in Table 6–1. It is not necessary to know which list is being accessed. You just need to know the name of the function that returns the needed information. The inquiry functions are fully described in each of the binding manuals.

Table 6–1 DEC GKS Data Structures

Table/List	Description
GKS description table	This table contains information about the constants for the DEC GKS implementation you are using, such as the level of GKS (with DEC GKS, level 2c), the number of available workstation types, the list of workstation types, the maximum allowable open workstations, and so on. If you are transporting your programs from one implementation of GKS to another, you may need to inquire about the implementation of GKS on a given system, so your program does not call unsupported functions.

(continued on next page)

Table 6–1 (Cont.) DEC GKS Data Structures

Table/List	Description
Workstation description table	This type of table contains information about the constants for one particular workstation, such as the workstation type, the workstation category, the device-specific maximum coordinate values, the different bundled output attribute values, and so on. Each graphics handler contains a workstation description table describing that particular device. If your DEC GKS application uses more than one workstation at a time, or if you are unsure of the capabilities of your workstation, you may need to inquire about the values contained in the workstation description table.
GKS operating state list	This list contains one of the following five operating states: GKCL (GKS closed), GKOP (GKS open), WSOP (workstation open), WSAC (workstation active), SGOP (segment open). If you are unsure of the control functions used to regulate the activity of your workstation, you may need to inquire about the present condition of the DEC GKS operating states.
GKS state list	This list contains entries that specify the current DEC GKS values, such as the set of open workstations (if any), the current normalization transformation number, the current character height, and so on. If you need to check the alterable DEC GKS values, you may need to inquire about the values contained in the DEC GKS state list.
Workstation state list	For each workstation you open, DEC GKS creates a workstation state list. This list contains entries that specify whether output is deferred, whether the surface has to be redrawn to fulfill an output request, whether the workstation surface is “empty” as defined by GKS, whether the picture on the surface represents all of the requests for output made thus far by the application program, and so on. If you need information concerning the current state of a particular workstation, you may need to inquire about the values contained in the workstation state list.

(continued on next page)

Table 6–1 (Cont.) DEC GKS Data Structures

Table/List	Description
Segment state list	The segment state list contains entries that specify the segment name, the set of associated workstations, the detectability of the segment, and so on. If you need information concerning a particular segment, you may need to inquire about the values contained in the segment state list.
Error state list	This list contains information about the current error state and error file identifier, as well as identification of any logical input device that caused an input queue overflow. If you need information about run-time errors, you may need to inquire about the values in the error state list.

6.1 Writing Device-Independent Code

Considering the effort of altering your program every time you run it using a different physical device, DEC GKS provides environment options, constants, and inquiry functions so you can write device-independent programs. The inquiry functions also let you ask for values associated with both the default and current state of DEC GKS or of a particular supported device.

The following sections describe application program functionality for which using device-independent code is beneficial.

6.1.1 Specifying the Connection Identifier and Device Type

In the call to OPEN WORKSTATION, you can specify 0 for the connection identifier and the workstation type, or you can specify the constants in the binding header file that represent the default workstation identifier and the workstation type. DEC GKS reads the VMS logical names, GKS$CONID and GKS$WSTYPE, or the ULTRIX environment variables, GKSconid and GKSwstype, to set the connection identifier and the workstation type, respectively. The environment options can be changed before each program run. See Chapter 2 and *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for more information.

6.1.2 Checking and Specifying the Workstation Category

Check the category of the workstation type. For example, if the application requires output, then the workstation type must be of category OUTPUT or OUTIN. Use the INQUIRE WORKSTATION CATEGORY function to determine the category of your workstation.

6.1.3 Checking the GKS Level

Use the INQUIRE LEVEL OF GKS function to check the GKS level, which determines the functional capabilities of the GKS implementation. This is important for applications that may need to be transported to other systems. Figure 1–3 illustrates the GKS levels.

6.1.4 Checking and Setting the Deferral State

Use the INQUIRE DEFAULT DEFERRAL STATE VALUES function to check the deferral mode and the implicit regeneration mode. The deferral mode controls the time at which the output functions have their visible effects. The implicit regeneration mode controls the time at which picture changes can occur. Use SET DEFERRAL STATE to change these values. You can also use INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES (3) to check the *dynamic modification accepted* value of the different attributes. These values cannot be changed.

The workstations have a default setting for the output deferral mode and implicit regeneration mode. For device independence, check the current or default values and change them as your program requires.

By setting the deferral mode, you can **buffer** the generation of output images, if the given workstation supports such buffering, before transmission to the workstation surface. In this manner, you improve overall rate of data transmission. In the application program, you can periodically release buffered output so the display surface reflects the picture defined by the application up to that point in execution. (See Section 2.2 for more information on deferral modes.)

Note

When debugging your DEC GKS programs, you may wish to see generated output as you debug. To do this, set the deferral mode to ASAP. After you debug your program, you can set the deferral mode to any desired mode. Workstations supporting output data deferral or buffering will perform better when the deferral mode is set to ASTI.

For workstations that support output data deferral or buffering (the DECwindows workstation, for example), DEC GKS implements the deferral modes defined in Section 2.2.

6.1.5 Setting Workstation-Dependent Output Attributes

Output attributes are divided into two categories: geometric attributes, which are device independent, and nongeometric attributes, which are device dependent.

When setting device-dependent attributes in a device-independent program, either query the device using the appropriate INQUIRE ... FACILITIES function or check the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for acceptable values of the attributes. Minimally, DEC GKS supports all values defined by the GKS–3D International Standard (ISO-8805).

Use the INQUIRE ... FACILITIES functions to obtain information concerning types, sizes, and limits of polylines, polymarkers, fill areas, edges, patterns, text, colors, views, and HLHSR. Using this function prevents you from using values that are not supported by the workstation. Geometric and nongeometric attributes are described in Chapter 4.

6.1.6 Checking the Color Capabilities of the Device

In GKS, color is specified in a number of situations. It may be an aspect of a primitive, where it is specified in the bundle for that primitive or by the individual color attribute for that primitive. Color may be part of a pattern for FILL AREA, in which case an array of color is specified. Color may also be part of a primitive itself—CELL ARRAY for example—in which case an array of color is also specified. In each case, the color is specified as an index into a color table on the workstation. On each workstation, there is one color table into which all the color indexes point.

A device may be monochrome or it may support an array of colors. Use the INQUIRE COLOR FACILITIES function to determine the color capabilities of the workstation before using code that requires the output of a nonmonochromatic color scheme. This function also returns the list of available color models. The color model of a workstation affects the interpretation of the color components used in setting the representation of a color. See Appendix A for more information on the DEC GKS color models.

6.1.7 Determining the Size of the Display

Use the INQUIRE DISPLAY SPACE SIZE function to obtain the size of the display. This information can be used to establish a maximum echo area for input or to create windows and viewports that are proportional to the workstation display surface.

6.1.8 Determining the Number of Logical Input Devices

Use the INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES to obtain the number of logical input devices in each class for a given workstation type. This tells you which logical input devices are supported by the workstation.

6.1.9 Checking the Available Prompt and Echo Types

Use the INQUIRE DEFAULT ... DATA function to obtain a list of the available prompt and echo types, and the maximum number of echo areas or volumes for a particular workstation.

6.2 Writing System-Independent Programs

The ISO 8805 GKS–3D standard and the ISO 7942 GKS standard specify the functionality of GKS–3D and GKS in a manner that is independent of the formal grammar of any programming language. Standard language bindings then map the GKS–3D and GKS standard functions, and their arguments, to the grammar of a programming language. In addition to the standard bindings, Digital has created two proprietary language bindings, the GKS$ and GKS3D$ language bindings, that provide an application program interface for programs written in any language shown in Table 6–2.

The GKS application developer must choose which programming language and which language binding to use to develop the GKS application. Application portability may be a major consideration when making this choice. Table 6–2 summarizes GKS portability with regard to language bindings and programming language. The only standard language bindings are the GKS FORTRAN language binding, ISO 8651/1, and its superset, the GKS–3D FORTRAN language binding, ISO 8806/1. The DEC GKS FORTRAN binding conforms to these standard language bindings. The DEC GKS C language binding conforms to a draft version of the standard language binding for the C programming language, which was not published at the time of this writing. The options presently available are summarized in Table 6–2.

Table 6–2 Portability of Language Bindings

Binding	Language	DEC GKS VMS	DEC GKS ULTRIX	Other GKS Systems
FORTRAN	FORTRAN	Yes	Yes	Yes
C	Any C	Yes	Yes	Possible
GKS3D$ and GKS$	VAX FORTRAN™	Yes	No	No
	Any C	Yes	Yes	No
	VAX Ada™	Yes	No	No
	VAX BASIC™	Yes	No	No
	VAX Pascal™	Yes	No	No
	VAX PL/1	Yes	No	No
	MACRO–32	Yes	No	No

6.3 Presentation of Program Examples in This Manual

Example 6–1 is located on the VMS kit in the following directory:

```
sys$common:[syshlp.examples.gks]
```

Example 6–1 is located on the ULTRIX kit in the following directory:

```
/usr/lib/GKS/examples
```

Example 6–1 illustrates a device-independent program. If you run this program on two different output devices, a VWS workstation and a VT240™ terminal for example, you will see that the picture occupies a larger or smaller portion of the workstation surface depending on the width-to-height ratio of the device surface. By default, DEC GKS generates output on the largest square that the workstation can produce, with the lower left corner of the default WC space corresponding with the lower left corner of the workstation surface.

Example 6–1 Device-Independent Program Example

```
/* Header files */

#include <stdio.h>              /* standard C library I/O header file */
#include <gks.h>                /* GKS C binding header file */
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
                                 /* Constant definitions */
#define     NUM_SIDE_COLORS  6
#define     NUM_ROAD_COLORS  2
#define     NUM_STAR_PTS     6
#define     NUM_MOON_PTS     2
#define     NUM_TREE_PTS    29
#define     NUM_FRONT_PTS   10
#define     NUM_LROOF_PTS    4
#define     NUM_RROOF_PTS    4
#define     NUM_BACK_PTS     6
#define     NUM_RIGHT_PTS    5
#define     NUM_LEFT_PTS     5
#define     NUM_LAND_PTS    15
#define     NUM_SIDE_PTS     4
#define     NUM_ROAD_PTS     4
#define     BACK_IND         0
#define     RED_IND          1
#define     GREEN_IND        2
#define     BLUE_IND         3

/*********************************************************************/
/*                                                                   */
/*  MAIN ()                                                          */
/*                                                                   */
/*********************************************************************/

main ()

{
    void    set_up( );
    void    draw_picture( );
    void    clean_up( );
    Gint    ws_id   = 1;       /* workstation identifier */
    Gint    title   = 1,       /* segment identifier for the title */
            stars   = 2,       /* segment identifier for the stars and moon */
            tree    = 3,       /* segment identifier for the tree */
            side    = 4,       /* segment identifier for the sidewalk */
            road    = 5,       /* segment identifier for the road */
            land    = 6,       /* segment identifier for the land */
            house   = 7;       /* segment identifier for the house */

    Gwstype ws_type = GWS_DEF;  /* workstation type */
    Gconn   conid   = GWC_DEF;  /* connection identifier */

    set_up (ws_id, &ws_type, conid);

    draw_picture (ws_type, ws_id, title, stars, tree, side, road,
                  house, land);

    clean_up (ws_id);

} /* end main */
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
/*******************************************************************/
/*  SET_UP ()                                                      */
/*                                                                 */
/*  This subroutine sets up the DEC GKS and workstation            */
/*  environments.                                                  */
/*                                                                 */
/*******************************************************************/

void set_up (ws_id, ws_type, conid)

Gint       ws_id;    /* workstation identifier */
Gwstype   *ws_type;  /* workstation type */
Gconn      conid;    /* connection identifier */

{
    Glong      memory = GDEFAULT_MEM_SIZE;  /* memory size */
    Gint       buf_size;                    /* buffer size */
    Gwscat     category;                    /* workstation category */
    Gchar      conid_buff[80];              /* buffer for workstation
                                               connection identifier */
    Gwsct      ct;                          /* workstation connection
                                               identifier and type */
    Gdefmode   def_mode;                    /* deferral mode */
    Girgmode   irg_mode;                    /* implicit regeneration mode */
    Gint       ret_size;                    /* returned size */
    Gint       status;                      /* returned status */
    Gfile     *error_file;                  /* error file */

    /* Initialize GKS.                                             */
    /* A pointer to a specified error file is passed in the call to */
    /* OPEN GKS.  DEC GKS writes all errors to this file.          */

    error_file = fopen ("example_6_1.error", "w");
    status = gopengks (error_file, memory);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error opening GKS.\n");
        exit (0);
       }

    /* Open and activate a workstation.                              */
    /* Both the workstation type and the connection have been        */
    /* initialized to 0 so DEC GKS reads the VMS logical names,      */
    /* GKS$WSTYPE and GKS$CONID, or the ULTRIX environment variables, */
    /* GKSwstype and GKSconid, for the values.                       */

    status = gopenws (ws_id, &conid, ws_type);
    if (status != NO_ERROR)
       {
        gemergencyclosegks ( );
        fprintf (error_file, "Error opening workstation.\n");
        exit (0);
       }
    gactivatews (ws_id);
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
    /* Defer output as long as possible and suppress implicit       */
    /* regeneration.                                                */
    /* The deferral mode specifies when calls to the output         */
    /* functions have their effect.  The implicit regeneration mode */
    /* specifies when changes to the existing picture are seen.     */
    /* This application chooses to control the time when attribute  */
    /* changes are seen on the display so it suppresses the         */
    /* implicit regeneration mode in case it is IMMEDIATE for the   */
    /* workstation.                                                 */

    def_mode = GASTI;
    irg_mode = GSUPPRESSED;
    gsetdeferst (ws_id, def_mode, irg_mode);

    /* Determine the workstation connection and type.               */
    /* The application specifies the default workstation and type,  */
    /* which are defined by environment options.  This inquiry      */
    /* returns the connection and type to the application program   */
    /* so it has the information.                                   */

    buf_size = sizeof(conid_buff);
    ct.conn = conid_buff;
    ct.type = ws_type;
    ginqwsconntype (ws_id, buf_size, &ret_size, &ct, &status);
    if (status)
       {
 gemergencyclosegks ( );
 fprintf (error_file,
  "Error: Inquire workstation connection and type.\n");
 fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

    /* Inquire about the category of the workstation. The inquiry   */
    /* function verifies the workstation has a workstation type     */
    /* that indicates it can perform output.                        */

    ginqwscategory (ws_type, &category, &status);
    if ( (status) || ((category != GOUTIN) && (category != GOUTPUT)) )
       {
 gemergencyclosegks ( );
 fprintf (error_file, "The workstation category is invalid.\n");
 fprintf (error_file, "Error status: %d\n", status);
        exit (0);
       }

} /* end set_up */

/*******************************************************************/
/*  DRAW_PICTURE ()                                                */
/*                                                                 */
/*  This subroutine draws the picture and places each primitive in  */
/*  a segment.  All objects are defined within the default world    */
/*  coordinate range ([0, 1] x [0, 1] x [0, 1]).   If the          */
/*  workstation has sufficient color capabilities, the objects are  */
/*  drawn in four colors which are defined with the SET COLOUR     */
/*  REPRESENTATION function.  This code assumes an RGB color model. */
/*  Device-independent code should inquire about the available     */
/*  color models and set the color model explicitly.  If the       */
/*  workstation is monochrome, the objects are drawn in the        */
/*  default foreground color.                                      */
/*                                                                 */
/*******************************************************************/
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
void draw_picture (ws_type, ws_id, title, stars, tree, side, road, house, land)
Gwstype  ws_type;  /* workstation type */
Gint     ws_id,    /* workstation identifier */
         title,    /* segment identifier for the title */
         stars,    /* segment identifier for the stars */
         tree,     /* segment identifier for the tree */
         side,     /* segment identifier for the sidewalk */
         road,     /* segment identifier for the road */
         house,    /* segment identifier for the house */
         land;     /* segment identifier for the land */

{
                                         /* array of the sidewalk colors
                                            for the cell array call */
    static Gint     side_colors[NUM_SIDE_COLORS] = {1, 2, 3, 1, 2, 3},
                                         /* array of the road colors
                                            for the cell array call */
                    road_colors[NUM_ROAD_COLORS] = {1, 3};

           Gint     status,              /* returned status */
                    bufsize,             /* buffer size */
                    colour,              /* flag to indicate if the workstation
                                            is color or monochrome */
                    fac_size;            /* number of facilities */

           Gfloat   char_height = 0.04,  /* character height */
                    line_width  = 3.0,   /* line width */
                    edge_width  = 2.0;   /* edge width */
           Glnfac   line_fac;            /* line facility */
           Gintlist line_types_st;       /* integer list stucture of line
                                            types */
           Gint     line_types[20];      /* list of line types */
           Gcofac   col_fac;             /* color facility */
    static Gchar    *title_str = "Starry Night";  /* title string */

                                    /* 3 sidewalk "corner" points for the
                                       cellarray call */
    static Grect3   side_rectangle_coordinates = { {0.2,  0.0, 0.8},
                                                   {0.25, 0.0, 0.8},
                                                   {0.2,  0.3, 0.7} };

                                    /* 3 road "corner" points for the
                                       cellarray call */
    static Grect3   road_rectangle_coordinates = { {0.0, 0.0, 1.0},
                                                   {0.0, 0.0, 0.8},
                                                   {1.0, 0.0, 1.0} };
                                    /* dimensions of the array with the
                                       with the sidewalk colors */
    static Gidim    side_rectangle_dim = {1, 6},
                                    /* dimensions of the array with the
                                       with the road colors */
                    road_rectangle_dim = {2, 1};

                                    /* color definitions with RGB values */
    static Gcobundl black = {0.0, 0.0, 0.0};
    static Gcobundl red   = {1.0, 0.0, 0.0};
    static Gcobundl green = {0.0, 1.0, 0.0};
    static Gcobundl blue  = {0.0, 0.0, 1.0};

    Ggdprec  moon_data_rec;  /* GDP data record for the moon */
                                    /* moon point values for the GDP */
    static Gpoint   moon_pts[NUM_MOON_PTS] = { {0.9,0.9}, {0.9,0.84} };
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
                                /* number of points in each of the fill
                                   areas comprising the fill area set */
static Gint     front_sizes[2] = {6, 4};

                                /* text font and precision */
static Gtxfp    text_font_prec = {1, GP_STROKE};
                                /* text position */
static Gpoint3  title_start = {0.3, 0.1, 0.8},
                                /* text direction vectors */
                text_vec1 = {1.0, 0.0, 0.0},
                text_vec2 = {0.0, 1.0,-1.0},

                                /* star coordinates */
                stars_pts[NUM_STAR_PTS] =
                  { {0.05,0.70,0.0}, {0.06,0.86,0.0}, {0.36,0.81,0.0},
                    {0.66,0.86,0.0}, {0.835,0.7,0.0}, {0.92,0.82,0.0} },

                                /* land coordinates */
                land_pts[NUM_LAND_PTS] =
                  { {0.0,0.35,0.0}, {0.04,0.375,0.0}, {0.055,0.376,0.0},
                    {0.08,0.36,0.0},  {0.1,0.365,0.0},  {0.3,0.366,0.0},
                    {0.375,0.38,0.0}, {0.44,0.385,0.0}, {0.49,0.375,0.0},
                    {0.56,0.36,0.0},  {0.68,0.38,0.0},  {0.8,0.35,0.0},
                    {0.9,0.359,0.0},  {0.95,0.375,0.0}, {1.0,0.385,0.0} },

                                /* tree coordinates */
                tree_pts[NUM_TREE_PTS] =
                  { {0.425,0.28,0.0}, {0.5,0.3,0.0},     {0.52,0.26,0.0},
                    {0.54,0.3,0.0},   {0.6,0.28,0.0},    {0.575,0.33,0.0},
                    {0.56,0.42,0.0},  {0.559,0.49,0.0},  {0.64,0.53,0.0},
                    {0.69,0.57,0.0},  {0.689,0.61,0.0},  {0.66,0.64,0.0},
                    {0.63,0.66,0.0},  {0.645,0.71,0.0},  {0.59,0.76,0.0},
                    {0.53,0.78,0.0},  {0.48,0.75,0.0},   {0.45,0.71,0.0},
                    {0.42,0.65,0.0},  {0.375,0.645,0.0}, {0.35,0.6,0.0},
                    {0.375,0.55,0.0}, {0.44,0.54,0.0},   {0.45,0.5,0.0},
                    {0.515,0.5,0.0},  {0.51,0.425,0.0},  {0.495,0.38,0.0},
                    {0.475,0.33,0.0}, {0.425,0.28,0.0} },

                                /* house front coordinates */
                front_pts[NUM_FRONT_PTS] =
                  { {0.1,0.3,0.7},  {0.3,0.3,0.7},  {0.3,0.6,0.7},
                    {0.2,0.75,0.7}, {0.1,0.6,0.7},  {0.1,0.3,0.7},
                    {0.2,0.3,0.7},  {0.25,0.3,0.7}, {0.25,0.4,0.7},
                    {0.2,0.4,0.7} },

                                /* house back coordinates */
                back_pts[NUM_BACK_PTS] =
                  { {0.1,0.3,0.3},  {0.1,0.6,0.3}, {0.2,0.75,0.3},
                    {0.3,0.6,0.3},  {0.3,0.3,0.3}, {0.1,0.3,0.3} },

                                /* house right side coordinates */
                right_pts[NUM_RIGHT_PTS] =
                  { {0.3,0.3,0.7},  {0.3,0.3,0.3}, {0.3,0.6,0.3},
                    {0.3,0.6,0.7},  {0.3,0.3,0.7} },

                                /* house left side coordinates */
                left_pts[NUM_LEFT_PTS] =
                  { {0.1,0.3,0.3},  {0.1,0.3,0.7}, {0.1,0.6,0.7},
                    {0.1,0.6,0.3},  {0.1,0.3,0.3} },

                                /* roof left side coordinates */
                lroof_pts[NUM_LROOF_PTS] =
                  { {0.1,0.6,0.7},  {0.2,0.75,0.7},
                    {0.2,0.75,0.3}, {0.1,0.6,0.3} },
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
                                /* roof right side coordinates */
                rroof_pts[NUM_RROOF_PTS] =
                  { {0.3,0.6,0.7},  {0.3,0.6,0.3},
                    {0.2,0.75,0.3}, {0.2,0.75,0.7} },

                                /* sidewalk coordinates */
                side_pts[NUM_SIDE_PTS] =
                  { {0.2,0.0,0.8},   {0.25,0.0,0.8},
                    {0.25,0.3,0.7},  {0.2,0.3,0.7} },

                                /* road coordinates */
                road_pts[NUM_ROAD_PTS] =
                  { {0.0,0.0,1.0},    {1.0,0.0,1.0},
                    {1.0,0.0,0.8},    {0.0,0.0,0.8} };

/* Check if the initialized line width is too wide.            */
/* The value of line_width is initialized to 3.00.  To avoid    */
/* requesting a line width that is wider than the workstation's */
/* widest line, call INQUIRE LINE FACILITIES.  If the line is   */
/* too wide, set it to the widest available width.              */

bufsize = sizeof(line_types);
line_types_st.integers = line_types;
line_fac.types = &line_types_st;
ginqlinefacil (&ws_type, bufsize, &fac_size, &line_fac, &status);
if (line_width * line_fac.nom_width > line_fac.max_width)
    line_width = line_fac.max_width / line_fac.nom_width;

/* Check if you are working with a color workstation.          */
/* If 4 colors are available, set each of the 4 color          */
/* representation indices for the specified workstation to the */
/* desired colors.  This type of coding is useful since        */
/* different workstations may have different default color     */
/* representations.                                            */

ginqcolourfacil (&ws_type, bufsize, &fac_size, &col_fac, &status);
colour = (col_fac.coavail == GCOLOUR) && (col_fac.predefined >= 4);
if (colour)
   {
    gsetcolourrep (ws_id, BACK_IND, &black);
    gsetcolourrep (ws_id, RED_IND, &red);
    gsetcolourrep (ws_id, GREEN_IND, &green);
    gsetcolourrep (ws_id, BLUE_IND, &blue);
   }

/* Create a segment for the title.                             */
/* If the workstation is color, make the text red. Set the     */
/* character height, and text precision and font.  All values  */
/* are initialized above. Use TEXT 3 to generate the string.   */
/* The position of the text string, the text vectors, and the  */
/* text string are initialized above.                          */

gcreateseg (title);
   if (colour)
       gsettextcolourind (RED_IND);
   gsetcharheight (char_height);
   gsettextfontprec (&text_font_prec);
   gtext3 (&title_start, &text_vec1, &text_vec2, title_str);
gcloseseg ( );
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
/* Create a segment for the stars and moon.                 */
/* If the workstation is color, make the stars blue. Make the  */
/* star shape a '+' and set the marker size scale factor. Use  */
/* POLYMARKER 3 to generate the star markers. The macro        */
/* NUM_STAR_PTS specifies the number of points. The list of    */
/* positions, stars_pts, is initialized above.                 */
/* Use a generalized drawing primitive (GDP) for the moon.     */
/* The moon object uses GDP_FCCP (-333), which is a filled     */
/* circle described by the center and a point on the           */
/* circumference.  The description in the Device Specifics     */
/* Reference Manual indicates that the data record is null so  */
/* the numbers of integers, floats, and strings in the the     */
/* data record are set to 0.  The number of points, 2, is      */
/* defined by the macro NUM_MOON_PTS, and the center and       */
/* circumference point are passed in the list moon_pts.        */
/* GDP_FCCP is a macro defining the GDP number.  It is in the  */
/* C binding include file.                                     */

gcreateseg (stars);
   if (colour)
       gsetmarkercolourind (BLUE_IND);
   gsetmarkertype (GMK_PLUS);
   gsetmarkersize (2.0);
   gpolymarker3 (NUM_STAR_PTS, stars_pts);
   gsetfillintstyle (GSOLID);
   moon_data_rec.gdp_datarec.number_integer = 0;
   moon_data_rec.gdp_datarec.number_float = 0;
   moon_data_rec.gdp_datarec.number_strings = 0;
   ggdp (NUM_MOON_PTS, moon_pts, GDP_FCCP, &moon_data_rec);
gcloseseg ( );

/* Create a segment for the tree.                              */
/* If the workstation is color, make the tree green. Set the   */
/* fill area interior style to solid so the tree is a solid    */
/* shape. Draw the fill area described by the number of points */
/* defined in the NUM_TREE_PTS macro and the list of points in */
/* tree_pts.  The points are initialized above.                */

gcreateseg (tree);
    if (colour)
       gsetfillcolourind (GREEN_IND);
   gsetfillintstyle (GSOLID);
   gfillarea3 (NUM_TREE_PTS, tree_pts);
gcloseseg ( );

/* Create a segment for the sidewalk.                          */
/* If the workstation is color, use CELL ARRAY 3 to describe   */
/* the sidewalk.  If it is monochrome, use FILL AREA 3.        */
/* The CELL ARRAY 3 function divides a parallelogram into      */
/* cells and displays each cell in a specified color.  The     */
/* call requires 3 points on the parallelogram (lower left     */
/* front corner, upper right front corner and upper right back */
/* corner), the number of rows and columns into which the      */
/* parallelogram will be divided, and a two -dimensional array */
/* containing the color index values of the cells.  The        */
/* dimensions of the color index array correspond to the       */
/* dimension (number of rows and columns) of the parallelogram */
/* for the C binding.                                          */
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
gcreateseg(side);
    if (colour)
       gcellarray3 (&side_rectangle_coordinates, &side_rectangle_dim,
                   side_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_SIDE_PTS, side_pts);
       }
gcloseseg ( );

/* Create a segment for the road.                              */
/* If the workstation is color, use CELL ARRAY 3 to describe   */
/* the road.  If it is monochrome, use FILL AREA 3.  All       */
/* values are initialized above.                               */

gcreateseg(road);
    if (colour)
       gcellarray3 (&road_rectangle_coordinates, &road_rectangle_dim,
                   road_colors);
    else
       {
        gsetfillintstyle (GSOLID);
        gfillarea3 (NUM_ROAD_PTS, road_pts);
       }
gcloseseg ( );

/* Create a segment for the land.                              */
/* If the workstation is color, make the line outlining the    */
/* land green.  Set the line width and make the line dashed.   */
/* Set the line width, make the line dashed, and use the       */
/* POLYLINE 3 function to draw the land horizon.               */

gcreateseg (land);
   if (colour)
       gsetlinecolourind (GREEN_IND);
   gsetlinewidth (line_width);
   gsetlinetype (GLN_DASHED);
   gpolyline3 (NUM_LAND_PTS, land_pts);
gcloseseg ( );

/* Create a segment for the house and draw it.                 */
/* If the workstation is color make the back, left and  right  */
/* sides of the house red, and the left and right sides        */
/* of the roof blue.  Use FILL AREA 3 to draw the back, left   */
/* side, and right side of the house.  Use FILL AREA SET 3     */
/* to draw the front of the house.  FILL AREA SET 3 draws      */
/* the door and the front of the house.  The area for          */
/* the door is hollow and is contained within the house front  */
/* area.  The green from the back of the house shows through.  */
```

(continued on next page)

Example 6–1 (Cont.) Device-Independent Program Example

```
    gcreateseg (house);
       gsetfillintstyle (GSOLID);
       if (colour)
           gsetfillcolourind (GREEN_IND);
       gfillarea3 (NUM_BACK_PTS, back_pts);
       if (colour)
           gsetfillcolourind (RED_IND);
       gfillarea3 (NUM_LEFT_PTS, left_pts);
       gfillarea3 (NUM_RIGHT_PTS, right_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gfillarea3 (NUM_LROOF_PTS, lroof_pts);
       gfillarea3 (NUM_RROOF_PTS, rroof_pts);
       if (colour)
           gsetfillcolourind (BLUE_IND);
       gsetedgeflag (GEDGE_ON);
       gsetedgewidthscfac (edge_width);
       gfillareaset3 (2, front_sizes, front_pts);
    gcloseseg ( );

} /* end draw_picture */

/*******************************************************************/
/*  CLEAN_UP ()                                                    */
/*                                                                 */
/*  This subroutine calls UPDATE WORKSTATION to display the        */
/*  completed picture since the implicit regeneration mode was set  */
/*  to SUPPRESSED.  When the operator indicates he/she has finished */
/*  viewing the picture, the subroutine cleans up the DEC GKS and   */
/*  workstation environments.                                      */
/*                                                                 */
/*******************************************************************/

void clean_up (ws_id)

Gint ws_id;  /* workstation identifier */

{
                                    /* event and timeout are used with the
                                       AWAIT EVENT call to pause program
                                       execution */
    Gevent  event;                  /* an input event */
    Gfloat  timeout = 8.00;         /* pause time in seconds */

    /* Update the workstation surface.                             */
    /* The PERFORM flag indicates that the update is to do a full  */
    /* regeneration of the image.  Any output outside of segments  */
    /* is lost.  Use the AWAIT EVENT call to pause the program     */
    /* execution.                                                  */

    gupdatews (ws_id, GPERFORM);
    gawaitevent( timeout, &event );


    /* Deactivate and close the workstation, close GKS.            */
    /* All active workstations need to be deactivated and closed   */
    /* before the call to CLOSE GKS.                               */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ( );

} /* end clean_up */
```

7 Storing the Picture in a Metafile

This chapter describes metafiles and explains how to work with them. It also provides a brief overview of the internal format of DEC GKS metafiles.

7.1 Using Metafile Functions

The DEC GKS metafile functions provide a mechanism for long-term storage, communication, and reproduction of a graphic image. Metafiles created by an application using DEC GKS can be used by other applications, also using DEC GKS, in other computer systems to reproduce a picture. When you store information in a metafile, you store specific information concerning the output primitives contained in the picture, the corresponding output attributes, and other information that may be needed to reproduce the picture.

DEC GKS supports two-dimensional and three-dimensional metafile output. This chapter highlights the similarities and the differences between the two-dimensional and three-dimensional metafiles.

The DEC GKS metafiles (GKS3 for three-dimensional and GKSM for two-dimensional) interface is a GKS workstation of type MO (metafile output) or MI (metafile input). For output and input, several different workstations of categories MO and MI can be used concurrently. Some of the workstation control and inquiry functions are not applicable to these workstations.

DEC GKS uses the metafile functions in the following list for both two-dimensional and three-dimensional metafiles. Environment variables or logical names are used to specify the dimensions for metafile output. Metafile input functions read the dimensions from the metafile that is being interpretted.

- WRITE ITEM TO GKSM writes data to the metafile.
- GET ITEM TYPE FROM GKSM delivers the type and data record of the current item.
- READ ITEM FROM GKSM copies the contents of the item data record of the current item into a data area supplied by the application program and makes the next item the current item.
- INTERPRET ITEM inspects the item copied by READ ITEM FROM GKSM. This function makes the required changes to the GKS state list and generates the graphic output specified by the metafile. If the data length argument is 0, the function ignores the item.

7.2 Creating and Reading Metafiles

DEC GKS defines a metafile output workstation for use when creating metafiles. It defines a metafile input workstation to use when reading metafiles. The following sections describe how metafile output is used to create metafiles, and how metafile input is interpretted.

7.2.1 Metafile Output

Once the metafile output workstation is active, DEC GKS records information about the current state of the picture, such as output attribute information. Then, as you call DEC GKS functions, pertinent information about the function call is recorded in a metafile record. MO workstations record the following information:

- The control functions that affect the appearance of the picture on the workstation surface.
- Output primitives, if the MO workstation is active at the time of the function call. The primitives are stored in a form equivalent to NDC points.
- Output attribute settings that are current at the time of primitive generation.
- Segments, if the MO workstation is active at the time of the call to CREATE SEGMENT.
- Geometric attribute data (such as character height, character-up vector, and so on) affecting stored text primitives, in a form equivalent to NDC points.
- Normalization transformation information such as the clipping rectangle. DEC GKS does not record workstation transformations.
- Data specific to the application, or information that DEC GKS metafiles cannot store through standard calls to the DEC GKS functions (stored using the function WRITE ITEM TO GKSM).

The metafile output default is three-dimensional (GKS3). If you want to create a two-dimensional metafile, you must specify a two-dimensional metafile output using a logical name or an environment variable.

Table 7–1 summarizes the environment options you can use when specifying metafile output.

Table 7–1 Environment Options for Metafile Output

Operating System	Environment Option	Values
VMS	GKS$METAFILE_TYPE	GKS3
	GKS$METAFILE_TYPE	GKSM
ULTRIX	GKSmetafile_type	GKS3
	GKSmetafile_type	GKSM

The value of the GKS metafile type is read in when DEC GKS is opened. Changing the logical name or environment variable during your open GKS session has no effect. If you want to use two-dimensional metafile output, you must set the environment variable prior to opening your GKS session.

User item numbers for GKSM output are integers greater than 100, while GKS3 user item numbers are integers less than 0.

It is important to understand the expected output is not guaranteed if you mix two-dimensional and three-dimensional calls.

DEC GKS uses the unity transformation (not the current normalization transformation) when generating graphic output from a metafile. It may be necessary to reestablish the application program's normalization transformation when the metafile image has been regenerated.

7.2.2 Metafile Input

To reproduce a graphic image from a GKS3 or GKSM metafile, you must open a metafile input (MI) workstation. DEC GKS defines the metafile input constant (numeric value 3) as the workstation type for MI workstations. Also, when you open the metafile input workstation, specify the name of the file containing the recorded data items as the connection identifier argument. (DEC GKS uses the file name exactly as specified, without using a default file extension.) You can open only one metafile input workstation for every corresponding physical file.

When you open a metafile input workstation, the first item written to the metafile becomes the current item. The current item is the item processed when you call the function GET ITEM TYPE FROM GKSM. As with metafile output workstations, you can open as many metafile input workstations as DEC GKS permits in total workstations, interpreting items from the appropriate metafile on the appropriate active workstations.

To reproduce the graphic image stored in the metafile, you must call GET ITEM TYPE FROM GKSM, READ ITEM FROM GKSM, and INTERPRET ITEM for all the applicable items in the metafile, until you reach the item of type 0 (specifying the last item). The function GET ITEM TYPE FROM GKSM returns the item type and the length of the data record of the current item. The function READ ITEM FROM GKSM returns the item data record and causes the next item in the metafile to become the current item. The function INTERPRET ITEM reads information about an item and reproduces the desired action on all active OUTPUT and OUTIN workstations.

In most applications, you call INTERPRET ITEM for all items in a metafile. However, there are cases when you may not wish to do this. For example, if the creator of the metafile called the function WRITE ITEM TO GKSM to pass user-defined data to the metafile, you need to handle this information in a special manner. If the user-defined data is a text string containing information for the application programmer, instead of passing the record to INTERPRET ITEM, you should store or write the text string as desired.

As another example, if you checked the item type and found it to be 3 (which is a call to the function UPDATE WORKSTATION), you may not want to interpret that item if it would delete important output primitives already on the workstation surface. For more information concerning the effects of a call to UPDATE WORKSTATION, see the chapter on control functions in your DEC GKS binding manual.

If after calling GET ITEM TYPE FROM GKSM, you decide that you do not want to interpret the item, pass the value 0 as the data length argument to READ ITEM FROM GKSM. This skips the current item, causing the next item in the file to become the current item.

7.3 Understanding Metafile Structure

Details of the GKS three-dimensional and two-dimensional metafile formats are given in the GKS–3D and GKS standards, respectively.

The standards define the metafile for the purpose of storing and retrieving information about the generation of a picture. The metafile contains information about output function calls from level 0 to level 2.

When you create the GKS3 or GKSM metafile, DEC GKS produces a metafile header, and for each function call necessary to reproduce the current environment, DEC GKS writes a series of items to the metafile. The items generated by a function call roughly correspond to the actual function call or to the state of the picture when the call was made.

For each item, DEC GKS produces an item header and an item data record. The DEC GKS standard specifies this general format for data storage with GKS3 or GKSM metafiles (metafile header, followed by an item header, followed by an item data record, and so on), but the individual item data record format is implementation specific. For example, some implementations may store all item data as a string of characters, whereas some implementations may store some information as binary-encoded integer values, and some information in character strings.

An item type is an integer value that corresponds to a DEC GKS function. For example, an item type of 3 corresponds to a call to UPDATE WORKSTATION. The item type is contained in the item header.

When creating GKS3 or GKSM metafiles, you do not need to know the information contained in the item header or the item data record. Once you activate a metafile output workstation and call output functions, DEC GKS formats the graphic output information within the metafile for you.

When you close the MO workstation, DEC GKS writes an item type of 0 to the metafile to specify that it is the last item in the metafile.

The design of the metafile structure defines a sequence of logical data items. The data items include information in both a clear text encoding and an unspecified binary format. Section 7.3.1 describes the format of logical data items, and Section 7.3.2 further describes and illustrates metafile structure.

7.3.1 Data Format Information

Integers are formatted in decimal ASCII characters in the output metafile. Floating point numbers are formatted in the standard F–Floating or E–Floating formats of decimal ASCII characters, depending on their value. DEC GKS does not support the use of a comma in place of a period in floating point numbers.

The GKS metafile allows four possible ways to represent integers and floating point numbers:

- Both integer and floating point numbers are specified by their character representations.
- Integer numbers are specified by their character representations. Floating point numbers are represented as scaled integers.
- Both integer and floating point numbers are specified by their internal binary representations.
- Integer numbers are specified by their internal binary representations. Floating point numbers are represented as scaled integers.

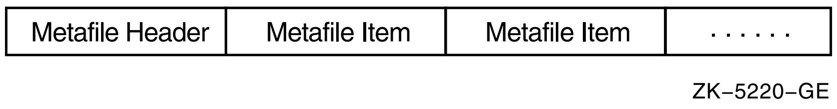
GKS metafiles also allow differing field length specifications for different fields of the metafile. The input workstation recognizes all the different field length specifications.

7.3.2 GKS Metafile Structure

A GKS metafile consists of a metafile header followed by metafile items. Each metafile item consists of an item header followed by item data.

Figure 7–1 illustrates the GKS metafile structure.

Figure 7–1 Metafile Structure



7.3.2.1 Metafile Header Structure

The three-dimensional and two-dimensional metafile headers each contain 90 bytes. The bytes are divided into 13 fields as shown in Figure 7–2.

Figure 7–2 GKS Metafile Header Structures

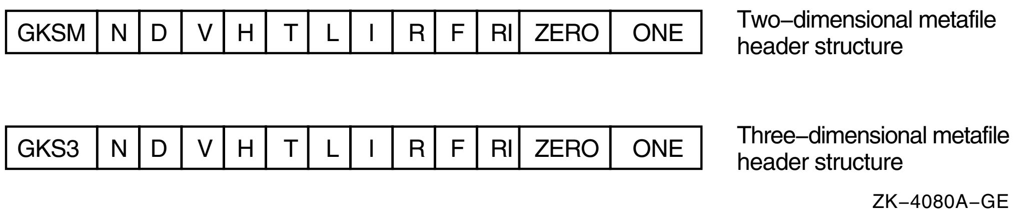

GKSM
N
D
V
H
T
L
I
R
F
RI
ZERO
ONE
Two–dimensional metafile header structure
GKS3
N
D
V
H
T
L
I
R
F
RI
ZERO
ONE
Three–dimensional metafile header structure
ZK–4080A–GE

Table 7–2 describes the fields within the metafile header.

Table 7–2 GKS Metafile Header Fields

Field	Size	Description
GKS3	4 bytes	String “GKS3”. This indicates a three-dimensional string.
GKSM	4 bytes	String “GKSM”. This indicates a two-dimensional string.
N	40 bytes	Name of author and installation. In DEC GKS, the author is the process name at the time of metafile creation (16 bytes), and installation is “DEC GKS Version n.n.”
D	8 bytes	Date (yy/mm/dd).
V	2 bytes	Version number (nn).
H	2 bytes	Integer specifying how many bytes of the string “GKS3” or “GKSM” occupy the beginning of each record (04).
T	2 bytes	Length of item type indicator field (03).
L	2 bytes	Length of item data record length indicator field (08).
I	2 bytes	Length of field for each integer in the item data record (12).
R	2 bytes	Length of field for each real in the item data record (14).

(continued on next page)

Table 7–2 (Cont.) GKS Metafile Header Fields

Field	Size	Description
F	2 bytes	Flag indicating if numbers are formatted as characters (1) or are stored in an internal binary format (2). The DEC GKS value is 01.
RI	2 bytes	Flag indicating if real numbers are stored as real numbers (01) or as scaled integers (02). The DEC GKS value is 01.
ZERO	11 bytes	Scaling information. Not used.
ONE	11 bytes	Scaling information. Not used.

7.3.2.2 Metafile Item Structure

There are several types of metafile items. Each item consists of an item header and an item data record. The item header format is the same for all types of metafile items, but the item data record varies in length and format for each type of metafile item. Figure 7–3 illustrates the structure of a metafile item.

Figure 7–3 GKS Metafile Item Structure

Item Header
Item Data Record . . .
ZK–5222–GE

7.3.2.3 Item Header Structure

Each three-dimensional and two-dimensional item header contains 15 bytes, divided into three fields. Figure 7–4 illustrates the item header structures.

Figure 7–4 GKS Metafile Item Header Structure

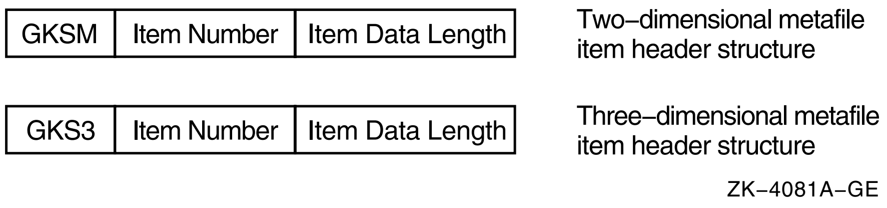

GKSM
Item Number
Item Data Length
Two–dimensional metafile item header structure
GKS3
Item Number
Item Data Length
Three–dimensional metafile item header structure
ZK–4081A–GE

Table 7–3 presents the item header fields.

Table 7–3 GKS Metafile Item Header Fields

Field	Size	Description
GKS3	4 bytes	Contains the string “GKS3”. This indicates that the item is three-dimensional.
GKSM	4 bytes	Contains the string “GKSM”. This indicates that the item is two-dimensional.

(continued on next page)

Table 7–3 (Cont.) GKS Metafile Item Header Fields

Field	Size	Description
Item Number	3 bytes	Contains an integer identifying the item.
Item Data Length	8 bytes	Contains an integer specifying the length, in bytes, of the item data record.

7.3.2.4 Layout of Item Data Records

Each item data type, identified by a unique item number (an integer), has a specific format associated with it. Table 7–4 and Table 7–5 list the possible item numbers and their associated formats.

Table 7–4 GKS Three-Dimensional Metafile Data Record Fields

Number	Format
< 0	User data.
0	END ITEM—Last item of the metafile. No data record.
1	CLEAR WORKSTATION—For all active workstations. a) Integer, 0 = CONDITIONAL or 1 = ALWAYS.
2	REDRAW ALL SEGMENTS ON WORKSTATION—No data record.
3	UPDATE WORKSTATION—For all active workstations. a) Integer, 0 = PERFORM, 1 = POSTPONE.
4	SET DEFERRAL STATE—a) Integer = deferral mode, 0 = ASAP, 1 = BNIG, 2 = BNIL, 3 = ASTI; b) Integer = regeneration mode, 0 = ALLOWED, 1 = SUPPRESSED.
5	MESSAGE—a) Integer = number of characters in string; b) string with specified number of characters.
6	ESCAPE—For all active workstations. a) Integer = function id; b) Integer = number of integers in integer array d; c) Integer = number of reals in real number array e; d) Array of integers; e) Array of reals.
11	POLYLINE—a) Integer = N, number of points of the polyline; b) N triples of real numbers. Each triple specifies the X, Y, and Z coordinates of a point as real numbers.
12	POLYMARKER—a) Integer = N, number of points of polymarker; b) N triples of real numbers. Each triple specifies the X, Y, and Z coordinates of a point as real numbers.
13	TEXT—a) Three real numbers specifying the starting position of string; b) Three coordinates of first text direction vector; c) Three coordinates of second text direction vector; d) Number N of characters in the string; e) N characters of the string.
14	FILL AREA—a) Integer = N, number of points delineating the fill area; b) N triples of real numbers. Each triple specifies the X, Y, and Z coordinates of a point as real numbers.
15	FILL AREA SET—a) Integer = number of fill areas; b) Array of integers = list of the number of points in each area; c) Array containing list of points.

(continued on next page)

Table 7–4 (Cont.) GKS Three-Dimensional Metafile Data Record Fields

Number	Format
16	CELL ARRAY—a) Three sets of point coordinates, describing three corners of the cell parallelogram; b) Integer = number of columns in array; c) Integer = number of rows in array; d) Integer array of color indexes stored row by row.
17	GDP—a) Integer = GDP identifier; b) Integer N = number of points; c) Number of integers in the integer array f; d) Number of reals in the real array g; e) Array containing coordinate points; f) Array containing integer data; g) Array containing real data; h) Number of strings; i) List of lengths of strings; j) List of strings.
21	POLYLINE INDEX—a) Integer = polyline index.
22	LINETYPE—a) Integer = line type.
23	LINEWIDTH SCALE FACTOR—a) Real number = line width scale factor.
24	POLYLINE COLOUR INDEX—a) Integer = polyline color index.
25	POLYMARKER INDEX—a) Integer = polymarker index.
26	MARKER TYPE—a) Integer = marker type.
27	MARKER SIZE SCALE FACTOR—a) Real number = marker size scale factor.
28	POLYMARKER COLOUR INDEX—a) Integer = polymarker color index.
29	TEXT INDEX—a) Integer = text index.
30	TEXT FONT AND PRECISION—a) Integer = text font; b) Integer precision, 0 = STRING, 1 = CHAR, 2 = STROKE.
31	CHARACTER EXPANSION FACTOR—a) Real number = character expansion factor.
32	CHARACTER SPACING—a) Real number = character spacing.
33	TEXT COLOUR INDEX—a) Integer = text color index.
34	CHARACTER VECTORS—a) Two real numbers specifying character height vector; b) Two real numbers specifying character width vector.
35	TEXT PATH—a) Integer, 0 = RIGHT, 1 = LEFT, 2 = UP, 3 = DOWN.
36	TEXT ALIGNMENT—a) Integer = Horizontal component, 0 = NORMAL, 1 = LEFT, 2 = CENTRE, 3 = RIGHT; b) Integer = Vertical component, 0 = NORMAL, 1 = TOP, 2 = CAP, 3 = HALF, 4 = BASE, 5 = BOTTOM.
37	FILL AREA INDEX—a) Integer = fill area index.
38	FILL AREA INTERIOR STYLE—a) Integer, 0 = HOLLOW, 1 = SOLID, 2 = PATTERN, 3 = HATCH.
39	FILL AREA STYLE INDEX—a) Integer = fill area style index.
40	FILL AREA COLOUR INDEX—a) Integer = fill area color index.
41	PATTERN REFERENCE POINT AND VECTORS—a) Three real numbers describing pattern reference point coordinates; b) Three real numbers describing first reference vector; c) Three real numbers describing second reference vector.
42	EDGE INDEX—a) Integer specifying edge index.
43	EDGE FLAG—a) Integer, 0 = OFF, 1 = ON.
44	EDGETYPE—a) Integer specifying edge type.

(continued on next page)

Table 7–4 (Cont.) GKS Three-Dimensional Metafile Data Record Fields

Number	Format
45	EDGEWIDTH SCALE FACTOR—a) Integer specifying edge width scale factor.
46	EDGE COLOUR INDEX—a) Integer specifying edge color index.
47	ASPECT SOURCE FLAGS—a) 17 integers specifying aspect source flags, 0 = BUNDLED, 1 = INDIVIDUAL.
48	PICK IDENTIFIER—a) Integer = pick identifier.
51	POLYLINE REPRESENTATION—a) Integer = polyline index; b) Integer = line type; c) Real = line width scale factor; d) Integer = polyline color index.
52	POLYMARKER REPRESENTATION—a) Integer = polymarker index; b) Integer = marker type; c) Real = marker size scale factor; d) Integer = polymarker color index.
53	TEXT REPRESENTATION—a) Integer = text index; b) Integer = text font; c) Integer = text precision, 0 = STRING, 1 = CHAR, 2 = STROKE; d) Real = character expansion factor; e) Real = character spacing; f) Integer = text color index.
54	FILL AREA REPRESENTATION—a) Integer = fill area index; b) Integer = interior style, 0 = HOLLOW, 1 = SOLID, 2 = PATTERN, 3 = HATCH; c) Integer = style index; d) Integer = fill area color index.
55	PATTERN REPRESENTATION—a) Integer = pattern index; b) Integer = number of columns in color array; c) Integer = number of rows; d) Integer table of the number of columns and rows specified containing color index values.
56	EDGE REPRESENTATION—a) Integer = edge index; b) Integer = edge flag, 0=OFF, 1=ON; c) Integer = edge type number; d) Real = edge width scale factor; e) Integer = edge color index.
57	COLOUR MODEL—a) Integer = color model.
58	COLOUR REPRESENTATION—a) Integer = color index; b) Three real numbers specifying color.
61	CLIPPING VOLUME—a) Six real numbers specifying XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX respectively.
62	CLIPPING INDICATOR—a) Integer, 0=CLIP, 1=NOCLIP.
63	VIEW INDEX—a) Integer specifying view index.
64	VIEW REPRESENTATION—a) Integer specifying view index; b) Real = 16-element view orientation matrix; c) Real = 16-element view mapping matrix; d) Real = view clipping limits (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX); e) Integer = XY clipping indicator, 0=CLIP, 1=NOCLIP; f) Integer = back clipping indicator, 0=CLIP, 1=NOCLIP; g) Integer = front clipping indicator, 0=CLIP, 1=NOCLIP.
65	HLHSR ID—a) Integer, HLHSR identifier.
66	HLHSR MODE—a) Integer, HLHSR mode.
71	WORKSTATION WINDOW—a) Six real numbers specifying XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX respectively.
72	WORKSTATION VIEWPORT—a) Six real numbers specifying XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX respectively.
81	CREATE SEGMENT—a) Integer = segment name.

(continued on next page)

Table 7–4 (Cont.) GKS Three-Dimensional Metafile Data Record Fields

Number	Format
82	CLOSE SEGMENT—No data record.
83	RENAME SEGMENT—a) Integer = old name; b) Integer = new name.
84	DELETE SEGMENT—a) Integer = segment name.
91	SET SEGMENT TRANSFORMATION—a) Integer = segment name; b) 12 real numbers specifying the transformation matrix values.
92	SET VISIBILITY—a) Integer = segment name; b) Integer = visibility, 0 = VISIBLE, 1 = INVISIBLE.
93	SET HIGHLIGHTING—a) Integer = segment name; b) Integer = highlighting, 0 = NORMAL, 1 = HIGHLIGHTED.
94	SET SEGMENT PRIORITY—a) Integer = segment name; b) Real = priority.
95	SET SEGMENT DETECTABILITY—a) Integer = segment name; b) Integer = detectability, 0 = UNDETECTABLE, 1 = DETECTABLE.

Table 7–5 GKS Two-Dimensional Metafile Data Record Fields

Number	Format
0	END ITEM—Last item of the metafile. No data record.
1	CLEAR WORKSTATION—For all active workstations. a) Integer, 0 = CONDITIONAL or 1 = ALWAYS.
2	REDRAW ALL SEGMENTS ON WORKSTATION—No data record.
3	UPDATE WORKSTATION—For all active workstations. a) Integer, 0 = PERFORM, 1 = POSTPONE.
4	SET DEFERRAL STATE—a) Integer = deferral mode, 0 = ASAP, 1 = BNIG, 2 = BNIL, 3 = ASTI; b) Integer = regeneration mode, 0 = ALLOWED, 1 = SUPPRESSED.
5	MESSAGE—a) Integer = number of characters in string; b) String with specified number of characters.
6	ESCAPE—For all active workstations. a) Integer = function id; b) Integer = number integers in integer array d; c) Integer = number of reals in real number array e; d) Array of integers, e) Array of reals.
11	POLYLINE—a) Integer = N, number of points of the polyline; b) N pairs of real numbers. Each pair specifies the X and Y coordinates of a point as real numbers.
12	POLYMARKER—a) Integer = N, number of points of polymarker; b) N pairs of real numbers. Each pair specifies the X and Y coordinates of a point as real numbers.
13	TEXT—a) Two real numbers specifying the starting position of string; b) Number N of characters in the string; c) N characters of the string.
14	FILL AREA—a) Integer = N, number of points delineating the fill area; b) N pairs of real numbers. Each pair specifies the X and Y coordinates of a point as real numbers.
15	CELL ARRAY—a) Three sets of point coordinates, describing three corners of the cell parallelogram; b) Integer = number of columns in array; c) Integer = number of rows in array; d) Integer array of color indexes stored row by row.

(continued on next page)

Table 7–5 (Cont.) GKS Two-Dimensional Metafile Data Record Fields

Number	Format
16	GDP—a) Integer = GDP identifier; b) Integer N = number of points; c) Number of bytes of the integer array f; d) Number of bytes of the real array g; e) Array containing coordinate points; f) Array containing integer data; g) Array containing real data; h) Number of strings; i) List of lengths of strings; j) List of strings.
21	POLYLINE INDEX—a) Integer = polyline index.
22	LINETYPE—a) Integer = line type.
23	LINEWIDTH SCALE FACTOR—a) Real number = line width scale factor.
24	POLYLINE COLOUR INDEX—a) Integer = polyline color index.
25	POLYMARKER INDEX—a) Integer = polymarker index.
26	MARKER TYPE—a) Integer = marker type.
27	MARKER SIZE SCALE FACTOR—a) Real number = marker size scale factor.
28	POLYMARKER COLOUR INDEX—a) Integer = polymarker color index.
29	TEXT INDEX—a) Integer = text index.
30	TEXT FONT AND PRECISION—a) Integer = text font; b) Integer precision, 0 = STRING, 1 = CHAR, 2 = STROKE.
31	CHARACTER EXPANSION FACTOR—a) Real number = character expansion factor.
32	CHARACTER SPACING—a) Real number = character spacing.
33	TEXT COLOUR INDEX—a) Integer = text color index.
34	CHARACTER VECTORS—a) Two real numbers specifying character height vector; b) Two real numbers specifying character width vector.
35	TEXT PATH—a) Integer, 0 = RIGHT, 1 = LEFT, 2 = UP, 3 = DOWN.
36	TEXT ALIGNMENT—a) Integer = Horizontal component, 0 = NORMAL, 1 = LEFT, 2 = CENTRE, 3 = RIGHT; b) Integer = vertical component, 0 = NORMAL, 1 = TOP, 2 = CAP, 3 = HALF, 4 = BASE, 5 = BOTTOM.
37	FILL AREA INDEX—a) Integer = fill area index.
38	FILL AREA INTERIOR STYLE—a) Integer, 0 = HOLLOW, 1 = SOLID, 2 = PATTERN, 3 = HATCH.
39	FILL AREA STYLE INDEX—a) Integer = fill area style index.
40	FILL AREA COLOUR INDEX—a) Integer = fill area color index.
41	PATTERN VECTORS—a) Two real numbers describing the pattern width vector; b) Two real numbers describing the pattern height vector.
42	PATTERN REFERENCE POINT—a) Two real numbers describing the coordinates of the reference point.
43	ASPECT SOURCE FLAGS—a) 13 integers specifying aspect source flags, 0 = BUNDLED, 1 = INDIVIDUAL.
44	PICK IDENTIFIER—a) Integer = pick identifier.
51	POLYLINE REPRESENTATION—a) Integer = polyline index; b) Integer = line type; c) Real = line width scale factor; d) Integer = polyline color index.
52	POLYMARKER REPRESENTATION—a) Integer = polymarker index; b) Integer = marker type; c) Real = marker size scale factor; d) Integer = polymarker color index.

(continued on next page)

Table 7–5 (Cont.) GKS Two-Dimensional Metafile Data Record Fields

Number	Format
53	TEXT REPRESENTATION—a) Integer = text index; b) Integer = text font; c) Integer = text precision, 0 = STRING, 1 = CHAR, 2 = STROKE; d) Real = character expansion factor; e) Real = character spacing; f) Integer = text color index.
54	FILL AREA REPRESENTATION—a) Integer = fill area index; b) Integer = interior style, 0 = HOLLOW, 1 = SOLID, 2 = PATTERN, 3 = HATCH; c) Integer = style index; d) Integer = fill area color index.
55	PATTERN REPRESENTATION—a) Integer = pattern index; b) Integer = number of columns in color array; c) Integer = number of rows; d) Integer table of the number of columns and rows specified containing color index values.
56	COLOUR REPRESENTATION—a) Integer = color index; b) Three real numbers specifying color.
61	CLIPPING RECTANGLE—a) Four real numbers specifying XMIN, XMAX, YMIN, YMAX respectively.
71	WORKSTATION WINDOW—a) Four real numbers specifying XMIN, XMAX, YMIN, YMAX respectively.
72	WORKSTATION VIEWPORT—a) Four real numbers specifying XMIN, XMAX, YMIN, YMAX respectively.
81	CREATE SEGMENT—a) Integer = segment name.
82	CLOSE SEGMENT—No data record.
83	RENAME SEGMENT—a) Integer = old name; b) Integer = new name.
84	DELETE SEGMENT—a) Integer = segment name.
91	SET SEGMENT TRANSFORMATION—a) Integer = segment name; b) Six real numbers specifying the transformation matrix values.
92	SET VISIBILITY—a) Integer = segment name; b) Integer = visibility, 0 = VISIBLE, 1 = INVISIBLE.
93	SET HIGHLIGHTING—a) Integer = segment name; b) Integer = highlighting, 0 = NORMAL, 1 = HIGHLIGHTED.
94	SET SEGMENT PRIORITY—a) Integer = segment name; b) Real = priority.
95	SET SEGMENT DETECTABILITY—a) Integer = segment name; b) Integer = detectability, 0 = UNDETECTABLE, 1 = DETECTABLE.
>100	User data.

7.4 GKS Metafile Physical File Organization

The GKS metafile has variable length record format, with a limit on the maximum record size of 512 bytes. File organization is sequential.

Each metafile item occupies two or more RMS records; one for the item header and one or more for the item data record. The metafile header occupies one RMS record. The record item data record occupies at least one RMS record. If the item data record has a length greater than 512 bytes, then the data record is split into two or more RMS records.

A Color Models

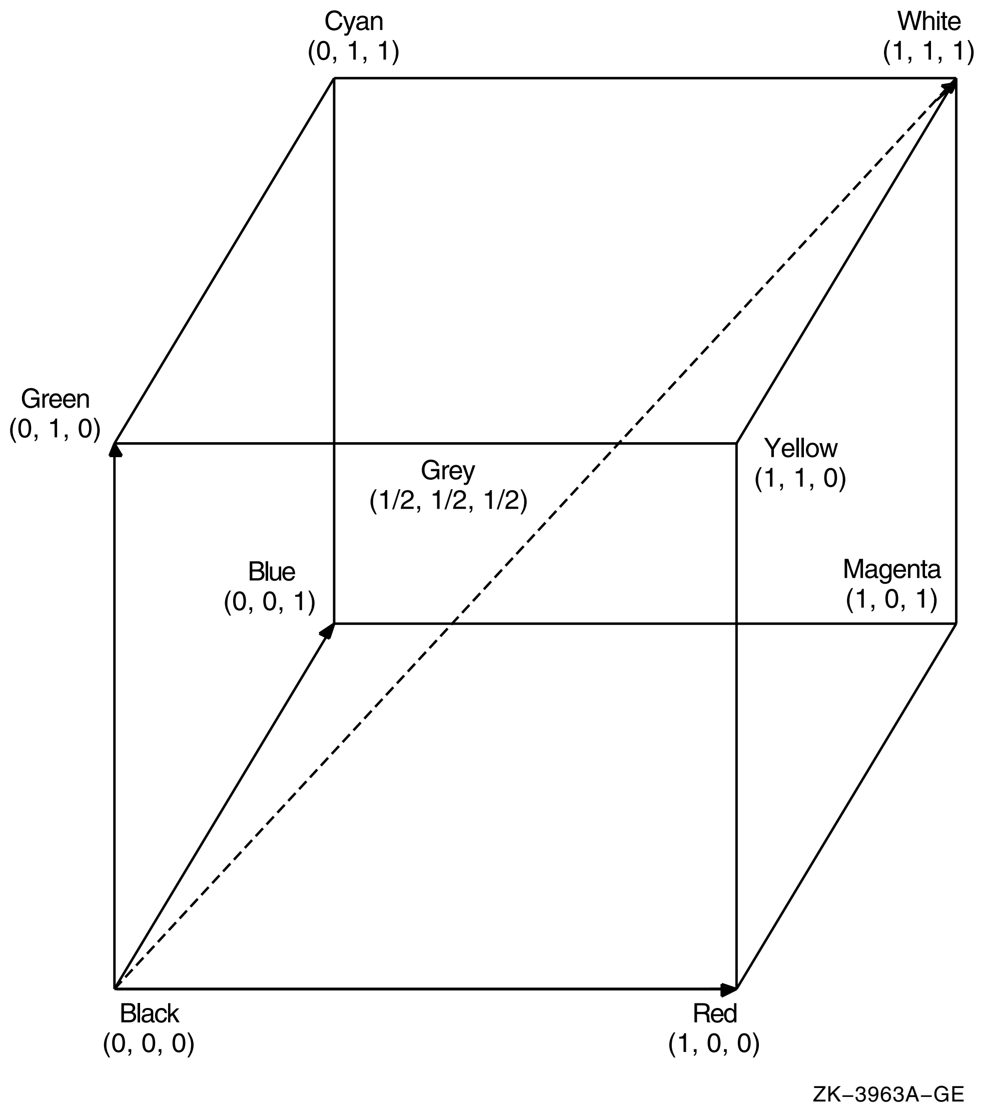
The use of color is an important aspect of working with DEC GKS. DEC GKS supports color models that let you specify colors within some color range. A color model is a specification of a three-dimensional color coordinate system and a three-dimensional subspace in the coordinate system where each displayable color is represented as a point. This appendix describes the color models supported by DEC GKS.

A.1 RGB Color Model

The RGB color model is a cube that has as its axes the primary additive colors, red, green, and blue. Each point within the cube has a red intensity value (X coordinate), a green intensity value (Y coordinate), and a blue intensity value (Z coordinate). Thus, the color triplet values are for the red, green, and blue intensities that form the final colors. Each axis or intensity ranges from 0 to 1, and each point defines a unique color.

The exact shade of the final color depends on the capabilities of the workstation. For example, the intensities 0.0000, 0.0000, 0.0000 produce a black color representation on some devices, 1.0000, 1.0000, 1.0000 produces white, and 0.0000, 0.0000, 0.8400 a shade of blue, and so on. The number of colors a device can support is also dependent on the device. For more information about color capabilities, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*. Figure A–1 illustrates the RGB color model.

Figure A–1 The RGB Color Model



A.2 CIE Color Model

The CIE (Commission International de l'Eclairge) 1931 XYZ color space defines colors independently of the specific primaries and alignment white of the display device. It describes all physically realizable colors. The CIE 1931 XYZ color space definition is derived from experiments in color matching; therefore, it is based on properties of the human visual system rather than the properties of a specific device. Use of the CIE 1931 system lets color-matched colors be produced on different hardcopy and display devices. This is accomplished through use of constants derived from the specification of the primary colors of the specific device.

CIE XYZ colors are specified by three independent quantities or tristimulus values. The CIE system uses hypothetical primaries chosen outside the range of real colors.

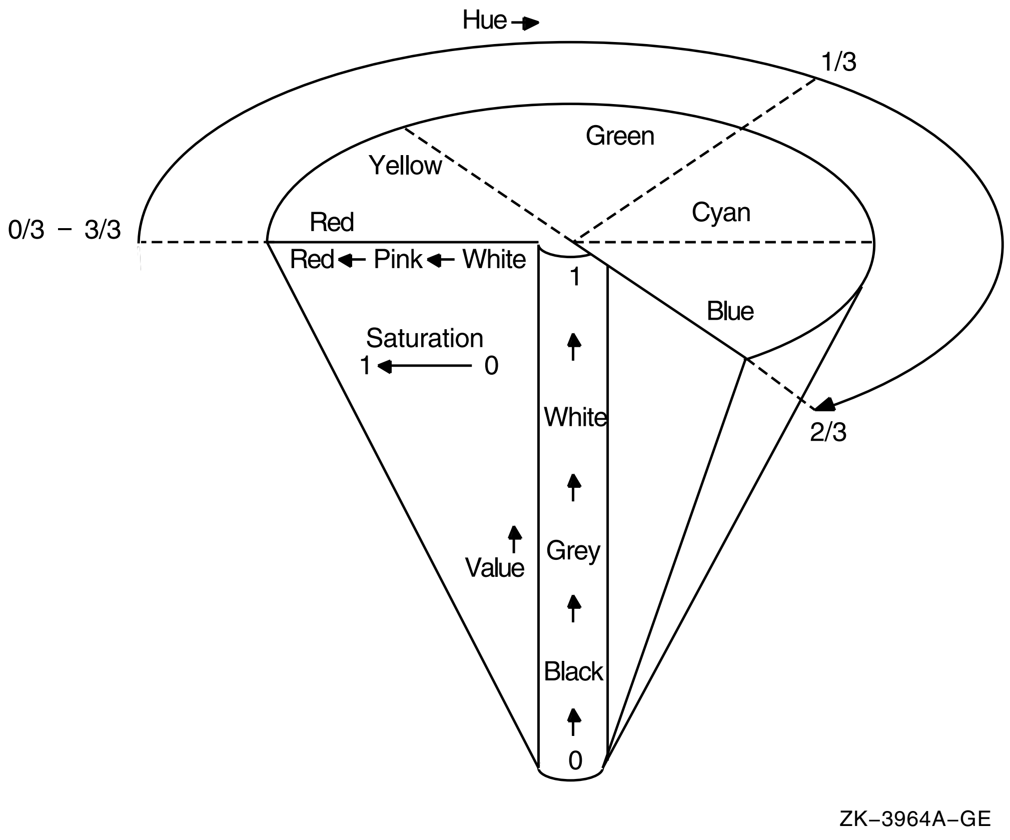
The main drawback of the CIE 1931 space is that it is nonuniform. Equal distances in this space do not correspond to equal perceptual differences. The CIE 1976 (L*u*v*) space, often referred to as the CIELUV system, is a transformation of the CIE 1931 color space that is approximately uniform for small color differences.

For indepth information on the CIE color model, see the American National Standard X3.144-1988.

A.3 HSV Color Model

The HSV color model is a cone. The color triplet values are for hue, saturation, and value. Hue is the angle about the axis of the color cone, in fractions of a circle; saturation is the radius; and value is the height, which specifies the intensity or brightness per unit area. Saturation and value range from 0 to 1, while hue ranges from 0 to 360. Each point within the cone defines a unique color with a hue value, a saturation value, and a height value. Figure A–2 illustrates the HSV color model.

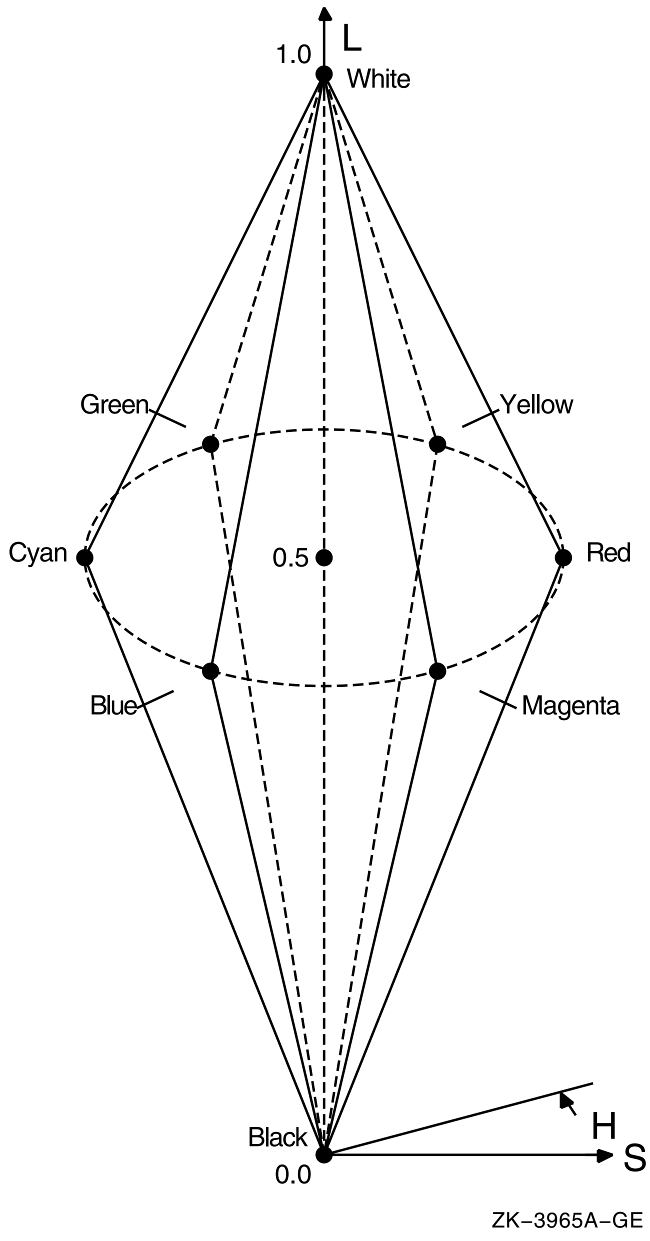
Figure A–2 The HSV Color Model



A.4 HLS Color Model

The HLS color model is a double-ended color cone. The color triplet values are for hue, lightness, and saturation. Hue is the angle about the axis of the color cone, in fractions of a circle; the radius is saturation; and lightness is the height, which specifies the intensity or brightness per unit area. Lightness and saturation range from 0 to 1, while hue ranges from 0 to 360. Each point within the cone defines a unique color with a hue value, a saturation value, and a lightness value. Figure A–3 illustrates the HLS color model.

Figure A–3 The HLS Color Model



Glossary

active devices

Logical input devices whose prompts currently appear on the workstation surface. To deactivate a logical input device, you remove its prompt from the workstation surface by placing the device in request mode.

aperture

A pick logical input cursor that a user positions on a segment to be picked.

aspect ratio

The ratio of lengths along the principal axes of an object. In two dimensions, the aspect ratio is the ratio of Y to X used to describe the shape of a rectangle in a particular coordinate system, such as a workstation window or workstation viewport.

aspect source flag (ASF)

A setting that tells DEC GKS to use either individual or bundled attribute settings when generating output primitives.

asynchronous input

A method of interaction between an application and its user. DEC GKS sample and event asynchronous input modes allow the user to enter input values while the application continues to execute. See also *synchronous input*.

attribute

A particular property that applies to an output primitive (such as character height) or to a segment (such as highlighting).

back plane

The furthest plane parallel to the view plane, which illustrates how the back side of the view volume is made finite.

background color

The color of a blank workstation surface.

baseline

A horizontal line within a character body which, for many character definitions, has the appearance of being the lower limit of the character shape. A descender passes below this line. All baselines in a font are in the same position in the character bodies.

bound attributes

Attribute values that DEC GKS assigns to a primitive at the time of output generation. Bound attribute values cannot be changed.

break

A device-dependent method the user has of terminating the input process without changing the measure of a logical input device. See also *measure*.

buffer

A temporary storage area.

bundle

A collection of primitive attributes.

bundle index

An integer value that specifies a single entry in a bundle table for a particular output primitive. A bundle table entry contains settings for each nongeometric attribute of an output primitive.

bundle table

A workstation-dependent table associated with a particular output primitive. Entries in the table specify all the workstation-dependent aspects of a primitive. Bundle tables exist for the output primitives polyline, polymarker, text, and fill area. See also *attribute*.

caplines

A horizontal line within a character body, which, for many character definitions, has the appearance of being the upper limit of the character shape. An ascender may pass above this line and in some languages an additional mark (for example, an accent) over the character may be defined above this line. All caplines in a font are in the same position in the character bodies.

cell array

An output primitive consisting of a parallelogram of equal size cells. Each cell is a parallelogram that has a single color.

centerline

A vertical line bisecting the character body.

character body

A rectangle used by a font designer to define a character shape. All character bodies in a font have the same height.

choice device

A logical input device providing a nonnegative integer defining one of a set of alternatives. An example of a choice prompt is a menu with a single highlighted choice.

clipping

Removing parts of output primitives that extend outside a window or viewport.

clipping indicator

A flag the enables or disables clipping.

clipping limits

Boundaries that specify the region of space in which visible data may appear, providing clipping is turned on.

color index

Integer values specified in two-dimensional arrays that define colors.

color table

A workstation-dependent table in which the entries associate red, green, and blue intensity values to color indexes. Color indexes are integer values that indicate individual color table entries.

composition

The putting together of individual pieces into a scene.

current event report

The oldest event report, which is removed from the event input queue and placed into the GKS state list by a call to AWAIT EVENT (as long as the queue contains at least one report). See also *event input queue* and *event report*.

cycling

A process by which a user activates the prompt of only one logical input device and deactivates all other prompts, activating each prompt in succession in some device-specific order. See also *active devices*.

deferral mode

This mode specifies when, after changes to a segment have occurred, the picture must be made visually correct.

deferred output

A process of delaying the transmission of output primitives to the surface of a workstation.

detectability

An attribute that controls whether an application program can detect a selection of an object from the display screen made by a user with a logical pick input device.

device coordinate system

A physical device's coordinate system used by a DEC GKS workstation. Device coordinate units are device dependent, and are expressed either in meters or some device-specific unit of measure. You use all or part of the device coordinate system to present graphic pictures.

device dependent

A property that is unique to a particular device (terminal, plotter, workstation, and so on). For example, the device coordinate system range is device dependent; its minimum and maximum X and Y values differ from device to device.

device handler

A series of routines that perform device-specific operations. Handler operations include performing output, getting input, and responding to inquiries for workstation-specific information.

device independent

A property that remains consistent no matter which device you use (terminal, plotter, workstation, and so on). For example, text height, as it is expressed in WC units, is a device-independent attribute; a single value can apply to any workstation you use.

display surface

See *workstation surface*.

dynamic changes

The attribute or transformation changes that DEC GKS can implement without having to redraw the entire picture on the workstation surface.

echo

A visual indication on the workstation surface of the current logical input device measure. See also *prompt and echo types*.

escape

A function used to access implementation or device-dependent features other than output generation (generalized drawing primitives access device-dependent output features).

event input queue

A time-ordered queue on which a device handler places an input report, which the input process generates each time the user triggers a device in event mode. See also *event report* and *trigger*.

event mode

An asynchronous input operating mode that allows the user to trigger input devices, placing event reports on the event input queue. When the application chooses, it removes the oldest report from the queue, places it in the current event report, and processes the information. See also *asynchronous input* and *event report*.

event queue overflow

A condition that occurs when the queue cannot accept subsequently generated reports. To allow the user to generate further reports, you need to empty the event input queue. See also *event report*.

event report

A data structure that the input process creates and places on the queue when a user triggers a device in event mode. The data structure contains a workstation identifier, a device number, an input class specification, a simultaneous event flag, and input data. See also *event mode*, *simultaneous events*, and *trigger*.

fill area

An output primitive consisting of a polygon that may be hollow or may be filled with a uniform color, a pattern, or a hatch style.

fill area bundle table

A table associating specific values, for all fill area nongeometric attributes, with a fill area bundle index. This table contains entries consisting of interior style, style index, and color index. See also *index*.

foreground color

The color a workstation uses to represent output primitives.

front plane

The foremost plane parallel to the view plane, which illustrates how the front side of the view volume is made finite.

generalized drawing primitive (GDP)

An output primitive used to address special geometrical workstation capabilities, such as a curve drawing. For example, a workstation can support a GDP that draws circles.

geometric attribute

A primitive attribute that is device dependent. For example, character height, character path, and pattern size are geometric attributes.

GKS level

A value from both the output levels (m, 0, 1, 2) and the input levels (a, b, c) that together define the minimal functional capabilities provided by a specific GKS implementation.

GKS metafile (GKSM)

A standard metafile structure used by DEC GKS. See also *metafile*.

graphics handler

A device-dependent part of a DEC GKS implementation that supports a physical device. The DEC GKS kernel uses graphics handlers to perform the device-dependent tasks involved with output generation and input requests.

halfline

A horizontal line between the capline and the baseline within the character body, on which a horizontal string of characters in a font would appear centrally placed in the vertical direction. All halflines in a font are in the same position in the character bodies.

hatch

One possible method of filling the interior of a fill area primitive. DEC GKS fills the interior with an arrangement of one or more sets of parallel lines. When generating hatches, DEC GKS overlays the hatch so that portions of the underlying primitives are still visible.

hidden line hidden surface removal (HLHSR)

The process of selecting for inclusion in a two-dimensional image of a three-dimensional scene only those primitives (and parts of primitives) that are visible from the view point in three-dimensional space; that is, only those primitives and parts of primitives that are not occluded by intervening opaque objects.

highlighting

A device-independent way of emphasizing a segment by modifying its appearance on the workstation surface. For example, an implementation of GKS can highlight a segment by causing all segment primitives to blink on and off.

identity segment transformation

A default segment transformation (number 0) that makes no changes to the segment as stored on the NDC plane.

implicit regeneration

A process of clearing the workstation surface and redrawing only the stored segments. DEC GKS performs implicit regenerations if you request an attribute or transformation change that cannot be implemented dynamically. If an implicit regeneration occurs, you lose all primitives not stored in segments. See also *dynamic changes* and *segment*.

index

An integer value that specifies a single entry in a bundle table. See also *bundle table*.

input class

A set of input devices that returns the same DEC GKS data type. The input classes are locator, stroke, valuator, choice, pick, and string.

input data record

A record that contains information about the prompt and echo type of a logical input device.

input trigger

An action performed by the user on a physical input device to return the measure. This action is performed in event and request modes.

inquiry function

A function that returns default or current values contained in DEC GKS data structures. Calls to these functions have no effect on the DEC GKS operating state or on the currently generated picture.

insert transformation

A transformation that is associated with the INSERT SEGMENT (3) function.

interaction

A request for input from an application user.

kernel

A part of a GKS implementation that performs device-independent tasks. To perform device-dependent tasks, the DEC GKS kernel calls a specified graphics handler.

locator device

A logical input device that accepts a device coordinate point and returns the corresponding WC points. See also *world coordinate system*.

lock

A disabling of an active logical input device prompt so that its measure cannot be changed. See also *active devices*, *event mode*, and *measure*.

logical device number

An identifier used to distiguish one logical input device from another of the same input class on the same workstation.

logical input device

An abstraction of one or more physical devices that delivers logical input values to the program. Logical input device classes are locator, stroke, valuator, choice, pick, and string.

mapping

A process of transferring the contents of a window to the interior of a viewport. Mapping in a normalization transformation establishes a one-to-one correspondence between the points in both the window and the viewport. Mapping in a workstation transformation establishes a one-to-one correspondence between the points in the window, and the points in the section of the viewport that maintains the aspect ratio of the picture. See also *aspect ratio*.

marker

A symbol with a specified appearance that you use to identify a particular location.

measure

A current value of a logical input device.

metafile

An audit of a DEC GKS picture generation session. Metafiles are used for long-term graphic data storage. You can use a metafile to reproduce a picture generated by another application program.

MI

An abbreviation for the DEC GKS metafile input workstation category.

MO

An abbreviation for the DEC GKS metafile output workstation category.

nominal sizes

A default line width or marker size as determined by the graphics handler. DEC GKS adjusts these sizes by multiplying the nominal size by the current scale factor value.

nongeometric attribute

A primitive attribute that is device independent. For example, line type, marker size scale factor, and fill area interior style are nongeometric attributes.

normalization transformation

A process of mapping a window in WC space into a viewport in NDC space. You use normalization transformations to compose a device-independent picture.

normalization viewport

The portion of the NDC plane onto which the normalization window is mapped.

normalization window

The rectangular portion of the WC system in which you plot a graphic image.

normalized device coordinate (NDC) system

A device-independent coordinate system, normalized to a range, typically 0 to 1.

normalized projection coordinate (NPC) system

A three-dimensional coordinate system in which the composition of an image is specified to the graphics system. The projection viewpoint and workstation window are specified in NPC space.

occludes

Overlaps.

operating modes

The synchronous and asynchronous methods of input. The three input operating modes are request, sample, and event mode.

operating state

The ability to access a given number of DEC GKS data structures depending on the previously called control functions. The current DEC GKS operating state determines which DEC GKS functions you can or cannot call at a given point in an application program.

output attribute

See *attribute*.

output primitive

See *primitive*.

overflow

See *event queue overflow*.

painter's algorithm

An algorithm that deals with hidden line hidden surface removal by telling the programmer to enter first the polygons that are farthest from the viewer (background), and enter last the objects closest to the viewer (foreground). The polygons in the background can be covered by the polygons in the foreground; therefore, hidden surfaces can be "covered up" by choosing the correct order to draw them.

parallel projection

The type of projection in which a three-dimensional object remains constant with regard to relative distance or depth.

parallelpiped

A prism whose bases are parallelograms.

pattern

A cell array that alternates its cells in a sequence of colors or shades. Patterns always overwrite any underlying primitives.

perspective projection

The type of projection in which a three-dimensional object recedes or appears closer with respect to relative distance or depth.

physical input device

A tool used to generate graphic output or accept graphic input. Terminals, plotters, printers, and workstations are examples of physical devices.

pick device

A logical input device that accepts a device coordinate point and returns the name of the segment that contains the picked primitive. DEC GKS segment names are integer values. See also *segment*.

pick identifier

An output attribute that allows you to name output primitives within a segment. At the time of primitive generation, DEC GKS assigns the current pick identifier integer value to the primitive. During pick input, the pick logical input device returns both the name of the segment that contains the picked primitives, and that primitive's pick identifier.

picture

A collection of output primitives or segments displayed at any one time on a workstation surface.

pixel

The smallest element of a display surface that can be independently assigned a color or intensity.

polyline

An output primitive consisting of a set of connected lines.

polyline bundle table

A table associating specific values, for all polyline nongeometric attributes, with a polyline bundle index. This table contains entries consisting of line type, line width scale factor, and polyline color index. See also *index*.

polymarker

An output primitive consisting of a set of locations indicated by a symbol.

polymarker bundle table

A table associating specific values, for all polymarker nongeometric attributes, with a polymarker bundle index. This table contains entries consisting of marker type, marker size scale factor, and polymarker color index. See also *index*.

presentation

The viewing of a particular portion of an entire scene.

primitive

An element that you use to construct a graphic picture. The output primitives are polyline, polymarker, text, fill area, cell array, and generalized drawing primitive.

primitive attribute

See *attribute*.

priority

A segment attribute used to determine which of several overlapping segments takes precedence.

projection reference point (PRP)

A VRC point that determines (together with the center of the view window) the direction of all projectors in a parallel projection, or the point from which all projectors emanate in a perspective projection.

projection viewport

A rectangular parallelepiped in the NPC system with edges parallel to the NPC axes.

prompt

A visual indication on the workstation surface of the current value of a logical input device.

prompt and echo types

Different visual indications used by devices of a logical input class. For example, a locator logical input device can prompt a user by placing cross hairs, a tracking plus sign, or a cross on the workstation surface.

queue

See *event input queue*.

raster graphics

Computer graphics in which a display image is composed of an array of pixels arranged in rows and columns.

raster units

Number of pixel rows and columns on a physical device.

representation

The attribute or RGB settings for a single bundle table entry.

request mode

A synchronous input operating mode that allows the user to trigger a logical input device once, returning its current measure. See also *synchronous input* and *trigger*.

rotation

Turning all or part of a segment around a fixed point axis.

sample mode

An asynchronous input operating mode that allows the application program to read the current measure of a logical input device; the user can only control the current measure and cannot trigger the device. See also *asynchronous input* and *measure*.

scaling

Enlarging or reducing all or part of a segment toward or away from a fixed point axis.

segment

A collection of output primitives that can be manipulated as a unit.

segment attributes

Properties that apply to segments. Segment attributes are visibility, highlighting, detectability, priority, and segment transformation. See also *attributes*.

segment transformation

A number specifying an associated matrix that expresses values for segment scaling, rotation, and translation.

simultaneous events

Several event reports generated by a trigger that affects several devices active at the same time. The device handler places the simultaneously generated events on the queue in some device-specific order. See also *event mode* and *event report*.

string device

A logical input device that returns a character string.

stroke device

A logical input device that returns the set of WC points. See also *world coordinate system*.

synchronous input

A method of interaction between an application and its user. DEC GKS request mode causes the application to pause while the user alters the measure of a device and triggers input. After the user triggers the device, the device handler removes the device's prompt from the workstation surface and application execution resumes. See also *asynchronous input*.

text

An output primitive consisting of a character string.

text bundle table

A table associating specific values, for all text nongeometric attributes, with a text bundle index. This table contains entries consisting of text font and precision, character expansion factor, character spacing and text color index. See also *index*.

text font and precision

A text attribute having the components *font* and *precision*, which together determine the shape of the characters being output on a particular workstation. In addition, the precision describes the effectiveness of the other text attributes. In order of increasing permitted effectiveness, the precisions are string, character and stroke.

transformation

A mapping of primitives from one coordinate system to another coordinate system.

translation

Altering a segment's coordinate points so that the segment appears in a new position in the picture.

trigger

An indication from the user telling a logical input device to accept the current input value. An example of an input trigger is the pressing of the Return key.

unit cube

The default clipping volume in the NPC system that extends from the origin (0, 0, 0) at the lower left corner to (1, 1, 1) at the upper right front corner.

unity transformation

A default normalization transformation (number 0) that uses the default normalization window and viewport.

update

A process of releasing deferred output and implementing all previously requested changes to the picture, if necessary.

valuator device

A logical input device that accepts and returns real values.

view attribute

A label that identifies an attribute associated with viewing.

view clipping limits

Boundaries, located inside the unit cube, defining a parallelepiped volume that specifies the region of NPC space in which visible data can appear, providing clipping is turned on.

view mapping matrix

A 4x4 matrix used to define the view transformation.

view orientation matrix

The matrix that defines what direction the object is being viewed from, as well as what direction is up.

view plane

A plane onto which three-dimensional objects are projected. The view plane is established by giving the view reference point, a view plane normal, and a view plane distance.

view plane distance

The distance of the view plane from the view reference point measured in VRC points along the view plane normal.

view plane normal

A vector normal to the view plane used to orient the plane.

view reference coordinate (VRC) system

The three-dimensional coordinate system in which the parameters for the view mapping transformations are specified. The axes of the VRC system are labeled U, V, and N. The position and orientation of this system, relative to world coordinates, are defined by the view reference point, the view plane normal, and the view up vector.

view reference plane

A plane in the WC or NDC system that contains the view reference point and that is perpendicular to the view plane normal.

view reference point

A WC or NDC point that defines the origin of the VRC system. Typically, the point is on or near the object being viewed.

view table

An array of view definitions. Every defined workstation state list contains a view table. Views are referenced by view index elements.

view transformation

The transformation that maps primitives from NDC points to the NPC system.

view up vector

A vector defined in WC space. The V axis of the VRC system is defined as an orthogonal projection of the view up vector onto the plane through the view reference point and perpendicular to the view plane normal. Vectors that are parallel to the view up vector in WC space will appear vertical in the final image.

view volume

The view volume is either a frustrum for perspective projections or a parallelepiped for parallel projections. The view volume is defined using the following:

- The view plane
- The front and back clipping planes

- The projectors from the projection reference point through the window center (for parallel projection) or from the projection reference point through the corners of the view window (for perspective projection)

view window

A rectangular region in the view plane.

viewing pipeline

A series of operations that let a three-dimensional object be displayed on the surface of a workstation.

viewport

A defined rectangular area (for two-dimensional systems) or volume (for three-dimensional systems) on a coordinate system into which DEC GKS maps primitives contained in a window.

visibility

An attribute that controls whether segments are visible or invisible on a display surface.

window

A defined rectangular area (for two-dimensional systems) or volume (for three-dimensional systems) on a coordinate system from which DEC GKS maps primitives to a viewport.

workstation

An abstract graphic device that includes a physical device and a software graphics handler that drives the physical device.

workstation category

Groupings of workstations determined by the various capabilities of each physical device.

workstation dependent

The logical software layers of DEC GKS that consist of the modules associated with the metafile input and metafile output logical workstation handlers, workstation independent segment storage, workstation handlers for three-dimensional devices, and the workstation manager for two-dimensional devices.

workstation dependent segment storage (WDSS)

Segment storage on a workstation other than a workstation of the category workstation independent segment storage. Segments cannot be transferred from WDSS to another workstation.

workstation handler

See *graphics handler*.

workstation identifier

An identifier that is used to reference the workstation being operated.

workstation independent

A logical software layer of DEC GKS that is independent of the hardware being used.

workstation independent segment storage (WISS)

A special workstation type, where segments can be stored and later transferred to other workstations.

workstation surface

The portion of the device space corresponding to the area available for displaying images or for input working space (such as a digitizer).

workstation transformation

A transformation that maps the boundary and interior of a workstation window into the boundary and interior of a workstation viewport (part of display space), preserving aspect ratio. The effect of preserving aspect ratio is that the interior of the workstation window may not map to the whole of the workstation viewport. You use workstation transformations to control the amount of the picture shown on a given portion of a workstation surface.

workstation viewport

A portion of the display space selected for output.

workstation window

A rectangular parallelepiped within the NPC system represented on the display surface.

world coordinate (WC) system

An imaginary device-independent Cartesian coordinate system that you use to plot output primitives. Once you plot a primitive in world coordinates, you must establish the proper normalization transformation, and then pass the world coordinate points to an output function.

Index