Chapter 1

Graphics Development Environment

This chapter provides a brief overview of the graphics development environment. The graphics development environment contains the tools and systems that you work with when you write GL programs.

The tools available in the graphics development environment include your workstation, the IRIX operating system, the IRIS Graphics Library, system libraries, include files, the X Window System™, and a programming language that provides the interface to the GL. This guide describes the ANSI C interface to the GL.

The IRIS workstation processes graphics operations through the Geometry Pipeline™. The Geometry Pipeline is like an assembly line that performs the specialized graphics tasks that create and display graphics.

The IRIX operating system provides commands for setting up and maintaining your system, creating and editing files, and using IRIX system calls in your GL applications.

1.1 Using the Graphics Library and System Libraries

The GL is a library of subroutines that you call from a program, written in C, or another language, to draw and animate 2-D and 3-D color graphics scenes. You build your application using GL commands within the framework of the programming language. The programming language provides the logical structure for your program, and GL commands provide the interface to the graphics software and hardware.

The GL is network-transparent, so you can send graphics information over the network to a remote host, send graphics information to multiple display screens, or share processing tasks with other systems. To use network-transparent features, you must link with the shared libraries, as described in Section 1.3.2, "Compiling C Programs." See Chapter 19, "Using the GL in a Networked Environment," for more information about network transparency. The X Window System, described in Section 1.2, "Using the X Window System," manages operations over the network.

1.1.1 System Libraries

System libraries allow you to use capabilities such as graphics, fonts, and math routines. Shared libraries provide the optimum use of system resources and the best portability and compatibility between IRIS platforms. Libraries such as the shared graphics library (*libgl_s.a*), the math library (*libm.a*), and the font library (*libfm.a*) are invoked when you link to them when compiling your program.

1.1.2 Include Files

Include files provide standard definitions that your program uses. The include file *gl/gl.h* provides the standard definitions for graphics. Include files such as *math.h*, *device.h*, and *stdio.h* provide definitions for system facilities that your program uses.

1.2 Using the X Window System

Your workstation uses the X Window System. The X Window System provides resource management and communication through a client/server system, known as the X client and X server, that you can read about in your X Window System documentation. A few of the facilities that it includes are a terminal emulator (*xterm*), a window manager (*4Dwm*), a manual page browser (*xman*), mail managing utilities, user customizing options, font utilities, demos, and toolkits for creating graphical user interfaces.

The client/server model allows for remote display of graphics output. The DISPLAY environment variable determines where output is to be displayed.

Most of the time you display graphics on the default display, which is the screen attached to your workstation.

The X Window System designates displays by:

```
hostname:server.screen
```

where:

hostname	is the name of the server on which the display resides.
server	is the number of the X server. Some setups have more than one server, so they are numbered starting with 0; the default is 0.
screen	is the screen to display on. Some systems have more than one screen; the default is 0.

A typical display is:

```
mysystem.mydomain:0.0
```

If you are displaying on the system in front of you, it is called a *local host*. When displaying on a local host, you do not have to specify the *hostname*, so your DISPLAY is `:0.0` if you have only one screen on your system. Because this is the default, your system already knows that DISPLAY is `:0.0`, and you do not have to set it explicitly. You can learn more about X servers and displays in your X Window System documentation. See Chapter 19, "Using the GL in a Networked Environment," for information about how to share graphics across a network.

The *4Dwm* window manager supplied with your system provides a default appearance for windows and manages window operations. You can create X windows, GL windows, and *mixed-model applications*, which are either X windows that accept GL input, or GL windows that intermix X and GL subroutines. Creating, manipulating, and displaying windows are activities central to your task as a GL programmer. Working with GL windows is covered in depth in the *Graphics Library Windowing and Font Library Programming Guide*.

1.3 Programming in C

The C programming language provides the framework for developing GL programs. This guide assumes that you are comfortable writing C programs.

1.3.1 Using the ANSI C Standard

Ideally, GL programs are independent of the particular IRIS platform and the installed graphics hardware that is running the application. The best way to develop machine-independent code is to follow the ANSI C standard and to query the system about available hardware to establish its graphics performance capabilities. You can learn about the ANSI C standard in your C programming language documentation. To compile non-ANSI compliant code, use the **-cckr** flag to invoke the standard C compiler, as described in Section 1.3.2.

1.3.2 Compiling C Programs

To compile C programs, use the *cc* (C compiler) command:

```
% cc myprogram.c -lc_s -lgl_s -lm -o myprogram
```

The first two options in this list allow the same binary to run on all IRIS-4D Series systems:

-lc_s links with the shared C library.

-lgl_s links with the shared Graphics library.

-lm links with the math library.

-o defines the name of the output file.

Some other compile options that you might want to use are:

-cckr invokes the standard C compiler rather than ANSI C.

-lfm_s links with the IRIS shared Font Manager library.

-lsun links with the Network Information Service (NIS) to supply a hostname list for network-transparent GL applications.

1.4 GL Program Structure

Steps that you perform in a GL program include:

1. Querying the system for the availability of graphics resources.
2. Initializing the Graphics Library.
3. Calling GL subroutines to set and get global state attributes and to create and render graphics.
4. Exiting the Graphics Library.

The best way to learn about what a GL program contains is to look at a sample program. The sample program for this chapter is in the *ch01* directory of */usr/people/4Dgifts/examples/glpg*. Change directories (*cd*) to that directory now so you can follow along with the discussion.

This program is named *green.c*:

```
#include <gl/gl.h>

main()
{
    prefsize(400, 400);
    winopen("green");
    color(GREEN);
    clear();
    sleep(10);
    gexit();
    return 0;
}
```

Compile the program using the compile line:

```
% cc green.c -lgl_s -lc_s -o green
```

Run the sample program by typing:

```
% green
```

Move the mouse cursor to the location where you want the window to appear and click the left mouse button. A solid green window opens, remains visible for 10 seconds, and then disappears.

Look at the program in detail. The first line

```
#include <gl/gl.h>
```

includes all the standard definitions for graphics. It must be included in every graphics program. You must include *gl/gl.h* in *green.c* to get the definition of "GREEN" in the call to `color()`.

The subroutine

```
prefsize(400,400);
```

tells the window system that when a window is opened, it should be 400 pixels on a side. It doesn't create a window—it just establishes the initial size of the next window to be opened.

The call to `winopen()` actually creates the window and initializes the GL. Calling `color()` with an argument of `GREEN` sets the drawing color for subsequent operations to the value of the constant `GREEN`, defined in *gl/gl.h*. The call to `clear()` fills the window with the current drawing color. You set the drawing color for other operations in the same manner—that is, by calling `color()` with the color specification you wish to use. See Chapter 4, "Display and Color Modes," for more information. The call to `gexit()` closes the window and tells the system that the process is finished using graphics.

As you work through the rest of the chapters, run the sample programs, copy them, and modify their parameters to see what effects are possible. The next sections examine the basic elements of a GL program that are in the sample program.

1.4.1 Initializing the System

Initializing the Graphics Library means telling the system to set up the graphics software and hardware environment in which a program will run. Use `winopen()` to initialize the GL and open a graphics window to tell the system where to display the graphics output of your program.

`winopen()` initializes the hardware, allocates memory for symbol tables and display list objects, and sets up default values for global state attributes. `winopen()` must be called before you call most GL routines.

You can call these GL subroutines before you call `winopen()`, `ginit()`, or `gbegin`:

```
fudge
foreground
imakebackground
iconsize
keepaspect
maxsize
minsize
noborder
noport
prefposition
prefsize
stepunit
getgdesc
gversion
ismex
scrnselect
```

1.4.2 Getting Graphics Information

You can make your application more portable by including code that queries the system for its graphics capabilities before commencing with the program.

Querying the System for Graphics Resources

`getgdesc()` allows you to inquire about characteristics of the graphics system, such as screen size, number of bitplanes, and the existence of optional hardware such as *z*-buffer. `getgdesc()` returns a number that describes the hardware configuration specified by its parameter, *inquiry*. See the *getgdesc*(3G) man page for a complete list of parameters. `getgdesc()` returns a value of -1 if the *inquiry* is invalid, or if the specified hardware is not installed. You can call `getgdesc()` at any time, including before the first `winopen()`.

The values that `getgdesc()` returns are not affected by changes to software modes or software configuration.

Querying the System for Graphics Hardware and Software Versions

`gversion(`*v*`)` returns information about the current graphics hardware and the Graphics Library version.

The argument *v* is a pointer to a location into which `gversion()` copies a null-terminated string. Reserve at least 12 characters at this location. `gversion()` fills the buffer pointed to by *v* with a null-terminated string that specifies the graphics hardware type and the version number of the Graphics Library.

You can call `gversion()` before the first `winopen()`.

Caution: Using `gversion()` makes programs machine-specific. In almost all cases, `getgdesc()` is preferable to `gversion()`.

Table 1-1 lists the descriptors returned by `gversion()`. In the table, *m* and *n* represent the major and minor release numbers, respectively, of the IRIX software release to which the current Graphics Library belongs.

Graphics Type	String Returned
B or G	GL4D-*m.n*
GT or GTB	GL4DGT-*m.n*
GTX or GTXB	GL4DGTX-*m.n*
VGX	GL4DVGX-*m.n*
VGXT, Skywriter	GL4DVGXT-*m.n*
RealityEngine	GL4DRE-*m.n*
Personal IRIS	GL4DPI2-*m.n*
Personal IRIS with Turbo Graphics	GL4DPIT-*m.n*
Personal IRIS (early serial numbers)	GL4DPI-*m.n*
IRIS Indigo Entry	GLDLG-*m.n*
XS	GL4DXG-*m.n*
XS24	GL4DXG-*m.n*
Elan	GL4DXG-*m.n*

Table 1-1 System Types and Graphics Library Versions

Note: Personal IRIS units with early serial numbers do not support the complete Personal IRIS graphics functionality.

Setting Compatibility Modes

`glcompat()` gives control over details of the compatibility among systems. `glcompat()` controls two compatibility modes. The first, `GLC_OLDPOLYGON`, offers compatibility with old-style polygons, described in Chapter 2, "Drawing." The second, `GLC_ZRANGEMAP`, controls the state of *z*-range mapping mode, described in Chapter 8, "Hidden-Surface Removal."

Setting and Getting the Graphics Resource Configuration

The GL is a modal system—it maintains settings, or *modes*, that determine how graphics resources are set up. When you change certain modes, you need to call `gconfig()` to configure them. Table A-2 in Appendix A, "Scope of GL Subroutines," contains a notation in its third column that indicates which commands require a `gconfig()` in order to take effect. Call `gconfig()` following the group of all calls that require a `gconfig()`.

The current configuration can be read back with `getgconfig()`.

Table 1-2 lists the `getgconfig()` tokens and the resource they describe.

Token	Resource
`GC_BITS_CMODE`	Color index size
`GC_BITS_RED`	Red component size
`GC_BITS_GREEN`	Green component size
`GC_BITS_BLUE`	Blue component size
`GC_BITS_ALPHA`	Alpha component size
`GC_BITS_ZBUFFER`	z-buffer size
`GC_ZMIN`	Minimum depth value that can be stored in the z-buffer
`GC_ZMAX`	Maximum depth value that can be stored in the z-buffer
`GC_BITS_STENCIL`	Stencil size
`GC_BITS_ACBUF`	Accumulation buffer size
`GC_MS_SAMPLES`	Number of samples in multisample buffer

Table 1-2 Tokens for Graphics Resource Inquiries

Token	Resource
GC_BITS_MS_ZBUFFER	Multisample zbuffer size
GC_MS_ZMIN	Minimum depth value that can be stored in the multisample z-buffer
GC_MS_ZMAX	Maximum depth value that can be stored in the multisample z-buffer
GC_BITS_MS_STENCIL	Multisample stencil size
GC_STEREO	Stereoscopic buffer state
GC_DOUBLE	Display double buffer state

Table 1-2 (continued) Tokens for Graphics Resource Inquiries

Requests for GGC_ZSIZE, GGC_ACSIZE, and GGC_MSZSIZE return a negative size if the buffer is signed.

1.4.3 Global State Attributes

The *global state attributes* are options that specify information that the GL uses. Many of the GL subroutines allow you to change the values of these attributes. Unless you specify otherwise, the global state attributes use their default values. See Appendix B for the default values of the global state attributes.

1.4.4 Exiting the Graphics Environment

gexit() performs housekeeping functions associated with the termination of graphics programming, such as freeing memory used for GL data structures. Call gexit() as the last step in a GL program.