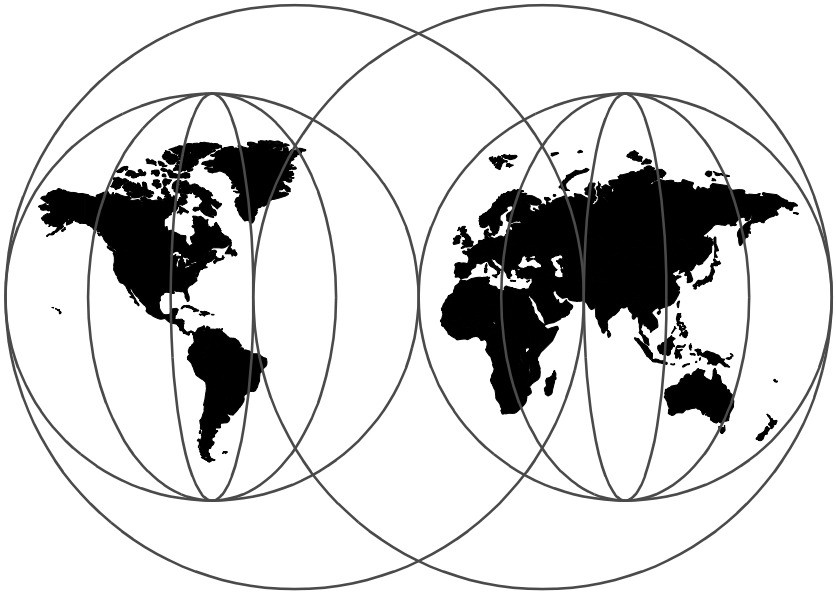
RS/6000 Graphics Handbook

Laurent Vanel, Mike Carline, Shigeo Murohashi



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5130-00

SG24-5130-00

International Technical Support Organization

RS/6000 Graphics Handbook

March 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 267.

First Edition (March 1999)

This edition applies to AIX Version 4.3.2, program number 5765-C34, PEX AND PHIGS for AIX Version 4.2, program number 5765-660 and OpenGL and GL3.2 for AIX V4.2, program number 5765-587.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This redbook summarizes the graphics capabilities of RS/6000 systems and discusses the various graphics adapters that can be installed, as well as the APIs available.

We introduce the reader to graphic hardware and explain the terms used in the announcement letter of graphics adapters. We describe graphics adapters for the RS/6000 systems, including withdrawn graphics adapters. We summarize the displays available to connect those graphics adapters as well as the cables that are needed to connect those displays with the corresponding adapters. We describe the various peripherals that can be connected to an RS/6000 system to enhance the user's interaction with the applications.

We describe the 2D APIs (X11, motif and CDE) and provide answers to frequently asked questions. We then describe the X Virtual Frame Buffer and softgraphics and the APIs that let the software take the place of the hardware.

Then we describe in detail each API, so that the reader can knowledgeably compare the four 3D APIs.

Finally, we discuss benchmark programs. We focus on the standard benchmark from the Graphics Performance Characterization Group. We also supply the latest results for all the IBM graphics adapters.

This redbook will be useful to sales people who have to propose a graphic configuration and must choose among many adapters, peripherals and software. This redbook can also be used as a reference for support people who need to know the technical characteristics of graphics adapters and which filesets to install to support various functions in a given environment.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Laurent Vanel is an AIX specialist at the International Technical Support Organization Austin Center. He is from Paris, France, where he joined IBM in February 1990, when the first RS/6000s were announced. Since then, he has provided AIX support to both field engineers and customers.

Mike Carline is a Senior I/T Specialist in Advanced Technical Support, Westlake, Texas. He has 19 years of experience in IBM. His areas of expertise include nine years providing technical support on AIX and the RS/6000 platform. He has taught many IBM classes on AIX, PSSP, and related subjects.

Shigeo Murohashi is an IT Specialist of IBM Japan. From 1990 to 1995, he was with IBM Tokyo Research Laboratory and studied computer graphics systems. He was one of initial members of development of Soft5080 product in 1992. Since 1996, he has been involved in RS/6000 product management and marketing and has supported CAD/CAM applications on AIX and NT. He has written extensively on 3D graphics APIs on AIX.

Thanks to the following people for their invaluable contributions to this project:

Diane Weires
Manager for PHIGS Development

Iliese Chelstowski
graPHIGS Development

Jeff Piaget
Graphics Performance Analyst

Jeanne Sparlin
AIXwindows Architect

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 283 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Part 1. Hardware

Chapter 1. Hardware Technology

This chapter is an overview of the concepts and elements which are the basis of graphic adapter technology. We introduce terms used in this book and explain the role of the various components you find on the latest IBM graphic adapter, the GXT3000P.

1.1 How Do You Build a Graphics Adapter?

The goal of any graphics adapter is to get the pixels that represent an image onto the screen for the user to view. The exact method for doing this varies widely among the different hardware vendors. Emphasis is placed on the particular aspects that differentiate their products by allowing them to perform certain tasks faster than others.

The problem for the user is in understanding what the various adapters are able to do, defining the user's needs, and which adapter is capable of satisfying those needs at a reasonable price. In this chapter, we attempt to provide you with some basic information regarding graphics functions and the hardware that supports them.

1.1.1 The Chips

The basic chip requirements are: an interface to the system bus, (bus interface), a buffer for storing the pixel information (frame buffer), and a method for converting the digital information to an analog signal that the display device utilizes, a Random Access Memory Digital Analog Converter (RAMDAC). When you add 3D capabilities to the adapter, you need additional chips to handle textures, lighting, transformations, stereographics, rasterization, and blitting.

1.1.2 Typical Graphics Adapter Functions

This section describes the adapter functions.

Alpha Buffer A fourth alpha channel for color control of a pixel is possible. It is used for controlling the first three channels. This merging of the fourth channel with the other three is often used to control the transparency or opacity of a pixel.

Antialiasing A method for smoothing out jagged lines on the screen.

Bit Planes	An individual bit of color information from each pixel stored in the frame buffer.
Bus Interface	Provides the connection between the system bus and the graphics adapter components.
Culling	Removal of pixel information that is not to be viewed from the rendering process.
Depth Cueing	Provides a perception of depth by rendering lines that are further away from viewer dimmer than closer lines.
Double Buffering	A method of dividing the available memory in half to improve performance. Half of the buffer is used to hold the currently displayed image while the next image is drawn into the other half of the buffer. After the next image is ready, the buffers are swapped, and the following image is drawn into the first half of the buffer.
Flat Shading	Calculating a single shade of color for each individual polygon.
Frame Buffer	Stores the pixel information for the display output.
Gamma Correction	A logarithmic assignment of color intensities to a lookup table to adjust the shading of objects to more closely match the human eye's perception.
Gouraud Shading	Calculating the shade of color for each pixel in a polygon based on the color of each of the polygon's vertices.
Hardware Colormaps	Hardware storage for color lookup tables. Since there are a rather small number of colors possible in the X server's colormap, hardware storage is used to hold multiple colormaps. This allows a colormap to be assigned to an individual window instead of to the entire screen. This helps to reduce the incidence of color flashing while changing focus between windows.
HLHSRemoval	Hidden Line/Hidden Surface Removal is used to ensure that objects that appear behind other objects are not rendered.
Luminance	This signal describes the amount of light in each pixel.

Motion Blur	The perception of motion blurring is achieved by storing previously rendered images in the accumulation buffer and mixing them with the current image on display.
Overlay Buffer	Storage for pixel information that is independent of the frame buffer data and is meant to appear on top of the frame buffer pixel data on the screen. Useful for weather maps and pop-up menus.
RAMDAC	Digital to analog convertor that also has the color lookup tables and the gamma correction table.
Rasterizer	Performs complex calculations for converting the polygon information from the application to pixel information for the display output. Some of the typical functions performed are blending, shading, texturing, and lighting.
Rendering	The final stage in the process of producing images for display on the screen.
RGB	Indicating the three channels for color control of a pixel: Red, Green and Blue.
Setup/BLIT unit	Processes blit commands and performs any necessary pixel scaling; performs some lighting calculations.
Stencil Buffer	Used to hold pixel information regarding areas that are not to be drawn. It acts like a typical cardboard stencil.
Stereo	The frame buffer is essentially split in half to provide for 12-bit RGB on a per-window basis. Utilizing special shutter-type glasses, a realistic 3D effect may be achieved by rendering a left-eye image into the top half of the buffer and a right-eye image into the bottom half and then drawing them interlaced onto the screen and synchronizing the glasses to the changes.
Texture Buffer	Stores images that can then be mapped onto a surface, like wrapping a box in patterned paper. Images are represented in collections of texels.
Underlay Buffer	Storage for pixel information that is independent of the frame buffer data and is meant to appear underneath the frame buffer pixel data on the

	screen. This data will only appear on the screen if the frame buffer pixel has a value of 0.
Utility Planes	A portion of the frame buffer used for utility functions. For example, they can be used to store clipping planes so that odd-shaped windows are supported with minimal performance degradation.
Video Scaling	Increasing or decreasing the size of an object without changing its position or orientation.
Window ID Buffer	Stores the X Window System Window ID information.
Z-Buffer	Used to store the depth information for the individual pixels to be displayed on the screen. Used in Hidden Surface/Hidden Line Removal.

1.1.3 A Look at the GXT3000P Design

Now that we have the basic graphics adapter information, let's take a look at the design of the GXT3000P Graphics Accelerator (shown in Figure 1).

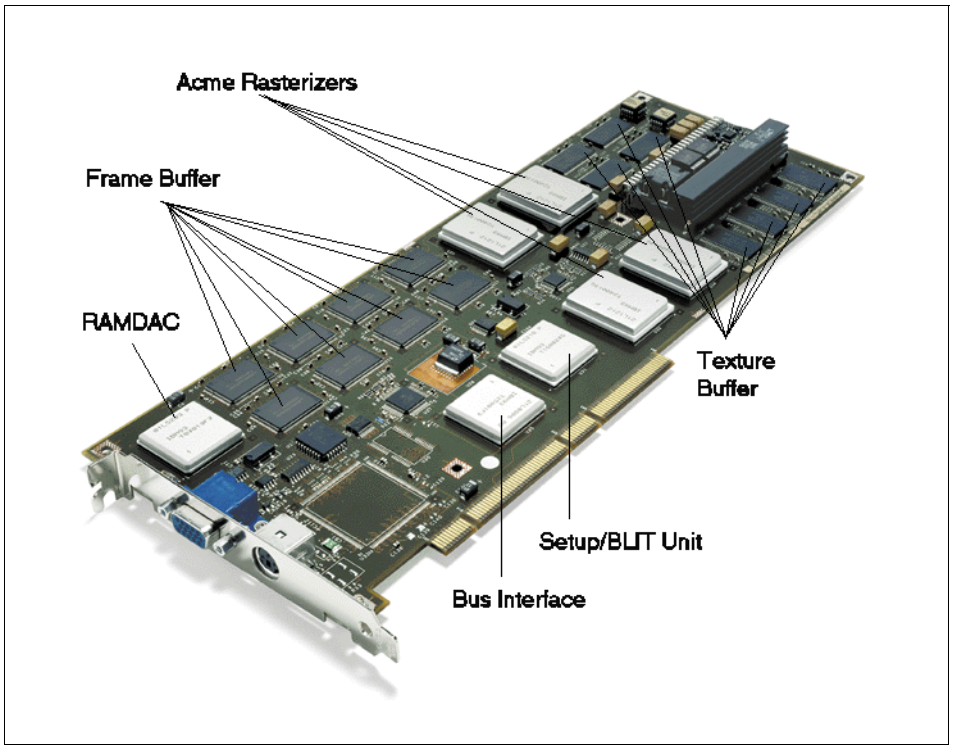


Figure 1. The IBM GXT3000P Graphics Accelerator (without Shields)

Bus Interface

The interface between the accelerator and the system bus provides for a 64-bit, PCI revision 2.1-compliant, connection. The chip has dual command input FIFOs, one for the current 3D API and the other for the 2D GUI. This minimizes the need for synchronization of the accelerator during context switches.

The interface also provides a high-function DMA (Direct Memory Access) controller that the APIs can exploit. This controller can retrieve commands from system memory and feed them to the 2D or 3D command FIFO. You can even embed commands for the DMA controller itself in the stream to cause it to automatically switch between the two FIFOs. The controller also performs sub-image DMA blits whereby it extracts a small image from within a larger one, striding across unneeded sections of the larger one as required. The DMA controller uses scatter-gather techniques to access system memory so that commands in memory or blitted images may be held in non-contiguous memory addresses.

There is also an asynchronous frame buffer access path that bypasses the rendering pipeline and allows the frame buffer to be read and written directly without a context swap.

Lighting and Blit Chip

The lighting and blit chip accepts commands from the bus interface chip. It processes blit commands by converting the incoming pixel data stream into one of the native frame buffer formats. It then performs any required scaling and distributes the data to the appropriate rasterizer chip.

This chip also accepts drawing commands and vertex data from the bus interface chip. The vertex data can contain pre-vertex material definitions and normals that are used to calculate the lit color of each vertex. The results from the lighting calculations are passed to the rasterizer chips as vertex colors. Unlike the blit commands and pixel data, drawing commands and their associated vertex data are sent to all four rasterizer chips at the same time.

Rasterizer Chips

In order to maximize the performance of the GXT3000P, the four rasterizer chips are designed to operate asynchronously during most operations. Each rasterizer chip drives one fourth of the frame buffer interleaved on a pixel-column by pixel-column basis. These chips contain most of the setup and interpolation logic for rasterizing the drawing primitives. Each chip also processes textures from its own copy of the texture memory so that there is

four times as much texture memory physically on the graphics accelerator as the amount seen by the user. They also contain memory controllers for the 3D-RAM and for the texture SDRAM. Pixel fragment processing is split between these chips and the 3D-RAM chips.

3D-RAM Chips

The 3D-RAMs are an intrinsic part of the rasterizer data path. They handle much of the OpenGL per-fragment operations as well as the pixel logic operations for the 2D GUI. The cache and Arithmetic and Logic Unit (ALU) inside the 3D-RAMs can be thought of as a special purpose SIMD (Single Instruction, Multiple Data) computer with the rasterizer chip acting as the tag RAM for the cache and as the controller for the data path.

Each of the 3D-RAM chips has a 256-bit wide path between its cache and the DRAM array. The sixteen 3D-RAMS thus have a total bus width of 4096-bits. This provides a much greater bandwidth to memory than conventional designs.

The frame buffer's pixels may be arranged in one of three ways to support different applications as shown in the table below.

Table 1. Frame Buffer Pixel Formats

Pixel Format	Color Buffer A				Color Buffer B				Stencil/Depth		Auxiliary		
32-bit color	A	R	G	B	A	R	G	B	S	Z	W	O	U
16-bit color	$a_l a_r$	$r_l r_r$	$g_l g_r$	$b_l b_r$	$a_l a_r$	$r_l r_r$	$g_l g_r$	$b_l b_r$	S	Z	W	O	U
8-bit index	x	x	x	I	x	x	x	I	S	Z	W	O	U

ARGB = 8-bits each of Alpha, Red, Green, and Blue
S = 8-bit stencil, Z = 24-bit depth, W = 8-bit Window ID
O = 8-bit Overlay, U = 16-bit Utility
$a_l a_r r_l r_r g_l g_r b_l b_r$ = Stereo: 4-bits each of left/right Alpha, Red, Green, Blue
I = 8-bit Index, x = reserved

The 32-bit color mode provides for double-buffered, true color applications. The 16-bit color mode organizes each pixel into four buffers - front/back and left-eye/right-eye for use with stereo-in-a-window animation. The 8-bit index mode is also double buffered and supports OpenGL's color index mode applications.

Palette Digital Analog Converter (DAC) Chip

The palette DAC is a customized version of the IBM RGB640 palette DAC modified to accommodate the 3D-RAM interface and to support advanced features used in engineering workstations. It has a 2Kx8 palette RAM for each color band. This RAM can be used as eight independent 256-entry palettes or partially subdivided into smaller 64-entry palettes for even more independent palettes.

The palette DAC also has a gamma correction RAM table that can be activated on a pixel-by-pixel basis. The window ID Byte that is scanned in from the frame buffer is allocated as five bits to control the primary buffer and the remaining three to control the overlay buffer. Thus, there can be up to 32 primary and 8 overlay window IDs concurrently on the screen. Each window ID can independently select a pixel format and palette, enable gamma correction, and select from A/B animation or left-eye/right-eye buffers. In typical usage, many windows will share a common window ID, but a few will make use of unique IDs. The large number of independent window IDs and palettes minimizes the annoyance of color flashing when several applications that use large color palettes are run concurrently.

Other features of this chip include a 10-bit monotonic DAC that is capable of up to 160 MHz operation in this application and of even higher speeds in other applications. The chip also provides both crosshair and 64x64 sprite hardware cursors.

The DAC output is provided to the monitor through a standard DDC-2B interface. Screen sizes of 1280x1024 at up to 85 Hz and 1024x768 at up to 120 Hz are supported. The 1024x768 at 120 Hz mode is well suited to stereo display with a minimum of flicker. The stereo output is provided through a VESA-standard mini-DIN interface.

With all this discussion about the IBM chip's roles in the graphics accelerator, we should add that they are implemented in IBM's 5sa (0.27 micron) and SA12 (0.18 micron) silicon technologies. The seven IBM chips on the GXT3000P Graphics Accelerator contain over forty million transistors and are mounted in IBM's flip-chip Ceramic Ball Grid Array (CBGA) packages providing them unparalleled performance and density.

1.2 Different Classes of Graphics Adapters

There are three classes of graphics adapters. Their attributes can be roughly described as follows:

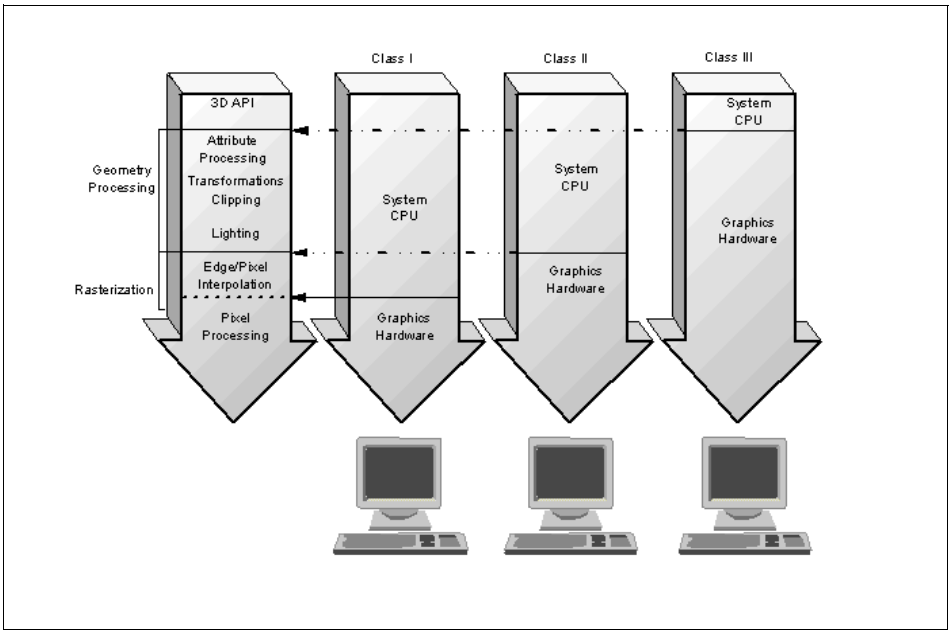


Figure 2. The Three Classes of Graphic Adapters

Class I 2D & Entry-Level 3D
All computing and rasterization work is performed on the CPU. 3D capabilities are achieved through use of the Softgraphics product.

Class II Mid-Range 3D
These adapters provide hardware acceleration support for antialiasing, texture mapping and rasterization. Geometry processing (lighting & transformations) is done on the CPU.

Class III High-End 3D
Hardware acceleration for rasterization and geometry processing is off-loaded to Digital Signal Processors (DSPs) on the adapter which, in turn, make use of custom rasterization chips.

Chapter 2. The Graphics Adapters

This chapter summarizes the hardware information available on the graphic adapters that can be plugged into an risc system/6000. This information not only contains the pure hardware details, such has bus type and number of bits for the colormap, but also elements such as the name of the fileset containing the device driver for each adapter (when they are still supported). This information comes from the announcement letter.

2.1 The Ancient Adapters

The adapters discussed in this paragraph are not supported anymore; so why mention them? Because sometimes you discover an old box in a dark room, and you wonder what adapters are in this box and what can I do with these? This section tries to answer those questions.

2.1.1 Grayscale Graphics Display Adapter

The Grayscale Graphics Display is a monochrome graphics adapter for the deskside systems. It supports a single 4-bit frame buffer and allows the display of 16 simultaneous shades of gray from a palette of 256 shades of gray. It attaches to the RS/6000 workstation through a single Micro Channel slot and supports a display resolution of 1280 x 1024. This adapter is suitable for applications, such as desktop publishing, CASE, monochrome drafting, and network management.

Here are some of its characteristics:

- Frame buffer: single 4-bit (one 4-bit buffer)
  - API support: Xlib, Graphic Kernel System (GKS), Display PostScript, graPHIGS
  - Maximum adapters per system: Two
  - Options: none
  - Monitor support: 8508 - 1280x1024 at 67 Hz

Its feature code was #2760, and it was withdrawn in the U.S. on June 17, 1994.

You can physically recognize the graphic adapters by the white sticker you find on the metallic frame. The sticker for this adapter shows: 1-2.

2.1.2 Color Graphics Display Adapter

The Color Graphics Display Adapter is an entry 2D color graphics adapter for the deskside systems. It supports a single 8-bit frame buffer and allows the display of 256 simultaneous colors from a palette of approximately 16.7 million colors. The Color Graphics Display Adapter supports a display resolution of 1280 x 1024 and attaches to the RS/6000 workstation via a single Micro Channel slot. It is suitable for 2D graphics applications such as: business graphics/desktop publishing, 2D geographic mapping, entry 2D mechanical drafting, and entry electrical CAD.

Here are some of its characteristics:

- Frame buffer: single 8-bit (one 8-bit buffer)
- API support: Xlib, GKS, Display PostScript, graPHIGS, PEXlib
- Maximum adapters per system: Two.
- Options: none.
- Monitor support:
  - 6091-016 - 1280 x 1024 at 60Hz
  - 6091-19i - 1280 x 1024 at 60 Hz
  - 6091-023 - 1280 x 1024 at 60 Hz

Its feature code was #2770 and it was withdrawn in the U.S., on January 6, 1995.

You can physically recognize the graphic adapters by the white sticker that you can find on the metallic frame. The sticker for this adapter shows: 1-1.

2.1.3 POWER Gt1

The POWER Gt1 is a low-priced 1-bit graphics frame buffer that attaches directly to the local processor I/O bus. It supports resolutions of 1280 x 1024 or 1024 x 768 and does not require a Micro Channel slot.

Here are some of its characteristics:

- Hardware assist.
  - 64 x 64 hardware cursor
  - Rectangle clip function
  - Window offset
  - XY pixel addressing

- Meets ISO 9241 Part 3 at 1024x768 resolution.
- Maximum of one POWER Gt1 per system can be installed.
- Option: The POWER Gt1 VRAM Upgrade is an optional feature that is installed on a POWER Gt1 or POWER Gt1b adapter. This upgrade is factory- or field-installable. Up to two upgrades can be installed on a POWER Gt1 or POWER Gt1b. Either adapter can be upgraded to a 4-bit graphics (16 gray shades or colors) frame buffer with the installation of a single VRAM Upgrade. The POWER Gt1 or POWER Gt1b is upgraded to an 8-bit graphics (256 colors) frame buffer when a second VRAM Upgrade is installed.

Its feature code was #4208, and it was withdrawn in the U.S. on September 19, 1995.

Note

The POWER Gt1b is FCC Class B-certified and supports both 1280x1024 and 1024x768 displays. The base configuration is 1-bit per pixel. VRAM options allow the POWER Gt1b to be configured with 4-bits or 8-bits per pixel. Note: Since the bandwidth of the POWER Gt1b is limited to a maximum of 111 MHz pixel frequency, there may be a very slight difference in fine graphics detail, though most users will not be able to see the difference. The POWER Gt1b is functionally identical to the POWER Gt1 EXCEPT that the POWER Gt1b has been certified by the FCC for Class B devices and therefore is more suitable for home use. Performance and price remain the same for both adapters. The POWER Gt1 should be chosen over the POWER Gt1b whenever home use is not an issue.

The Gt1B (#2803) was withdrawn in the U.S. on September 19, 1995.

2.1.4 POWER Gt1x

The POWER Gt1x is a low-priced 8-bit, 256 color, graphics adapter that attaches directly to the local processor I/O bus. It does not require a Micro Channel slot.

Here are some of its characteristics:

- 2D hardware drawing support
  - Points
  - Lines
  - Filled triangles

- Filled quadrilaterals
- Color-expanded bit block transfer
- 16x16 pattern fill
- Logical operations
- Plane mask

- 64x64 three-color programmable hardware cursor
- 60 to 77 Hz refresh modes
- Supports a variety of displays
- Meets ISO 9241 Part 3 at 1280x1024 resolution on IBM POWERdisplays 16S, 16 and 19
- Maximum of one POWER Gt1x per system can be installed

Its feature code was #4207, and it was withdrawn in the U.S. on September 19, 1995.

2.1.5 IBM E15-type Graphic

The IBM E15-type graphics is a low-cost, medium performance DRAM-based graphics solution for systems employing a PCI system bus. The E15-type graphics employs the 64-bit S3 Vision864 graphical user interface (GUI) accelerator, and the 16-bit 135 MHz S3 SDAC. It comes standard with a 2 MB frame buffer, providing for up to 1280x1024 resolution, and integrates on the system board. It uses a standard 15-pin D-shell connector.

This adapter supports the following resolution:

- 1280x1024x8 bits per pixel at 60 Hz refresh
- 1024x768x8 bits per pixel at up to 76 Hz refresh
- 800x600x8 bits per pixel at up to 76 Hz refresh
- 640x480x8 bits per pixel at up to 72 Hz refresh

2.1.6 S15 Graphics Adapter

The IBM S15 Graphics Adapter is a high-performance VRAM-based PCI graphics adapter with integrated video coprocessor for use as a premium graphics solution. The S15 comes in a 2 MB fixed version.

Here are some of its characteristics:

- The S15 supports the standard 15-pin D-shell (DB-15) monitor cable.
- Monitors supported: The S15 supports multisync monitors having at least a 64 KHz horizontal scan capability. Default display modes are structured

around the factory presets for the IBM Personal Computer family of monitors.

Its feature code was #2657, and it was withdrawn in the U.S. on January 21, 1997.

2.1.7 The (#2839) - POWER GXT110P Graphics Adapter - PCI

The IBM POWER GXT110P Graphics Adapter is an 8-bit, 256-color adapter that attaches through one 32-bit PCI slot. It is designed as an entry-level 2D graphics adapter.

Here are some of its characteristics:

- 32-bit PCI bus interface
- 2 MB DRAM
- Pattern fill support
- Rectangular and non-rectangular clipping
- 256 colors from a palette of 256,000 colors
- 1 hardware colormap
- Monitor support
  - Resolution: 640x480, 800x600, 1024x768, 1280x1024
  - Refresh rates: 60 to 85 Hz

Its feature code was #2839, and it was withdrawn in the U.S. on August 14, 1998.

2.1.8 POWER Gt3

The POWER Gt3 is a high-performance 2D color graphics adapter for deskside systems. It has a single 8-bit frame buffer and allows the display of 256 simultaneous colors from a palette of approximately 16.7 million colors. The POWER Gt3 supports monitors with a display resolution of 1280x1024. The POWER Gt3 attaches through a single Micro Channel slot and is suitable for 2D graphics applications, such as drafting, 2D mechanical CAD, electronic CAD, civil/architectural engineering, 2D mapping applications, and CASE.

Here are some of its characteristics:

- Frame buffer: single 8-bit (one 8-bit buffer)

- API support: Xlib, GKS, Display PostScript, graPHIGS, PEXlib
- Maximum adapters per system: Two
- Options: none
- Monitor support:
  - 6091-016 - 1280x1024 at 60 Hz
  - 6091-019 - 1280x1024 at 60 Hz
  - 6091-023 - 1280x1024 at 60 Hz

Its feature code was #2777, and it was withdrawn in the U.S. on April 30, 1993.

You can physically recognize the graphic adapters by the white sticker you can find on the metallic frame. The sticker for this adapter shows: 1-6.

2.1.9 POWER Gt3i

The POWER Gt3i is a high-performance 2D color graphics adapter for deskside systems. It is also the lowest priced color graphics adapter available for these systems. It has a single 8-bit frame buffer and allows the display of 256 simultaneous colors from a palette of approximately 16.7 million colors. The POWER Gt3i supports monitors with a display resolution of 1280x1024 including monitors that comply with Part 3 of the ISO 9241 ergonomic standard. These ergonomic displays provide users with improved viewing and physical comfort, minimized reflections and sharper images. The POWER Gt3i attaches through a single Micro Channel slot and is suitable for 2D graphics applications, such as drafting, 2D mechanical CAD, electronic CAD, civil/architectural engineering, 2D mapping applications, and CASE.

Here are some of its characteristics:

- Frame buffer: single 8-bit (one 8-bit buffer)
- API support: Xlib, GKS, Display PostScript, graPHIGS, PEXlib
- Maximum adapters per system: Two
- Options: none.
- Monitor support:
  - 6091-016 - 1280x1024 at 60 Hz, 77 Hz
  - 6091-19i - 1280x1024 at 60 Hz, 77 Hz
  - 6091-019 - 1280x1024 at 60 Hz
  - 6091-023 - 1280x1024 at 60 Hz

Its feature code was #2768, and it was withdrawn in the U.S.,on September 19, 1995.

You can physically recognize the graphic adapters by the white sticker you can find on the metallic frame. The sticker for this adapter shows: 1-9.

2.1.10 POWER GXT150L

This graphics adapter is designed for superior 2D performance in an AIXwindows 2D environment. The POWER GXT150L Graphics Adapter is an 8-bit single buffer, 256 color, graphics adapter that attaches to the PowerPC local bus graphics expansion slot and does not use a Micro Channel slot. When used in conjunction with the AIXwindows 3D feature and Softgraphics, it provides cost-effective 3D performance.

The POWER GXT150L graphics adapter provides 1280x1024, 1152x900 and 1024x768 resolution support, three color palettes and hardware window support.

Here are some of its characteristics:

- Frame buffer: single 8-bit (one 8-bit buffer)
- Z-buffer: none
- Advanced hardware functions: Three color palettes, hardware window support
- API support: Xlib, GKS, Display PostScript, OpenGL, graPHIGS, PEXlib
- Maximum adapters per system: 1
- Options: none

Its feature code was #2660, and it was withdrawn in the U.S. on September 24, 1997.

2.1.11 POWER GXT155L

This 8-bit double-buffered 2D adapter is designed to provide enhanced performance for the Softgraphics implementation of the OpenGL Application Programming Interface (API). It provides customers with a cost-effective, entry-level platform for their OpenGL applications.

The POWER GXT155L attaches directly to the PowerPC local bus in models 41W and 41T and does not require the use of any Micro Channel slots. It supports display resolutions of 1280x1024, 1152x900 and 1024x768. It is

available as an upgrade from the POWER GXT150L on the 42W and 42T. Softgraphics OpenGL is provided through optional software support.

Here are some of its characteristics:

- Frame buffer: Double 8-bit (two 8-bit buffers)
- Z-buffer: none (Z-buffer is implemented by Softgraphics)
- Advanced hardware functions: T
- Three color palettes, hardware window support
- API support: Xlib, GKS, Display PostScript, OpenGL, graPHIGS, PEXlib
- Maximum adapters per system: 1
- Options: none

Its feature code was #2665, and it was withdrawn in the U.S. on July 18, 1997.

2.1.12 High-Performance 3D Color Graphics Processors

The High-Performance 8- or 24-bit Color Graphics Processor provides 3D color graphics capability for deskside systems. The 8-bit adapter (#2780) supports a single 8-bit frame buffer that allows 256 simultaneous colors from a palette of approximately 16.7 million colors. The 24-bit adapter (#2781) supports a single 24-bit frame buffer that allows approximately 16.7 million simultaneous colors from a palette of approximately 16.7 million colors. This 24-bit frame buffer can be partitioned to support double-buffering. A 24-bit Z-buffer option that assists with hidden line and surface removal is also available. Both of these adapters support a display resolution of 1280x1024. The High-Performance 8- and 24-bit Color Graphics Processors attach to the 7013 through two MicroChannel slots. These adapters are suitable for 3D applications, such as 3D wireframe CAD (mechanical), solid modeling, architectural rendering, structural design, and seismic processing.

Here are some of its characteristics:

- Frame buffer: single 8-bit (#2780); single 24-bit (#2781)
- Z-buffer: 24-bit (optional)
- API support: Xlib, GKS, Display PostScript, Graphics Library (GL), graPHIGS
- Maximum adapters per system: Two
- Options:

    - 24-bit Z-buffer Solid Rendering Option (#2782, No Longer Available). This option provides hidden line, hidden surface removal for 3D applications. It is a daughter card which attaches to the existing two-card set and does not require an additional Micro Channel slot.
    - 8-bit to 24-bit Upgrade (#2783) for (#2780) only, turns it into a 24-bit graphic adapter.
- Monitor support:
    - 6091-016 - 1280x1024 at 60 Hz
    - 6091-019 - 1280x1024 at 60 Hz
    - 6091-023 - 1280x1024 at 60 Hz
    - 5081-016 - 1280x1024 at 60 Hz
    - 5081-016 - 1280x1024 at 60 Hz

Their feature codes were #2780 for the 8-bit frame and #2781 for the 24-bit frame buffer, and it was withdrawn in the U.S., date unknown.

You can physically recognize the graphic adapters by the white sticker you can find on the metallic frame. The sticker for this adapter shows: 1-3.

2.1.13 POWER Gt4e

The POWER Gt4e is a single card which provides 3D graphics capabilities for deskside systems. It has a double 8-bit frame buffer which allows the display of 256 simultaneous colors from a palette of approximately 16.7 million colors (24-bit color is NOT available on this adapter). The adapter also comes standard with a 24-bit Z-buffer which assists with hidden line and surface removal. The POWER Gt4e supports monitors with a display resolution of 1280x1024, including monitors that comply with Part 3 of the ISO 9241 ergonomic standard. These ergonomic displays provide users with improved viewing and physical comfort, minimized reflections and sharper images. The POWER Gt4e attaches to the workstation through a single Micro Channel slot and is suitable for applications, such as mechanical CAD, engineering analysis, architectural design, and geographical mapping.

Here are some of its characteristics:

- Frame buffer: Double 8-bit (two 8-bit buffers)
- Z-buffer: 24-bit
- Overlay planes: Two
- Advanced hardware functions: Dithering, depth cueing, antialiased lines, Gouraud shading, locallighting, Non-Uniform Rational B-Spline (NURBS)

- API support: Xlib, GKS, Display PostScript, GL, graPHIGS, PEXlib
- Maximum adapters per system: Two
- Options: none
- Monitor support:
  - 6091-016 - 1280x1024 at 60 Hz, 77 Hz
  - 6091-19i - 1280x1024 at 60 Hz, 77 Hz
  - 6091-019 - 1280x1024 at 60 Hz
  - 6091-023 - 1280x1024 at 60 Hz

Its feature code was #2776, and it was withdrawn in the U.S. on October 25, 1996.

You can physically recognize the graphic adapters by the white sticker you can find on the metallic frame. The sticker for this adapter shows: 1-9.

2.1.14 POWER Gt4 and POWER Gt4x

The POWER Gt4 and POWER Gt4x adapters provide low to mid-range 3D graphics capabilities, respectively, for deskside models in addition to maximum configuration flexibility. Both adapters provide double 8- or 24-bit frame buffers that allow 256 or approximately 16.7 million simultaneous colors, respectively, from a palette of approximately 16.7 million colors. These adapters also include a standard 24-bit Z-buffer that assists with hidden-line and surface removal. Performance differentiates the functionally equivalent POWER Gt4 from the POWER Gt4x. The POWER Gt4x has four additional processors that boost its performance to meet the requirements of more demanding graphics applications. The POWER Gt4 and POWER Gt4x support monitors with a display resolution of 1280x1024, including monitors that comply with Part 3 of the ISO 9241 ergonomic standard. These ergonomic displays provide users with improved viewing and physical comfort, minimized reflections and sharper images. The POWER Gt4 and Gt4x 8-bit adapters both require two Micro Channel slots while the 24-bit configuration for both requires an additional Micro Channel slot (total of three slots). The POWER Gt4 is suitable for the same types of applications as the POWER Gt4e. The POWER Gt4x is suitable for a larger range of applications including 3D mechanical CAD, aerospace applications, reservoir simulations, molecular modeling, and geographical mapping.

Here are some of its characteristics:

- Frame buffer: POWER Gt4 8-bit (#2795) - Double 8-bit (two 8-bit buffers); POWER Gt4 24-bit (#2796) - Double 24-bit (two 24-bit buffers); POWER

Gt4x 8-bit (#2790) - Double 8-bit (two 8-bit buffers); POWER Gt4x 24-bit (#2791) - Double 24-bit (two 24-bit buffers)

- Z-buffer: 24-bit
- Overlay planes: Two
- Advanced hardware functions: Dithering (8-bit), depth cueing, antialiased lines, Gouraud shading, local lighting, NURBS
- API support: Xlib, GKS, Display PostScript, GL, graPHIGS, PEXlib
- Maximum adapters per system: One
- Options:
  - The POWER Gt4 8-bit to 24-bit Upgrade (#2792) - Provides an upgrade to a double 24-bit frame buffer for both the 8-bit POWER Gt4 and 8-bit POWER Gt4x. This upgrade requires one additional Micro Channel slot.
  - POWER Gt4 Performance Upgrade (#2794) - Boosts the performance of the Gt4 to that of the Gt4x and requires no additional Micro Channel slots. This option is a daughter card which attaches to the adapter.
- Monitor support:
  - 6091-016 - 1280x1024 at 60Hz, 77 Hz
  - 6091-19i - 1280x1024 at 60 Hz, 77 Hz
  - 6091-019 - 1280x1024 at 60 Hz
  - 6091-023 - 1280x1024 at 60 Hz
  - 5081-016 - 1280x1024 at 60 Hz

Their feature codes were #2790, #2791, #2795, #2796 and they were withdrawn in the U.S. on December 21, 1993.

You can physically recognize the graphic adapters by the white sticker you can find on the metallic frame. The sticker for this adapter shows: 1-5.

Note

The 24-bit POWER Gt4i graphics adapter (feature #2713) and the 8-bit and 24-bit POWER Gt4xi graphics adapters (features #2711 and 2712, described below) were announced in September 1993 and replace the POWER Gt4 and POWER Gt4x (feature #2796, 2790 and 2791). The enhanced adapters primarily improve the 3D performance of the original adapters. These were withdrawn in the U.S. on October 25, 1996.

2.1.15 POWER GTO Accelerator

The 7235 POWER GTO subsystem is a high-performance 3D color graphics subsystem for deskside systems. The GTO has its own product number (7235) and cannot be ordered as a feature. It is referred to as a subsystem because it is an external device with its own power supply. The POWER GTO is available in two different models, Model 01i which provides a double 8-bit frame buffer and Model 02i which provides a double 24-bit frame buffer. These models allow 256 or approximately 16.7 million simultaneous colors, respectively, from a palette of approximately 16.7 million. Model 02i additionally provides an enhanced shading processor (Model 01i does not have a shading processor) and a standard 24-bit Z-buffer for the support of hidden line and surface removal. Model 02i also provides subpixel antialiasing of lines in hardware. The POWER GTO models support monitors with a display resolution of 1280x1024, including monitors that comply with Part 3 of the ISO 9241 ergonomic standard. These ergonomic displays provide users with improved viewing and physical comfort, minimized reflections and sharper images. The POWER GTO is an external subsystem that attaches to the RS/6000 workstation through the POWER GTO Accelerator (Feature #4350) card which occupies one Micro Channel slot. This card is ordered as a feature of the RS/6000 and is MANDATORY for support of the 7235 POWER GTO product. The POWER GTO is suitable for high-performance applications such as high-end CAD design and analysis, entertainment graphics, scientific visualization, and aerospace applications.

Here are some of its characteristics:

- Frame buffer:
  - Model 01i - Double 8-bit (two 8-bit buffers)
  - Model 02i - Double 24-bit (two 24-bit buffers)
- Z-buffer: 24-bit (Model 02i).
- Overlay planes: Four
- Advanced hardware functions: depth cueing (Model 02i), antialiased lines, Gouraud shading (Model 02i), local lighting (Model 02i), NURBS
- API support: Xlib, GKS, Display PostScript, GL, graPHIGS, PEXlib
- Maximum adapters per system: One
- Options: Model conversions for all of the POWER GTO Models (001, 002, 01i, 02i) only available through RPQ
- Monitor support:
  - 6091-016 - 1280x1024 at 60 Hz, 77 Hz

- 6091-19i - 1280x1024 at 60 Hz, 77 Hz
- 6091-019 - 1280x1024 at 60 Hz
- 6091-023 - 1280x1024 at 60 Hz

Its feature code was #4350, and it was withdrawn in the U.S. on November 4, 1994.

You can physically recognize the graphic adapters by the white sticker that you can find on the metallic frame. The sticker for this adapter shows: 1-4.

2.2 The Current Graphic Adapters

This section discusses in more detail the supported graphic adapters for the RS/6000.

2.2.1 The MVP Power Multi-Monitor Graphics Accelerator

The MVP Power Multi-Monitor Graphics Accelerator supports two displays on a single adapter thus conserving an expansion slot. This allows for the viewing of multiple data sets simultaneously and is especially beneficial for applications such as security trading, financial analysis and software engineering. The unique design of this adapter lets a single mouse move seamlessly between the two monitors for easy operation.

Some of its characteristics are:

- 2 MB EDO RAM
- RAMDAC frequency 135 MHz
- Resolution: from 640x480x8 to 1280x1024x8

2.2.2 The GXT120 Family

The GXT120P provides entry-level 2D graphics for the PCI RS/6000 servers and workstations. It is an excellent choice for business graphics or for attaching a graphical control display to one of these PCI servers.

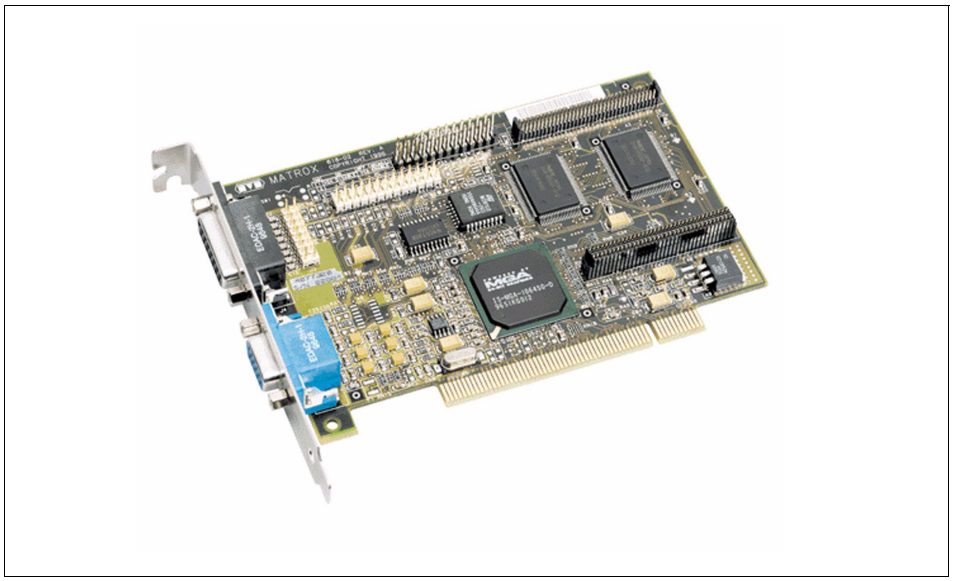


Figure 3. The IBM GXT120P Graphics Accelerator

Here are some of their characteristics:

- 2 MB EDO DRAM
- 32-bit PCI bus interface
- 8-bit color support
- Screen resolutions:
  - 640x400 at 60 - 85 Hz vertical refresh
  - 800x600 at 60 - 85 Hz vertical refresh
  - 1024x768 at 60 - 85 Hz vertical refresh
  - 1280x1024 at 60 - 75 Hz vertical refresh

Its feature code is #2837.

2.2.3 The GXT150M

The GXT150M is a Micro Channel adapter. It provides high-performance and is supported by Softgraphics for RS/6000 workstations. It is standard on the RS/6000 Model 397.

Here are some of its characteristics :

- Bus Master

- Hardware acceleration with a 32-bit Graphics Dedicated Processor for:
  - Points
  - Lines
  - Arc
  - Circle
  - Rectangles
  - Font
  - Bit block transfer
- Pattern fill
- Hardware rectangular clipping.
- Hardware cursor (crosshair cursor and image cursor-64 X 64)
- Monitor support
- Resolution: 1280x024
- Refresh rates: 60 to 77 Hz
- 256 colors from a palette of 16 million
- Meets ISO 9241 Part 3 on appropriate displays

Its feature code is #2650.

2.2.4 The GXT250P

The GXT250P provides the top 2D performance for PCI accelerators. It is an 8-bit adapter that is capable of displaying up to 256 concurrent colors from a palette of approximately 16.7 million colors. The double buffer 8-bit is ideal for entry to midrange OpenGL applications through Softgraphics. The 24-bit capability provides true color.

Here are some of its characteristics:

- 64-bit or 32-bit PCI bus interface
- 3 MB of VRAM
- Hardware acceleration for points, lines, triangles, rectangles, quadrilaterals, bit block transfer
- Rectangular and non-rectangular clipping
- Up to 256 colors from a palette of 16.7 million
- Hardware window support (4 window ID bits)

- Dithering
- Three hardware color maps
- Monitor support
  - Resolution: 1024x768 or 1280x1024 (limitation to 1024x768 with double buffer)
  - Refresh rates: 60 to 85 Hz
  - Capable of driving monitors that meet ISO 9241 Part 3 Specification

Its feature code is #2851.

2.2.5 The GXT255P

The GXT255P provides performance similar to the GXT250P with the added advantage of either double buffer 8-bit or single buffer 24-bit color support.

Here are some of its characteristics:

- 64-bit or 32-bit PCI bus interface
- 8 MB of VRAM
- Hardware acceleration for points, lines, triangles, rectangles, quadrilaterals, bit block transfer
- Pattern fill support
- Rectangular and non-rectangular clipping
- Up to 16.7 million colors from a palette of 16.7 million
- Hardware window support (4 window ID bits)
- Dithering
- Three hardware color maps
- Monitor support
  - Resolution: 1024x768 or 1280x1024
  - Refresh rates: 60 to 85 Hz
  - Capable of driving monitors that meet ISO 9241 Part 3 specifications

Its feature code is #2852.

2.2.6 The GXT550P

The POWER GXT550P graphics accelerator offers exceptional and affordable 3D graphics for the PCI-based RS/6000 43P Models 140, 150 and

240 and for Model F40 workstations. Coupled with IBM's implementation of many of the industry's most popular application programming interfaces (APIs), these accelerators are an excellent fit for today's most demanding 3D graphics applications, such as mechanical design and analysis.

The GXT550P further establishes IBM as a leader in providing price/performance solutions for a wide range of demanding customer application requirements.

The GXT550P combines 3D rasterization with the strength of the latest PowerPC microprocessors. This rasterization technology provides customers with a cost-effective solution that delivers excellent 3D graphics acceleration that scales with the PowerPC processor performance of the RS/6000 workstation.

The GXT550P provides hardware acceleration for OpenGL and PHIGS. IBM is a leader in the industry by offering native support of OpenGL and PHIGS on the same accelerator. These adapters accelerate advanced 3D graphics functions, such as Gouraud shading, antialiasing, hidden surface removal, depth-cueing and transparency. This helps enable your 3D applications to run more quickly and interactively. For OpenGL, texture maps up to 1024x1024, and a 64-bit accumulation buffer are also supported through software.

The GXT550P offers highly flexible frame buffers that can be dynamically configured to provide a broad set of color and feature options. When using OpenGL and PHIGS, the GXT550P supports 8-bit, 12-bit, and 24-bit double-buffered color and also includes 8-bits of double-buffered alpha buffers for more realistic transparency control. It provides 8-bit overlay buffers, which enhance the speed of the Graphical User Interface (GUI), 8-bit stencil buffers, and 24-bit Z-buffer for hidden surface removal operations.

The GXT550P graphics accelerator is a 32-bit PCI card that plugs into a single PCI slot. It supports display resolutions of up to 1280x1024 and 1024x768 and refresh rates from 60 Hz to 85 Hz, including refresh rates that comply with the ISO 9241 Part 3 ergonomic standard. When used with other compliant components, including monitors, this standard provides improved viewing, reduced flicker, reduced reflection, and sharper characters. These features, coupled with IBM's powerful RS/6000 43P Models 140, 150, 240, and Model F40 workstations, make the POWER GXT550P an ideal solution for demanding 3D application needs.

Its feature code is #2855.

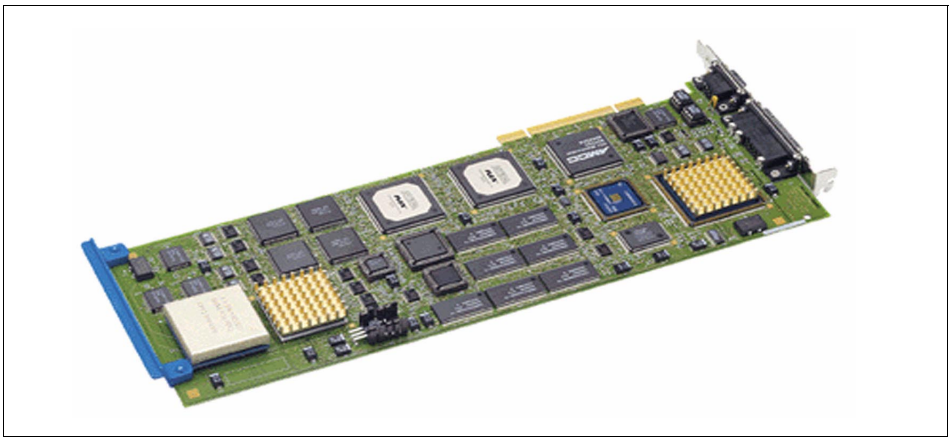


Figure 4. The IBM GXT550 Graphics Accelerator

2.2.7 The GXT800 Family

The POWER GXT800M graphics accelerator provides an enhanced level of powerful and advanced 3D graphics for the Micro Channel-based RS/6000 Models 397 desktop and 595 deskside workstations. Every GXT800M features hardware-accelerated texture mapping for generating more realistic images at interactive speeds.

The GXT800M is particularly well suited for users requiring powerful graphics performance in systems designed for floating point-intensive applications. When coupled with IBM's implementation of many of the industry's most popular Application Programming Interfaces (APIs), this accelerator is an excellent fit for today's most demanding 3D graphics applications in fields such as mechanical design and analysis, petroleum exploration and production, molecular modeling, and scientific research. The GXT800M further establishes IBM as a leader in providing advanced graphics solutions for a wide range of demanding customer application requirements.

The POWER GXT800P graphics accelerator provides a new level of powerful and advanced 3D graphics for the PCI-based RS/6000 43P Models 140 and 240, Model F40, and Model F50 systems. The GXT800P offers an optional configuration that accelerates texture mapping in hardware, generating more realistic images at interactive speeds.

The GXT800P is particularly well suited for users requiring the next step in performance above that provided by the GXT550P graphics accelerator. Coupled with IBM's implementation of many of the industry's most popular Application Programming Interfaces (APIs), this accelerator is an excellent fit

for today's most demanding 3D graphics applications in fields such as mechanical design and analysis, petroleum exploration and production, molecular modeling, and scientific research. The GXT800P further establishes IBM as a leader in providing advanced graphics solutions for a wide range of demanding customer application requirements.

The GXT800P utilizes a 5-way rendering engine that processes advanced 3D graphics in parallel, providing the throughput required to work with complex geometries at interactive speeds. When this is combined with the latest PowerPC microprocessors in supported systems, the GXT800P 3D graphics accelerator delivers increasing performance as you upgrade to more powerful systems.

The feature code is #2853 for the GXT800P, #2859 for the GXT800P with texture, and #2850 for the GXT800M.

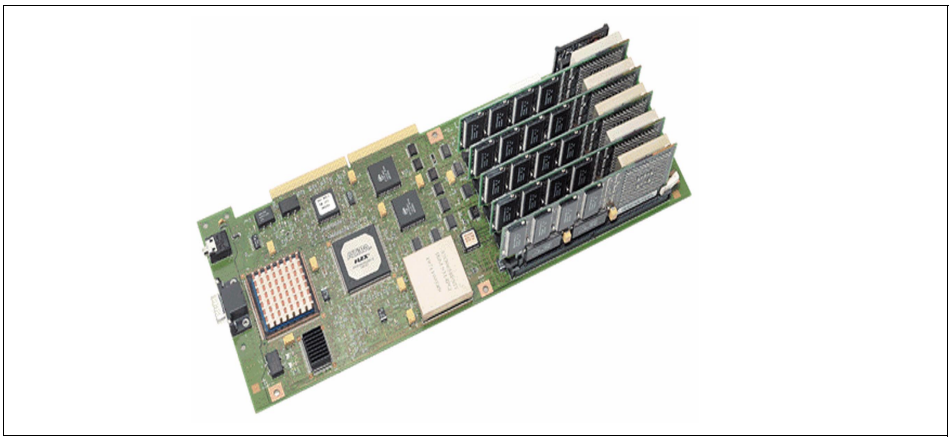


Figure 5. The IBM GXT800P Graphics Accelerator

2.2.8 The GXT3000P

The POWER GXT3000P graphics accelerator is all about productivity. The faster the combination of graphics accelerator and RS/6000 workstation, the more work engineers and scientists can accomplish. The GXT3000P is designed to deliver outstanding performance at a fraction of the usual cost. It is a win/win situation when using the GXT3000P for visualizing very large MCAD models and advanced simulations.

The POWER GXT3000P graphics accelerator marks a speed and feature breakthrough for the most demanding design and visualization solutions such as MCAD, MCAE, petroleum and scientific applications. This is because of its

native support for PHIGS and Open GL application programming interfaces (APIs). Attached to the 64-bit 43P Model 260, the GXT3000P delivers leading GPC benchmark and application graphics performance. Combining the GXT3000P and the 43P Model 150 produces winning price/performance, with over two times the GXT800PT/43P Model 140 MCAD graphics speed.

Features that help ensure the transfer of visual information include:

- Hardware setup for faster shading and blending
- Hardware lighting both front and back sides of up to eight infinitely positioned light sources for OpenGL compatibility
- Hardware 3D texturing with mipmaps and level-of-detail filtering
- Hardware luminance filtering and separate specular lighting, applied after texturing in hardware
- Hardware support of advanced API rendering modes
- Stereo in a window without obscuring other windows
- 16 MB of on-board texture RAM for more texture

The pacesetting capabilities of the GXT3000P are made possible by innovative electronics design throughout. The PCI bus interface provides a high transfer rate of 3D graphics commands and data from the CPU. A lighting and setup circuit prepares the 3D objects for rasterization on the fly, funneling the elements to be rendered into four-way parallel rasterization units that match full-instruction microprocessors in complexity. Here, blending, shading, texturing, and lighting effects are added, and both sides of a polygon can be simultaneously textured and lighted. The elements are realized as colored pixels into lightning-fast 3D-RAM graphics memory and then double-buffered for seamless animation on displays of resolutions up to 1280x1024 at 85 Hz refresh rate. The result is a rich and content-filled display of the most complex 3D models.

The GXT3000P was designed to support the features of PHIGS and OpenGL 1.2 (including hardware texturing), plus many OpenGL extensions. The four raster engines on the GXT3000P accept setup data through the APIs in the form of polygons to be rendered. The rasterization subsystem processes 3D graphics in parallel, reaching drawing speeds many times those of previous IBM graphics accelerators.

The GXT3000P graphics accelerator supports display resolutions of up to 1280x1024 and 1024x768, and refresh rates from 60 Hz to 85 Hz, including refresh rates that comply with the ISO 9241 Part 3 ergonomic standard when used with other ISO 9241 Part 3-compliant components, including monitors.

This standard provides improved viewing, reduced flicker, reduced reflections, and sharper characters. The GXT3000P is the right choice for great graphics performance for getting the job done fast. Whether designing a new avionics system in 3D or animating a flyby of the entire fuselage, the GXT3000P has the graphics speed you demand.

When you need features and throughput to get your product out to the market sooner, use the GXT3000P.

Its feature code is #2825.

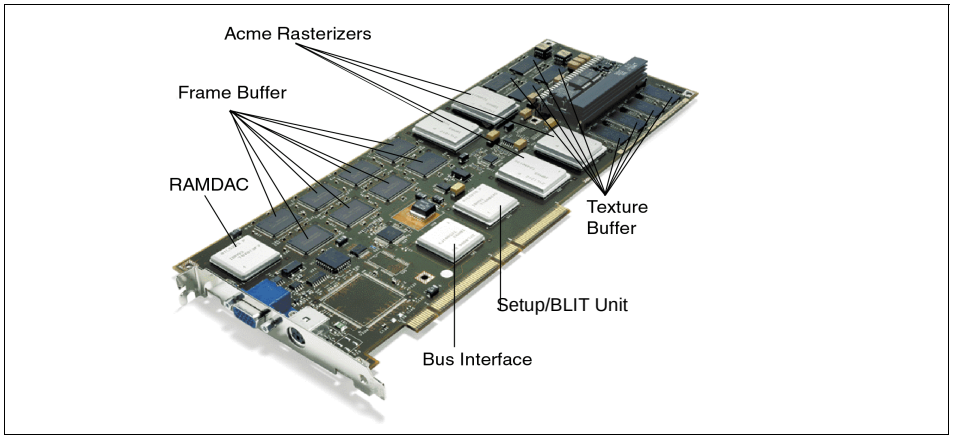


Figure 6. The IBM GXT3000P Graphics Accelerator

2.2.9 Device Drivers

This table summarizes the device drivers used by each adapter. Use the `lslpp` command to check if the correct device driver is installed on your system.

Table 2. Device Information for Current Graphics Accelerators

Adapter	Development Codename	Class	Device Name	Files Included in Install
GXT110	Brushy	I	iga0	devices.pci.3353c088.rte devices.pci.3353c088.X11 devices.pci.3353c088.diag
MVP		I	iga*	devices.pci.33531188.rte devices.pci.33531188.X11 devices.pci.33531188.diag devices.pci.33531188.com

Adapter	Development Codename	Class	Device Name	Files Included in Install
GXT120P	Sagebrush	I	mga0	devices.pci.2b101a05.rte devices.pci.2b101a05.X11 devices.pci.2b101a05.diag
GXT150M	Neptune	I	nep0	devices.mca.8f9a.rte devices.mca.8f9a.X11 devices.mca.8f9a.diag devices.mca.8f9a.ucode
GXT250P GXT255P	SkyBlue	I	b10	devices.pci.14103c00.rte devices.pci.14103c00.X11 devices.pci.14103c00.diag
GXT550P	Mint	II	mint0	devices.pci.14105400.rte devices.pci.14105400.X11 devices.pci.1415400.diag
GXT800M	SuperMint	II	smint0	devices.mca.8f61.rte devices.mca.8f61.X11 devices.mca.8f61.diag
GXT800P	SuperMint	II	smint0 smintdm0	devices.pci.14105e00.rte devices.pci.14105e00.X11 devices.pci.14105e00.diag
GXT3000P	Sierra	II+	mtn0	devices.pci.14108e00.rte devices.pci.14108e00.X11 devices.pci.14108e00.diag

2.2.10 Properties of the Graphic Adapters

This table summarizes the properties of the graphic adapters:

Table 3. Standard Graphics Features

Adapter	Max. # of Colors	# of Color Tables	Supports Stereo	Maximum Resolution Supported	Gamma Correction Tables
GXT120P	256	1	No	1280x1024	No
GXT150M	256	2	No	1280x1024	No
GXT250P	256	3	No	1280x1024	No
GXT255P	16.7M	3	No	1280x1024	No
GXT550P	16.7M	4	Yes	1280x1024	No

Adapter	Max. # of Colors	# of Color Tables	Supports Stereo	Maximum Resolution Supported	Gamma Correction Tables
GXT800M	16.7 Million	4	Yes	1280x1024	No
GXT800P	16.7 Million	4	Yes	1280x1024	No
GXT3000P	16.7 Million	8	Yes	1280x1024	Yes

2.2.11 Buffer Configuration

This table summarizes the existence and size of frame, underlay, overlay, alpha, stencil, Z-buffer, utility or window ID buffers on the graphic adapters:

Table 4. Buffer Configuration

Adapter	# of Bit Planes	Frame Buffer	Z Buffer	Overlay Buffer	Stencil Buffer	Window ID	Alpha Buffer	Utility Buffer
GXT120P	8	8						
GXT250P	8	8						
GXT255P	24	24						
GXT550P	112	24+24	24	8	8	8	8+8	
GXT800M	112	24+24	24	8	8	8	8+8	
GXT800P	112	24+24	24	8	8	8	8+8	
GXT3000P	128	24+24	24	8	8	8	8+8	16

2.2.12 Advanced 3D Functionalities

This table summarizes the advanced 3D functions of the graphic adapters:

Table 5. Advanced 3D Functionality Supported in Hardware

Adapter	Hidden Line	Anti-aliasing	Gouraud Shading	Depth Cueing	Texture Mapping	Transparencies	Blur	Lighting	Video Scaling	16-bit Color
GXT550P	Y	Y	Y	Y	Y	Y	N	N	N	N
GXT800M	Y	Y	Y	Y	Y	Y	Y	N	N	N
GXT800P	Y	Y	Y	Y	O	Y	Y	N	N	N
GXT3000P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Y = Yes, O = Optional attachment required										

2.2.13 Limitations

This table summarizes the connectivity and maximum number of adapters per system and some possible limitations:

Table 6. Number of Slots/Max Number of Adapters

System	150M	800M	MVP	110P	120P	250P	255P	550P	800P	3000P
7012-397	1/1	3/1[1]								
7013-J50	1/1									
7013-595	1/1	3/1								
7015-R50										
7017-S70					1/1					
7017-S7A					1/1					
7024-E30				1/2		1/2				
7025-F40			1/1[1]	1/1[1]	1/2	1/2	1/2	1/1	3/1	
7025-F50				1/1[3]	1/2				4/1[1]	
7026-H50					1/2					
7043-140			1/1[2]	1/2[4]		1/2[4]	1/2[4]	1/1[2]	3/1[2]	
7043-150					1/4[6]	1/4[6]	1/4[6]	1/1[5]		2/1[1]
7043-240			1/1	1/2	1/3	1/2	1/2	1/1[2]	3/1[2]	
7043-260					1/4[6]	1/4[6]	1/4[6]			1/1 [2]

[1]Must be located in slot 3.
[2]Must be located in slot 2.
[3]Must be located in slot 3,4,5.
[4]Must be located in slot 1,2.
[5]Must be located in slot 2,3.
[6]Must be located in slot 1,2,3,4,5.

Please refer to the *PCI Adapter Placement Reference,* SA38-0538, for important information regarding permissible and optimum slot placement of your PCI graphics adapters.

You should also be aware of the interaction with other PCI adapters in the same bus with your graphics adapters. A particularly important point is the performance degradation that may be seen by placing a 32-bit adapter in the same bus as the 64-bit GXT3000P adapter.

Chapter 3. Displays and Cables

This chapter describes the possible displays you can connect to the graphic adapters discussed in Chapter 2, “The Graphics Adapters” on page 11.

3.1 The Supported Displays

This section lists the displays supported on the RS/6000 systems. For each of these displays, it gives the viewable size, the supported resolution, maximum refresh rate, and as the aperture grille. Further information for the displays can be found at the following URL:

`http://srvteam.greenock.uk.ibm.com/hardware/displays/manual/index.html`

3.1.1 The IBM P72 Color Monitor

The P72 monitor is a 17-inch Trinitron CRT with a viewable image size of 16.0 inches (407 mm), incorporating a 0.25 mm aperture grille for bright, high-definition images. A maximum horizontal frequency of 85 KHz provides reduced flicker operation at an optimum 1024x768 pels at up to 85 Hz non-interlaced with a maximum addressability of 1280x1024 pels.

The IBM P72 color monitor comes in two colors: stealth black and pearl white.

3.1.2 The IBM P92 Color Monitor

The P92 monitor is a 19-inch Trinitron CRT with a viewable image size of 17.9 inches (456 mm), incorporating a 0.25 - 0.27 mm aperture grille for bright, high-definition images. A maximum horizontal frequency of 94 KHz provides reduced flicker operation at an optimum 1280x1024 pels at up to 85 Hz non-interlaced with a maximum addressability of 1600x1200 pels.

The IBM P92 color monitor comes in two colors: stealth black and pearl white.

3.1.3 The IBM P202 Color Monitor

The P202 monitor is a 21-inch Trinitron CRT with a viewable image size of 19.8 inches (503 mm), incorporating a 0.25 - 0.27 mm aperture grille for bright, high-definition images. A maximum horizontal frequency of 107 KHz provides reduced flicker operation at an optimum 1600x1200 pels at up to 85 Hz non-interlaced with a maximum addressability of 1600x1200 pels.

The IBM P202 color monitor comes in two colors: stealth black and pearl white.

3.2 The Cables

Now that we have seen the various graphic adapters and displays available for the RS/6000 systems, the last difficult step is to connect one with the other, and this is done through cables. The correct cables depends on the connectors present on the adapter and the display. This section describes the various cables, and Table 7 on page 39 summarizes the cables needed for each pair of adapter and display. Let's start with the list of available cables:

- The 6091 Attachment Cable:

  This display cable is required for attachment of an IBM 5081 or IBM 6091 display. The cable is 1.8 meters (6 ft) in length. Its feature code is #4217.

- The 15-pin-d-shell to 13W3 Display Cable:

  This cable attaches monitors with a 13W3 connector to a graphics adapter with a 15-pin-d-shell connector. Its feature code is #4235.

- The 13W3 to 15-pin DDC/ID Bits Switchable Display Cable:

  This cable attaches displays with 13W3 video connectors to graphics adapters with 15-pin connectors. There is a switch in the cable that is used to enable or disable Plug and Play. When disabled, ID bits are presented to the attaching system. Its feature code is #4237.

- The DDC 15-pin to 13W3 Display Cable:

  This converter cable attaches displays with a 13W3 connector that either are DDC (Display Data Channel) capable or do not need ID bits to a graphics adapter with a 15-pin D-shell connector. Its feature code is #4238.

- The 15-pin D-shell cable to the 13W3 connector on the adapter. Its feature code is #4213.

- The 15-D Adapter to 3BNC ID Cable:

  This cable attaches displays with a 3BNC video connector to graphics adapters with a 15-pin D-shell connector. Its feature code is #4239.

Table 7. Cable Feature Number Required per Monitor/Adapter Configuration

	GXT110P GXT120P	GXT250P GXT255P	GXT800M GXT800P	GXT3000P
G52	Display	NS	NS	NS
P70/P200	4238	4238	4238	4238
P201	4237	4237	4237	4237
P72/P92/P202	Display	Display	Display	Display
6091-19I	NS	4239	4239	4217
PowerDisplay	4217	4239	4239	4217

Table 8. Cable Feature Number Required per Monitor/Adapter Configuration

	GXT150M	GXT500P GXT550P	GXT1000	7250
G52	NS	NS	NS	NS
P50 P72	4213	4213	NS	NS
P70 P200	4234	4240	4234	4234
P201	4234	4241	4234	4234
6091-19I	4214	4219	3252	3252
POWERdisplay 17	4214	4219	3253	3252
POWERdisplay 20	4214	4219	3253	3253

Chapter 4. Graphics Peripherals

This chapter briefly examines peripheral devices that enhance the user/application interaction in the manipulation and preprogramming of graphic images and applications.

4.1 Mice

There are two pointing devices available for RS/6000 systems. The announcement letter regarding those devices is #198-242.

- 3-Button Mouse

  The 3-Button Mouse is a dynamic tracking pointing device which utilizes opto-mechanical technology. Its resolution is 320 DPI, and a 2.74 meter (9 ft.) attachment cord is included. Its feature code is #6041

- 3-Button Mouse, Black

  It is a 3-button dynamic tracking pointing device. It utilizes opto-mechanical technology, has a resolution of 320 dpi, a 3-meter cable, is stealth black color, and has standard 6-pin mini-DIN connection.

  Its feature code is #8741.

4.2 The Keyboards

The keyboards you connect to the RS/6000 systems are grouped under two families.

The Quiet Touch Keyboard

- Feature codes from #6600 to #6640
- Quiet and soft-touch key depression
- 3-meter keyboard cable
- Removable wrist rest
- Pearl white color
- Standard 6-pin mini-DIN connection
- 37 languages supported
- 101 to 106 keys, depending on language specified
- Optional keyboard cable with speaker (Feature #6599)
- Announcement letter #197-279

The Quiet Touch Keyboard, blac:

- Feature code from #8700 to #8740
- Quiet and soft-touch key depression
- 3-meter keyboard cable
- Removable wrist rest
- Stealth Black color
- Standard 6-pin mini-DIN connection
- 37 languages supported
- 101 to 106 keys, depending on language specified
- Euro currency sign support on appropriate national language versions
- Announcement letter is #198-242

Here is a list of the supported languages:

- US English, #103P
- French, #189
- Italian, #142
- German/Austrian, #129
- UK English, #166
- Spanish, #172
- Japanese, #194
- Brazilian Portuguese, #275
- Canadian French, #058
- Belgian/UK-Flemish, #120
- Belgian/French, #120
- Swedish/Finnish, #153
- Danish, #159
- Bulgarian, #442
- Swiss, French/German, #150
- Norwegian, #155
- Dutch, #143
- Portuguese, #163
- Greek, #319
- Hebrew, #212
- Hungarian, #208
- Icelandic, #197
- Polish, #214
- Romanian, #446
- Slovakian, #245
- Czech, #243

- Turkish, #179
- Turkish, #440
- LA Spanish, #171
- Arabic, #238
- Serbian-Cyrillic, #118
- Korean, #413
- Chinese/US, #467
- French Canadian, #445
- Thailand, #191
- Russian, #443
- Croatian, #105
- US English ISO9995, #103P(EMEA)

4.3 The Tablets

The IBM 6093 CursorPad and Tablet are digitizing devices that are used for data input and can be attached to the RS/6000 series of workstations. The announcement letter for those devices is #197-278.

The following table summarizes the main properties of the three tablet models:

Table 9. Main Properties for the Tablet Models

Model	Programmable Resolution	Tablet Size	4-Button Cursor	6-Button Cursor	3-Button Pen
011	1.279 Lines per inch	Small	Yes	Yes	No
012	1.279 Lines per inch	Large	Yes	Yes	No
021	2.540 Lines per inch	Small	Yes	No	Yes

A more detailed description for each model is included in the following section.

4.3.1 The 6093-011 Model

The 6093 Model 011 CursorPad is a small cursor pad that can be programmed with a resolution of up to 1,279 lines per inch. There is a programmable button function, which allows customers to adapt the four-button cursor as well as the six button cursor to their own design of button

configuration and identification. With appropriate features, it may be used in place of the 5083 Model 021 CursorPad and Model 022 Tablet.

Its features include:

One selectable feature from:

- Feature #1511 Four Button Cursor
- Feature #1512 Six Button Cursor
- Feature #6351 Two Button Stylus (pen) with a tip switch and two side buttons

Optional features include:

- Feature #4015 RS/6000 Attach Kit
- Feature #4030 Serial Port Cable Kit for attachment to 25-pin connectors
- Feature #4035 Serial Port Cable Kit for attachment to 9-pin connectors

4.3.2 The 6093-012 Model

The 6093 Model 012 Tablet is essentially the same as the Model 011. The tablet measures approximately 15"x16.5" as opposed to the 8"x10.5" measurements of the Model 011.

Its features include:

One selectable from:

- Feature #1511 Four Button Cursor
- Feature #1512 Six Button Cursor
- Feature #6351 Stylus (pen) with a tip switch and two side buttons

Optional features include:

- Feature #4015 RS/6000 Attach Kit
- Feature #4030 Serial Port Cable Kit for attachment to 25-pin connectors.
- Feature #4035 Serial Port Cable Kit for attachment to 9-pin connectors.

4.3.3 The 6093-021 Model

The 6093 Model 021 Tablet input device is a state-of-the-art device to ease the manipulation of computer graphics images. This electromagnetic tablet, with cordless and batteryless, four-button cursor or three-button pen, and a 4x5 work area provides better accuracy (+/- 0.025 inches) and higher

resolution capability (2540 lines per inch, Model 011 = 1280). The 6093 Model 021 Tablet is capable of higher data rates (200+ points per second, Model 011 = 100) and has better connectivity (fewer cabling options).

The 6093 Model 021 Tablet is a follow-on product to the 6093 Model 011 Tablet. The Model 021 Tablet is 8x10.5 inches, with a 4x5 inch active area.

The Model 021 is only supported on AIX Version 4.1.5, 4.2.1, 4.3, or later. It is functionally compatible to the Model 011, such as a cursor with a four-button physical layout, and an interface having 8-pin mini-DIN and microcode compatibility. The Tablet has an attached (non-removable) cable, with an 8-pin mini-DIN connector for attachment to the system tablet port.

Its features include:

One selectable feature from:

- Feature #1511 Four Button Cursor
- Feature #6351 Stylus (pen) with a tip switch and two side buttons

Optional features include:

- Feature #4030 Serial Port Cable Kit for attachment to 25-pin connectors.
- Feature #5999 Stealth Black Color Selection for Covers: selects the stealth black color for covers. If this feature is not selected, the default cover color is pearl white. This feature is not available on Models 011 and 012.

4.3.4 Additional Features

There are additional features that can be connected or used with these tablets. Here is a detailed list:

- Four Button Cursor:

  This feature (#1511, P/N 74F3131) allows the customer to have four different functional options at his/her fingertips. The cursor is functionally the same as the IBM 5083 cursor. The cursor has a targeting sight to accurately pick the points as required by the user.

- Six Button Cursor:

  This feature (#1512, P/N 74F3132) allows the customer to have six different functional options at his/her fingertips. The button configuration can be changed by the user to the most advantageous and comfortable for him. The cursor has a targeting sight to accurately pick the points as required by the user.

- 5086 Attach Cable Kit:

  This feature (#3888, P/N 74F3355) contains the cable for attachment to the 5086 Graphics Processor and the appropriate publications. The cable is 1.19m in length.

- 6095 Attach Cable Kit:

  This feature (#3898, P/N 74F3356) contains the cable for attachment to the 6095 Graphics Processor and the appropriate publications. The cable is1.19m in length.

- RS/6000 Attach Kit:

  This feature (#4015, P/N 74F3357) contains the cable for attachment to the RS/6000 and the appropriate publication. The cable is 2.13 meters long.

  Without this feature, the 6093 CursorPad Model 011 cannot be attached to the RS/6000 processors.

- PC, PC/XT PS/2 Attach Kit (25-Pin):

  This feature (#4030, P/N 74F3358) contains the cable for attachment to the 25-pin PC, PC/XT, and PS/2 Serial Port and the appropriate publications.

  Also included is a 3.5 inch diskette and appropriate power supply (default by country code). The cable is 2.43m in length.

- PC/XT, PC/AT, PS/2 Attach Kit (9-pin):

  This feature (#4035, P/N 74F3368) contains the cable for attachment to the 9-pin connector on the Dual-Sync Adapter for the PC/AT and the PS/2 and the appropriate publications. Also included is a 3.5 inch diskette and the appropriate power supply (default by country). The cable is 2.43m in length.

- Two Button Stylus:

  This feature (#6351, P/N 74F3133) is a hand-held, pen-like, manual input device that is available with a nylon tipped cartridge. The Stylus assembly houses a transmitting coil as well as a dome-type switch (tip switch) to sense if the Stylus is being pressed against the tablet surface.

4.3.5 Configuring the 6093 Tablet

This section now describes how to install and configure a tablet to an RS/6000 system.

1. Attach the graphics tablet to either the tablet port or to the serial ports of your machine. The tablet port is not available on all machines. When a tablet port is present, it is located next to the mouse and keyboard ports. On some systems, the tablet port is labeled with a T. If the graphics tablet is attached to a serial port, you may need to add a tty to the system. To add a tty:
    1. Enter `smit devices`:
    2. Select **TTY**.
    3. Select **Add a TTY**.
    4. Select the serial port to which the graphics tablet is connected.
    5. Install the following device-dependent software package for the graphics tablet. Its name is AIXwindows Graphics Input Adapter Software, and the file is devices.mca.edd5
    6. Also install the co-requisite software for the AIXwindows Serial Graphics Input Adapter. The name of the file is devices.serial.gio.

4.4 The Dials

The 6094 dials Model 010 is an image-manipulation device that allows the user to manually input scalar values in order to affect the visual image displayed on the monitor. The dials are a separate desktop unit consisting of eight potentiometers (commonly used for Pan, Zoom, and Rotation of 2D and 3D figures). The dials may be programmed to provide specific functions through the application.

The following publication is shipped with the product. Additional copies are available immediately by ordering *IBM 6094-010 Dials Option Instructions,* GA23-2404.

4.4.1 Attachment of the Dials to an RS/6000 System

The 6094 dials may be attached to and used with a variety of IBM and non-IBM systems configurations. The usual way to attach the dials to an RS6000 is through a serial line and to use the 6091 display to provide the power supply.

> **Important**
>
> The POWER displays 17 and 20 do not offer video redrive or auxiliary power as did the 6091 displays.

If a Micro Channel expansion slot is available in the RS/6000, you can order the Graphics Input Device Adapter (feature code 2810) and appropriate signal cables (feature code 2811). The dials and LPFK each need a signal cable.

Since the Graphics Input Device Adapter supplies power, no additional power supplies or auxiliary Power cables are needed.

If a Micro Channel expansion slot is not available in the RS/6000, separate power supplies are needed. For the 6094-010 dials and the 6094-020 LPFKs please order from the following feature codes (order one per peripheral).

- Feature #4063 100 volt power supply
- Feature #4064 120 volt power supply
- Feature #4065 220/240 volt power supply

4.4.2 Setting Up the 6094-010 on a Workstation

This section describes how to install and configure the dials on the serial port of your system.

1. Connect all the equipment:
   - Dials unit
   - Plug-in power supply
   - Cable that can accept power cord
2. Plug into serial port on workstation.
3. Use SMIT to add a tty:
   1. Choose parent from the list corresponding to port used.
   2. Port number is s1 or s2.
   3. Terminal type is dumb.
   4. Disable login.
   5. Default /etc/getty.
4. Verify the correct configuration with the 5080 emulator:
   - With the parameter `-ttyd ttyDialNum` for example `-ttyl 0`

   or
   - Add the following X resource:

     ```
     Soft5080.ttyDial: 1
     ```

4.5 The Lighted Program Function Keyboard (LPFK)

The 6094 Lighted Program Function Keyboard (LPFK) Model 020 is an application aid device that allows the user to preprogram application functions to be initiated by striking either a single lighted key or a combination of lighted keys. The LPFK is a separate desktop unit consisting of 32 keytops in which bright and easily viewed indicators are imbedded. The keyboard indicator lights can be turned on and off under processor or host application control.

The following publication is shipped with the product: *IBM 6094-020 LPFK Options Instructions,* GA23-2403. Additional copies are available and orderable.

4.5.1 Attachment of the LPFK to an RS/6000 System

The LPFK can be attached to an RS/6000 through a special card or through one of the standard serial ports. The serial port option does not use one of the card slots (this is an important consideration for desktop RS/6000s), but does have some drawbacks. The serial port lighted PF keys draw their power from the 6091 display. If the display is powered-off, the lighted PF keys need to be redefined to the system before they can be used.

To attach the LPFK using the serial port, you must order Serial Attach feature #4060 for the lighted PF Keys and Serial Attach Cable #4061.

If you use the display to power your serially connected dials and LPFKs, you can power down the display without powering down the workstation. However, you must power down the display for at least 20 seconds before powering it up. After the display is powered up, it will take at least four seconds for the dials/LPFKs to be reinitialized. During this period, input events are ignored. If, after this period, the dials/LPFKs do not appear to work, power down the display again, wait at least 20 seconds, and then power up the display. The dials/LPFKs should now work.

If a Micro Channel expansion slot is available in the RS/6000, you can order the Graphics Input Device Adapter (feature code 2810) and appropriate signal cables (feature code 2811). The dials and LPFK each need a signal cable.

4.5.1.1 Setting Up the 6094-020 on a Workstation:

You will need:

- The LPFK unit
- The plug-in power supply

- The cable that can accept power cord and plug into serial port on workstation

The first step is to put the hardware together; the LPFK should light when the DIN cable is plugged in last. You may need small black cable on the 5xx workstation to get to the 25 pin D-shell.

Use SMIT to add a tty:

- Choose the parent from the list corresponding to the port you plugged the device into.
- Select the port number (it can be either s1 or s2).
- The terminal type is dumb.
- Disable login.
- Default **/etc/getty.**

Start the 5080 emulator with the parameter `-ttyl ttyLpfkNum` for example `-ttyl 0` (for tty0 or 1 for tty1) or add the following X resource:

```
Soft5080.ttyLpfk: 0
```

The LPFK should light when moved into the emulator window.

4.5.2 Additional Features

This section describes additional features that can be connected or used with the 6094-010 dials or 6094-020 LPFK.

- (#4000) - Attachment Kit for PAC or 6095 Graphics Processor:

  A feature on the 6094-010 dials that is required when attaching to the Peripheral Adapter Connector (PAC - #4200) or the IBM 6095 Graphics Processor. The kit consists of a single attachment cable.

- (#4015) - RS/6000 Workstation Attachment Kit:

  This feature allows the 6094-010 dials and the 6094-020 LPFK to be attached to an RS/6000 workstation. The kit consists of a single attachment cable.

  Attributes required: The RS/6000 workstation must already have the Graphics Application Input Adapter installed.

- (#4020) - RS/6000 43P Series, PC-PS/2 (9) Attachment Kit:

  This feature allows the 6094-010 dials and the 6094-020 LPFK to be attached to D serial port interfaces on IBM PC, PC-XT, PS/2 computers or 43P Series (7248-100/120/132 and 7043-140/240) and other RS/6000 models announced after September 1996. The kit consists of an attaching

cable, a power supply, and a 3.5-inch diagnostic diskette. The attaching cable part number is different on each of the products.

Limitations: The number of dials and LPFKs attachable at the same time to any model of the IBM PC or PS/2 is directly related to the number of Asynchronous Adapters installed to accommodate them. The 43P Series machines use two 9-pin serial ports.

NOTE: #9525 must be ordered at the time of the original order placement if a 5.25-inch diagnostic diskette is required. There are no specific codes for power supplies; the power supply is based upon the country code of order origin. There is no default option.

- (#4025) - PC-PS/2 (25) Attachment Kit:

  This feature allows the 6094-010 dials to be attached to the PC-AT or PS/2 computer. The kit consists of an attaching cable, a power supply, and a 3.5-inch diagnostic diskette. The attaching cable part number is different on each of the products.

  Limitations: The number of dials attachable at the same time to any model of the IBM PC or PS/2 is directly related to the number of Dual Async or Serial Parallel Adapters installed to accommodate them.

  NOTE: #9525 must be ordered at the time of the original order placement if a 5.25-inch diagnostic diskette is required. There are no specific codes for power supplies; the power supply is based upon the country code of order origin. There is no default option.

- (#4060) - 6094/RS/6000 Serial Attachment:

  This feature allows attachment of the 6094 dials Model 010 to the Standard S1/S2 Serial Port of the RS/6000 POWERstation Models 2XX, 3XXX 5XXX and 7XX.

  Feature #4060 does not include a power facility for the 6094 dials or the 6094 LPFKs. If using a 5081 or 6091 display, or POWERdisplay 16 or 19, choose feature #4061 to obtain power for the 6094 dials or LPFKs. The IBM POWERdisplay 16-inch monitor has an actual, viewable screen size of 14.8 inches when measured diagonally. The IBM POWERdisplay 19-inch monitor has an actual viewable screen size of 17.3 inches when measured diagonally. If using a display other than the 5081, 6091 or POWERdisplay 16 or 19, choose either feature #4063 (100 volts), #4064 (120 volts) or #4065 (220/240 volts).

  NOTE: If your RS/6000 POWERstation configuration already includes a 6094 Serial Attachment Cable, 6094 Serial Power Cable, or 6094 Power Supply, the 4060, 4061, 4063, 4064, or 4065 features are not required.

Limitation: Diagnostics for the 6094 dials and 6094 LPFK cannot be executed from a diskette when feature #4060 is installed.

However, disk-based (hardfile) diagnostics for the 6094 dials and 6094 LPFK can be executed after these devices have been configured.

Once a standard serial port has been configured to support the 6094 dials or the 6094 LPFK, the remaining standard serial port will support only a 6094 dials or 6094 LPFK.

- (#4061) - 6094/RS/6000 Serial Attachment Display Power Cable:

  This feature allows the connection of 6094 dials Model 010 to the power source provided by the 5081, 6091, or POWERdisplay 16 or 19 Monitor. The IBM POWERdisplay 16-inch monitor has an actual, viewable screen size of 14.8 inches when measured diagonally. The IBM POWERdisplay 19-inch monitor has an actual, viewable screen size of 17.3 inches when measured diagonally. This feature is a dual power connection cable. Only one feature #4061 is necessary for each RS/6000 POWERstation.

  NOTE: If your RS/6000 POWERstation configuration already includes a Serial Attachment Power Cable, no further order for feature #4061 is required.

- (#4063) - RS/6000 Serial Attachment 100 Volt Power Supply:

  This feature is a 100 Volt Power Supply that provides power through the serial attachment cable to the 6094 peripheral. This feature is only to be used to power the 6094 Peripheral from a facility 100 volt power source. Quantity: One power supply is needed for each 6094-010 or 6094-020 Peripheral when the Serial Attachment Display power cable (#4061) cannot be used.

- (#4064) - RS/6000 Serial Attachment 120 Volt Power Supply:

  This feature is a 120 Volt Power Supply that provides power through the serial attachment cable to the 6094 peripheral. This feature is only to be used to power the 6094 Peripheral from a facility 120 volt power source. Quantity: One power supply is needed for each 6094-010 or 6094-020 Peripheral when the Serial Attachment Display power cable (#4061) cannot be used. Maximum: Two (2) per workstation. Limitations: See Limitations. Cable Orders: None. Field Installable: Yes. Prerequisites: None. Co-requisites: For RS/6000 orders, Serial Attachment Cable (#4060). Customer Setup: Yes.

- (#4065) - RS/6000 Serial Attachment 220/240 Volt Power Supply:

  This feature is a 220/240 Volt Power Supply that provides power through the serial attachment cable to the 6094 peripheral. This feature is only to be used to power the 6094 Peripheral from a facility 220/240 volt power

source. Quantity: One power supply is needed for each 6094-010 or 6094-020 Peripheral when the Serial Attachment Display power cable (#4061) cannot be used.

- (#4200) - Peripheral Adapter Connector:

  The Peripheral Adapter Connector (PAC) is an external passive device that allows attachment of multiple IBM graphics peripheral devices, such as the 6094 Model 010 dials, 6094 Model 020 LPFK, or 6094 Model 030 Spaceball, to RS/6000 workstations through the serial ports, S1 and S2. The PAC eliminates the need for individual external power sources for each attached peripheral by using the required power through the system's dedicated tablet port. The feature includes a 9-pin D-shell serial port attachment cable to be used exclusively on the RS/6000 7025-F40, 7043-140 and 240 workstations.

  CAUTION: The use of this feature on NON-SUPPORTED systems could cause damage to the system unit due to excessive power requirements through the tablet port.

  NOTE: 6094 feature 4000 (Peripheral Cable) is required for new peripherals. It is also required when the PAC replaces an existing external power supply (black power pack). This cable feature is required for each peripheral that is used.

- Attachment Cable (P/N 6247457):

  Used to connect the 6094-020 LPFK to the LPFK to the PAC (#4200) and also to the 6095 Graphics Processor through the Peripheral Connector Assembly (PCA) under the 5081 or 6091 displays.

- Attachment Cable (P/N 39F8228):

  Used to connect the 6094 Lighted Programmable Function Keyboard Model 020 to the Standard Serial Port of the RS/6000. See 6094/RS/6000 Serial Attachment (#4060) in 4.5.2, “Additional Features” on page 50.

- Attachment Cable (P/N 39F8302):

  Used to connect the 6094 Lighted Programmable Function Keyboard Model 020 to the power source provided by the 5081 or 6091 Monitor. See 6094/RS/6000 Serial Attachment Power Cable (#4061) in 4.5.2, “Additional Features” on page 50.

- Attachment Cable (P/N 6247480):

  Used to connect the 6094-020 LPFK to the Graphics Application Input Adapter (#2810) of the IBM RS/6000 POWERstation.

- Attachment Cable (P/N 39F8228):

Used to connect the 6094-020 LPFK to the Serial Port of the IBM PS/2 system and/or to the Async Communications Adapter of the IBM PC or PC-XT.

- Attachment Cable (P/N 39F8229):

  Used to connect the 6094-020 LPFK to the Dual Async Adapter of the IBM PS/2 system and/or the Serial Parallel Adapter of the IBM PC-AT.

- Power Supply (P/N 74F3089):

  100 volt power supply that must be used in conjunction with an attachment cable when attaching the 6094 to any model of the IBM PS/2 or RS/6000 POWERstation.

- Power Supply (P/N 6247468):

  120 volt power supply that must be used in conjunction with an attachment cable when attaching the 6094 to any model of the IBM PS/2 or RS/6000 POWERstation.

- Power Supply (P/N 6247469):

  220/240 volt power supply that must be used in conjunction with an attachment cable when attaching the 6094 to any model of the IBM PS/2 or RS/6000 POWERstation.

- Power Supply (P/N 6247470):

  220/240 volt power supply that must be used in conjunction with the attachment cable when attaching the 6094 to any model of the IBM PS/2 or RS/6000 POWERstation to be used in the United Kingdom.

- Diagnostic Diskette (P/N 39F8224):

  3.5-inch diagnostic diskette that MUST be used when attaching the 6094-020 LPFK to any model of the IBM PC or PS/2 with a 3.5-inch disk drive.

- Diagnostic Diskette (P/N 39F8225):

  5.25-inch diagnostic diskette that MUST be used when attaching the 6094-020 LPFK to any model of the IBM PC with a 5.25-inch disk drive.

4.6 The Spaceballs

The Spaceball provides users with an easier, more intuitive way to interact with models and images displayed by their applications. This type of interaction makes it easier and more productive for users to manipulate and control the viewing of their data. There are two different models available for the RS/6000 systems.

4.6.1 The IBM 6094 Model 031 Spaceball

The Spaceball is a 3D Input Device that puts 3D manipulation into your hands. Simply by pressing with your fingertips, to push, pull or twist the ball, you can rotate and view a computer-generated model from many different angles, manipulating the on-screen image as if you were grasping and handling it without any intermediate controls. This simple, intuitive interaction makes it easier and more productive for you to zoom, pan and rotate your model data.

A high-performance, 3D input device that attaches to all models of the RS/6000 workstation family, the Spaceball provides six degrees of freedom and allows complete interactive spatial control, including simultaneous translations and rotations about the X, Y and Z axes. In addition, the Spaceball gives you the ability to lock an object in motion along one axis. Its fully customizable interface allows you to create and modify button functions. Using online help, you can set up groups of functions for multiple user configurations, or modify individual sensitivity settings for panning, zooming and rotating.

The Spaceball is ideally suited for a wide variety of user-written applications requiring 3D input. Applications can offer Spaceball users additional functionality, including selectable viewpoints (object control mode, eyepoint control mode and orbit mode), the ability to constrain movement of an object within a certain boundary, and the ability to replay an object's movements.

Figure 7. The IBM 6094-031 Spaceball with Stealth Black Feature

Here are some of its characteristics:

- LPFK support - provides the functionality of LPFKs through software emulation, allowing you to have one less input device and one more available port on the system.
- Movable ball control - increases precise intuitive control, making it easier to apply force equally in any direction.
- The 6094 Model 031 does not require an external power source. The new technology uses minimal power acquired through the system serial port.
- The 6094 Model 031 is enhanced by a new SpaceWare 7.4.7 AIX(R) driver with many new functions. It delivers a fully customizable interface that allows users to create and modify button functions. This allows execution of user-defined controls, it can create groups of functions for multiple-user configurations, and even modify individual sensitivity settings for panning, zooming and rotating - complete with online help.

The *6094-031 Spaceball 3D Input Device Installation and User's Guide* is shipped with the product. This product is fully described in the announcement letter #198-157.

4.6.2 The 6094 Spaceball Model 040

The 6094 Spaceball Model 040 does not support dials emulation. It is not intended to be a replacement for the Model 031. The Model 040 is recommended for use when the application has provided support for full Spaceball support.

Here are some of its characteristics:

- Six degrees of freedom:

  The Spaceball allows complete interactive spatial control, simultaneous translations and rotations about the X, Y, and Z axes.

- Dominant axis filter:

  Allows you to lock in motion along pure X, Y or Z axes for finer control of larger images.

- SoftButtons user interface:

  You can select functions from the on-screen, SoftButton pop-up menu, allowing you to focus your complete attention on the screen and on your design.

- Ease of use:

  The ergonomic design coupled with the ability to adjust the ball sensitivity encourages a very natural, relaxed hand position and helps eliminate arm

or hand stress and fatigue while encouraging proper hand positioning on the ball.

- The 6094 Model 040 does not require an external power source.

4.7 Magellan

The 6094-600 Magellan is a 3D input device that allows simultaneous, six degrees of freedom control with one hand. It is used to help view a computer-generated object or model from any angle. It unifies the features of a conventional 2D mouse with those of a device for interactive motion control of 3D graphics in up to six degrees of freedom. The Magellan units are serially attached through the serial ports and conform to the RS-232 architecture. Magellan is fully described in announcement letter #197-278.

Figure 8. Magellan

4.8 Stereographics Capabilities

Stereo viewing is a method used to add depth to an image rendered on the two-dimensional surface of a monitor. It is used to enhance the 3D effects of various applications by providing support for stereoscopic viewing. For a good review of 3D and stereoscopic concepts and development see `http://www.qualixdirect.com/html/chapter1.html`

There are two elements that you need to take advantage of a stereographics environment:

- A graphic adapter capable of output in stereo mode. Table 3 on page 32 provides the list of adapters supporting stereo mode.

- Active shutter glasses that alternately block the view for one eye and then the other as the image is drawn. The application must be written to take advantage of this functionality. IBM does not sell these glasses. See `http://www.stereographics.com` for details regarding this device.

4.8.1 How to Connect the Emitter

The emitter plugs into the mini-DIN connector on the GXT550P, GXT800M, GXT800P, and GXT3000P graphics accelerator cards. You need to ensure that you have the correct emitter for the adapter you are using.

To install the emitter on a system with a GXT3000P Graphics Accelerator, shut down and remove power from the system. This prevents a blown fuse in the emitter. After power has been removed, insert the mini-DIN cable end into the graphics adapter. When you reboot the system, the emitter is detected, and the graphics adapter is auto configured for 120 Hz operation.

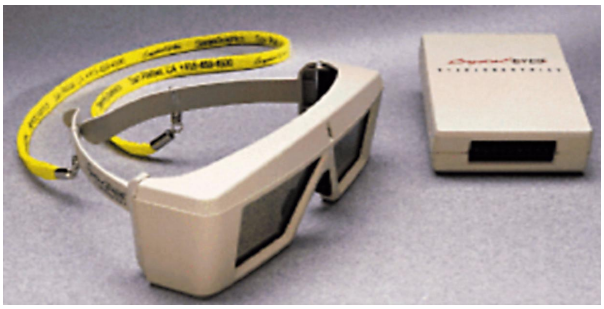


Figure 9. The StereoGraphics CrystalEyes Glasses and Emitter

Part 2. Software

Chapter 5. X11, Motif and CDE

This chapter briefly describes the 2D environment provided with every Risc System/6000. This 2D environment is made up of several APIs, X Windows, Motif, and CDE. Since, there are many other useful sources of information, this chapter focuses on describing the filesets to install to obtain the desired functions. The second part contains answers to frequently asked questions.

5.1 The 2D Environment

Since its introduction in 1990, the AIX operating system includes a graphical environment, including the X11 (that is the X server, the associated libraries and clients) and Motif (that is the window manager and application). Version 3.1 of AIX included X11R3 with Motif 1.0 and has evolved to X11R4 with Motif 1.1 and then to X11R5 and Motif 1.2.3. The latest version of AIX 4.3 includes X11R6 and Motif 2.1. Introduced with AIX Version 4 was the COSE Desktop, also known has CDE.

5.1.1 Configuration

When you set up your system, the installation process will automatically detect any graphic adapter installed on your system and add the correct filesets to drive this adapter, as well as the base filesets to obtain the graphical environment. The default is to have this graphical environment at boot time. Let's see now what files are included in the filesets for this 2D environment and their associated functions so you will know which one to install if you want to add new functions.

5.1.1.1 Filesets

Here is the description for the filesets associated with the 2D graphical environment:

X11.Dt — Includes all the components for the CDE environment. The minimal installation includes:

X11.Dt.ToolTalk AIX CDE ToolTalk Support

X11.Dt.bitmaps AIX CDE Bitmaps

X11.Dt.helpmin AIX CDE Minimum Help Files

X11.Dt.helprun AIX CDE Runtime Help

X11.Dt.libAIX CDE Runtime Libraries

X11.Dt.rteAIX Common Desktop Environment (CDE) 1.0

You can also decide to add these optional filesets:

X11.Dt.helpinfo	AIX CDE Help Files and Volumes
X11.Dt.xdt2cde	Migration Tool from the xdt to the CDE environment
X11.Dt.adt	AIX CDE Application Developers' Toolkit
X11.Dt.compat	AIX CDE Compatibility to the earlier version of the desktop
X11.adt	Includes the necessary files for developing applications for the X11 environment. It includes:
X11.adt.bitmaps	AIXwindows Application Development Toolkit Bitmap Files
X11.adt.ext	AIXwindows Application Development Toolkit for X Extensions
X11.adt.imake	AIXwindows Application Development Toolkit imake
X11.adt.include	AIXwindows Application Development Toolkit Include Files
X11.adt.lib	AIXwindows Application Development Toolkit Libraries
X11.adt.motif	AIXwindows Application Development Toolkit Motif
X11.apps	Includes most of the default X windows clients applications. Some are installed by default:
X11.apps.clients	AIXwindows Client Applications
X11.apps.config	AIXwindows Configuration Applications
X11.apps.custom	AIXwindows Customizing Tool
X11.apps.msmit	AIXwindows msmit Application
X11.apps.rte	AIXwindows Runtime Configuration Applications
X11.apps.aixterm	AIXwindows aixterm Application
X11.apps.xterm	AIXwindows xterm Application
X11.apps.util	AIXwindows Utility Applications

But you can decide to add these additional filesets:

X11.apps.pm	AIXwindows Power Management GUI Utilities
X11.apps.xdm	AIXwindows xdm Application
X11.apps.pcmcia	AIXwindows PCMCIA GUI Utility

X11.base	It includes the core files for X11, the X server, the libraries, the directories, or the shared memory transport extensions.
X11.compat	It contains all the filesets that provide compatibility with older versions of X11 or Motif. Those compatibility environments are valid for X11R3, X11R4, or X11R5 and Motif 1.0, Motif 1.1.4, or Motif 1.2
X11.fnt	It includes the fonts and the font utilities needed by the X11 environment. The default fonts are installed based on the specification for your selected language, the example for a default language of ISO8859-1 would be:
X11.fnt.coreX	AIXwindows X Consortium Fonts
X11.fnt.defaultFonts	AIXwindows Default Fonts
X11.fnt.iso1	AIXwindows Latin 1 Fonts
X11.fnt.util	AIXwindows Font Utilities

But you can add any other fonts filesets:

X11.fnt.iso2	AIXwindows Latin 2 Fonts
X11.fnt.iso3	AIXwindows Latin 3 Fonts
X11.fnt.iso4	AIXwindows Latin 4 Fonts
X11.fnt.iso5	AIXwindows Cyrillic Fonts
X11.fnt.ibm	AIXwindows Arabic Fonts
X11.fnt.iso7	AIXwindows Greek Fonts
X11.fnt.iso8	AIXwindows Hebrew Fonts
X11.fnt.iso9	AIXwindows Turkish Fonts
X11.fnt.Gr_Cyr_T1	AIXwindows Greek-Cyrillic Type1 Fonts
X11.fnt.ibm1046_T1	AIXwindows Arabic Type1 Fonts
X11.fnt.iso_T1	AIXwindows Latin Type1 Fonts
X11.fnt.iso8_T1	AIXwindows Hebrew Type1 Fonts
X11.fnt.ksc5601.ttf	AIXwindows Korean KSC5601 TrueType Fonts
X11.fnt.ucs	Includes support for the Unicode fonts
X11.info	Up to AIX Version 4.2.1, this fileset provides the graphical environment for exploring the AIX

	documentation. This fileset is not needed with AIX Version 4.3, where the documentation is HTML based.
X11.motif	Includes the libraries and binaries for the Motif environment. Those filesets are installed by default.
X11.samples	Includes the samples provided by the X consortium. It can help developers learning how to develop applications, but it also provides useful tools for end users.
X11.samples.lib.Core	AIXwindows Sample X Consortium Core Libraries Binary /Source
X11.samples.common	AIXwindows Imakefile Structure for Samples
X11.samples.apps.aixclients	AIXwindows Sample AIX Clients Source
X11.samples.apps.clients	AIXwindows Sample X Consortium Clients binary/Source
X11.samples.apps.demos	AIXwindows Sample X Consortium Demos Source
X11.samples.apps.motifdemos	AIXwindows Sample Motif Demos Source
X11.samples.doc	AIXwindows Sample Documents Source
X11.samples.ext	AIXwindows Sample X Extensions Source
X11.samples.fnt.util	AIXwindows Sample Font Server Utilities Source
X11.samples.rgb	AIXwindows Sample Color Database Source
X11.vfb	Includes the code to use the X Virtual Frame Buffer described Chapter 6, “The X Virtual Frame Buffer and Softgraphics” on page 75.
X11.vsm	Includes the components of the Visual System Management.

5.1.2 Answers to Frequently Asked Questions

This section will now focus on trying to answer the most frequently asked questions.

5.1.2.1 How Do I Manage Several Graphics Adapters?

Yes, you can have multiple graphics adapters connected to your system. See Chapter 2, “The Graphics Adapters” on page 11 for the maximum number of adapters you can have on your system. The first thing to do is to list those adapters and to decide which one will be the default adapter. The default adapter will be the one on which the ASCII or the graphic login will be displayed. Remember, even if you have multiple graphics adapters and screens, you still have only one keyboard and mouse; so only one screen is active at a time.

The pair of commands, `lsdisp` and `chdisp`, will list the adapters connected on your systems.

```
# lsdisp


DEV_NAME  SLOT  BUS  ADPT_NAME  DESCRIPTION
========  ====  ===  =========  ===========
iga0      C0    pci  E15        E15 Graphics Adapter
gga0      01    pci  S15        IBM Personal Computer Power Series S15 Graphics
Adapter

   Default display = gga0
```

The `chdisp` command allows to change the default graphic adapter. In our case the default graphics adapter is the IBM Personal Computer Power Series S15 adapter. If we decide that we want to change that to the E15 adapter, we use the `chdisp` command, and we can also decide if we want to change adapters for this session only and get back to the S15 adapter after this session, or if we want to pick the E15 graphic adapter but only after reboot, and that is the role of the `-d` and `-p` options.

```
# chdisp -help
Usage: chdisp [-dDeviceName] [-pDeviceName]
        Changes the default display.

        -d Changes the default display for this session.
        -p Changes the default display in the database.  This
           is effective at the next IPL.
        DeviceName is the logical name of the display.  This is
        the name in the first column of output of the lsdisp command.
# chdisp -p iga0
# lsdisp


DEV_NAME  SLOT  BUS  ADPT_NAME  DESCRIPTION
========  ====  ===  =========  ===========
iga0      C0    pci  E15        E15 Graphics Adapter
gga0      01    pci  S15        IBM Personal Computer Power Series S15 Graphics
Adapter

   Default display = iga0
```

Finally, the `chdisp` command should run from the `lft`, but redirecting the input allows you to use this command from any session.

```
# chdisp
chdisp: 0468-002 Run this command only at an LFT workstation.
# chdisp < /dev/lft0
Usage: chdisp [-dDeviceName] [-pDeviceName]
        Changes the default display.

        -d Changes the default display for this session.
        -p Changes the default display in the database.  This
           is effective at the next IPL.
        DeviceName is the logical name of the display.  This is
        the name in the first column of output of the lsdisp command.
```

The next part of the question may be, "How do I start my X server on one screen rather than the other or on both at the same time?" The default with the latest versions of AIX starts the X server on every screen connected to the system, and the mouse crosses the screen limits seamlessly. From AIX Version 4.2.1, the CDE window manager is able to display a front panel on each of the screen parts of the X server (refer to *AIX Version 4.2 Differences Guide,* SG24-4807). So, the real question is now how to start the server on only one screen. The answer is to use the `-P` option of the X command that allows you to organize your screens as a matrix of rows and columns; then you access each screen by row and column number. If you want to define only one screen, choose a matrix with one row and one column and add the

`-P11 display name` to the command line. Here is as example of how to start the server on the display connected to the iga0 adapter:

```
xinit -- -P11 iga0
```

If you are using CDE, you will have to modify the X server's file in the /etc/dt/config directory to add this `-P` option.

5.1.2.2 How Do I Change My Display Resolution?

The resolution you can obtain on your screen depends on two factors: the graphic adapter on your system and the display itself. In order to specify the maximum resolution for your adapter, you first have to specify the type of display you are using. Use `smit chdisptype` to associate your display with its graphic adapter.

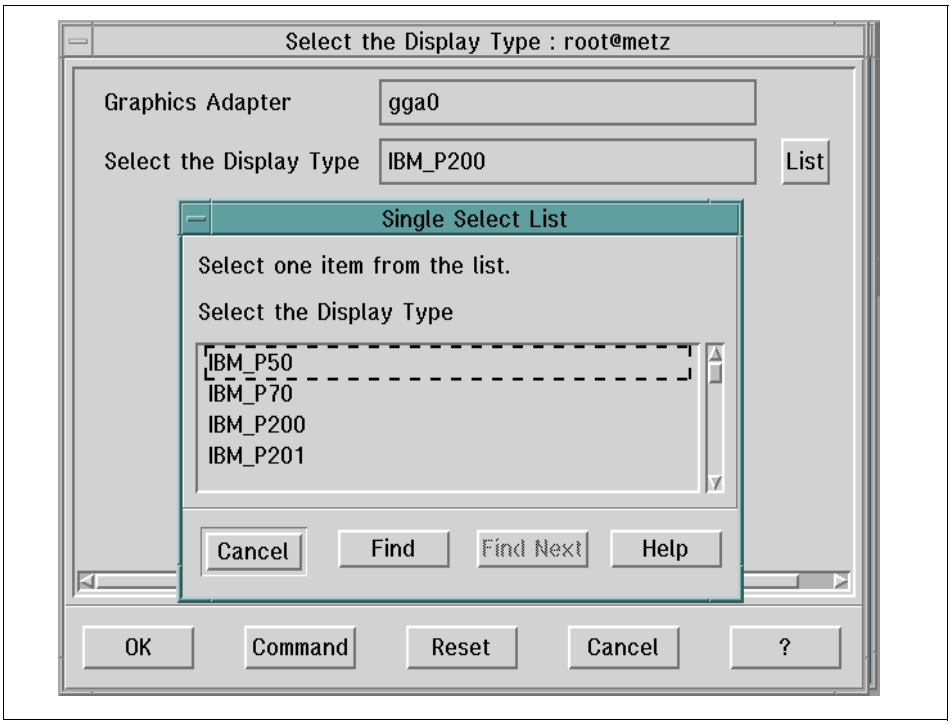


Figure 10. The SMIT chdisptype Screen

Then, you can use `smit chres_refrt` to select the best resolution and refresh rate.

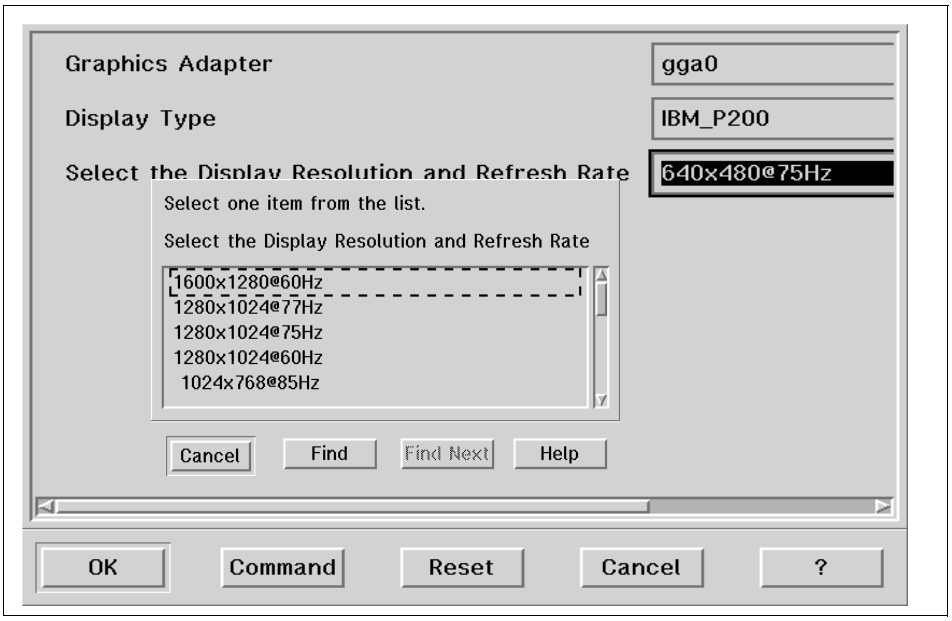


Figure 11. The SMIT chres_refrt Screen

Even if you can run this command from any session, you will have to quit and reenter the graphical environment for the changes to take effect. Within CDE, that implies logging out of your CDE session and clicking on the **Reset Login Screen menu** under the Options button.

5.1.2.3 How Do I Specify if I Want a Graphic Login Screen?

Within AIX, you can decide whether you will have a graphical environment at boot time and if you want a graphical login screen (with CDE) or if you want to keep the regular ASCII screen. Use the `smit dtconfig` command to customize your environment.

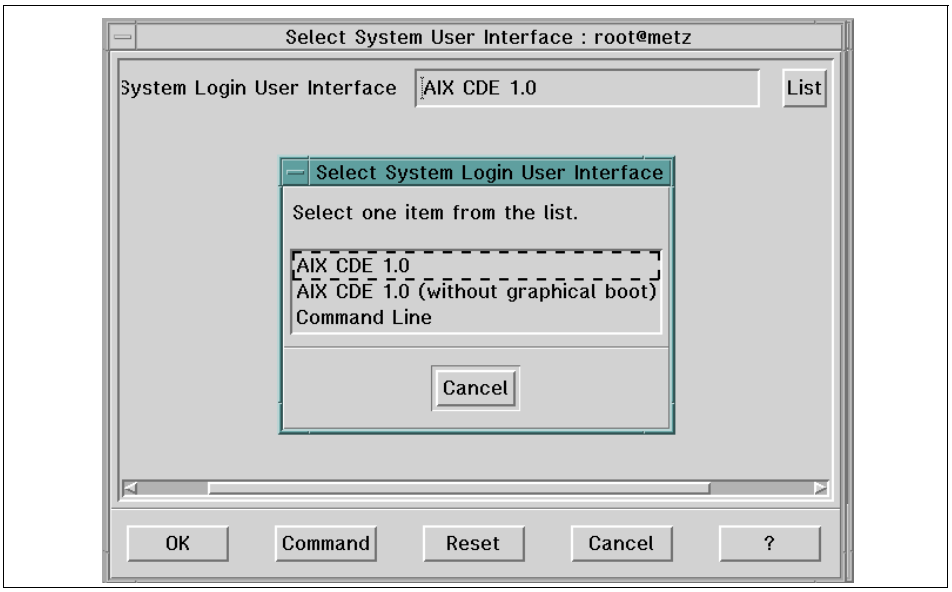


Figure 12. The SMIT dtconfig Screen

5.1.2.4 How Do I Add Extensions to the Server?

If you decide that you will use 3D API or special functions of the X server, chances are that you will have to configure an X server extension (refer to the *AIX Version 4.2 Differences Guide,* SG24-4807, for a complete description of X server extensions). These extensions are functions that are not part of the X server, but can be added if you need them. There are several ways to indicate which extensions to add the X server. Those extensions are referenced in two files located in /usr/bin/X11. The first file is static_ext, and it describes the extensions that are loaded automatically each time the X server is started. The second one is dynamic_ext, and it contains all possible extensions for the X server. After that, you control from the command line or by the EXTENSIONS environment variable which one to load.

A typical line would look like this:

```
abx             /usr/lpp/X11/bin/loadAbx
```

Where `abx` is the short name of the extension and `/usr/lpp/X11/bin/loadAbx` is the location of the module to be loaded.

If an extension is not specified in the file static_ext, you must use the `-x` option on the X command:

```
xinit -- -x abx -x mbx
```

If you specify an extension which is not installed, the X server will not start. After the X server has started, you can use the `xwininfo` command to see which extensions are really loaded.

5.1.2.5 How Do I Configure My CDE Login Screen?

Figure 13 is an example of the default CDE login screen in AIX Version 4.3.2.

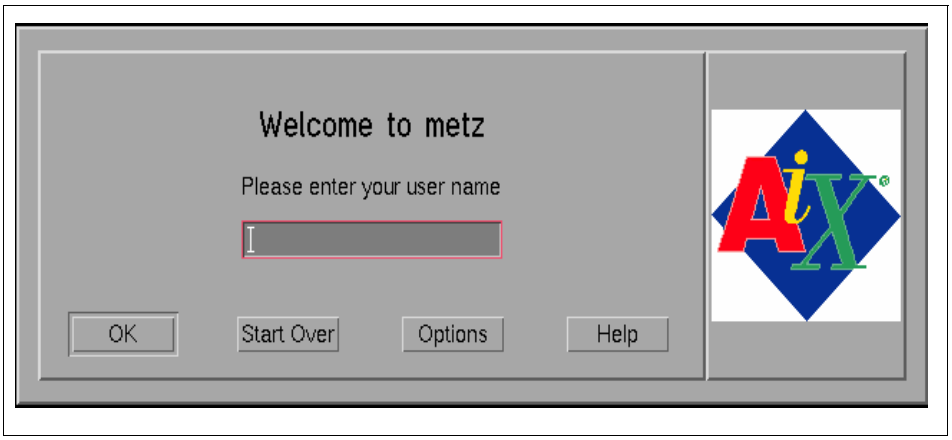


Figure 13. Example of the Default CDE Login Screen

You may decide that you want to customize this screen with your own logo or text. This is done by modifying the X resources file. The template for this file is located in the /usr/dt/config/$LANG directory. Copy this file into /etc/dt/config/$LANG directory and change the entries that command the logo file name, color, fonts, or message.

For example, to change the logo, modify the resource:

```
Dtlogin*logo*bitmapFile: < your file name location >
```

Note

The file name that you specify for the `logobitmapfile` name must contain a valid pixmap. One of the easiest ways to create such a file is to use the icon editor included with CDE. You can display any image on the screen and then grab this image from the icon editor. The last step is to save that image into the desired location.

To change the welcome message, modify the resource:

```
Dtlogin*greeting.labelString: < Your welcome message>
```

To change the fonts, modify the resource:

```
Dtlogin*labelFont: <your favorite font>

Dtlogin*textFont: <your favorite font>

Dtlogin*greeting.fontList: <your favorite font>
```

The resulting CDE login screen could look like the following figure:

Figure 14. A Customized CDE Login Screen

To obtain more information on how to customize and use the other components of CDE, refer to the *AIX Version 4 Desktop Handbook,* GG24-4451.

5.1.2.6 How Do I Change X Resources?

An X resource is a property for an object that can be defined either in the source code of the application using this object or in many exterior configuration files. The value for this resource can then be set in the program and then customized by users. Users can specify from the command line, for example, the background color for an aixterm, by specifying the `-bg` option. But there are many other locations that can be checked to find out what this color can be. Here is the list of all the locations inspected when you start an aixterm:

/usr/lpp/X11/defaults/$LANG/Xdefaults	This is the template if you have implemented an international environment.
/usr/lpp/X11/defaults/Xdefaults	This is the template for a common environment.

/usr/lib/X11/$LANG/app-defaults/Xdefaults	This is the default system location to specify system wide resources for users based on their language environment.
/usr/lib/X11/app-defaults/Xdefaults	This is used for an international environment.
/usr/lpp/X11/defaults/app-defaults/Xdefaults	This is the same as /usr/lpp/X11/defaults/Xdefaults since they are linked.
/usr/lpp/X11/defaults/$LANG/Aixterm	This is specific for the aixterm command and is based on the $LANG.
/usr/lpp/X11/defaults/Aixterm	This is specific to the aixterm command, but does not include the $LANG variable.
/usr/lib/X11/$LANG/app-defaults/Aixterm	This adds the app-defaults subdirectory.
/usr/lib/X11/app-defaults/Aixterm	This is the default systemwide location.
$HOME/$LANG/Aixterm	This is located in the home directory if you have implemented an international environment.
$HOME/Aixterm	This is for a common environment.
$HOME/$LANG/aixterm	You can spell aixterm with uppercase 'A' or not.
$HOME/$LANG/.Xdefaults	This is the .Xdefaults file based on your $LANG variable.
$HOME/.Xdefaults	This is the most used file.
$HOME/$LANG/.Xdefaults-displayname	You can also specify a different file based on the display where you display your applications.

$HOME/.Xdefaults-displayname	Same thing as the previous item but without the $LANG subdirectory.
$XENVIRONMENT	The file you have specified in the $XENVIRONMENT variable.

Chapter 6. The X Virtual Frame Buffer and Softgraphics

This chapter discusses two features included in AIX that belong to the software but are also hardware related since they will either fully or partially replace the hardware. The X Virtual Frame Buffer allows an application to render into the main memory of the computer instead of the hardware graphic adapter, and Softgraphics allows an application to use advanced 3D functionalities on a 2D adapter.

6.1 The X Virtual Frame Buffer

IBM has enhanced the AIX X server to support a technology called the X Virtual Frame Buffer. The X Virtual Frame Buffer (XVFB) allows the X server to initialize and run without the presence of any physical graphics adapter. In the past, the X server has required one or more graphics adapters in order to run and would exit with an error if none were present.

Furthermore, in a standard X Window System environment, each 3D application that is running on a system must share the same hardware frame buffer. While this is fine for viewing clients locally, it is bad for viewing clients remotely when the graphics windows overlap in the screen space. This overlap causes the rendering to be serialized and slows overall performance.

Utilizing the XVFB, each application has a private 3D rendering area. Since there is no window overlap, rendering can take place in parallel. Of course, there is a downside for any locally attached display. You cannot see the image on the screen. It must be displayed over the network to be viewed.

The XVFB environment, shipped with the AIX operating system, is an environment where an application renders images on a server machine. These images are then distributed to viewing stations on a network, saved to a database, or used in some other way.

The XVFB environment fills the need for a method to facilitate viewing 3D graphics on a low-end system for casual users.

The XVFB software is supported on AIX Versions 4.1.5, 4.2.1, and 4.3.1 or later. The XVFB is also supported on the RS/6000 SP. You need to install the X11.vfb package. It requires approximately 1 MB. It will be necessary to reboot the system after installing this package as it requires a kernel extension to be loaded at boot time.

6.1.1 Installing XVFB

This section describes how to install the software for XFVB on various levels of AIX.

6.1.1.1 AIX 4.1.5

XVFB on AIX 4.1.5 is installed from the following filesets.

- U454863 - OpenGL.OpenGL_X.dev.vfb.04.01.0005.0000
- U454864 - X11.vfb.04.01.0005.0000

Two PTFs are needed for XFVB to fix a window resizing problem:

- U456568 - OpenGL.OpenGL_X.dev.vfb.4.1.5.2
- U455198 - OpenGL.OpenGL_X.rte.soft.P1.4.1.5.8

6.1.1.2 AIX 4.2.1

XVFB on AIX 4.2.1 is installed from the following filesets.

- U455397 - OpenGL.OpenGL_X.dev.vfb.04.02.0001.0000
- U455398 - X11.vfb.04.02.0001.0000

Two PTFs are needed for XVFB to fix a window resizing problem. They are available on fixdist:

- U456210 - OpenGL.OpenGL_X.dev.vfb.4.2.1.2
- U456371 - OpenGL.OpenGL_X.rte.soft.4.2.1.8

6.1.1.3 AIX 4.3.0

XVFB on AIX 4.3.0 is installed from the following filesets.

- U454163 - OpenGL.OpenGL_X.dev.vfb.04.03.0000.0000
- U454162 - X11.vfb.04.03.0000.0000

Two PTFs are needed to fix a window resizing problem. They are not available. There is no plan to create these two PTFs for AIX 4.3.0 at Austin. To use XVFB with AIX 4.3, you must migrate to AIX 4.3.1.

If the PTF U452591 is installed, CATweb Navigator Version 1 runs OK.

- U452591 - OpenGL.OpenGL_X.rte.soft.4.3.0.1

6.1.1.4 AIX 4.3.1

XVFB on AIX 4.3.1 is installed from the AIX 4.3.1 product CDs. Install the following filesets:

- OpenGL.OpenGL_X.dev.vfb.04.03.0001.0000
- X11.vfb.04.03.0001.0000

Two PTFs are needed to fix a window resizing problem. They are available on fixdist:

- U456096 - OpenGL.OpenGL_X.dev.vfb.4.3.1.1
- U456079 - OpenGL.OpenGL_X.rte.soft.4.3.1.1

6.1.1.5 AIX 4.3.2

XVFB on AIX 4.3.2 is installed from the AIX 4.3.2 product CDs. Install the following filesets:

- OpenGL.OpenGL_X.dev.vfb.04.03.0002.0000
- X11.vfb.04.03.0001.0000

6.1.2 Starting the XVFB

The XVFB is loaded into the X server with the `-vfb` flag.

```
/usr/bin/X11/X -force -vfb -x abx -x dbe -x GLX
```

This will start up the X server without using the graphics adapter if it is present as well as load the OpenGL extensions to the X server.

You can also use the `xinit` command which will not only start the X server but also the window manager.

```
/usr/bin/X11/xinit -- -force -vfb -x abx -x dbe -x GLX
```

You can also add the `-vfb` flag to the EXTENSIONS line in your .xserverrc file.

If you desire to have the XVFB effective at system boot, the system administrator can add an entry in the /etc/inittab file. The following entry will cause the X server to be started at system boot time as well as cause it to be restarted automatically if the server ever exits or dies.

```
xvfb:2:respawn:/usr/bin/X11/X -force -vfb -x abx -x dbe -x GLX
>/dev/null
```

You can run more than one X server at the same time, with the following restrictions:

- No COSE Desktop
- Only a single instance of the XVFB X server
- Only a single instance of the X server running to a graphics adapter

If you have a system with a graphics adapter, and you want to run an XVFB X server and an X server to your graphics adapter, do the following:

Start your X server to the graphics adapter:

```
xinit
```

From an xterm/aixterm, start your XVFB server:

```
X -vfb -x GLX -x abx -x dbe -force
```

6.1.3 Testing the XVFB

Since you can't see the frame buffer when using XVFB, it is difficult to confirm that things are working correctly. Fortunately, several X clients ship with AIX that can be used to query and view window contents, and can be used to help verify that XVFB is rendering the correct images.

These clients include `xwininfo`, `xwd`, and `xwud`.

6.1.3.1 Verifying that You Are Using XVFB

To verify that the X server is running with the XVFB, you can use the following command:

```
/usr/lpp/X11/Xamples/bin/xprop -display sysname:0 -root | grep VFB
```

`XVFB_SCREEN(STRING) = "TRUE"` <== indicates XVFB is being used

6.1.3.2 Verifying that XVFB is Working

If you are unsure whether or not XVFB is installed and started properly, you can use the following method to test XVFB. Your system needs to be on a network, and you need access to another system (with a screen) in order to view the contents of the XVFB.

1. On the XVFB system, start the X server using the `-vfb` flag.

   ```
   /usr/lpp/X11/bin/X -force -vfb -x GLX -x abx -x dbe
   ```

2. On the XVFB system, run the xclock client program.

   ```
   xclock -display :0.0 &
   ```

3. On the other system, make sure X is running and that clients can connect.

   ```
   xhost +
   ```

4. Find the window ID for the `xclock` client.

   ```
   xwininfo -root -tree | grep xclock
   ```

   The first number (0x12345678) is the window ID.

5. On the XVFB machine, use `xwd`/`xwud` to display the client window of the XVFB system on the other system.

   ```
   xwd -id 0x12345678 | xwud -display othersystem:0.0
   ```

You should see an image of the `xclock` you started on the XVFB system appear on the other system.

6.1.4 Implementing XVFB in Application Code

XVFB allows developers to write Web-based 3D graphics applications for an RS/6000 server without the need of a 3D graphics adapter. XVFB-enhanced applications in most environments should achieve near linear scalability by adding processors to multiple-processor systems. This is because each client can render into its own frame buffer without interaction with the X server. End-users viewing data at client systems receive the benefit of XVFB with no changes to software or invocation methods to applications.

To enable an application to operate in a XVFB environment, the application must be enhanced in the following way:

1. Method to extract rendered image from rendering server:

   The image to be displayed should be extracted or retrieved from the rendering server. Typically, an application would do an `XGetImage` for X applications or a `glReadPixels` for OpenGL applications.

   The application decides the frequency and type of actions that will result in the extraction of the image from the rendering server. One criteria might be each time the buffer is swapped using the `XdbeSwapBuffers` or `glXSwapBuffer` subroutine.

2. Method to send commands to remote application on server platform:

   To give the application input to perform, such as opening files or transforming objects, there should be a method to send the application input from a remote source. This could be a command language, a socket connection, interaction with an HTTP server or some other type of communication services. The XRecord and XTest extensions could be used to send events to the X server and communicate with the application through traditional X events.

3. Method to display image on a display station:

   The XVFB environment should contain a method to display the rendered image on the display station. The method of display of the extracted image is at the discretion of the programmer. If the display station is an AIX workstation executing an AIXwindows X server, the extracted image could be displayed using the `xwud` command. The `xwud` takes an image in the `xwd` format, creates a window, and does an `XPutImage` to display the extracted image.

   More sophisticated methods could include image compression and Java display applets. The advantage of a Java display applet is that the image could be displayed on a variety of platform types.

Applications can also check for the value of the XVFB_SCREEN property in order to determine if they are running with the XVFB. The following code shows how this can be done:

```
int isXVFB(Display *display,Screen screen)
{
     Atom atom,actual_type;
     int actual_format,status;
     unsigned long nitems,bytes_after;
     unsigned char *prop;
     atom = XInternAtom(display,"XVFB_SCREEN",True);
     if (atom == None)
         return False;
     status = XGetWindowProperty(display,RootWindow(display,screen),
     atom,0,100,False,atom,&actual_type,&actual_format,&nitems,
     &bytes_after,&prop);
     if (strcmp((char*)prop,"TRUE") == 0)
         return True;
     return False;
}
```

6.1.5 How Does It Work?

The XVFB works by providing an X DDX layer (Device-Dependent X) that drives a software graphics adapter. In other words, the frame buffer is stored in system memory, and all graphics processing (lines, polygons, text, and so on) is done completely in software using the system CPU.

The XVFB is intended to be used in a “rendering server” environment. With XVFB, X applications can run and render images, and the images can be queried back into the application for saving to a file, distributing across the network, saving to a database, and so on. In this mode, the X application is not being used directly by an end user in an interactive way. Instead, the X application is being driven remotely as a rendering server.

With no physical graphics device, there is no RAMDAC to generate RGB signals. This means that it is impossible to connect a monitor to your system and view the contents of the X server Virtual Frame Buffer. While this idea takes a little while to get used to, it is a good solution for rendering server environments, where the added expense of a physical graphics adapter and display are not required. When you cannot see the X frame buffer directly, it does make debugging of your application more difficult. It is suggested that applications be developed with a physical graphics adapter and then ported to the XVFB. X client tools like xwininfo, xwd, and xwud can be used to help verify your application is running correctly with the XVFB.

In addition, input devices in the XVFB environment are not required. Since you cannot see the frame buffer, moving the mouse around and typing on the keyboard are not very useful. Rendering server applications instead will be driven remotely, either with direct socket communication, interaction with an HTTP server, through CORBA connections, message passing interface (MPI), and so on. Most applications need modifications in order to be controlled remotely for a rendering server environment.

OpenGL is currently supported with the XVFB; however, PEX, GL 3.2 and graPHIGS are not supported. For additional information point, your Web browser at:

`http://www.rs6000.ibm.com/solutions/interactive/rendserv.html .`

6.2 CATweb and the XVFB

CATweb Navigator allows end users with Java-enabled Web browsers to view and navigate product information created with CATIA Solutions. By using the intuitive set of Java applets that make up the CATweb Navigator client, users can connect to a CATweb Navigator server machine, select models for viewing, and then view and navigate the models with a 3D viewer, 2D schematic viewer, or a report style viewer. One CATweb Navigator server machine can support multiple concurrently active CATweb Navigator clients. For each CATweb client that attaches to the CATweb server, one or more CATweb processes is started on the CATweb server to handle the requests of that particular CATweb client. One of these CATweb processes is a 3D rendering application. This application runs on the CATweb server and is responsible for:

- Loading the requested CATIA model
- Rendering the 3D model on the fly as the CATweb client requests new views
- Compressing the final rendered image
- Transferring the image to the Java CATweb client

This application uses both the X Window System and OpenGL libraries to quickly and accurately render 3D images.

XVFB and DirectSoft OpenGL are both important new IBM technologies that enhance the RS/6000's capabilities as a CATweb Navigator server platform. XVFB and DirectSoft OpenGL allow CATweb Navigator server machines to operate without expensive graphics adapters and to effectively exploit Symmetric Multi Processing (SMP) machines. In addition, XVFB and

DirectSoft OpenGL allow a CATweb Navigator server to be usable at boot time without requiring a user to log in interactively, making it easier to set up and administer a CATweb Navigator server. The RS/6000, AIX, XVFB, DirectSoft OpenGL combination makes the RS/6000 the most cost-effective, scalable, and easy to administer CATweb Navigator server platform available.

The CATweb Navigator does not use the XVFB by default. The image server rendering is done through the X real frame buffer and all the concurrent CATweb renderings are synchronized to share this single resource (CATweb rendering lock).

In the XVFB mode, each CATweb process uses its own X Virtual Frame Buffer, and the rendering synchronization is no longer necessary, thus improving the overall performance in a multi-user environment.

You can deactivate the default CATweb rendering lock by changing the following value in the runServerCATIA file under the directory .../CATwebNavigator/bin:

```
VirtualFrameBufferOn=1 (default value = 0)
```

The CATweb rendering unlock will be taken into account at the next CATweb connection.

6.2.1 DirectSoft OpenGL

DirectSoft OpenGL (DSO) is a new IBM OpenGL rendering technology for AIX and RS/6000. Implemented to work with the XVFB, DSO was designed specifically to enhance CATweb server performance by eliminating extraneous interprocess communication, eliminating process context switching overhead, and making rendering and image reading more direct and efficient. In addition, when combined with the XVFB, DSO allows the CATweb server to effectively exploit multiple processors in an SMP machine.

DSO is a pure software implementation of OpenGL that runs as a direct OpenGL Context. Basically, this means that all of the CPU-intensive OpenGL work (3D rendering) is part of the application process (not part of the X server process). By running direct, all of the interprocess communications with the X server are eliminated, making 3D rendering much more efficient. In addition, the AIX operating system is not having to perform context switching between the X server and the 3D rendering applications, making system utilization more efficient.

Also, DSO and XVFB enable SMP machines as viable and scalable CATweb servers. DSO actually creates a private rendering area for each 3D rendering

application. When it comes time to render a new image, the application draws to its private rendering area. If two CATweb clients request new images at exactly the same time, each of the 3D rendering applications can draw new images concurrently, each to their own private rendering area. Since multiple 3D rendering applications can draw concurrently, multiple CPUs can be exploited concurrently. Without DSO and XVFB, the 3D rendering applications are drawing to a physical, shared frame buffer, which means they must take turns and operate serially. With each 3D rendering application taking turns to draw, only a single CPU can be effectively exploited.

If you are using an OpenGL application with DirectSoft OpenGL and XVFB, the testing scenario with `xwd` and `xwud` discussed earlier in this chapter will not work. This is because, to increase performance, the default OpenGL DirectSoft behavior does not blit the contents of the private frame buffer to the X Virtual Frame Buffer. This behavior will be fine for most OpenGL rendering server applications since the image will be queried from the private frame buffer using `glReadPixels`. So, using `xwd` and `xwud` just grabs the contents of a blank window.

To verify that DirectSoft OpenGL is working, set the `_OGL_MIXED_MODE_RENDERING` environment variable to 1, and then run your OpenGL application. This will force OpenGL to actually blit the rendered image from the private software frame buffer to the X Virtual Frame Buffer. Once this is done, you can once again use xwd and xwud to grab and dump the contents of your OpenGL window. Note: It is recommended that you do not keep this environment variable set all the time, since this will slow the overall operation of DirectSoft OpenGL.

Applications can determine if they are using DirectSoft OpenGL by checking the OpenGL rendering string (`glGetString`), and checking the OpenGL context (`glXIsDirect`). The following code can be used:

```
int isDirectSoftOpenGL(Display *display,GLXContext context)
{
   if (glXIsDirect(display,context) == FALSE)
     return FALSE;
   if (strcmp(glGetString(GL_RENDERER),"SoftRaster") == 0)
     return TRUE;
   return FALSE;
}
```

6.3 Softgraphics

Traditionally, 3D graphics APIs were implemented using specialized graphics hardware. In this kind of API implementation, all 3D operations are performed on the graphics adapter. The graphics adapter offers frame buffers, additional specialized buffers, graphics processing power, a specialized graphics pipeline, and hardware-implemented algorithms. Geometry processing as well as raster processing is done on the graphics adapter. This creates applications that are strongly dependent on certain functions provided by certain graphics hardware. Two things changed over time in the world of technical workstations:

- The power of workstation CPUs increased and entry-level workstations started requiring 3D graphics capabilities because technical workstations were now used for a wide variety of different types of applications.
- Technically, highly specialized graphics hardware could be replaced with a simple frame buffer adapter plus a strong CPU, system memory and software.

But this requires a different kind of 3D API implementation. Specialized graphics operations formerly done in hardware can now be done in software. The API software implementation works as a glue between the application code and the hardware. This allows you to utilize the CPU capacity and also allows applications to be hardware independent. It offers consistent graphics functions across all graphics adapters independent of their specific capabilities. It therefore offers integration across hardware products.

6.3.1 What is Softgraphics?

For customers requiring 3D capability, AIX provides the facilities for the development and execution of 3D applications using a variety of industry standard APIs. This includes hardware-accelerated support for graPHIGS and GL 3.2 as well as a pure software implementation of OpenGL, and graPHIGS referred to as Softgraphics.

Softgraphics allows all 3D functions to be performed by software where the graphics adapter is used simply as a frame buffer to display the image. This implementation makes it possible to run 3D applications on any 2D graphics adapter. Softgraphics performance scales with the performance of the RS/6000 processor. The faster the system processor, the faster Softgraphics will perform. The choice between the added price of the 3D adapter, which provides graphics acceleration, versus the execution of all 3D functions by the system processor, should be evaluated very carefully on an application by application basis.

Softgraphics provides a uniform development environment for 3D applications on RS/6000s with entry-level graphics adapters. Programmers can develop advanced 3D applications for the industry's most popular APIs which can easily be moved to any 3D graphics adapter with little or no change to the source code.

6.3.2 Installation of Softgraphics

Softgraphics is now shipped as part of the AIX base installation. It is installed with the following code:

- `OpenGL.OpenGL_X.rte.soft`
- `PEX_PHIGS.graPHIGS.rte.soft`

An installation verification program is supplied with the graPHIGS Base Run-Time Environment fileset. Run `/usr/lpp/graPHIGS/etc/runivp` to start the program.

Chapter 7. graPHIGS

This chapter describes the IBM graPHIGS product, which is one of the two major 3D graphics rendering APIs (Application Programming Interfaces) supported by AIX (PHIGS for AIX 4.3 is included in AIX Version 4.3).

The graPHIGS API is IBM's implementation of the ISO/ANSI Programmer's Hierarchical Interactive Graphics System (PHIGS) standard, which includes a subset of the PHIGS PLUS extensions (such as independent lighting and shading controls, advanced primitives, and extended rendering attributes) as well as IBM's own extensions. The graPHIGS API is supported on the IBM Enterprise Systems and the RS/6000 computers equipped with 2D or 3D adapters.

7.1 Definition

To better understand the graPHIGS API, it is necessary to introduce its predecessors: Core, GKS and PHIGS.

7.1.1 Core, GKS, and PHIGS

In 1977, a Special Interest Group on Graphics (SIGGRAPH), formed by a committee of the Association of Computing Machinery (ACM), developed the first recognized standard computer graphics API known as the 3D Core Graphics System, or Core. Core, however, was not officially recognized as a standard by groups such as the American National Standards Institute (ANSI) or International Standards Organization (ISO). The first API to receive the endorsement of these organizations was the Graphical Kernel System (GKS).

Though Core was used as a model during its development, GKS turned out to be an enhanced version of Core (restricted to 2D geometry).

From the initial releases of GKS, it became apparent that a standard for 3D graphics was needed. In response to this requirement, two separate APIs appeared: GKS-3D and Programmer's Hierarchical Interactive Graphics System (PHIGS). The PHIGS API proved to be a more robust library for graphics and utility routines and was subsequently accepted by ISO as an international standard in 1988. PHIGS was also accepted by the ANSI committee and was adopted by many users in the graphics community.

Extensions to PHIGS which include advanced rendering functions such as lighting and shading are part of a new standard called PHIGS PLUS. The

PHIGS PLUS functional specification is complete, including a C language binding.

7.1.2 graPHIGS

IBM's implementation of the PHIGS standard is called graPHIGS and is available on all graphics adapters for the RS/6000 as well as on some Enterprise Systems platforms. Additionally, the graPHIGS API contains extensions beyond the PHIGS and those of the proposed PHIGS PLUS standards.

This graphics software contains a suite of advanced graphics functions for developing complex 3D applications in technical and commercial areas, including computer-aided design and manufacturing, industrial design, engineering analysis, and scientific visualization.

For 3D graphics applications, the PHIGS product provides interoperability in networked heterogeneous environments. This client/server implementation allows you to increase productivity by distributing your 3D applications between workstations and computer servers.

The IBM PHIGS product includes the graPHIGS API and ISO PHIGS language bindings as well as the Graphical Kernel System-Compatibility Option (GKS-CO), which is an extension to the graPHIGS API.

The advantages that give PHIGS its popularity are:

- Wide range of graphics functions - from the simple geometry to the sophisticated rendering of complex surfaces
- Interoperable across platforms - designed on open API standards to be fully portable
- Multi-threaded graphics pipeline - increases selected application performance by utilizing multiple processors simultaneously without recompilation
- Easy to use - includes example programs, application development tools, and online hypertext information

Though many graPHIGS applications have migrated to OpenGL, the graPHIGS API is still widely used, not only for CATIA but also for many customers' private applications.

Although the strength of graPHIGS is retained mode graphics, graPHIGS has been extended to provide immediate mode capabilities. For an explanation of these terms, see:

- Section 7.1.3, “Retained Mode Graphics” on page 89
- Section 9.1.1, “Immediate Mode Graphics” on page 130
- Section 7.3.3, “Explicit Traversal Control for Immediate Mode Graphics” on page 100

7.1.3 Retained Mode Graphics

Retained mode means that the geometry is stored in an internal hierarchical data structure for later display and modification.

An example of an application where retained mode graphics is well suited is a simulation of the motion of a robot arm consisting of a base, two links and a wrist element. Each portion of the arm is connected to the preceding portion; the base to the first link, the second link to the first, and the wrist to the second link. The data representing the physical elements of the arm and their locations is stored in a hierarchically arranged display list. When the base is rotated, every element is affected, and the transformation matrix which describes the rotation can be applied globally to the arm without recalculating the position of each link individually.

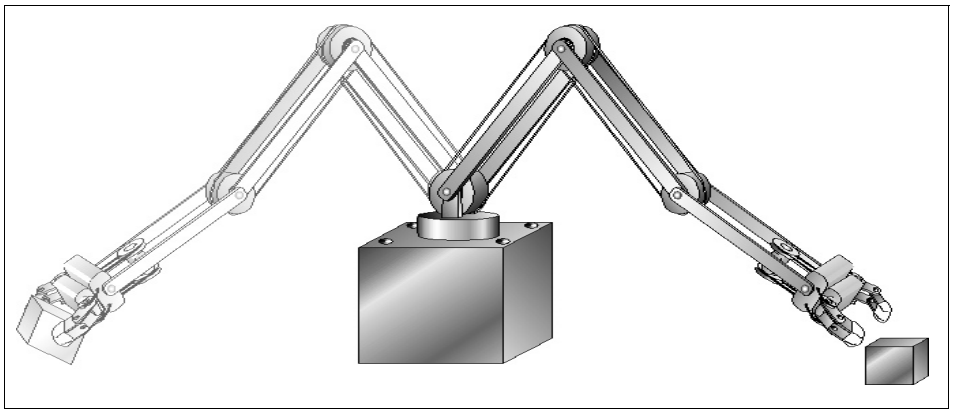


Figure 15. Retain Mode Graphics

7.1.4 Technical Content of the IBM graPHIGS Product

The graPHIGS API is architected to allow application programs to run in a distributed environment, in the same manner as the X Window System. This distributed capability is achieved by partitioning the graPHIGS API into a shell (client) and a nucleus (server). The graPHIGS API shell is tightly coupled to the application and provides communication between the application and the nucleus. The nucleus controls the resources used by the application program, such as graphical data storage.

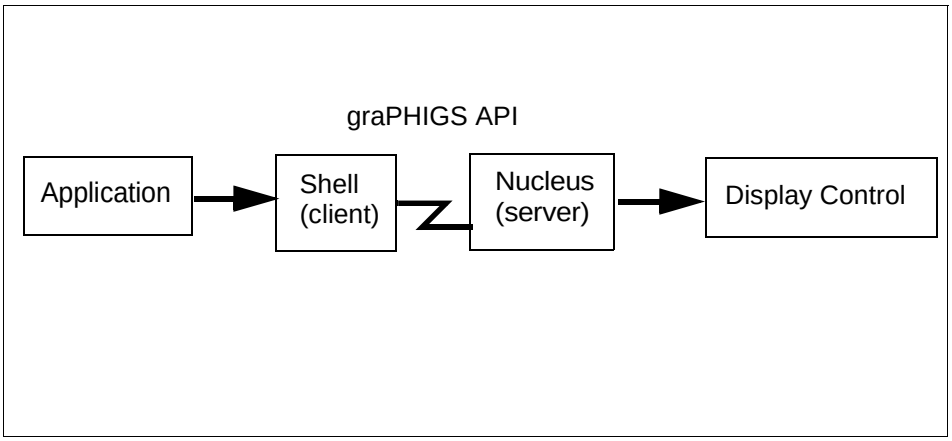


Figure 16. graPHIGS Shell and Nucleus

By utilizing the structure of the graPHIGS API, an application program may be partitioned into several application processes. Each process has its own shell. Collectively, they can communicate with a single graPHIGS API nucleus to share graphics resources.

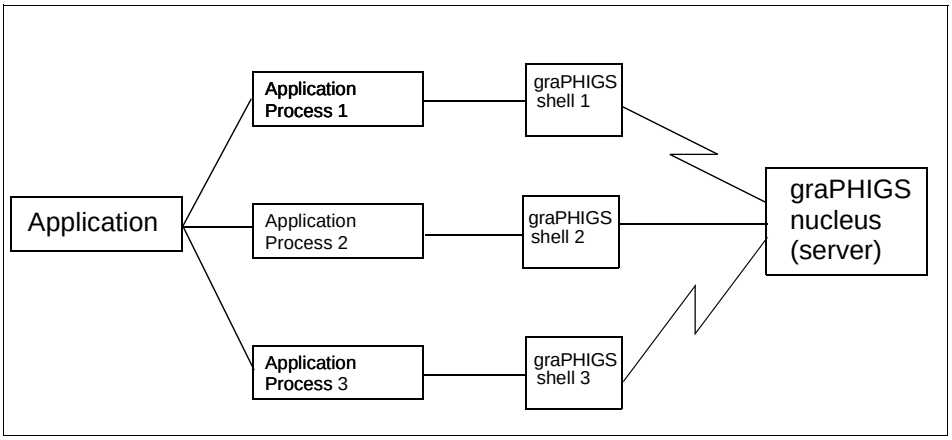


Figure 17. Distributed Capability of graPHIGS

In addition, graPHIGS provides the following functions:

- Basic Primitives:
  - 2D and 3D text (annotation text and geometric text)
  - Markers
  - Lines

    - Polygons
- Advanced primitives:
    - Triangle strips
    - Quadrilateral meshes
    - Concave and multi-contour polygons
    - Non-Uniform Rational B-Spline (NURBS) curves
    - Trimmed and untrimmed NURBS surfaces
- Line-on-line highlighting
- User-defined clipping volumes
- Antialiased primitives
- Transparency
- Dithering
- Morphing
- Explicit traversal control (for immediate and mixed mode rendering)
- Archiving
- Conferencing
- National language (font support):
    - Unicode standard
    - Traditional Chinese
    - Hangul (Korean)
    - Kanji (Japanese)
- 12-bit visual support
- Hardcopy support for HPGL2 devices

The graPHIGS Development Environment offers:

- Example programs
- Application development tools including:
    - Font editor
    - Debugger
    - Tutorial
- Installation verification program
- Online hypertext information

7.1.5 ISO PHIGS

In addition to the graPHIGS API, the IBM PHIGS products include the ISO/ANSI PHIGS Fortran and C language bindings. The current release of IBM's ISO PHIGS product does not fully adhere to the Federal Information Processing Standard (FIPS) because the incremental spatial search function is not supported. Documentation on the FIPS standard is contained in FIPS 153 available from the United States Federal Government. For complete documentation on adherence to the ISO PHIGS standards, refer to *Introducing the graPHIGS Programming Interface*, SC33-8190.

7.1.6 Graphical Kernel System (GKS)

Although the Graphical Kernel System (GKS) is a 2D graphics API, IBM's implementation, called the GKS-Compatibility Option (GKS-CO), is in fact an extension to the graPHIGS API. GKS-CO adheres to the ISO/ANSI GKS standard. IBM's implementation delivers a level of function available in the GKS standard known as level 2C, which is the highest level of ISO/ANSI compatibility. Be aware that the default values in GKS-CO are based on the graPHIGS default values and therefore may be different than those defined in other standard versions of GKS. Both Fortran and C programming language bindings are available for the GKS-CO on the RS/6000.

7.2 Basic Terminology and Concepts

This section briefly describes some graPHIGS terms and concepts that are used later in this chapter and throughout the graPHIGS manuals.

7.2.1 Common Terms

Some of the most common terms used are these.

- Primitives:

  The graPHIGS API defines a graphic system architecture that enables you to create, modify and display graphical objects. A sequence of elements defines an object, including output primitives, attributes, and transformations. Basic output primitive elements include lines, markers, polygons, and text definitions.

- Attributes:

  Attributes define the characteristics of an output primitive. An attribute, for example, may define the color or size of a polymarker primitive.

- Structures:

Graphical primitives and attributes group together to form structures. A structure may represent the geometry of an object, as well as information regarding the appearance of that object. Elements may be inserted into, or deleted from, structures at any time in an operation called structure editing. This editing capability minimizes the need to redefine data in order to modify it. Structures may be linked in a number of ways, including geometrically, hierarchically, or characteristically, according to your application needs.

- Input:

  The graPHIGS API supports a wide range of input devices and provides the essential tools for application interaction. Input devices operating synchronously or asynchronously relay information to the application, which in turn responds by defining, editing, or displaying the graphical data. The graPHIGS API supports six classes of input devices. These classes represent generic physical devices that differ from one another by the type of data they return to the application. Input device classes include the following:

  - Locator
  - Stroke
  - Valuator
  - Choice
  - Pick
  - String

- Operating Modes:

  The graPHIGS API supports three modes of interaction (Operating Modes) that allow you to request and obtain data from a logical input device:

  1=REQUEST

  Your application prompts for input and then waits until the operator either enters the requested input or performs a break action which terminates interaction.

  2=SAMPLE

  Your application obtains the current values of the input device by explicitly sampling it.

  3=EVENT

  An asynchronous environment is established between your application and a chosen device. In this mode, both your application and any

corresponding device operate independently of each other with the help of a centralized input queue.

- Workstations:

  The term workstation refers to an abstraction of a physical graphics device. It provides the logical interface through which your application program controls physical devices.

  The graPHIGS API provides an environment that supports multiple workstations. How your application interacts with a particular workstation depends on the interactive capabilities of that workstation and the design of your application.

  The graPHIGS API supports three categories of workstations: INPUT, OUTPUT, and OUTIN. The capabilities of a workstation determine its category. For example, an INPUT workstation, such as a digitizer, provides only input, while an OUTPUT workstation, such as a plotter, generates only output. The OUTIN workstation, on the other hand, is an interactive design station that offers the capability of providing both input and output.

  On the X Window System, this workstation concept corresponds to a window containing a graPHIGS drawable, and available workstation types are X, XSOFT, XLIB, and XDWA.

- Inquiry Subroutines:

  Inquiry subroutines allow the application programmer to access the program data contained in state lists, description tables or structures. They are useful for determining both error conditions and device characteristics.

- States:

  graPHIGS defines several states to track the environment in which your application runs:

  - The System State:

    Defines whether the graPHIGS API has been activated or deactivated using the Open graPHIGS (`GPOPPH`) or Close graPHIGS (`GPCLPH`) subroutines, respectively. No other subroutine calls can be accessed until the system is open.

  - The Workstation State:

    Defines whether a workstation has been activated or deactivated by using the Create Workstation (`GPCRWS`) or Close Workstation (`GPCLWS`) subroutines respectively.

The graPHIGS API structure display subroutines can only be used if a workstation is open.

- The Structure State:

  Defines whether the graPHIGS API display structure is open and able to be modified or closed and unavailable for modification. A structure is opened and closed with the Open Structure (GPOPST) and Close Structure (GPCLST) subroutines. Graphics primitives and attributes can only be created if the structure state is open.

7.2.2 Graphical Resources

The graPHIGS API system consists of graphical resources with subroutines to control and utilize them. Graphical resources available to your application include:

- Structure Stores:

  Collections of structures.

- Workstations:

  (See Workstation 7.2.1, “Common Terms” on page 92.)

- Font Directories:

  Collections of displayable characters, typically used for different languages, appearances, or special-purpose user-defined symbols.

- Image Boards:

  Data collections for displaying images.

These resources are controlled by the graPHIGS nucleus (server).

7.2.3 Resources and Capabilities

A typical use of the resources and capabilities of the graPHIGS API by your application might include the following:

- Create graphical data:

  Creating structures containing elements and attributes that define displayed objects. Those objects may include:

  - Figures formed by lines and filled areas (such as a robot arm created by lines in different locations and colors)
  - Text such as labels, menus of options, and status information
  - Images

- Open and control a workstation:

Identifying the current workstation and setting its values (such as the color table) to those required for your application. This includes viewing information that controls the parts of visible graphical data and their appearance on the display.

- Define the displayed content:

  Associating structures of graphical data with a workstation so that the workstation can draw the objects. The display content is modified by editing structures or changing the view tables used to display the graphical data.

- Accept user input:

  This allows you to provide input to the application, typically to change the displayed objects. For example, to change an object, a user might pick the object, select a choice provided by the function keys, or indicate a point or position using a tablet.

The graPHIGS API resources and facilities let you create a graphics application to display objects that a user can modify interactively. Your application can run in numerous environments, and inquiry subroutines provide information that enable your application to adapt to different hardware capabilities.

7.2.4 Subroutines

Types of subroutines available to your application include:

- Control subroutines:

  Provide the basic control functions. These allow your application to open and close the graPHIGS API and allocate, share, control, and free graphical resources.

- Structure subroutines:

  Provide control of structures, which are groupings of graphical elements. You can create, delete and modify structures. Modifications include changes to the whole structure content (such as emptying a structure)as well as changes to the elements in the structures (such as deleting or adding a single element in a structure).

- Element subroutines:

  Provide the basic drawing facilities. These include primitives (such as lines, text, filled areas, curves, and surfaces) and their attributes (color, size, and line type).

- Workstation subroutines:

Provide control of a workstation's facilities, such as setting a color table or view table entries.

- Display subroutines:

  Provide the controls to display structure content on a workstation.

- Input subroutines:

  Provide control of a workstation's input devices so that users may provide input to the application. For example, a user may want to pick a displayed object, provide text from a keyboard, or provide point or stroke (multiple points) input.

- Image subroutines:

  Define image content and controls, such as color mappings and image display.

- Inquiry subroutines:

  Provide your application information about the capabilities, state of resources and the systems

7.3 IBM Implementations

graPHIGS is implemented in two ways by IBM:

- IBM Softgraphics technology
- Hardware-accelerated graphics adapter

7.3.1 Softgraphics Technology

The Softgraphics technology uses the RS/6000 software for all graphics rendering operations. This provides full 3D function at lower cost. (See also 6.3, "Softgraphics" on page 84.)

graPHIGS applications may use the Softgraphics technology by simply opening a graPHIGS workstation of type XSOFT.

- Full functionality and adapter independence:

  XSOFT offers full-function graPHIGS support. This includes not only the basic graphics functions but also includes such capabilities as HLHSR (hidden line, hidden surface removal), lighting and shading, depth-cueing, transparency, blending, and antialiasing. Previously, this functionality was not available across all IBM RS/6000 workstations. It is now supported on all past and current IBM RS/6000 graphics workstations.

- No display hardware required:

Unique to graPHIGS is the fact that the XSOFT pipeline and rasterizer can be used when there is no display hardware at all. The graPHIGS IMAGE type workstation uses the graPHIGS Softgraphics routine to produce hardcopy output for PostScript and IOCA (IBM's Image Object Content Architecture) printer devices.

- Performance:

  Be aware that performance scales with the amount of CPU available. Softgraphics graPHIGS may provide an entry-level production environment on an RS/6000 43P 140 or 150 (with POWER GXT255P), which is beyond the POWER GXT1000 level of performance for applications like drafting. In the following table, PLBwire and PLBsurf results show the 3D wire frame and solid surface performance, respectively. The higher the number the better. For information of performance benchmarking, see Chapter 11, "Benchmarking" on page 205.

Table 10. Example of Performance Results with Softgraphics

Machine Type	Graphic Adapter	Using Softgraphic	PLBWire	PLBSurf	Comments
43P-150	GXT255P	Y	178.6	75.4	on AIX 4.3.2
43P-133	GXT255P	Y	91.5	31.9	older machine AIX 4.1.5
42T	GXT1000-2	N	112.9	153.2	older 3D adapter with AIX V4.1.5
42T	GXT500D	N	89.4	86.1	older 3D adapter with AIX V4.1.5

- graPHIGS Shared Memory Image (GP-MIT-SHM) X Extension:

  X has a limit on the size of the protocol buffer, requiring that the XSOFT rendering target is chunked before being transferred to the X server. This restriction impacts the visual quality of an update as well as interactive performance. The GP-MIT-SHM X extension bypasses the client-server protocol by transferring the rendering target in one piece through shared memory. This solution improves the image quality and performance. (As described below, this is valid as long as the graPHIGS nucleus and the X server run on the same machine.)

- Xstations and Distributed X Environments:

  Whenever the graPHIGS nucleus and the X server are not executing on the same machine, for example, Xstations, X on NetworkStations, X

emulators on PC, and other distributed X environments, the XSOFT type workstation cannot take advantage of the GP-MIT-SHM X extension, even if the extension is loaded. Interactive performance of the XSOFT workstation is severely impacted without this feature. However, the GP-MIT-SHM X extension can be used in the distributed graPHIGS configuration when the graPHIGS shell and nucleus are distributed but the X server is executing on the same machine as the nucleus. Distributing graPHIGS in this manner does not limit the performance of the XSOFT workstation.

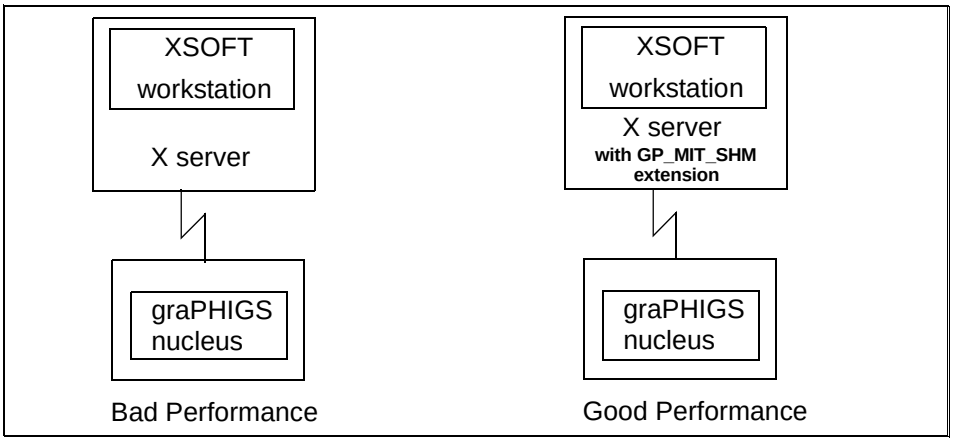


Figure 18. Importance of the GP_MIT_SHM Extension

On the Xstations, the X workstation type does not support lighting and interpolated shading. The XSOFT workstation provides this support that was not previously available. This configuration may be right for the occasional or view-only user of an application, or for applications with low frame rates or those that require minimal user interactions or user interactions implemented entirely through X.

Applications also have the option to use the X workstation type to provide interactive performance in the distributed environment and a second XSOFT workstation to provide a more advanced rendering. This is possible through the graPHIGS ability to share Structure Stores among more than one workstation.

7.3.2 Hardware-Accelerated

All or part of the rendering to the graPHIGS API is done by the adapter hardware and microcode. Section 1.2, “Different Classes of Graphics Adapters” on page 9, describes the role of class II and III graphics adapters.

7.3.3 Explicit Traversal Control for Immediate Mode Graphics

Immediate mode allows direct control of traversal processing and the graphics resources of the workstation. The graPHIGS extension is called Explicit Traversal Control (ETC). Using ETC enables an application to update a portion of the model's geometry without redrawing the entire data structure or rerendering the entire model. Instead, only the changed portion of the model is redrawn. Increased control over graphical resources, such as frame buffers and the Z-buffer, is also possible through ETC.

For information about the immediate mode graphics, please see 9.1.1, "Immediate Mode Graphics" on page 130.

7.3.4 Multi-Threaded Graphics Pipeline

Applications that use the graPHIGS API on the RS/6000 Models 240, 260, F40, and F50 SMP workstations with supported graphics accelerators are able to take advantage of the multiprocessor capabilities of those systems to increase application performance. The graPHIGS API has been enhanced to include support for a multi-threaded graphics pipeline, which will automatically be invoked to utilize up to four processors simultaneously on these SMPs.

Animations and interactive model manipulation would likely benefit the most from the performance improvements. These improvements can be observed without any changes to your application because of the graPHIGS API's embedded client/server capabilities.

7.3.5 graPHIGS on GXT3000P PCI Graphics Accelerator

The graPHIGS API has been enhanced to improve rendering performance. There is also support for the new high-performance RS/6000 POWER GXT3000P PCI graphics accelerator which attaches to the RS/6000 43P Models 150 and 260. The RS/6000 POWER GXT3000P graphics accelerator for 3D visualization marks a breakthrough in performance and functionality for design and visualization solutions. The graphics subsystem matched with IBM Power3 and PowerPC-based workstations delivers outstanding speed and performance for demanding 3D applications.

7.4 Configuration

The PHIGS for AIX 4.3 product is included in Version 4.3 of AIX. The installation of the filesets is done with the usual system administration tools, such as SMIT.

7.4.1 Filesets

There are four filesets as follows:

PEX_PHIGS.graPHIGS.rte	graPHIGS Runtime Environment
PEX_PHIGS.dev	graPHIGS Device Dependent Software
PEX_PHIGS.graPHIGS.adt	gP Application Development Toolkit
PEX_PHIGS.graPHIGS.fnt	graPHIGS Fonts

The first two filesets must be installed to run any graPHIGS applications on the system. To develop graPHIGS programs on the system, it is necessary to install the graPHIGS Application Development Toolkit fileset.

7.4.1.1 graPHIGS Runtime Environment

The fileset PEX_PHIGS.graPHIGS.rte contains the following modules:

PEX_PHIGS.graPHIGS.rte.base Base Runtime Environment

This provides:

- Runtime code under /usr/lpp/graPHIGS/bin/
- Library files such as libgP.a under /usr/lib/ and /usr/lpp/graPHIGS/lib/
- Geometric text fonts /usr/lpp/graPHIGS/fonts/afm0*.sym
- Sample profile and X defaults file and an installation verification script (runivp) under /usr/lpp/graPHIGS/etc/
- An archive utility and a script to run it under, /usr/lpp/graPHIGS/bin/ and /usr/bin, respectively

PEX_PHIGS.graPHIGS.rte.pipe Pipeline Runtime Environment

This provides Softgraphics pipeline code under /usr/lib/.

PEX_PHIGS.graPHIGS.rte.soft Soft Runtime Environment

This provides soft graphics code under /usr/lpp/graPHIGS/bin/ and /usr/lib/.

PEX_PHIGS.graPHIGS.rte.rnuc Remote Nucleus Support

This provides command files for remote nucleus and scripts to run them under /usr/lpp/graPHIGS/bin/ and /usr/bin/.

PEX_PHIGS.graPHIGS.rte.6098 6098 Support

This provides command files for 6098 with FDDI feature under /usr/bin/.

PEX_PHIGS.graPHIGS.rte.plot Plotter Support

This provides configuration files and utilities for plotters under /usr/lib/lpd.

7.4.1.2 graPHIGS Device Dependent Software

The fileset PEX_PHIGS.dev provides the following device dependent software modules:

PEX_PHIGS.dev.pci.14103c00	GXT250P/GXT255P
PEX_PHIGS.dev.pci.14105400	GXT500P/GXT550P
PEX_PHIGS.dev.pci.14105e00	GXT800P
PEX_PHIGS.dev.pci.1410b800	GXT2000P (N.A.)
PEX_PHIGS.dev.pci.14108e00	GXT3000P
PEX_PHIGS.dev.mca.8f61	GXT800M
PEX_PHIGS.dev.mca.8fbc	GXT1000
PEX_PHIGS.dev.mca.8ee3	GT4
PEX_PHIGS.dev.buc.00004002	GXT500

7.4.1.3 graPHIGS Application Development Toolkit (ADT)

The fileset PEX_PHIGS.graPHIGS.adt contains the following modules of the graPHIGS Application Development Toolkit:

PEX_PHIGS.graPHIGS.adt.include graPHIGS ADT Include Files

This provides include files, afm*.h, under /usr/include/.

PEX_PHIGS.graPHIGS.adt.samples graPHIGS ADT Samples

This provides the following subdirectories under /usr/lpp/graPHIGS/samples/.

- Samp/ contains simple sample sources in C, FORTRAN, Pascal.
- Gettingstarted/ contains sample programs written in *The graPHIGS Programming Interface: Getting Started*, SC33-8198.
- Widgets/ contains graPHIGS widget and its samples.
- The others, such as cbinding/, contain other samples.

PEX_PHIGS.graPHIGS.adt.clients graPHIGS ADT Clients

This provides the graPHIGS fonts editor (fe), a sample debugger (gPdbg), and a collection of slicing and contouring demo programs (sliceriso).

PEX_PHIGS.graPHIGS.adt.tutor graPHIGS ADT Tutorial

This provides gPtutor and its README under /usr/lpp/graPHIGS/clients/gPtutor/. gPtutor is linked to /usr/bin/gPtutor.

PEX_PHIGS.graPHIGS.adt.gks graPHIGS ADT GKS Library

This provides /usr/lib/libgksco.a containing the GKS-CO Library

7.4.1.4 graPHIGS Fonts

The fileset PEX_PHIGS.graPHIGS.fnt contains the following fonts modules:

PEX_PHIGS.graPHIGS.fnt.JP	Kanji (Japanese) Fonts
PEX_PHIGS.graPHIGS.fnt.KR	Hangul (Korean) Fonts
PEX_PHIGS.graPHIGS.fnt.SC_EUC	Simplified Chinese Font, EUC based
PEX_PHIGS.graPHIGS.fnt.TW	Traditional Chinese Fonts
PEX_PHIGS.graPHIGS.fnt.uni	Unicode Fonts

Each provides the corresponding font files under /usr/lpp/graPHIGS/fonts. A name of font file is like afm0*.sym.

7.4.2 Installation

Once SMIT has returned from the installation process, you should run the verification script to make sure that you have a working environment.

7.4.2.1 The Installation Verification Script: runivp

Type in an aixterm or other terminal window with the DISPLAY variable correctly set :

```
sh /usr/lpp/graPHIGS/etc/runivp
```

then a graPHIGS window comes up on the screen if the installation has successfully completed. In this window, a robot arm is displayed. As the directions in the window state, pressing the F1 key makes the arm rotate around. To exit after the animation, press any function key.

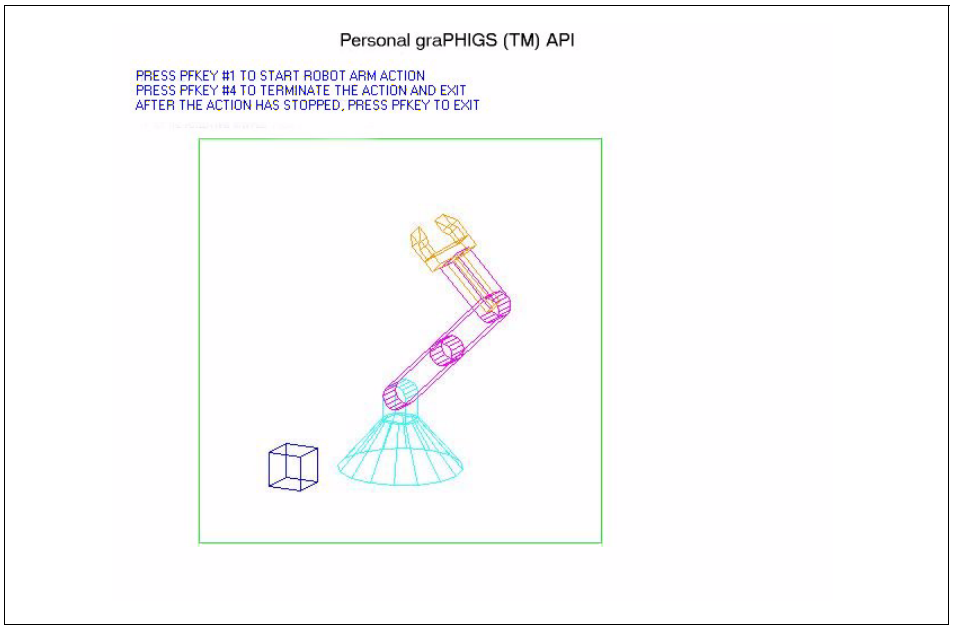


Figure 19. Initial Image of runivp

7.4.2.2 xinit Options for GXT3000P, 800P, 550P, 550P, 1000, and 255P

The graPHIGS window must be created in the color planes (layer 0, while the overlay plane's layer is 1). For the best performance, it is recommended that the X root window be created in the overlay planes.

If the X and graPHIGS windows are both created in the color planes, manipulating the X window causes an exposure event, and more graPHIGS redraws may occur. On the other hand, if X starts in the overlay planes, there will be fewer exposure events, and fewer graPHIGS are needed.

Starting X in the overlay planes is only available when the application specifies one of the supported visuals for the color planes and uses it to create the graPHIGS window. If the application does not specify visual to graPHIGS window, graPHIGS uses the visual associated with the root window and X should then be started in the color planes in this case.

Therefore, the necessary options for the `xinit` command should be chosen accordingly:

- For GXT3000P, 800P, 550P, 550P, 1000:
  - If the application specifies supported visuals for the color planes and uses it to create the graPHIGS window, type:

```
xinit -- -x dbe -x abx
```

X then starts in the overlay planes by default.

- If the application does not pass a selected visual to the graPHIGS window, the `-layer 0` option is required to start X in the color planes, and more options may be specified depending on the frame buffer configurations which the application uses:
  - The 8-bit visual:

    ```
    xinit -- -x dbe -x abx -layer 0
    ```

  - The 24-bit DirectColor visual:

    ```
    xinit -- -x dbe -x abx -layer 0 -d 24 -cc DirectColor
    ```

  - The 24-bit TrueColor visual:

    ```
    xinit -- -x dbe -x abx -layer 0 -d 24 -cc TrueColor
    ```

- For GXT255P:

  Almost the same as the above, but the `-x abx` option should not be specified.

The extensions `-x dbe` and `-x abx` can also be specified to the environment variable EXTENSIONS in /usr/lpp/X11/defaults/xserverrc file. If the OpenGL product is also installed on the same machine, these extensions may already be specified in this file.

If you use the CDE, modify the last line in /usr/dt/config/Xservers to specify command line options to `xinit`, such as `-layer 0` or `-d 24`. The line is originally:

```
:0 Local local@console /usr/lpp/X11/defaults/xserverrc -T -force :0
```

Insert the `-layer 0` option, for example, as follows:

```
:0 Local local@console /usr/lpp/X11/defaults/xserverrc -layer 0
-T -force :0
```

7.4.2.3 xinit Options for XSOFT Workstation Type

As mentioned in 7.3.1, “Softgraphics Technology” on page 97, the GP-MIT-SHM extension is only available to the XSOFT workstation device driver when the graPHIGS nucleus is executing on the same machine as the X server and the X server has the GP-MIT-SHM extension loaded.

To load the GP-MIT-SHM extension, start the X server with `-x gpshm` command line option. Alternately, to automatically load the GP-MIT-SHM extension, change the following line in the /usr/lpp/X11/defaults/xserverrc file

```
EXTENSIONS=""
```

to

```
EXTENSIONS="-x gpshm"
```

7.4.2.4 The External Defaults File (EDF)

The External Defaults File contains records which consist of User Defined Specifications (UDSs). The UDSs in this file allow you to change user default options at runtime without recompiling or rebuilding your application. The API accesses the External Defaults File as follows:

The file must be named PROFILE, or must be specified in the `gPPROFILE` environment variable. At the initialization phase, the graPHIGS API searches for a PROFILE in this order:

1. `gPPROFILE` Environmental Variable:

   The `gPPROFILE` environment variable allows you to specify an alternate file name or an alternate directory path containing the file, PROFILE, as the external defaults file.

   If the `gPPROFILE` environmental variable is not defined, is defined with an invalid file name or directory name, or there is no file named PROFILE in the defined valid directory name, the search continues.

2. Current Directory:

   The current directory is searched for a file named PROFILE. If there is no file named PROFILE in the current directory, the search continues.

3. /usr/lpp/graPHIGS/etc Directory:

   The graPHIGS API provides a sample External Defaults File as /usr/lpp/graPHIGS/etc/PROFILE.

The following examples illustrate a way of specifying the connection identifier and workstation type with PROFILE. It says that any connection to a workstation of type X should use the local display:0.

.

```
AFMMNICK TOWSTYPE=X,
         TOCONNID=:0
```

Figure 20. Sample of PROFILE

About the detail of this file, see Chapter 7. Controlling the Environment with Defaults and Nicknames in *The graPHIGS API: Technical Reference*, SC33-8193.

7.4.2.5 $(HOME)/.Xdefaults

graPHIGS is the default application name of graPHIGS application in X Window System. Using the application name, in the .Xdefaults file in your $HOME directory, you can define resources for your graPHIGS application such as the initial position, initial size, window title, icon name, icon bitmap, minimum aspect ratio, maximum aspect ratio, minimum window size, window border color, and the window border width.

Entries in the .Xdefaults file to force the window to be a square of 500 pixels located in the upper-left corner and have `My Title` in the title bar would look like this:

```
graPHIGS.geometry:   500x500+0+0
graPHIGS.title:          MyTitle
```

Figure 21. A Sample of $HOME/.Xdefaults

If you want to be more specific and associate properties to one specific graPHIGS application, the default application name, graPHIGS, can be changed to another using the above PROFILE stanza:

```
AFMMNICK  TOWSTYPE=X,
          TOCONNID=:0,
          PROCOPT=((XNAME,MyName))
```

Figure 22. Define the Application Name in PROFILE

Then, the entries in the .Xdefaults file in your $HOME directory would look like this:

```
MyName.geometry:   500x500+0+0
MyName.title:          MyTitle
```

Figure 23. Another Sample of $HOME/.Xdefaults

The following table contains available resources for graPHIGS window:

Table 11. Resources for graPHIGS Applications

Resource Name	Description	graPHIGS API Default Action
geometry	Initial Window Geometry	Half the width of the screen and with the same aspect ratio

Resource Name	Description	graPHIGS API Default Action
minSize	Minimum Window Size	100x100
title	Window Title	Blank Title
iconName	Icon Name	Default to the Window Name
iconBitmap	Icon Bitmap File Name	No icon bitmap
aspectMinimum	Minimum Aspect Ratio	None specified
aspectMaximum	Maximum Aspect Ratio	None specified

With the following definitions, the graPHIGS window keeps a square shape when it is resized:

```
graPHIGS.aspectMinimum:   1x1
graPHIGS.aspectMaximum:   1x1
```

After you have edited your $HOME/.Xdefaults file, execute the `xrdb` command to reload the file as follows:

```
xrdb $HOME/.Xdefaults
```

If you would like to remove the window title bar and the window resize frame, add the following line in $HOME/.Xdefaults; then restart the Motif window manager or CDE window manager:

```
(for Mwm) Mwm*graPHIGS*clientDecoration:  none
(for CDE) Dtwm*graPHIGS*clientDecoration:  none
```

If you need to resize the frame, replace `none` with `resize` in the above lines.

7.4.2.6 SYSPRINT

The name SYSPRINT is often used as the name of error log file where the graPHIGS API outputs error messages. You can change this to your favorite name through `GPOPPH()` subroutine as follows:

```
GPOPPH('SYSPRINT',0);
```

This file helps with debugging the application. The graPHIGS API does not overwrite this file; that is, if this file exists, it adds messages to the file.

The following is an example of what an error message looks like:

```
Mon Nov 16 hh:mm:ss 1998 GPOPWS*AFM2047 XOPENDISPLAY FAILED - CHECK THE
GRAPHIGS CONNID.
```

This line contains the date and time, the name of the API subroutine associated with the error, the message identifier with string AFM followed by a four-digit message number, and the message text which briefly explains the error.

With the message identifier, you can look for the detail information in *The graPHIGS API: Messages and Codes*, SC33-8196. There you will find an explanation of the error and description of system action and programmer response as follows.

```
2047        XOPENDISPLAY FAILED - CHECK THE graPHIGS CONNID
```

Explanation: The X workstation is not able to connect to the X server ...

System Action: The workstation is not opened.

Programmer Response: Verify that the connection identifier `connid` is ...

7.5 Overview for Programming

Learning graPHIGS programming is necessary to develop a brand new application, but may also be useful to realize the porting of an existing one to a new API.

To learn or look into graPHIGS programming, the following books may help you:

- *The graPHIGS API: Understanding Concepts*, SC33-8191
- *The graPHIGS API: Getting Started V2R2.2*, SC33-8198

Part one of *Understanding Concepts* describes the basics of using the graPHIGS API and is especially suited for a first-time user of the graPHIGS API. Samples are included in every chapter of the basic section to give you hands-on experience with the graPHIGS API applications, but these are written in fortran language.

On the other hand, *Getting Started* is a small book, but consists of sample programs written in the C language with comments and explanations for each. This book is also suited for a first-time user of the graPHIGS API.

Some subroutines in these guide books are obsolete; that is, these have been used since graPHIGS Version 1 was available and have been replaced with other subroutines since graPHIGS Version 2.

During your study, if you have to check the syntax for a command or its arguments, refer to *The graPHIGS API: Subroutine Reference*, SC33-8140.

However, the descriptions in this book are fortran like and it is necessary to be careful whether the type of an argument is a number or a pointer if you write a program in C.

To learn more technical details, such as information about the integration with X or lists of contents of workstation description table, see *The graPHIGS API: Technical Reference*, SC33-8193.

7.5.1 graPHIGS Subroutines

Subroutine names are prefixed by GP followed by two or four capitals or digits. Especially, the names of inquiry subroutines are prefixed by GPQ. For example, `GPOPPH()`, `GPPL3()`, `GPQWDT()`. There are no constants and types specifically defined for graPHIGS. The types of variables which are passed to graPHIGS subroutines are either int or float (32-bit integer or floating point values).

7.5.1.1 Long Names by afmgp.h

To make the source code easy to read

```
#include <afmgp.h>
```

provides a long name for each graPHIGS subroutines and enumerations defined by these subroutines. For example, the names in the left-hand side can be replaced with the ones in the right-hand side in the following:

`GPOPPH`	gPOpengraPHIGS
`GPPL3`	gPPolyline3
`GPQWDT`	gPInquireWorkstationDescription

These long names match the titles in the subroutine reference pages, such as `Open PHIGS`, `Polyline 3`, `Inquire Workstation Description`. And, for example, the following enumeration is used to define those arguments:

```
/*-- gP_istyle Interior Style -- */
typedef enum {
              GP_HOLLOW = 1;
              GP_SOLID,
              GP_PATTERN,
              GP_HACTH,
              GP_EMPTY,
              GP_ISTYLE_INVALID = 0x7fffffff
            } gP_istyle;
```

With these long names, the line:

```
GPIS( 2 );
```

can be changed to:

```
gPInteriorStyle( GP_SOLID);
```

Before you get familiar with graPHIGS subroutines, here is an easy way to find out a brief description for each subroutines:

```
grep GPXXXX /usr/include/afmgp.h
```

Then, you can replace the short names with the corresponding long names in your sample or existing application source code to make it easier to read.

7.6 graPHIGS References

These documents are available:

- On the AIX Version 4.3 Base Documentation CD
- On the following IBM Web pages:

`http://www.rs6000.ibm.com/resource/aix_resource/Pubs/`

select **AIX Version 4.3 Documentation Library**

AIX Version 4.3 Base Documentation

List of Books

graPHIGS

There are nine documents, here are their titles and a brief description of each:

- *The graPHIGS API: Understanding Concepts*, SC33-8191:

  This guide helps you understand the use of the graPHIGS API functions in your application to create, display, and interact with graphics data. It contains two parts: basic and advanced. Part One describes the basics of using the graPHIGS API and is especially suited for a first-time user of the graPHIGS API. Included in every chapter of the Basic section are sample fortran subroutines to give you hands-on experience with the graPHIGS API applications.

  For the experienced graPHIGS API programmer, Part Two offers advanced functions and capabilities to further enhance your application program. It also expands upon some of the concepts introduced in Part One. Used in conjunction with other graPHIGS API publications, this guide helps you create complete graphics application programs.

- *The graPHIGS API: Technical Reference*, SC33-8193:

This reference provides technical information about the functions and limitations of the graPHIGS API and its supported workstations. It also contains reference information, both general and specific, about particular aspects of writing applications, namely on Character Set Facilities and on Defaults and Nicknames. The purpose of this manual is to provide application programmers with a comprehensive volume of the technical information they need to accurately code or modify applications using the graPHIGS API screen.

- *The graPHIGS API: Subroutine Reference*, SC33-8194:

  This reference manual contains the information you need to code your calls and to declare variables correctly. Each subroutine listed in this manual has information about error codes and functional relations to help you identify more readily the source of errors resulting from data, program flow. Each subroutine description explains the result to the subroutine call and a list of the errors associated with the subroutine.

- *The graPHIGS API: Customization and Problem Diagnosis*, SC33-8130:

  This manual is divided into two parts. Part One describes the hardware supported by the IBM Personal graPHIGS Application Programming Interface and hardware and software requirements. It explains how to customize the IBM Personal graPHIGS Programming Interface for optional environments and plotter support. Part Two provides information needed to diagnose problems in the IBM Personal graPHIGS Programming Interface and the graPHIGS gateway. The following information is also provided: guidelines for locating the symptoms of the graPHIGS API problem, techniques for collecting the supporting data required for further analysis, an Authorized Program Analysis Report (APAR) form, and a description of the information required for submitting the APAR.

- *The graPHIGS API: Messages and Codes,* SC33-8196*:*

  This book gives programmers a comprehensive guide to error messages that can result from installing, using, and maintaining the graPHIGS API.

- *The graPHIGS API: Getting Started,* SC33-8198*:*

  This book guides programmers who are familiar with the C language and an AIX editor through the steps of writing and running their first graPHIGS API programs. The book consists of sample programs with comments and explanations for each.

- *The graPHIGS API: Quick Reference*, SC33-8195:

  This booklet provides a quick reference for the graPHIGS API. It is intended as a supplement to graPHIGS Programming Interface Technical Reference, in which the subroutines are described in detail.

- *ISO PHIGS Subroutine Reference*, SC33-8140:

  This reference manual contains the information you need to code your ISO PHIGS calls and to declare variables correctly. Each subroutine listed in this manual has information about error codes and functional relations to help you identify more readily the source of errors resulting from data and program flow. Each subroutine description explains the result of the subroutine call and a list of the ISO PHIGS standard errors associated with the subroutine.

- *ISO PHIGS Quick Reference*, SC28-2705:

  This booklet provides a quick reference for the graPHIGS API. It is intended as a supplement to the ISO PHIGS Subroutine Reference, in which the subroutines are described in detail. To help you find information quickly in the reference book, each listed subroutine includes a page reference. This manual contains both a reference-by-function as well as an alphabetical-by-subroutine listing for both the C and fortran bindings.

Chapter 8. GL 3.2

This chapter describes the GL 3.2 API and IBM's implementation for AIX. This API has almost become obsolete as a 3D graphics API since OpenGL has taken its place in the industry. AIX Version 4.3.2 still supports this API for all previous 3D graphics adapters, but the GL 3.2 API should not be implemented in the graphics pipeline for the latest POWER GXT3000P.

IBM encourages GL 3.2 customers to migrate to OpenGL.

8.1 Definition

The Graphics Library (GL) API is a 3D API which was developed by Silicon Graphics, Incorporated (SGI) as a proprietary API for use on their graphics platforms. This API has been licensed to IBM and several other vendors. It was once very popular and used by many independent software vendors and was the basis for their 3D applications.

The strength of GL is immediate mode graphics. See 9.1.1, "Immediate Mode Graphics" on page 130, for an explanation of this term.

The biggest disadvantage of the GL API is that it does not provide interoperability in heterogeneous networked environments, and it is not easily portable even between the few platforms that support it.

For these reasons, and in response to the growing popularity of open systems, SGI initiated an effort to standardize GL so that it could be easily supported by multi-vendor platforms. This effort led to the development of an open 3D API standard called OpenGL, see Chapter 9, "OpenGL" on page 129.

8.1.1 Technical Content of GL 3.2

GL 3.2 is a set of graphics and utility subroutines that provide high- and low-level support for graphics as well as X Window System integration facilities. The GL 3.2 library provides the following features:

- Basic 3D rendering services including:
  - Flat or Gouraud shaded points, polylines, triangle mesh, polygons
  - Matrix and Attribute Stacks
  - Depth (Z-buffering)
- Non-Uniform Rational B-Spline (NURBS) support (curves and surfaces)

- Circle and arc capabilities
- Concave polygon capabilities
- Antialiased points and lines
- Display list creation and manipulation facilities
- Depth-cueing effect
- Logical image operations
- Picking and selecting operations
- Integration with the X Window System including:
  - Window control (initialization, size change, window title, icon, window mapping)
  - Cursor management
  - Pop-up menu creation
  - Event handling
  - Color map control
  - Query about system resource
  - Renderer access to X11 fonts

8.2 IBM Implementation

IBM's GL implementation is integrated with AIXwindows. This is significant because IBM was one of the first vendors to successfully integrate GL with an open system such as the X Window System. IBM's offering of GL does not, however, support a distributed computing environment.

IBM's current GL implementation is based on SGI GL Version 3.2 and is provided to support applications that continue to require that interface.

C and fortran bindings for GL are supported on the RS/6000 through hardware-acceleration only. IBM's offering of GL does not support the software rendering capability.

Note that the POWER GXT3000P does not support GL 3.2. Therefore, any GL 3.2 application has to be migrated to OpenGL to run on GXT3000P. GL 3.2 for AIX Version 4.3.2 is supported on systems with the following graphics adapters (or subsystems):

- POWER Gt4e, Gt4, Gt4i, Gt4x, Gt4xi
- POWER GXT500, GXT500D

- POWER GXT500P, GXT550P, GXT800P (with or without texture memory)
- POWER GXT800M, GXT1000

8.3 Configuration

OpenGL and GL 3.2 for AIX are standard features of IBM AIX Version 4.3, while these has been offered as Licensed Program Products for IBM AIX Version 4.2 or 4.1.

8.3.1 Filesets

The GL 3.2 products includes three filesets :

OpenGL.GL32.rte	GL Runtime Environment
OpenGL.GL32.dev	GL Device Dependent Software
OpenGL.GL32.adt	GL Application Development Toolkit

The first two filesets must be installed to run GL 3.2 applications on the system. To develop GL 3.2 programs on the system, it is necessary to install the GL Application Development Toolkit fileset.

8.3.1.1 GL Runtime Environment

The fileset, OpenGL.GL32.rte, contains only one module:

OpenGL.GL32.rte.base GL Base Runtime Environment

This provides:

- /usr/lpp/GL/README
- Run time code: /usr/lpp/OpenGL/bin/loadR5Proc
- Library files: /usr/lib/libgl.a, /usr/lib/libfgl.a
- /usr/lpp/X11/bin/GLcmap and /usr/lpp/X11/bin/GLcmap_R5
- A directory link for compatibility purpose : /usr/lpp/ibmgl -> /usr/lpp/GL

8.3.1.2 GL Device-Dependent Software

The OpenGL.GL32.dev fileset provides the following device-dependent software modules:

OpenGL.GL32.dev.pci.14105400	GXT500P/GXT550P
OpenGL.GL32.dev.pci.14105e00	GXT800P
OpenGL.GL32.mca.8ee3	GT4 family
OpenGL.GL32.mca.8f61	GXT800M
OpenGL.GL32.mca.8fbc	GXT1000
OpenGL.GL32.buc.00004002.com	GXT500 (Common)
OpenGL.GL32.buc.00004002	GXT500

8.3.1.3 GL Application Development Toolkit (ADT)

The OpenGL.GL32.adt fileset contains the following ADT modules:

OpenGL.GL32.adt.include GL ADT Include Files

This provides /usr/include/gl/gl.h, /usr/include/gl/device.h, and so on. These files are also linked to /usr/include/.

OpenGL.GL32.adt.demos GL ADT Demos

This provides three demo programs and a README file. See 8.3.3.1, “Demo Programs in /usr/lpp/GL/demo/” on page 120.

OpenGL.GL32.adt.samples GL ADT Sample Source

This provides a number of samples source code and a README file. See 8.3.3.2, “Sample Source Code in /usr/lpp/GL/examples/” on page 121.

OpenGL.GL32.adt.util GL ADT Utilities Source

This provides a number of utility programs with source code and README files. See 8.3.3.3, “Utility Programs in /usr/lpp/GL/utilities/” on page 123.

8.3.2 Installation

The installation of the GL 3.2 product is performed using the standard AIX applications such as SMIT.

8.3.2.1 X Server Extensions

To run GL 3.2 applications on adapter such as the GXT1000, GXT5xx or GXT800, the X Windows Ancillary Buffer Extension (ABX) and the X Windows Double Buffer Extension (DBE) should be loaded when the X server starts. To start-up manually, use `xinit` command-line options as follows:

```
xinit -- -x abx -x dbe
```

Or to start-up automatically under `xinit`, edit /usr/lpp/X11/defaults/xserverrc and change the line reading

```
EXTENSIONS=""
```

to

```
EXTENSIONS="-x abx -x dbe"
```

If the OpenGL product is also installed on the same machine, these extensions are specified.

DBE is an official X standard and has replaced the non-standard X Multi-Buffer Extension (MBX).

Note

Starting the X server with `-x mbx` instead of `-x dbe` still currently works on an AIX Version 4.2.1 system, but not on an AIX Version 4.3.2 system.

If `-x abx` is not specified, you will find the following error message when you run a GL 3.2 application.

```
Xlib:  extension "xAncillaryBufferExtension" missing on display ...
Xlib:  extension "xAncillaryBufferExtension" missing on display ...
Xlib:  extension "xAncillaryBufferExtension" missing on display ...
winopen: PseudCol Visualgl: winopen: 1345-072 The requested X-Windows
Server extension is not installed
```

And if `-x dbe` is not specified, you will find the following error message.

```
Xlib:  extension "DOUBLE-BUFFER" missing on display ...
Xlib:  extension "DOUBLE-BUFFER" missing on display ...
```

To check a list of extensions, the currently loaded to the X server, run `xdpyinfo` (`/usr/bin/X11/xdpyinfo`) on local system, or run it with `the -display` option on a remote system.

8.3.2.2 Environment Variable

With AIX Version 4.3.1, a modification is brought to GL 3.2 to enable the default X cursor (black with white border) to appear instead of the hardcoded red cursor that GL 3.2 normally creates.

To avoid the red cursor, simply export the following environment variable:

(for ksh) `export _GL_USE_UNDERLYING_CURSOR=TRUE`

(for csh) `setenv _GL_USE_UNDERLYING_CURSOR TRUE`

8.3.3 Demo Programs, Sample Source Code and Utilities

This section describes demo programs, samples, and utilities provided with the GL 3.2 product.

8.3.3.1 Demo Programs in /usr/lpp/GL/demo/

The following list shows the program file names with a brief description of the purpose of each program. These demo programs can be used to verify the correct installation of the GL product.

Example Program	Purpose
lorenz	Fast Sphere Demo RIGHTMOUSE click will exit demo.
robotarm	Animation/Lighting Demo RIGHTMOUSE click will exit demo. LEFTMOUSE held down will animate jaws.
lmodtest	Lighting Demo explanation is displayed on the screen.

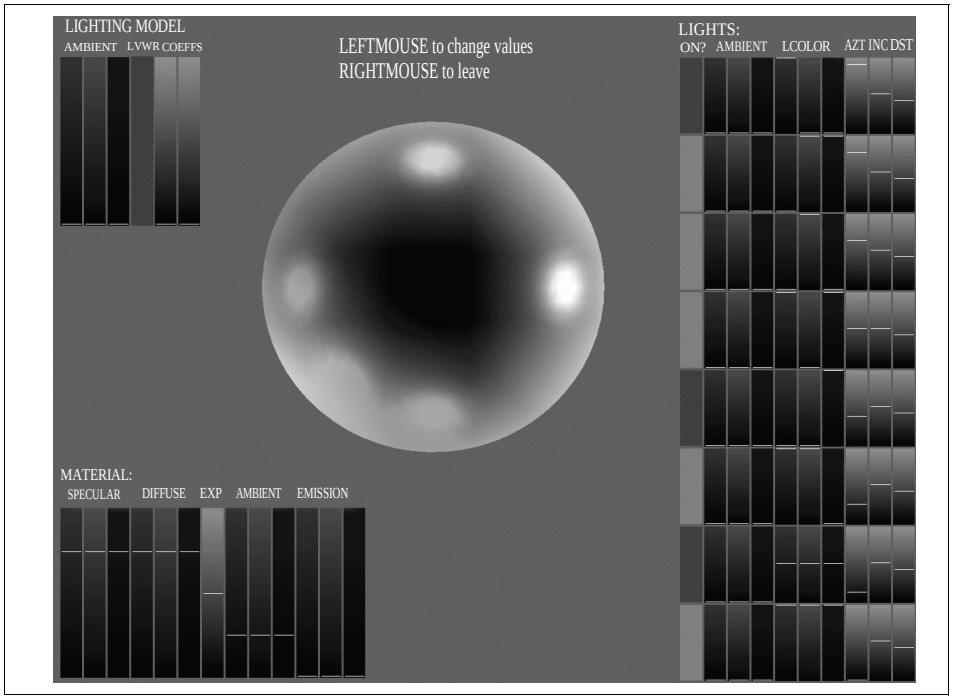


Figure 24. Image from the lmodtest Program

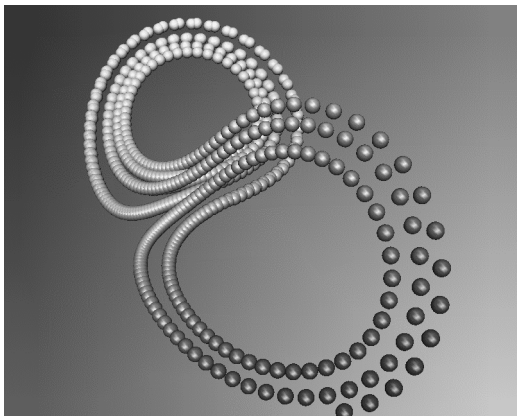


Figure 25. Image from the lorenz Program

8.3.3.2 Sample Source Code in /usr/lpp/GL/examples/

This directory contains a number of GL example programs. Each program illustrates a different aspect of GL or shows how to solve particular problems using GL and other AIX operating system features.

Table 12. GL Samples Programs

Example Program	Purpose
clover.c	Clears the overlay bit planes.
decor.c	Example of X/GL integration. Demonstrates how to change window decorations for GL windows.
depthche.c	Example of depth-cued lines.
fork_examp.c	A very simple example involving GL and the system `fork()` subroutine. Shows how the GL subsystem must be shut down before using the system `fork()` routine.
fork_examp2.c	A more sophisticated `fork()` example.
GLexec.c	A very simple example involving GL and the system exec( ) routine. Shows how the GL subsystem must be shut down before using any of the system exec( ) routines.
gen_sig.sh	See rqenter.c
getXdpy.c	Simple example of X/GL integration. Shows how to obtain the X Display of a GL session.

glwininfo.c	Example of X/GL integration. Prints information about the X visual being used.
input_test.c	Example of use of new X input function. Demonstrates the new SpaceBall support.
mapwin.c	Example of X/GL integration. Demonstrates how to map and unmap a GL window.
nurbs.c	Example of GL NURBS.
rqenter.c	Example showing how to write re-entrant queueing code. The example creates a pipe and then checks for input on both the pipe and on the GL queue. USR1 signals caught by the process result in input being placed on the pipe. (The shell script gen_sig.sh can be used to generate signals). See the related example: xinput.c
smoothln.c	Example of Anti-Aliased Lines. LEFTMOUSE activates linesmooth (TRUE). RIGHTMOUSE deactivates (FALSE).
Xcolormap.c	Example of X/GL integration. Shows how X11 can be used to manipulate the colormap associated with a GL window.
wincmap.c	Example of X/GL integration, colormap snooping utility.
xinput.c	Example of X/GL integration. Shows how to simultaneously manipulate both the GL and the X11 event queues within the same process. The example can be extended to simultaneously read other (pipe, socket, or file-based) event sources. See the related example: rqenter.c.
zover_examp.c	Example Z-buffered overlays. Shows how depth-testing (Z-buffering) can be enabled for overlays.

IBM does not guarantee that the contents of the source code examples, whether individually or as one or more groups, will meet your requirements or that the source code examples are error-free.

The contents of this directory are subject exclusively to the terms set forth in the Notice to Users that can be found in each of the source code example files.

8.3.3.3 Utility Programs in /usr/lpp/GL/utilities/

The utilities/gutil/ directory contains the Makefile and source files for libgutil.a, which includes useful graphics utilities as listed in the following table. The directory utilities/examples/ contains samples using this library.

Table 13. Utilities Examples for GL

Example Program	Purpose
dbcs.c	DBCS interface to charst.r
gammacorrect.c	Create gamma corrected gammaramps.
gltosnf.c	Convert defrasterfonts to SNF.
hashutil.c	Hash utilities.
partition.c	Colorindex color partitions.
rotaxis.c	Rotate about an axis.
visual.c	Get the correct GL visual.

GL Widgets

The directory utilities/widgets/ contains source files for sample GL widget and its Motif version.

This version supports GLXlink, GLXgetconfig, GLXunlink, and GLXwinset. (This version of GL widget sample code replaces the Glib.c code in /usr/lpp/ibmgl/utilities/gutil).

GL widgets can be used with either Xt-based programs or with Motif-based programs. Use GlxDraw for Xt and GlxMDraw version for Motif-based programs.

Screen Dump

The directory utilities/screendump/ contains source files for RGB image dump utility for RS/6000 with 3D Graphics Adapter.

The following directories will be found:

x24wd	Directory containing screendump utility
x24wud	Directory containing undump utility

Usage:

- To compile and link the programs, x24wd and x24wud, type the following:

  ```
  make
  ```

This makes the two sub-directories, x24wd and x24wud, which will contain the executables, `x24wd` and `x24wud`.

- To take a dump of a double-buffered 24-bit RGB or double buffered 12-bit RGB, type:

  ```
  x24wd filename
  ```

  You will be prompted to click on the window you want to dump. The files will be called, filename.a.X24 and filename.b.X24. These contain the content of this window for the front and back buffer.

- To take a dump of single buffered 24-bit RGB window, type:

  ```
  x24wd -single filename
  ```

  You will be prompted to click on one window, and the file will be called filename.X24.

- To display the result of your dump and make sure it is valid, type:

  ```
  x24wud filename
  ```

  where `filename` does not contain the .X24 suffix.

- You can use the PBMPLUS (described in X11 R4 release tape from MIT) utilities, `xwdtopnm` and `pnmtops` to convert X24 files to PostScript. The convert subdirectory contains a script, `xwdtops`, to accomplish this task.

  ```
  xwdtops filename
  ```

  Where `filename` does not contain the .X24 suffix. This script converts xwd files to a color PostScript file that can be used with a printer or any PostScript visualizer.

Note: The utility only works for applications that use DirectColor (actual numbers in the frame buffer are color values and not indices of a colormap).

You can manipulate the data directly to generate and read your own data formats. Simply replace the write_data or read_data routines with your own.

8.4 Overview of Programming

This section presents the header files for the GL 3.2 program, how to compile and link GL 3.2 programs, a sample program showing a simple GL 3.2 program. Appendix A.1, “GL 3.2 Sample Code” on page 231, contains more elaborate examples. They introduce double-buffering, the event loop, drawing geometric objects, and so on.

8.4.1 Header Files

Here are the two header files for developing code with the GL API.

- /usr/include/gl/gl.h:

  This should be included in every GL 3.2 program. It includes constant, type, and function declarations needed for all GL 3.2 programs. For instance, it defines the preprocessor tokens BLACK and GREEN.

- /usr/include/gl/device.h:

  This file contains definitions and type definitions (typedefs) pertaining to GL devices.

8.4.2 Link Libraries

To compile and link a GL 3.2 program, enter the following AIX command:

```
cc -o sample sample.c -lgl
```

To tell the linker to link to the math library, where the sin and cos functions are located, for example, specify the `-lm` flag option on the `cc` command:

```
cc -o sample sample.c -lgl -lm
```

8.4.3 Sample Program

The following GL 3.2 program demonstrated some basic concepts of GL, such as how to open a window on the screen, how to set attributes, and how to draw. It prints the message "Hello, World!" inside.

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

void main( void )
{
     prefsize( 200, 100 );
     winopen( "HI THERE" );
     color( BLACK );
     clear();
     color( GREEN );
     cmov2( 50, 50 );
     charstr( "Hello, World!" );
     sleep( 5 );
 }
```

In detail for each line:

```
prefsize( 200, 100 );
```

The `prefsize` subroutine communicates to the window manager the suggested size of the window that is created with the `winopen` subroutine.

The prefsize subroutine does not actually create the window; it only specifies the window size preferences.

In this case, the preference is a window that is 200 pixels wide and 100 pixels high.

You can also control other window properties, such as the preferred position (by the `prefposition` subroutine) or the window title (by using the `wintitle` subroutine).

```
winopen( "HI THERE" );
```

The `winopen` subroutine creates a window as defined by the current values of the window constraints.

This new window becomes the current window. If this is the first time that a program has called the `winopen` subroutine, the system also initializes the graphics system.

All drawing (lines, polygons, and NURBS) is done in the current window. Lighting, depth-cueing, and Z-buffering all apply to the current window. Every window has an independent set of stacks: matrix stack, name stack, attribute stack, and viewport stack, and all stack manipulation routines, such as matrix multiplications', are directed at the current window.

A string argument, for example "HI THERE", specifies the window title displayed on the left-hand side of the title bar. A zero-length string displays no title.

```
color( BLACK );
```

The color subroutine sets the current color. Everything you draw is displayed in the current color until you change that color. The color subroutine itself does not draw anything.

```
clear();
```

The clear subroutine clears the entire window to the current color. Because the program line immediately preceding the clear subroutine call sets the current color to black, the window is cleared to black.

```
cmov2( 50, 50 );
```

The `cmov2` subroutine sets the current character position. Every character you draw is displayed at the current character position until you change that

position. In this example, the current character position is set to 50 pixels up and 50 pixels to the right of the lower left-hand corner of the window.

```
charstr( "Hello, World!" );
```

The `charstr` subroutine actually draws the text "Hello, World!". The text is drawn with the current color, which is now green (by `color( GREEN );`).

```
sleep( 5 );
```

The `sleep` subroutine is an AIX system call. In this example, the `sleep` subroutine prompts the system to do nothing for 5 seconds. After 5 seconds, the `sleep` subroutine returns and the program exits.

When a GL program exits, any window that it has created disappears. Without the sleep call, the window would be created, would flash to black, with some green text, and would then disappear immediately. You can replace the sleep call with anything that will keep the program running, such as an infinite loop. But when the program exits, the window definitely disappears.

8.5 GL 3.2 References

Additional information about GL 3.2 can also be found:

- On the AIX Version 4.3 Base Documentation CD
- On the following IBM Web pages:

  `http://www.rs6000.ibm.com/resource/aix_resource/Pubs/`

  Select **AIX Version 4.3 Documentation Library**

  AIX Version 4.3 Base Documentation

  List of Books

  AIX Programming Guides

  GL3.2 Version 4 for AIX: Programming Concepts

  AIX Programming Reference

  GL3.2 Version 4 for AIX:
  Graphics Library (GL) Technical Reference

There are two documents:

- *GL 3.2 Version 4 for AIX: Programming Concepts*, SC23-2612

  This book provides information on the Graphics Library (GL). GL is an application programming interface (API) for performing advanced 3D

rendering, window management, and input device support. It serves as both a tutorial and a guide and is a programmer's source book for learning about 3D graphics from a programmer's perspective.

- *GL 3.2 Version 4 for AIX: Graphics Library (GL) Technical Reference*, SC23-2630

  This book provides information on calls and subroutines for the Graphics Library (GL) and example programs that illustrate using basic GL subroutines. This application programming interface is for use with the AIX operating system.

Chapter 9. OpenGL

This chapter describes the OpenGL API and its implementation by IBM for AIX. OpenGL has become one of the two major graphic APIs available on AIX, and IBM encourages its use for any new development. OpenGL is easy-to-use, full-featured and network transparent.

9.1 Definition

OpenGL supports a broad array of advanced graphics rendering techniques, such as texture mapping, line and polygon antialiasing, transparency, and fog. It provides versatile graphics functions, from simple geometry to the sophisticated rendering of complex surfaces. Users can generate high-quality pictures from user-defined graphical objects through a simple low-level modular interface.

This suite of advanced graphics functions is ideal for developing complex 3D applications, including computer-aided design and manufacturing (CAD/CAM), industrial design, engineering analysis, petroleum and chemical engineering, scientific visualization, and entertainment.

The OpenGL API is licensed from Silicon Graphics, Inc. (SGI) and is governed by the OpenGL Architecture Review Board. See 9.1.5, “The OpenGL Architecture Review Board (ARB)” on page 134.

OpenGL is the direct descendant of the proprietary SGI GL (SGI Graphics Library) API. Since Version 1.0 was released in 1992, OpenGL has gained a lot of momentum in the industry and is attracting a large number of software vendors. See 9.6.1.1, “History — From GL to OpenGL” on page 173.

Some of the advantages of OpenGL are:

- A consistent, industry standard API across present and future IBM graphics adapters
- A robust conformance test suite
- Debugging tools for tracking and check pointing calls to the graphics libraries.

The strength of OpenGL, as well as SGI GL, lies in their powerful support for immediate mode graphics, while supporting retained mode graphics. These terms are described in the following subsections.

9.1.1 Immediate Mode Graphics

Immediate mode graphics is not specific to the Silicon Graphics, Inc. APIs. Almost all the graphics APIs that had been supplied to PCs are of this type, and it is familiar to those who make graphics programs on PCs.

In immediate mode, graphics commands are executed as soon as they are encountered. There is no required display list structure. The data describing the geometry is not retained for modification or display by the API. Instead, this data must be retransmitted across the distributed environment every time the orientation or appearance of the model changes. Immediate mode graphics is well suited for applications with frequently changing geometric definitions.

- An example of this type of application is a car crash simulation where a car model is heading for a crash wall in the first scene and is a crumpled wreck in the last. A new description of the car geometry must be used for each scene (step) of the simulation. In this application, there would be no benefit in using retained mode graphics since the data definition changes completely from frame to frame.
- Another example is an application that uses computational fluid dynamics. The natural transformation of a sea wave is an example of this function. As the data representing the wave is recomputed, it needs to be redisplayed. Since the data is different in each frame that is displayed, there is no benefit in retaining old data.

9.1.2 Retain Mode Graphics

OpenGL supports the retained mode graphics using the display list mechanism. A display list can store a sequence of OpenGL commands and has a unique name (also called an identifier) in an OpenGL context. These commands are executed in sequence when the display list is called from the client application. The commands and their arguments are sent from the client to the OpenGL server when a display list is created. So, in the case of calling a display list repeatedly, it will reduce traffic between the client and the server. But because a display list is a resource on the server side, this mechanism consumes memory on the server to store the data associated with the display list.

For more information about the retain mode graphics, see 7.1.3, “Retained Mode Graphics” on page 89.

9.1.3 Client/Server

In the OpenGL vocabulary, a client is an application which issues OpenGL commands, and a server is a window system, such as X Window System with GLX X extension, where these commands are interpreted and processed to be ultimately displayed on your screen. The server and the client may or may not run on the same machine. So the OpenGL is mentioned as network transparent.

A server may maintain a number of OpenGL contexts, each of which contains current OpenGL state. On the other hand, a client may connect to more than one server, but should select one or none of the contexts on these servers at each moment.

9.1.4 Technical Content of OpenGL

OpenGL consists of several functional pieces: a rendering library, a utility toolkit, a Window System integration suite, and a networking protocol.

- The rendering library provides the basic functions of OpenGL.
- The OpenGL Utility (GLU) toolkit provides some additional useful features to the rendering library. It internally calls the rendering library.
- The OpenGL extension to X (GLX) library provides function that integrates the rendering library with the X Window System, using the standard X11 extension mechanism. Thus, OpenGL has a network protocol and is thereby network transparent.

The above libraries can be accessed with C, Fortran and Ada language bindings.

9.1.4.1 The Rendering Library

It provides the following features and capabilities:

- Basic 3D rendering services including:
  - Flat or Gouraud shaded points, lines, triangles, and polygons
  - Matrix and attribute stacks
  - Per vertex Phong lighting
  - Depth (Z-buffering)
- Antialiased points, lines, and polygons
- Subpixel accurate point, line, triangle and polygon rasterization
- User-defined clipping planes (modeling clip)
- Texture-mapping support

    - Nearest and linear interpolation using texture mipmap images
    - Texture decaling, modulation, and blending
    - Automatic texture coordinate generation for environment mapping
- Display list creation and manipulation facilities
- Fog and atmospheric effects
- Stencil buffers (useful for constructive solid geometry, interference checking, mirror reflection, and shadow algorithms)
- Accumulation buffers (useful for image post processing, motion blur and depth of field effects)
- Alpha buffers (useful for certain transparency algorithms)
- Logical image operations

OpenGL 1.1 added new functions such as vertex array, polygon offset, logical operation in RGB mode, texture proxies, texture objects, and subtexture.

OpenGL 1.2 added new functions such as 3D texturing, additional texture mapping control, new pixel formatting capability, and support for the Vertex Array Draw Element function.

9.1.4.2 The OpenGL Utility (GLU) Toolkit

The GLU toolkit provides additional useful features:

- Non-uniform Rational B-Spline (NURBS) support
- Basic disk, cylinder, sphere, and quadric capabilities
- Concave and multicontour polygon tessellation capabilities
- Simple image scaling and mipmap generation utilities

OpenGL 1.2 updates the GLU with improved NURBS tessellation.

9.1.4.3 The OpenGL Extension to X (GLX) Library

The GLX library provides functions that integrate the rendering library with the X Window System:

- Creating and binding of rendering contexts to windows
- Window creation aids through the extended visual types mechanism
- Synchronization of the OpenGL with the X11 data stream
- Renderer access to X11 fonts

9.1.4.4 The OpenGL Utility Toolkit (GLUT)

The GLUT is different from the above GLU toolkit. It provides functions that make programming easier than directly using the GLX, Xt and X11 libraries:

- Window control (initialization, size change, window title, icon, push/pop)
- Cursor management
- Overlay management
- Menu creation
- Event handling using callback subroutines
- Colormap control
- Query about device, supported extensions, and layer
- Bitmap and stroke font management
- Prebuilt models (sphere, cone, cube, torus, decahedron, octahedron, tetrahedron, and teapot)

The OpenGL Auxiliary (aux) Library

Previously, the OpenGL Auxiliary library was provided (its library and header files are libaux.a and aux.h, respectively). It has been withdrawn from the latest OpenGL development environment.

In the *OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.0*, ISBN 0-201-46138-2, sample programs were written using the aux functions, but those are rewritten using the GLUT functions in its latest revision for OpenGL 1.1.

9.1.4.5 The OpenGL Widgets

The OpenGL widgets are also provided. These are convenient for creating OpenGL drawing area widgets, to control their resources, and to register callback subroutines.

9.1.4.6 The OpenGL Development Environment

The OpenGL development environment includes:

- The rendering library (included in libGL.a) and header files (gl.h, fgl.h).
- The GLU library (libGLU.a) and header file (glu.h)
- The GLX library (also included in libGL.a) and header files (glx.h)
- GLX X server extension
- The GLUT library (libglut.a) and header file (glut.h)
- The OpenGL widgets library (libXGLW.a) and header files
- ZAPdb, debugging and tracing tool

- Viewperf, performance characterization tool
- Source code for example programs
- Demo programs

For detailed information, see 9.3.1, “Filesets” on page 140.

Upper Compatibilities

Because OpenGL 1.2 is a superset of OpenGL 1.1, all programs written for OpenGL 1.1 run on OpenGL 1.2 without modification, recompiling, or relinking. OpenGL 1.1 is also a superset of OpenGL 1.0.

9.1.5 The OpenGL Architecture Review Board (ARB)

OpenGL is an industry standard API under the guidance of the OpenGL ARB. IBM is one of its founding members along with SGI, Digital Equipment Corporation, Intel and MicroSoft. The current ARB members are IBM, SGI, Compaq, Intel, MicroSoft, 3DLabs, Evans & Sutherland, Hewlett Packard, Intergraph, and NVIDIA. There are many OpenGL licensees to help achieve broad market support for the OpenGL API, related technologies, programming expertise, and applications.

The ARB meets regularly to define the future of the OpenGL API. It also resolves issues in the current version of OpenGL and discusses extensions to the OpenGL standard. Minutes of recent ARB meetings are available for public review.

The ARB can be contacted through the publicly accessible Internet news group comp.graphics.opengl. Answers to frequently asked questions (FAQs) are posted regularly to this news group.

9.1.6 Conformance Test Suite

In order to provide compatibility and interoperability between different OpenGL implementations, the ARB requires a robust conformance test suite that must be satisfied before the OpenGL trademark can be used. The level of testing is rigorous and unprecedented in the 3D graphics marketplace.

9.1.7 OpenGL Licensing

OpenGL is licensed by SGI to interested parties under its standard terms and conditions. There are many licensees, including software and hardware vendors, workstation manufacturers, and universities.

Several forms of licensing are available. Licensed materials include the OpenGL, GLX and GLU specifications and manual pages, a sample implementation of OpenGL, GLX and GLU written in the C and/or C++ languages, and a conformance test suite.

IBM has obtained a full license, including the right to relicense modified, derived, and OpenGL-related source code.

9.2 IBM Implementations

OpenGL and GL 3.2 for AIX provide the OpenGL Version 1.2 and 1.1 compatibility – an excellent choice for developing new applications that use 3D graphics for RS/6000 workstations.

Major applications on AIX systems need an OpenGL runtime environment. This includes I-DEAS Master Series (SDRC), Alias (Alias/Wavefront), UniGraphics, and Pro/Engineer, to name a few.

IBM offers OpenGL 1.2 or 1.1 on AIX 4.1, AIX 4.2, and AIX 4.3 for almost every graphics adapter available in the RS/6000 workstation product line, from the entry-level to the high-end POWER GXT3000P.

IBM intends to continue providing industry-leading functional enhancements to the OpenGL product. IBM has provided Softgraphics technology, multiprocessor support, interactive debugger, Direct Soft OpenGL using XVFB, and many useful extensions to OpenGL.

The latest OpenGL product (for AIX Version 4.3.2) provides the OpenGL Utility (GLU) Toolkit Version 1.2, and the OpenGL Utility Toolkit (GLUT) Version 3.3.

OpenGL is implemented in two ways by IBM:

- IBM Softgraphics technology
- Hardware-accelerated graphics adapter

9.2.1 Softgraphics Technology

Softgraphics technology provides a software emulation to the OpenGL functions for all rendering operations. This provides full 3D function at lower cost. (See also 6.3, “Softgraphics” on page 84.)

The software implementation of OpenGL API (Soft OpenGL) through this IBM's Softgraphics pipeline is provided for many of the older graphic adapters and adapters without direct support of 3D rendering (entry graphics

accelerators such as GXT150M, GXT250P and GXT255P). It supports rendering to 8- and 24-bit visuals.

The software implementation is complete and fully compliant (it satisfies the criteria set forth by the conformance test suite).

Because all the operations are implemented in software, be aware that performance scales with the amount of CPU available. In the following table, PLBwire and PLBsurf results are shown for 3D wire frame and solid surface performance; the higher number represents the better performance. For information on performance benchmarking, see Chapter 11, “Benchmarking” on page 205.

Table 14. Impact of Softgraphics on OpenGL Performance

Machine Type	Graphics Adapter	Softgraphics	CDRS-03	DX-03	Comment
43P 150	GXT255P	Y	10.05	5.02	AIX V4.3.2
43P 133	GXT255P	Y	4.87	0.93	AIX V4.1.5
42T	GXT1000-2	N	23.84	4.76	AIX V4.1.5
42T	GXT500D	N	13.42	3.10	AIX V4.1.5

9.2.2 Hardware-Accelerated

All or part of rendering to the OpenGL API is done by the adapter hardware and microcode. Section 1.2, “Different Classes of Graphics Adapters” on page 9, describes the role of class II and III graphics adapters.

3D adapters, including the POWER GXT550P, GXT800P, GXT800M and GXT3000P, provide OpenGL acceleration by implementing raster operations in hardware while maintaining lighting, transformation, and clipping operations in software. The GXT3000P performs some lighting and setup in hardware as well. Since the transformation pipeline is maintained on the system processor, most operations scale with the processing power of the host.

9.2.3 OpenGL 1.1

IBM shipped OpenGL 1.1 for the first time in AIX 4.2.1 in April 1997. In May 1997, OpenGL 1.1 was made available under AIX 4.1.5.

Users of AIX 4.2.1 should look in InfoExplorer for detailed information about the proper use of OpenGL 1.1.

With AIX Version 4.1.5, the major changes found in OpenGL 1.1 are described in PostScript files located in /usr/lpp/OpenGL/docs.

9.2.4 OpenGL 1.2

With AIX 4.3.2, we announce support of OpenGL Version 1.2 on the new POWER GXT3000P adapter. If you use this graphics adapter, you can make use of OpenGL 1.2 functionalities; however, the optional imaging extension subset is not supported by the IBM OpenGL 1.2 implementation at this time.

9.2.5 Performance Improvements in OpenGL for AIX 4.3.2

With AIX4.3.2, IBM OpenGL 1.1 and 1.2 are enhanced to improve performance in several areas. Users of uniprocessor systems will note faster drawing of primitives under most conditions. All users should see improvements in:

- Throughput and cache utilization
- Latency when lighting is enabled
- Overall performance on any primitives using the new MultiDrawArray extension

9.2.6 New Extensions to OpenGL for AIX 4.3.2

Three new extensions to OpenGL are valuable in AIX Version 4.3.2. These are the MultiDrawArray Extension, the Texture Mirrored Repeat Extension, and the Color Blend Extension. Some performance-related extensions (compiled vertex array, vertex culling, and static data) are also newly added and enhance performance of a number of applications. For more information about these extensions, see Appendix A.2.3, “OpenGL Extensions Supported on AIX” on page 247.

9.2.7 Easy MP

In June 1997, IBM shipped Easy Implementation of OpenGL MP, or Easy MP, which provides multi-processor support for OpenGL 1.1. This set of subroutine libraries offers some customers convenient access to MP performance while running OpenGL in the RS/6000 Models 43P-240, F40 and F50 PCI-based SMP workstation. In most cases, this support is accomplished without changing any source code or recompiling their existing programs, and also without having to learn about writing thread-safe applications.

While Easy MP is not a thread safe library (meaning that the customer should not attempt to instantiate multiple copies of Easy MP in multiple threads), it is

coded to create multiple threads internal to the library layer and split the application's drawing requests among multiple processors.

Easy MP is not guaranteed to double your performance. Please read the caveats in Appendix A.2.2, "Using Easy MP" on page 245.

9.2.8 64-bit OpenGL Support

OpenGL now runs 64-bit binary applications in indirect contexts. Support for direct contexts is not currently available. Attempts to run 64-bit binaries in direct contexts are routed to indirect contexts.

9.2.9 Direct Soft OpenGL or OpenGL for a Virtual Frame Buffer

OpenGL is currently supported with the Virtual Frame Buffer (VFB), while PEX and graPHIGS are not.

The implementation of OpenGL for a VFB screen renders into private pixmaps. These pixmaps are not shared between OpenGL and the X server, nor are they shared between two OpenGL direct rendering clients. In other words, mixed mode (X and OpenGL) rendering is not supported. X rendering requests are performed on the VFB frame buffer, and OpenGL rendering requests are performed on OpenGL's private pixmaps. See Chapter 6, "The X Virtual Frame Buffer and Softgraphics" on page 75, for more information about VFB and DirectSoft OpenGL.

9.2.10 The ZAPdb OpenGL Interactive Debugger

The ZAPdb OpenGL Interactive Debugger (formerly known as Zapp) is a full featured OpenGL 1.2 application debugger. This provides a unique set of debugging features tailored for OpenGL developer and performance analysts.

Note

ZAPdb was originally developed by IBM and is currently licensed to SGI, HP, DEC, and Sun. It is also available for Windows NT.

Unlike other debuggers, ZAPdb operates at the library level, intercepting all calls to the OpenGL shared library. Consequently, it does not require any special application modification, recompilation or source code.

ZAPdb provides a number of features designed specifically for the OpenGL developer, including the ability to:

- View the OpenGL data stream in a variety of ways
- Set break points for all OpenGL API functions

- Review OpenGL API function call usage by resetable call counters
- Query attributes of the OpenGL graphics context and state

One of this debugger's most powerful features is its ability to generate a C code trace of the application's OpenGL calls for performance tuning, problem determination, geometry capture, and test case generation.

For more information about ZAPdb and how to use it, run the command ZAPdbDoc or see :/usr/lpp/OpenGL/tools/ZAPdb/ZAPdb.html.

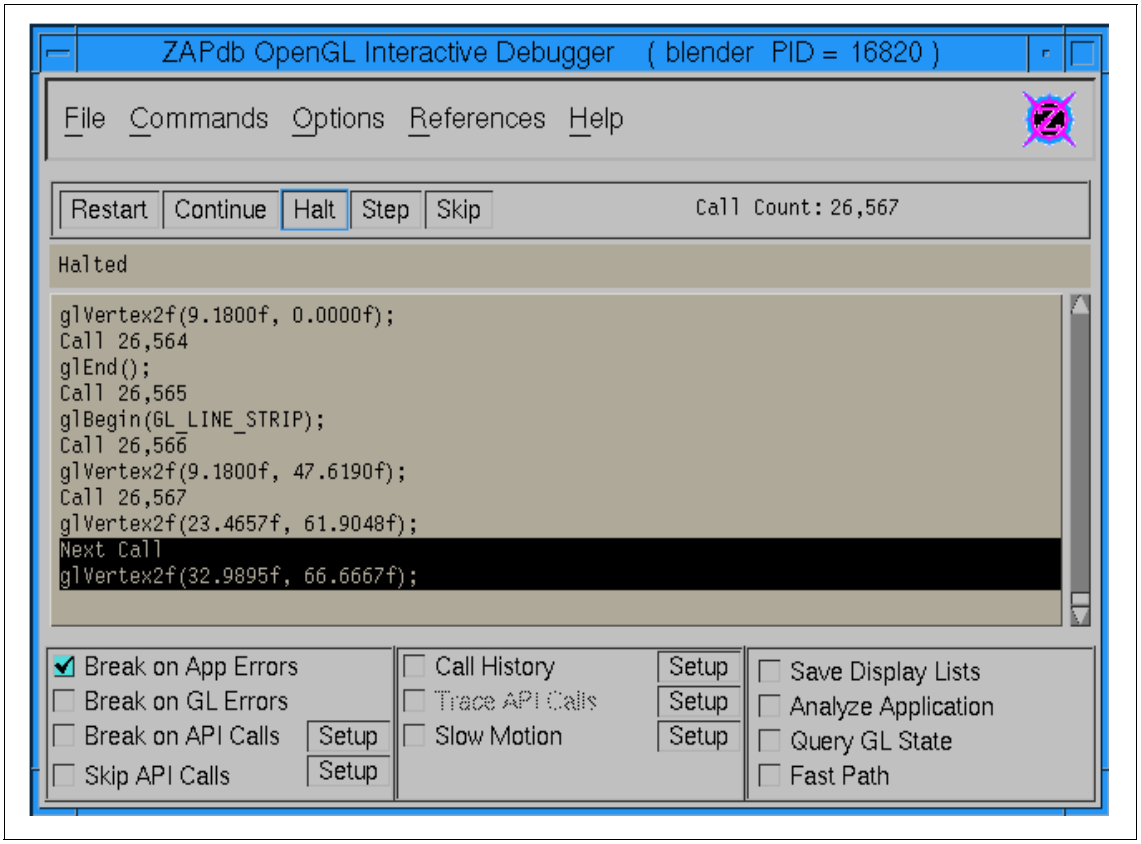


Figure 26. Main Panel of ZAPdb

9.2.11 Development History

The following table documents the history of AIX OpenGL product development.

Table 15. Main Steps in OpenGL Products Development

AIX Version	Date	Products/Features
4.1.3	June 1995	Support for GXT500, GXT500D
4.1.4	October 1995	Extensions
4.2.0	April 1996	Support for GXT250P, GXT255P
4.1.5	October 1996	Support for GXT500P, GXT550P, GXT800P
4.2.1	April 1997	Easy MP, OpenGL 1.1
4.3.0	October 1997	Integration with X11R6 and GXT800M
4.3.1	April 1998	VFB
4.3.2	October 1998	GXT3000P, OpenGL 1.2

9.3 Configuration

OpenGL and GL 3.2 for AIX are standard features of IBM AIX Version 4.3, but these had been offered as Licensed Program Products for IBM AIX Version 4.2 or 4.1.

9.3.1 Filesets

There are four filesets included in this product:

OpenGL.OpenGL_X.rte OpenGL Runtime Environment
OpenGL.OpenGL_X.dev OpenGL Device Dependent Software
OpenGL.OpenGL_X.adt OpenGL Application Development Toolkit
OpenGL.OpenGL_X.tools OpenGL Tools

The first two filesets must be installed to run any OpenGL applications on the system. To develop OpenGL programs on the system, it is necessary to install the OpenGL Application Development Toolkit fileset. The OpenGL Tools fileset includes the very useful OpenGL debugger.

9.3.1.1 OpenGL Runtime Environment

The fileset OpenGL.OpenGL_X.rte contains the following modules:

OpenGL.OpenGL_X.rte.base Base Runtime Environment

This provides:

- The /usr/lpp/OpenGL/README file.
- The runtime code under /usr/lpp/OpenGL/bin/.
- The library files, such as libGL.a or libGLU.a /usr/lpp/OpenGL/lib/, are linked from /usr/lib/.
- The /usr/lpp/OpenGL/samples/xglinfo/xglinfo (/usr/lpp/X11/bin/xglinfo) file.

OpenGL.OpenGL_X.rte.base+ Enhanced Geometry Pipeline
OpenGL.OpenGL_X.rte.base_mp MP Base Runtime Environment
OpenGL.OpenGL_X.rte.soft Soft Runtime Environment

These include the pipeline module under /usr/lpp/OpenGL/lib/ and so on.

9.3.1.2 OpenGL Device-Dependent Software

The fileset OpenGL.OpenGL_X.dev provides the following device-dependent software modules:

OpenGL.OpenGL_X.dev.common.bbl	GXT150L/155L/150P
OpenGL.OpenGL_X.dev.pci.14103c00.PPC	GXT250P/GXT255P
OpenGL.OpenGL_X.dev.pci.14105400.PPC	GXT500P/GXT550P
OpenGL.OpenGL_X.dev.pci.14105e00.PPC	GXT800P
OpenGL.OpenGL_X.dev.pci.1410b800.PPC	GXT2000P (N.A.)
OpenGL.OpenGL_X.dev.pci.14108e00.PPC	GXT3000P
OpenGL.OpenGL_X.dev.pci.14105400.PPC_mp	MP GXT500P/GXT550P
OpenGL.OpenGL_X.dev.pci.14105e00.PPC_mp	MP GXT800P
OpenGL.OpenGL_X.mca.8f61	GXT800M
OpenGL.OpenGL_X.dev.common.rby	GXT1000
OpenGL.OpenGL_X.buc.00004002	GXT500

It also provides:

OpenGL.OpenGL_X.vfb Virtual Frame Buffer Support

9.3.1.3 OpenGL Application Development Toolkit (ADT)

The fileset OpenGL.OpenGL_X.adt contains two ADT modules:

OpenGL.OpenGL_X.adt.include OpenGL ADT Include Files

This contains include files such as gl.h under /usr/lpp/OpenGL/include/. which is linked as /usr/include/GL

OpenGL.OpenGL_X.adt.samples OpenGL ADT Samples

This provides sample source code and READMEs under /usr/lpp/OpenGL/samples/. These samples demonstrate the use of OpenGL and the GLUT library. There are four directories as follows:

- prog_guide/

  Contains the source code to the examples referenced in the Addison & Wesley *OpenGL Programming Guide*, ISBN 0-201-46138-2

- glx_prog_guide/

  Contains the source code to the examples referenced in *OpenGL Programming for the X Window System* published by Addison-Wesley, ISBN 0-201-48359-9

- libglut/

  Provides the GLUT source code

- xglinfo/

  Provides `xglinfo` and its source code

9.3.1.4 OpenGL Tools

The fileset OpenGL.OpenGL_X.tools contains the following modules:

OpenGL.OpenGL_X.tools.base OpenGL Base Tools

This creates a link from /usr/lpp/OpenGL/tools/glxscope to /usr/lpp/X11/Xamples/aixclients/xscope.

OpenGL.OpenGL_X.tools.debugger OpenGL Debugger

This provides the followings under /usr/lpp/OpenGL/tools/ZAPdb:

- OpenGL debugger bin/ZAPdb, which is linked as /usr/bin/ZAPdb, and the README file for it
- Utilities under util/, which will be used with ZAPdb
- OpenGL library files under bin/, which are called from ZAPdb
- Manual pages (HTML files) and README under ZAPdb/doc/
- doc/ZAPdbDoc, which is linked as /usr/bin/ZAPdbDoc

9.3.2 Installation

Installation is done using the standard AIX tools, such as `smit install_latest`.

Please refer to the release notes in /usr/lpp/OpenGL/README first.

9.3.2.1 /usr/lpp/X11/defaults/xserverrc

The commands used to start up the OpenGL Extension to X Window System (GLX) automatically are in /usr/lpp/X11/defaults/xserverrc, which contains the following line:

```
EXTENSIONS="-x GLX -x abx -x dbe"
```

Otherwise, to start up automatically under `xinit`, edit the file and change the line reading:

```
EXTENSIONS=""
```

Or, to start up manually, use `xinit` command line options as follows:

```
xinit -- -x GLX -x abx -x dbe
```

On this command line, GLX refers to the OpenGL Extension to X, `abx` refers to the X Windows Ancillary Buffer Extension, and `dbe` refers to the X Windows Double Buffer Extension (DBE). DBE is an official X standard and has replaced the nonstandard X Multi-Buffer Extension (MBX).

Note

Starting the X server with `-x mbx` instead of `-x dbe` still works on an AIX Version 4.2.1 system, but not on AIX Version 4.3.2 system.

To check a list of extensions currently loaded to the X server, run `xdpyinfo` (`/usr/bin/X11/xdpyinfo`) on the local system, or run it with the `-display` option on a remote system.

9.4 Overview of Programming

The *OpenGL Programming Guide*, ISBN 0-201-46138-2, published by Addison-Wesley, is a good guidebook to learn OpenGL programming. See 9.7, “References” on page 189. It provides plenty of sample codes. Those sample codes also in the directory /usr/lpp/OpenGL/samples/prog_guide when OpenGL.OpenGL_X.adt.samples installed on your system.

This section describes an overview for OpenGL programming. Section 9.4.13, “Debugging Hints” on page 167 shows simple hints that may be useful for your first time programming.

9.4.1 OpenGL Programs

To summarize an OpenGL program:

1. The first part of OpenGL program is creation of both OpenGL windows and contexts.
2. Next, because most of OpenGL programs are event driven, proper callback subroutines should be registered for events your program will handle. Those events have to be selected.
3. Almost every OpenGL call is used in the callback subroutines. It is necessary to make current a pair consisting of one OpenGL window and one context as the drawing target before any OpenGL calls.
4. The OpenGL program usually has a main event loop.
5. Of course, in the main loop, the program flow should be changed dynamically by control of callback subroutine registrations.

Because OpenGL includes no Window Systems subroutines, to manage windows, OpenGL contexts and events, it is necessary to call the GLX or toolkit subroutines. Section 9.4.2, “Programming Styles” on page 146, describes how to implement those actions.

9.4.1.1 Creation of OpenGL Windows and Contexts

Both OpenGL window and context need reference to a visual that is one of those supporting OpenGL on the X server.

First, you should decide the requirements for the visual:

- Double-buffered or single-buffered
- RGBA or color indexed
- 8-bit RGBA or 24-bit RGBA
- With or without depth/stencil buffer
- With or without accumulation buffer
- Level (for example, level 1 and 0 correspond to overlays and normal windows, respectively)

The command `xglinfo` shows a list of visual configurations available for the X server. This command is described in 9.4.12, “xglinfo” on page 165.

OpenGL contexts and corresponding OpenGL windows should refer to the same visual.

9.4.1.2 Initialization of OpenGL Context

OpenGL program may or may not have a callback subroutine for a window-mapped event. If it exists, the initialization of the OpenGL context is done there. If you do not implement such a callback, you should first make

sure that the OpenGL window is mapped, and then initialize the OpenGL context.

The very first step of the initialization is to make current a pair consisting of one OpenGL window and one context. They will become the target for any rendering order.

9.4.1.3 Other Callback Subroutines

There should always be a callback for the window exposure event. It is the responsibility of the OpenGL application to redraw the content of a window when this one is exposed.

Callbacks for device inputs, such as keyboard, mouse, tablet, LPFK, dial, and spaceball, are registered if you want to consider them.

9.4.1.4 Creating a Scene

To create a scene:

1. Make a pair of the OpenGL window and context current.

   If the program manages only one OpenGL window and one context, this is not required. You should do that once at the initialization.

2. Set viewport.

3. Initialize and set the global view matrix (or in other words, view volume, or projection matrix).

   Here, you can choose a perspective view or an orthogonal view, and set near and far clipping planes.

   In a perspective view: Your eyes are located at the origin of the world coordinates, and they look in the direction of positive Z along the Z axis.

   In an orthogonal view: Your eyes are located at infinite point in the negative side of Z, and they look in the direction of positive Z in parallel with the Z axis.

   In both cases, the X and Y axes point to your right-hand side and your upper side, respectively.

4. Initialize the local view matrix (or in other words, model view matrix).

5. Often set the local view matrix first, so you can move all the objects in the view in one matrix modification.

6. Draw figures on the color buffer(s).

7. Flush OpenGL calls or swaps color buffers to show the scene on the screen.

Double-Buffered View

With a double-buffered view, one of the two color buffers is displayed on the screen each time. The visible buffer is called the front buffer, and the other invisible buffer is called the back buffer. If you swap these color buffers, the previous front buffer is now the back buffer — in other words, invisible.

You can draw figures either in the front or in the back buffer or in both at once. While the figures are drawn in the front buffer, you can see the drawings even before they are completed (but flushing OpenGL calls may be needed sometimes).

9.4.1.5 OpenGL Windows and Contexts

Several windows in a display can share an OpenGL context. On the other hand, several OpenGL contexts may exist for one display, but only one at a time can be attached to a window.

9.4.1.6 Multiple Windows, Multiple Viewports or Multiple Contexts

To implement multiview on your application window, there are different ways:

- First, there is only one viewport in each OpenGL context.
- You can set the viewport as many times as needed in a window so that the window has multiple viewports.
- You can attach multiple OpenGL contexts to a window even if only one is active at a time. You can then assign one of your multiple viewports to each OpenGL contexts.
- You can use a set of multiple subwindows in your application main window as multiple viewports in a window.

9.4.2 Programming Styles

Based on the way you want to manage your windows, OpenGL context and X events, you may choose to use either:

- The OpenGL Utility Toolkit (GLUT)
- The OpenGL widgets and the OpenGL extension to X (GLX) library
- The GLX library without the GLUT or the OpenGL widgets

The last option has more flexibility, but needs more lines of code. To learn OpenGL, programming with the GLUT is the easiest method, you can even create very sophisticated applications with the GLUT.

If you would like to make your application with other widgets, you should use the GLX library with or without the OpenGL widgets but including the GLUT. And you should not mix the GLUT and the OpenGL widgets in an application.

All OpenGL subroutines are available to use in your application whatever toolkit you choose.

9.4.3 Naming Conventions

Naming conventions in OpenGL are very simple and are designed to reduce name clashes with other packages.

Subroutine names, constants, and types are prefixed by gl, GL_, and GL, in C. And similar naming conventions are also applied in the GLU, GLX, and GLUT. Prefixes for these are listed in the following table:

Table 16. Naming Conventions for OpenGL Components

Library	Subroutines	Constants	Types
GL	gl	GL_	GL
GLU	glu	GLU_	GLU
GLX	glx	GLX_	
GLUT	glut	GLUT_	

For example, you will soon get familiar with such names:

- Subroutines : `glBegin()`, `glEnd()`, `glEnable()`, `glDisable()`, `glDepthFunc()`
- Constants : GL_LINE, GL_TRIANGLES, GL_DEPTH_TEST
- Types : GLubyte, GLint, GLfloat, GLdouble

Many OpenGL subroutines (commands) take a certain type of values and/or enumeration constants. These subroutines are suffixed to indicate the number of elements (if needed) and the type of values. For example:

- glVertex*2i*() requires two GLint values (as x, y).
- glVertex*4f*() requires four GLfloat values (as x, y, z, w).

- Suffix *v* in glVertex*2iv*() means that it has a pointer to an array that holds two GLint values.
- glVertex*() stands for all of such subroutines.

9.4.4 Header Files

OpenGL header files are located under /usr/include/GL/.

#include <GL/gl.h>	This is required in any OpenGL program; however, it is included in other header files as follows.
#include <GL/glu.h>	This is required to call the GLU subroutines. It includes gl.h.
#include <GL/glx.h>	This is required to call the GLX subroutines. It also includes gl.h.
#include <GL/glut.h>	This is required to call the GLUT subroutines. It includes both gl.h and glu.h.
#include <GL/GLwMDrawA.h>	This is required to use the OpenGL widgets. It includes both gl.h and glx.h. This is the same as writing #include <X11/GLw/GLwMDrawA.h>

9.4.5 Link Libraries

`-lGL` and `-lX11` should always be specified, for instance. In the simplest case, you can make an executable file as follows:

```
cc -o sample sample.c -lGL -lX11
```

If the `GLU` is used, then `-lGLU` should be specified.

```
cc -o sample sample.c -lGLU -lGL -lX11
```

If the OpenGL widgets and the `GLU` are used, then `-lXGLW` and `-lXm -lXt` are additionally required for the OpenGL widgets:

```
cc -o sample sample.c -lXGLW -lGLU -lGL -lXm -lXt -lX11
```

If your code includes the `GLUT` and the `GLU` subroutines, the link options should be:

```
cc -o sample sample.c -lglut -lGLU -lGL -lXmu -lXext -lX11
```

9.4.6 OpenGL Rendering Context

A context is a complete set of OpenGL state variables. The configuration of the frame buffer is also part of OpenGL state, while the contents of frame buffers are not.

Because performance is critical in 3D rendering, the OpenGL extension to X allows OpenGL to bypass the X server's involvement in data encoding, copying, and interpretation and instead renders directly to the graphics pipeline.

If the graphics adapter supports direct rendering and the X server connection is local, the application can use the direct context; otherwise, the indirect context is used.

9.4.7 Programming with the Rendering Library

The following subsections describe typical OpenGL subroutines.

9.4.7.1 Geometric Primitives

The general syntax of drawing primitives is described below:

- Start with `glBegin(primitive_type)`.
- Set attributes, using `glColor*()`, `glNormal*()`, and so on.
- Specify the first vertex with `glVertex*(1st_vertex_data)`.
- Set attributes, if any.
- Specify 2nd vertex with `glVertex*(2nd_vertex_data)`.
- Specify nth vertex with `glVertex*(nth_vertex_data)`.
- End with `glEnd()`.

Whenever a vertex is specified with `glVertex*()`, the current attributes, such as color and normal vector coordinates, are assigned to that vertex. You can change the values of those attributes before each `glVertex*()` call.

The argument passed to `glBegin()` specifies the type of primitive, and it is one of the following 10 values:

GL_POINTS, GL_POLYGON, GL_LINES, GL_TRIANGLES, GL_QUADS, G_LINE_STRIP, GL_TRIANGLE_STRIP, GL_QUAD_STRIP, GL_LINE_LOOP, GL_TRIANGLE_FAN.

Between a `glBegin()` and `glEnd()` pair, only the following nine OpenGL subroutines are valid, and making any other OpenGL call generates an error.

`glVertex*()`	set vertex coordinates
`glColor*()`	set current color
`glIndex*()`	set current index
`glNormal*()`	set normal vector coordinates
`glEvalCoord*()`, `glevalPoint*()`	generate coordinates
`glCallList()`, `glCallLists()`	execute display list(s)
`glTexCoord*()`	set texture coordinates
`glEdgeFlag*()`	control drawing of edges
`glMaterial*()`	set material properties
`glArrayElement()`	extract vertex array data

These subroutines also can be called outside of `glBegin()` and `glEnd()` block.

For example, this code draws a red colored loop with n+1 line segments:

```
glColor3f(1.0, 0.0, 0.0);/* red */
glBegin(GL_LINE_LOOP);
   glVertex3f(x0, y0, z0); /* GLfloat x0, y0, z0,..., xn, yn, zn;*/
   glVertex3f(x1, y1, z1);
   glVertex3f(x2, y2, z2);
   ...
   glVertex3f(xn, yn, zn);
glEnd();
```

9.4.7.2 Mode Control

Many state variables, such as texture coordinates, materials or test functions, refer to modes such as 2D or 3D texture, lighting or depth test. Those modes are enabled or disabled with `glEnable()` or `glDisable()` subroutines. For example:

```
glEnable(GL_TEXTURE_3D);
glEnable(GL_LIGHTING); /* enables lighting mode */
glEnable(GL_LIGHT2); /* enables one of eight lights */
glEnable(GL_DEPTH_TEST);
```

9.4.7.3 Shade Model

Flat shade or Smooth (Gouraud) shade is selected by the `glShadeMode()` subroutine. Flat shade is the default. This code shows a red object in flat shade mode or a smooth shaded object in smooth shade mode:

```
glShadeMode(GL_FLAT or GL_SMOOTH);
glBegin(primitive_type);
   glColor3f(1.0, 1.0, 1.0); /* white */
   glVertex3f(x0, y0, z0);
   glColor3f(1.0, 1.0, 0.0); /* yellow */
   glVertex3f(x1, y1, z1);
   ...
```

```
   glColor3f(0.0, 0.0, 1.0); /* blue */
   glVertex3f(xn, yn, zn);
glEnd();
```

9.4.7.4 Push and Pop Attributes and Transformation Matrix

Using `glPushAttrib()` and `glPopAttrib()` subroutines, you can save and later restore the values of a collection of state variables on an attribute stack, or in another words, attribute changes can be encapsulated in a block. For example, in this code the colors assigned to the 1st and the 2nd vertices are both blue:

```
glColor3f(1.0, 0.0, 0.0); /* red */
glColor3f(0.0, 0.0, 1.0); /* blue */
glVertex3f(x0, y0, z0); /* the 1st vertex */
glVertex3f(x1, y1, z1); /* the 2nd vertex */
```

But in the following code, the 1st vertex is assigned the blue color, while the 2nd vertex is assigned the red color:

```
glColor3f(1.0, 0.0, 0.0); /* red */
glPushAttrib(GL_CURRENT_BIT);
   glColor3f(0.0, 0.0, 1.0); /* blue */
   glVertex3f(x0, y0, z0); /* the 1st vertex */
glPopAttrib();/* we go back to the red */
glVertex3f(x1, y1, z1); /* the 2nd vertex */
```

`glPushMatrix()` and `glPopMatrix()` saves and later restores the transformation matrix in the same way.

9.4.7.5 Display List

Using a display list, an object, a group of OpenGL subroutines can be used repeatedly, in a scene or in different scenes. For example, this code makes a display list containing an object, such as a block, without specifying colors:

```
glNewList(listid, GL_COMPILE);
glBegin(primitive_type);
    glVertex3f(...);
    ...
    glVertex3f(...);
glEnd();
glEndList();
```

Later, this object can be called at a different location with a different color as follows:

```
glColor3f(1.0, 0.0, 0.0);
glPushMatrix();
   glTranslatef(...);
   glCallList(listid);
glPopMatrix();
```

```
glColor3f(0.0, 0.0, 1.0);
glPushMatrix();
   glTranslatef(...);
   glCallList(listid);
glPopMatrix();
```

Notice the difference between the following two codes:

```
glNewList(listid, GL_COMPILE);
glColor3f(1.0, 1.0, 1.0); /* white */
glBegin(primitive_type);
   glVertex3f(...);
   ...
glEnd();
glEndList();
```

The above display list changes the current color to white when it is called, and will remain white after this call. But the display list below will change the current color to white and then the restore previous color before exit:

```
glNewList(listid, GL_COMPILE);
glPushAttrib(GL_CURRENT_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(primitive_type);
       glVertex3f(...);
       ...
   glEnd();
glPopAttrib();
glEndList();
```

9.4.8 A Program with the GLUT

The code in this section is not the complete source code for an application. For instance, no error handling is described here, but it serves as an explanation for OpenGL programming. (The code in the following two sections is not complete.)

9.4.8.1 main()

The following subroutine is a typical main function in the sample programs you have in the *OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.1*, ISBN 0-201-46138-2

```
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(250, 200);
```

```
    glutInitWindowPosition(100, 150);
    glutCreateWindow("sample");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

In the `main()` subroutine:

- The first five GLUT subroutines configure a main window as an OpenGL drawable. This window is mapped on the screen after `glutMainLoop()` is executed.
- Next, `init()`, as shown below, initializes the OpenGL context.
- Each of the next three subroutines sets a callback subroutine to the corresponding event.

The detail for each line is:

```
glutInit(&argc, argv);
```

Initializes the application context. (`cf. XtAppInitialize()` or `XtOpenApplication()`.)

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

Requests a single-buffered RGB(A) visual with depth (Z) buffer.

Note

If all RGB(A) visuals have a Z-buffer, one of them is chosen by this command even if you do not specify GL_DEPTH. You do not have to handle the Z-buffer in that case.

```
glutInitWindowSize(250, 200);
glutInitWindowPosition(100, 150);
```

Sets the main window width and height to 250 and 200 pixel size and locates the window's upper-left corner at x = 100, y = 150 on the screen.

```
glutCreateWindow("sample");
```

Creates a main window with the above attributes (visual, size, and position) and gives a string "sample" to the window title.

```
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
```

Registers callback subroutines. In an event loop, if the main window is displayed (exposed) on the screen, a callback subroutine, `display()`, for the exposed event is executed. When the application user changes the application window sizes, `reshape()` is called. Any keystroke causes a call for `keyboard()`.

About 20 functions are ready to use to register a callback: `glutMouseFunc()`, `glutMotionFunc()`, `glutTabletButtonFunc()`, `glutDialsFunc()`, `glutTimerFunc()`, `glutIdelFunc()`, and so on.

If you want to remove a callback subroutine later, specify `NULL` as shown below :

```
glutKeyboardFunc(NULL);
glutMainLoop();
```

Starts the main loop. (`cf. XtAppMainLoop()`).

OpenGL Context

In this `main()` subroutine, you don't see the creation of any OpenGL context. But the GLUT subroutines create one context and make it current with the newly created window.

9.4.8.2 init()

This is the place where you should initialize the OpenGL context properly.

```
void init (void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   glEnable(GL_DEPTH_TEST);
   }
```

The detail of each line is:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
```

Sets the clear color to black. To make black the default, this line can be omitted. This command is valid in RGBA mode or for RGB(A) visual. In color index mode,

```
glIndexi(n); /* n is one of available color indices */
```

is valid and sets the clear color index to n.

```
glShadeModel(GL_FLAT);
```

After this call, polygons and lines are flat shaded. If you need Gouraud shading, pass GL_SMOOTH to this command. GL_FLAT is the default; so this line can also be omitted.

```
glEnable(GL_DEPTH_TEST);
```

Enables the depth test. By default, the test is disabled, so this line is necessary to make it available.

9.4.8.3 Callback Subroutines

The subroutines shown below are samples of callbacks.

First, this is a sample callback subroutine for the exposure event. This should be registered by `glutDisplayFunc()`, and will be called every time the main window is exposed — in other words, when it is initially mapped or when it is restored from icon.

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   /******************/
   /* drawing a scene */
   /******************/
   glFlush();

   glutSwapBuffers();
}
```

The detail for each line is:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Clears the drawable RGBA buffer and Z-buffer at once. To clear the RGBA buffer, the clear color is used. The Z-buffer is cleared with the maximum number. GL_COLOR_BUFFER_BIT is valid in both RGBA and color index modes.

```
glFlush();
```

Flushes all of the commands that are not yet executed on the OpenGL server. It is necessary to call it if you work on a single-buffered visual. For a double-buffered visual, this also works well; however, sometimes the things go well without this command because of buffer swapping. This happens

because the remaining commands are executed in the back buffer, and you don't see the result.

```
glutSwapBuffers();
```

If the visual is double buffered, this command swaps color buffers. For instance, it makes the invisible color buffer (back buffer) visible, and vice versa. If the visual is single-buffered, nothing will happen with this command.

Next, this is a sample callback subroutine for the window reconfiguration event. This should be registered by `glutReshapeFunc()` and will be called at the window creation as well as when the user changes the window sizes. It resets the viewing information in the OpenGL context:

```
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0 * (GLfloat) h / (GLfloat) w, -1.0, 1.0);
   /* or gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 30.0); */
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   /*... some transformations you need */
 }
```

The detail for each line is:

```
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
```

Adjusts the viewport to the reshaped window size and assures you that you can still use the maximum size of the window for your drawing.

Note

Without this line, the result can be quite surprising. For example, if the size of a window, which is originally 150" width and 100 in height, changes to 150x100, the bottom 50 rows are not available for drawing because the viewport is unchanged, and your figures are clipped by viewport boundary. See Figure 27 on page 157.

And, of course, if you want to keep the original viewport regardless of window size changes, you don't need to call `glViewport()` here. Further if you set both of the projection and modelview matrices properly in `init()`, you may not have to call `glReshapeFunc()` itself.

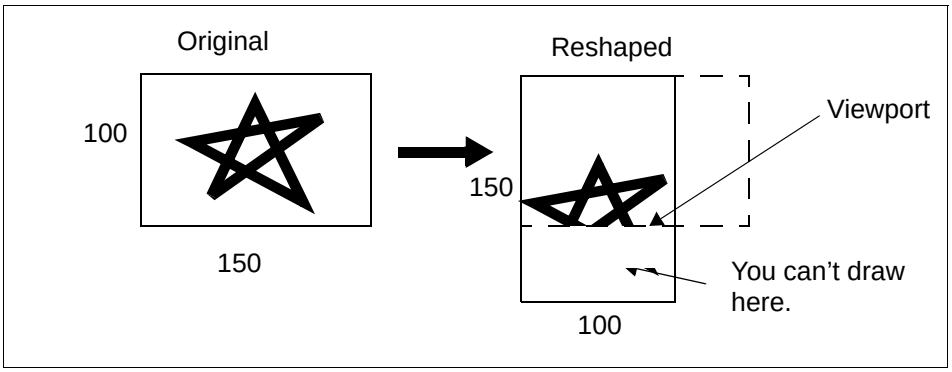


Figure 27. Importance of the Viewport Concept

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0 * (GLfloat) h / (GLfloat) w, -1.0, 1.0);
```

First, reset the projection mode matrix to unit matrix; then set the orthogonal view volume. This `glLoadIdentiy()` is important. `glOrtho()` generates a transformation matrix and applies it to the current projection matrix. Therefore, the following lines might create unexpected results.

```
glMatrixMode(GL_PROJECTION);
glOrtho(0.0, 1.0, 0.0, 1.0 * (GLfloat)h / (GLfloat)w, -1.0, 1.0);
```

To use a perspective view, call:

```
gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 30.0);
```

instead of `glOrtho()`.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Change the matrix mode to GL_MODELVIEW mode, and reset the modelview matrix to the unit matrix.

Next is a sample callback subroutine for key input event. This should be registered by `glutKeyboardFunc()`. It will be called every time a key is hit.

```
void keyboard(unsigned char key, int x, int y)
{
   switch(key) {
   case 'q':
   case 'Q':
   exit(0);
```

```
    default:
    break;
  }
}
```

9.4.9 A Program with the GLX Library and the OpenGL Widgets

Sample `main()` subroutine and some callback subroutines are shown here using the GLX library and GLw widget.

9.4.9.1 main()

The following subroutine is a typical `main()` function:

```
#include <X11/Intrinsic.h>
#include <GL/GLwMDrawA.h>

void ginitCB(Widget w, XtPointer client_data, XtPointer call_data);
void exposeCB(Widget w, XtPointer client_data, XtPointer call_data);
void inputCB(Widget w, XtPointer client_data, XtPointer call_data);

int main(int argc, char **argv)
{
    ...
    int glxconfig[] = {
                        GLX_DOUBLEBUFFER,
                        GLX_RGBA,
                        GLX_RED_SIZE, 8,
                        GLX_GREEN_SIZE, 8,
                        GLX_BLUE_SIZE, 8,
                        None
        };
    XVisualInfo *visualinfo;
    GLXContext glxcontext;
    Widget glw;
    ...

    toplevel = XtOpenApplication(&app_context, "sample",...);
    ...
    visualinfo = glXChooseVisual(display, screen, glxconfig);

    glxcontext = glXCreateContext(display, visualinfo, None, GL_TRUE);

    n = 0;
    XtSetArg(args[n], GLwNvisualInfo, visualinfo); n++;
    XtSetArg(args[n], XmNwidth, 256); n++;
    ...
    glw = GLwCreateMDrawingArea(toplevel, "glw", args, n);
```

```
    XtManageChildren(glw);

    XtAddCallback(glw, GLwNginitCallback, (XtCallbackProc)ginitCB,
                  (XtPointer)option1);
    XtAddCallback(glw, GLwNexposeCallback,
                    (XtCallbackProc)exposeCB, (XtPointer)option2);
    XtAddCallback(glw, GLwNresizeCallback,
                    (XtCallbackProc)resizeCB, NULL);
    XtAddCallback(glw, GLwNinputCallback,
                    (XtCallbackProc)inputCB, NULL);

    XSelectInput(display, XtWindow(glw),
                 XtBuildEventMask(glw) | PointerMotionMask);

    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
    return 0;
}
```

In this `main()` subroutine:

- The `glXChooseVIsualInfo()` subroutine returns a pointer to an XVisualInfo structure describing the visual that best meets the client's specified attributes.
- The `glXCreateContext()` subroutine creates a OpenGL context with visualinfo.
- The `GLwCreateMDrawingArea()` subroutine creates an OpenGL widget.
- Each `XtAddCallback()` subroutine registers a proper callback subroutine for corresponding event type.

The detail for each part is:

```
int glxconfig[] = {
                    GLX_DOUBLEBUFFER,
                    GLX_RGBA,
                    GLX_RED_SIZE,    8,
                    GLX_GREEN_SIZE,  8,
                    GLX_BLUE_SIZE,   8,
                    None
                  };
```

Describes the client's specified attributes.

```
XVisualInfo *visualinfo;
visualinfo = glXChooseVisual(display, screen, glxconfig);
```

Returns a pointer to an XVisualInfo structure describing the visual that best meets the attributes with the above `glxconfig[]`.

```
GLXContext glxcontext;
glxcontext = glXCreateContext(display, visualinfo, None, GL_TRUE);
```

Creates an OpenGL context with `visualinfo`.

```
Widget glw;
Arg args[10];
int n;
n = 0;
XtSetArg(args[n], GLwNvisualInfo, visualinfo); n++;
XtSetArg(args[n], XmNwidth, 256); n++;
...
glw = GLwCreateMDrawingArea(parent, "glw", args, n);
XtManageChildren(glw);
```

Creates and manages an OpenGL widget specifying `visualinfo` and the other attributes.

```
void ginitCB(Widget w, XtPointer client_data,
GLwDrawingAreaCallbackStruct *call_data);
...
XtAddCallback(glw, GLwNginitCallback,
(XtCallbackProc)ginitCB, (XtPointer)option1);
XtAddCallback(glw, GLwNexposeCallback,
(XtCallbackProc)exposeCB, (XtPointer)option2);
XtAddCallback(glw, GLwNresizeCallback,
(XtCallbackProc)resizeCB, NULL);
XtAddCallback(glw, GLwNinputCallback,
(XtCallbackProc)inputCB, NULL);
```

Registers a proper callback subroutine for each event type.

9.4.9.2 Callback Subroutines

Subroutines shown below are samples of callbacks.

This will be called just when the OpenGL widget is mapped on the screen. This is similar to the `init()` subroutine in the GLUT example except for the call of `GLwDrawingAreaMakeCurrent()`, which should be called in advance of any other OpenGL calls.

```
void ginitCB(Widget w, XtPointer client_data,
             GLwDrawingAreaCallbackStruct *call_data)
{
   GLint viewport[4];
```

```
    GLwDrawingAreaMakeCurrent(w, (GLXContext)client_data);

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glEnable(GL_DEPTH_TEST);
}
```

The following is called when the OpenGL widget is exposed, and it is similar to the `display()` subroutine in the GLUT example:

```
void exposeCB(Widget w, XtPointer client_data,
            GLwDrawingAreaCallbackStruct *call_data)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /******************/
    /* drawing a scene */
    /******************/
    glFlush();
    GLwDrawingAreaSwapBuffers(w);
}
```

The following function is called when the OpenGL widget is resized, and it is similar to the `reshape()` subroutine in the GLUT example:

```
void resizeCB(Widget w, XtPointer client_data,
            GLwDrawingAreaCallbackStruct *call_data)
{
    GLsizei w = call_data->width;
    GLsizei h = call_data->height;

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0 * (GLfloat) h / (GLfloat) w, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    /*... some transformations you need */
}
```

The following function is called when the OpenGL widget gets events from input devices:

```
void
inputCB(Widget w, XtPointer client_data,
       GLwDrawingAreaCallbackStruct *call_data)
{
    switch(call_data->event->type) {
```

```
    case KeyRelease:
                ...
    case ButtonPress:
                switch(call_data->event->xbutton.button) {
                case Button1:
                          ...
                }
                break;
    case MotionNotify:
                ...
    }
}
```

9.4.10 A Program with the GLX Library

Sample `main()` subroutine is shown here using the GLX library. The following subroutine is a typical `main()` function:

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
...
#include <GL/glx.h>
#include <GL/glu.h>

int main(int argc, char **argv)
{
    ...
    int glxconfig[] = {
                       GLX_DOUBLEBUFFER,
                       GLX_RGBA,
                       GLX_RED_SIZE,    8,
                       GLX_GREEN_SIZE,  8,
                       GLX_BLUE_SIZE,   8,
                       None
                      };
    XVisualInfo *visualinfo;
    GLXContext glxcontext;
    Window root, window;
    XSetWindowAttributes attr;
    XEvent event;
    ...

    display = XOpenDisplay(NULL);
    screen = DefaultScreen(display);
    root = RootWindow(display, screen);
    ...
    visualinfo = glXChooseVisual(display, screen, glxconfig);
```

```
    glxcontext = glXCreateContext(display, visualinfo, None, GL_TRUE);
    cmap = XCreateColormap(display, root, visualinfo->visual, AllocNone);

    attr.border_pixel = 0;
    attr.event_mask = ExposureMask | ButtonPress;
    attr.colormap = cmap;
    window = XCreateWindow(display, root, 0, 0, 512, 512,
                    0, visual_info->depth, InputOutput, visualinfo->info,
                    CWBorderPixel | CWColormap | CWEventMask, &attr);
    XSetWMColormapWindows(display, window, &window, 1);

    glXMakeCurrent(display, window, glxcontext);
    init();
    XMapWindow(display, window);

    while(!done) {
        status = XPending(display);
        if(status!= 0) {
              XNextEvent(display, &event);
              switch(event.type) {
              case Expose:
                          ...
              case ButtonPress:
                          ...
              }
        }
    }
    return 0;
}
```

9.4.11 Overlay Window

GLUT provides several subroutines to support drawing on overlay planes:

```
/* GLUT overlay sub-API. */
void glutEstablishOverlay(void);
void glutRemoveOverlay(void);
void glutUseLayer(GLenum layer);
void glutPostOverlayRedisplay(void);
void glutShowOverlay(void);
void glutHideOverlay(void);
/* GLUT callback sub-API. */
void glutOverlayDisplayFunc(void (*)(void));
```

These are not described in this book.

The following is additional information to the above GLX programming with or without OpenGL widgets.

To get an overlay visual, pass this to `glXChooseVisualInfo()`.

```
int overlay_config[] = {
                     GLX_LEVEL, 1,
                     GLX_TRANSPARENT_TYPE_EXT,
                     GLX_TRANSPARENT_INDEX_EXT,
                     None
                     };
XVisualInfo *overlay_visualinfo =
glXChooseVisual(display, screen, overlay_config);
```

The OpenGL context and window for overlay planes are created in the same manner as color planes.

```
GLXContext overlay_context =
glXCreateContext(display, overlay_visualinfo, None, GL_TRUE);
Widget glw_overlay;
...
XtSetArg(args[n], GLwNvisualInfo, overlay_visualinfo); n++;
glw_overlay = GLwCreateMDrawingArea(toplevel, "glw_overlay", args, n);
```

In the initialization subroutine (like `ginitCB()`), it is necessary to get the transparent pixel for the clear color index as follows:

```
int transparent_pixel;
glXGetConfig(display, overlay_visualinfo,
            GLX_TRANSPARENT_INDEX_VALUE_EXT, &transparent_pixel);
glClearIndex(transparent_pixel);
```

This process is required to keep portability of the application because the transparent pixel may change on the other machines. It may be 255 or 0. The `xglinfo` command shows the number for transparent overlay on a specified display.

To handle `MapNotify` events, whenever mapped, raise the overlay window on top of the window in color planes and add the event handler as described here:

```
XtAddEventHandler (glw_overlay,StructureNotifyMask,False,
                  (XtEventHandler)structureChanged,"");
```

The event handler can be defined with:

```
void structureChanged(Widget w, XtPointer client, XEvent *event)
{
       if(w == glw_overlay && event->type == MapNotify)
```

```
            XRaiseWindow(display, XtWindow(w));
}
```

Because the overlay visual is usually a color-indexed visual, you should assign colors to color indices. The following code assigns only one color.

```
Colormap    cmap;
ulong       pixels[1];
XColor      color;

XtVaGetValues(glw_overlay, XmNcolormap, &cmap, NULL);

/* Allocate a color cell */
XAllocColorCells(display, cmap, False, NULL, 0, pixels, 1);

color.pixel = pixels[0];
color.red = color.green = color.blue = 0x0000; /* black */
color.flags = DoRed | DoGreen | DoBlue;
XStoreColor(display, cmap, &color);

glIndexi(pixels[0]);
```

When the overlay planes are used in the application, there are at least two sets of OpenGL contexts and windows; therefore, `glXMakeCurrent()` or `GLwDrawingAreaMakeCurrent()` should be called when the drawing target changes.

9.4.12 xglinfo

The command `/usr/bin/X11/xglinfo` provides you with useful information for your OpenGL programming. You should again execute the DISPLAY where you are going to draw.

To get the information for the screen that is in front of you, just type `xglinfo` or `/usr/bin/X11/xglinfo`. To get the information for the screen on the remote host, if the X server is running on that machine, execute:

```
xglinfo -display remote_host_name:screen_number
```

A result of this command on the machine with GXT3000P is shown in Appendix A.2.1, “Output of xglinfo” on page 243. You have a similar one on your system. The following explains its contents.

In the output, the Display section shows the X server characteristics with the X server extension list and GLX and GLU libraries information.

The following Screen section shows the graphics adapter's name, or "SoftRaster" for the VFB, and a list of available GL extensions.

```
GL Extension: Vendor = IBM
              Renderer = GXT3000
              Version = 1.2.0
              Extensions = GL_EXT_texture_object GL_EXT_vertex_array
              GL_EXT_rescale_normal GL_IBM_rasterpos_clip
              GL_IBM_cull_vertex GL_EXT_multi_draw_arrays
              GL_EXT_abgr GL_EXT_bgra GL_EXT_blend_color
              GL_EXT_blend_logic_op GL_EXT_polygon_offset
              GL_EXT_subtexture GL_EXT_texture3D
              GL_EXT_texture_edge_clamp GL_EXT_texture_lod
              GL_IBM_texture_mirrored_repeat
```

The last section is information about visuals available on the screen. In the above case (with GXT3000P), all visuals are available for OpenGL use, but sometimes you find the following lines for the other type adapters.

```
OPENGL NOT SUPPORTED
```

For example:

```
TrueColor visual: ID = 0x2c (hex) 44 (decimal) screen = 0
DOUBLE buffered MONO RGB visual with (Alpha Z Stencil)
GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8,
Extensions: visualCaveat=None, OPAQUE
Core X:depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff)
bits=8
```

This visual has a double-buffered 32-bit RGBA color buffer, a 24-bit Z buffer and an 8-bit stencil buffer. `MONO` means nonstereo visual.

```
TrueColor visual: ID = 0x2e (hex) 46 (decimal) screen = 0
DOUBLE buffered MONO RGB visual with (Alpha Z Stencil Accum)
GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8, Accum=(16,16,16,16)
Extensions: visualCaveat=None, OPAQUE
Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff)
bits=8
```

This visual, in addition, has a 64-bit RGBA accumulation buffer.

```
TrueColor visual: ID = 0x28 (hex) 40 (decimal) screen = 0
DOUBLE buffered MONO RGB visual with (Z Stencil)
GL Sizes: RGBA=(8,8,8,0), Z=24, Stencil=8,
Extensions: visualCaveat=None, OPAQUE
Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff)
bits=8
```

This visual has a double-buffered 24-bit RGB color buffer but no alpha. It also has a 24-bit Z-buffer and an 8-bit stencil buffer.

```
TrueColor visual: ID = 0x25 (hex) 37 (decimal) screen = 0
DOUBLE buffered STEREO RGB visual with (Alpha Z Stencil Accum)
GL Sizes: RGBA=(4,4,4,4), Z=24, Stencil=8, Accum=(16,16,16,16)
Extensions: visualCaveat=None, OPAQUE
Core X: depth=12, colormapSize=16 RGB: masks=(0xf00,0xf0,0xf) bits=4
```

This visual has a double-buffered stereo RGBA color buffer, but each component has four bits. Therefore, its color resolution is less than the above visuals (They all support eight bits per component.)

```
PseudoColor visual: ID = 0x23 (hex) 35 (decimal) screen = 0
DOUBLE buffered MONO COLOR INDEX visual with (Z Stencil)
GL Sizes: ColorIndex=8, Z=24, Stencil=8
Extensions: visualCaveat=None, OPAQUE
Core X: depth=8, colormapSize=256
```

This visual has a double-buffered 8-bit indexed color buffer, a 24-bit Z-buffer and an 8-bit stencil buffer.

```
PseudoColor visual: ID = 0x22 (hex) 34 (decimal) screen = 0
OVERLAY(1) SINGLE buffered MONO COLOR INDEX visual GL Sizes:
ColorIndex=8,
Extensions: visualCaveat=None, TRANSPARENT INDEX, index value=255
Core X: depth=8, colormapSize=255
```

This visual has a single-buffered 8-bit indexed color buffer. This is an overlay visual. Because index 255 is used as the transparent index, the available color indices are [0,254]. The color map size is 255.

```
PseudoColor visual: ID = 0x21 (hex) 33 (decimal) screen = 0
OVERLAY(1) SINGLE buffered MONO COLOR INDEX visual GL Sizes:
ColorIndex=8,
Extensions: visualCaveat=None, OPAQUE
Core X: depth=8, colormapSize=256
```

This is also an overlay visual, but it is opaque.

9.4.13 Debugging Hints

If the result of your first program is not the one you expect, you may be interested in this section. ZAPdb is very useful for debugging.

- If the application is coded without GLUT, and the application closes without any drawing, it is very possible that MakeCurrent is not called in the correct place or maybe not at all.

- The depth test is disabled by default. And if you select a visual without the depth buffer on a hardware-accelerated graphics adapter, the depth test does not work. If you need the depth test, check the supported visuals by using `xglinfo` and specify GLX_DEPTH_SIZE in the configuration data for `glXChooseVisual`.
- If no object is seen in the black screen when the lighting is enabled, disable the lighting first. Then check if objects are rendered.
  - Replace `glColor*()` with a set of `glMaterial*()` calls, which are needed for lighting. There is another way using `glColorMaterial()` and `glEnable(GL_COLOR_MATERIAL)`, but it is not usually recommended to use this. That is, `glMaterial*()` should be used. And, when lighting is disabled, use `glColor*()` (or `glIndex*()` in color index mode).
  - If, even with `glMaterial*()`, the picture is not correct, then enable two-sided lighting as follows:

    ```
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
    ```

    and/or disable face cull.
- If no object is seen even when the lighting is disabled, check the view volume. First, remember that viewer (your eyes) are located at the origin of the world coordinate system, (0, 0, 0). If you use a perspective view, this is sometimes forgotten.
  - Using an orthogonal view is easier to find objects in view volume. Set near and far clipping planes to negative infinity and positive infinity, respectively. If you still cannot see your objects, check the right, left, top, and bottom boundaries.
  - With a perspective view, set the near clipping plane at near the origin and set the far clipping plane to positive infinity. And push objects to the positive direction of Z using `glTranslate*()`. If you find them, it is more convenient to move the viewer position to the negative direction of Z using the `gluLookAt()` subroutine than pushing objects away from you (indeed, `gluLookAt()` does the same thing.) The proper place of `gluLookAt()` is:

    ```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(...); /* or glOrtho(...),... */
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt(...);
    ```

- If the application uses a double-buffer visual and nothing occurs, change the draw buffer to front using `glDrawBuffer(GL_FRONT)`. Then draw a scene following `glFlush()`, but do not call the swap buffer subroutine.
- If drawing on overlay planes fails:
  - First change the clear color to any non-transparent color pixel, and clear with `glFlush()`.
  - If nothing occurs, check the location (x, y) and the size (width, height) of the overlay window.
  - If the clear color works correctly, make sure there is a `glFlush()` after drawing. For example, with the following code, nothing might be drawn.

    ```
    clear, draw rubber band, clear, draw rubber band, clear,...
    ```

    In such a program, insert `glFlush()` as follows:

    ```
    clear, draw rubber band, flush, clear, draw rubber band, flush,...
    ```

    Then, the rubber band will be drawn correctly.

9.5 Performance Tips

For general performance guidelines, consult the OpenGL Performance Tips in the *OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.1*, ISBN 0-201-46138-2.

Most of the following topics are described in the latest /usr/lpp/OpenGL/README file.

9.5.1 Additional Tips

- For a set of primitives, each composed of a uniform number of vertices, use the appropriate independent primitive type (GL_LINES, GL_TRIANGLES, GL_QUADS) instead of a connected primitive type (GL_LINE_STRIP, GL_TRIANGLE_STRIP, GL_QUAD_STRIP, GL_POLYGON). This reduces the number of calls to `glBegin()` and `glEnd()`.
- Use the display list rendering whenever possible, especially if you reuse rigid objects.
- To get better performance, make display lists as large as possible rather than making a large number of small ones.
- Use `glVertex3*()` commands instead of `glVertex4*()` if the W component is 1.0.

- Avoid `glColorMaterials()` unless you really need them; that is, disable GL_COLOR_MATERIAL and use `glMaterial*()` instead of `glColor*()` for lighting.
- Use `glLoadIdentity()`, `glRotate*()`, `glTranslate*()`, and `glScale*()` rather than calling `glLoadMatrix()` and `glMultMatrix()`.
- Avoid GL_NORMALIZE if possible; that is, provide unit length normals if it is possible to do so. And avoid using `glScale*()` when doing lighting because it almost always requires GL_NORMALIZE to be enabled.
- Avoid multiple OpenGL contexts in a scene. It may be convenient to have multiple contexts in an application, but context changing is expensive. Use one context for each visual.
- If you need to use accumulation buffers for scene antialiasing, try to use `glReadPixels()`, `glDrawPixels()` and your CPU memory in direct rendering context without accumulation buffers. It might be faster than using accumulation buffers in indirect context.
- Use texture objects to encapsulate texture data. OpenGL 1.1 supports it to improve performance in texture mapping.

9.5.2 Specific Implementation Notes

The following descriptions are performance tips for specific graphics systems.

9.5.2.1 GXT3000P OpenGL

- Use MultiDrawArrays or DrawArrays.
- Sometimes performance is better if face cull is disabled on this adapter. Even if you need face cull, compare performance with or without face cull.
- Use infinite light and infinite viewer unless you really need point light.
- Utilize direct rendering contexts when possible. Visuals supporting accumulation buffers do not support direct rendering contexts. Therefore, unless accumulation buffers are needed, avoid using visuals with one.

9.5.2.2 Soft OpenGL on VFB

For two OpenGL clients to render into a common pixmap buffer, both clients must render using indirect contexts. To achieve mixed mode rendering (X and OpenGL), the environment variable, `_OGL_MIXED_MODE_RENDERING` may be set to 1. Because of the performance impact, this is intended for verification purposes only. With mixed mode rendering enabled, the utilities `xwd` and `xwud` can be used to view the VFB frame buffer contents.

9.5.2.3 Soft OpenGL on the GXT250 Family/GXT150 Family

The softGL implementations accelerate some OpenGL rasterization in the following X server resolutions:

Table 17. Resolutions Supported by softGL on the GXT150 and GXT250 Families

Graphic Adapter	1280x1024	1024x768
GXT255P	X	X
GXT250P		X
GXT155L	X	X
GXT150L		X
GXT150P		X

Since all the operations are implemented in software, it is important that the following guidelines are observed in order to maximize system performance.

- Avoid rendering to unnecessary buffers. For example, if destination alphas are not needed, disable alpha writes using:

  `glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_FALSE)`

- If Z-buffering is not required, disable the depth test:

  `glDisable(GL_DEPTH_TEST)`

- If possible, use the default depth test of `GL_LESS`.
- Minimize synchronization function calls like `glFlush()`, `glFinish()`, `glXWaitX()`, and `glXWaitGL()`.
- Utilize display list rendering whenever possible.
- Avoid dithering for the clear color. This can be achieved by specifying a clear color, which is displayable without dithering, or by disabling dithering.
- Lines and polygons are rendered in hardware when the following states are selected:
  - `glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_FALSE).`
  - `glDisable(GL_DEPTH_TEST)`
  - `glShadeModel(GL_FLAT)`

  In addition, the GXT250 series renders color-interpolated lines.
  - `glShadeModel(GL_SMOOTH)`

GL_FLAT is still the fastest line renderer, though.

- If possible, disable dithering when rendering to color-index drawables.
- Pixmap rendering is supported in indirect context only.

9.5.2.4 GXT500 Family/GXT800 Family OpenGL

Performance can be maximized if the following guidelines are observed:

- Use direct rendering contexts when possible. Visuals including accumulation buffers do not support direct rendering contexts. Therefore, unless accumulation buffers are needed, avoid using visuals with one.
- Use MultiDrawArrays or DrawArrays.
- Use face cull, if possible.
- (GXT800 Family only) For those systems equipped with texture memory, primitives textured with texture maps larger than 512x512 (without borders, 256x256 with borders) are textured using software. To maximize texture mapping performance, make sure the texture images are no larger than these limits.

9.5.2.5 GXT1000 OPENGL

Consult the *7250 POWER GXT1000 Graphics Accelerator User's Guide*, SA23-2070, for more information on the GXT1000.

Performance can be maximized if the following guidelines are observed.

- Utilize direct rendering contexts whenever possible.
- When rendering in immediate mode, do not artificially extend `glBegin()`/`glEnd()` primitives by adding degenerate vertices.
- Use the GXT1000 Model 2 with the Advanced Graphics (AG) option to maximize texture mapping performance. Also use that model if large texture maps or deep accumulation buffers are required.
- If several textures are used repeatedly, wrap the `glTexImage()` calls by defining each texture in a display list. This allows the adapter to cache the images in the adapter texture memory, even if other texture images are subsequently defined.
- The GXT1000 can be configured to swap the frame buffers of a double-buffered window on horizontal retraces. The default configuration is to swap on vertical retraces. Swapping on horizontal retrace increases performance, but may produce small visual artifacts. Consult the *7250 POWER GXT1000 Graphics Accelerator User's Guide*, SA23-2070, for more information.

On the GXT1000, the value returned by a query of GL_MAX_TEXTURE_SIZE represents the size of the largest texture image that can be defined with *BOTH* mipmaps and borders. Images without mipmaps or borders that are larger than GL_MAX_TEXTURE_SIZE are supported by the GXT1000:

- Images with either mipmaps or borders (but not both) can be up to twice the size reported by GL_MAX_TEXTURE_SIZE.
- Images with neither mipmaps nor borders can be up to four times the size reported by GL_MAX_TEXTURE_SIZE.

Double-buffered windows with an accumulation buffer are only available on a GXT1000 Model 2 with the Advanced Graphic option where the environment variable HW_TEXTURE_CFG is set to TX96.

For information on rendering in stereo or using the GXT1000 Video Output Option (VOO), consult the *7250 POWER GXT1000 Graphics Accelerator User's Guide*, SA23-2070. The GXT1000 renderer currently does not support pixmap rendering.

9.6 Comparison with Other 3D Graphics APIs

In this section, OpenGL is compared with GL 3.2 and graPHIGS. This section also introduces Open Inventor.

9.6.1 Comparison with GL 3.2

As stated earlier in this chapter, OpenGL comes almost directly from the proprietary GL API developed by Silicon Graphics, Inc.

9.6.1.1 History — From GL to OpenGL

GL has been very popular as a 3D graphics API, but it has some disadvantages because it does not provide interoperability in heterogeneous networked environments and is not easily portable even between the few platforms that support it.

For these reasons, and in response to the growing popularity of open systems, SGI initiated an effort to standardize GL so that it could be easily supported by multivendor platforms. This effort led to the development of an open 3D API standard called OpenGL.

IBM, DEC, Microsoft, and Intel were invited by SGI to join the OpenGL ARB to help formulate the definition of the OpenGL API. OpenGL Version 1.0 was

released in 1992. IBM is still an active member of the ARB and continues to contribute to the OpenGL specification.

OpenGL is not backwards compatible with former GL versions; in fact, OpenGL calls are completely different from previous GL implementations. For this reason, OpenGL requires ports from older versions of GL, like 3.2, 3.3, and 4.0, on all vendors' platforms.

SGI's and IBM's strategic direction is away from a proprietary windowing system to an open system. Therefore, GL customers are encouraged to migrate to OpenGL by separating windowing and input calls (which are better handled by Xlib) from 3D rendering calls.

9.6.1.2 Advantages of OpenGL Over IBM's GL Offering

IBM encourages GL 3.2 customers to migrate to OpenGL. This is because of the many advantages of OpenGL over GL 3.2, as described here.

- Additional rendering capabilities:

  OpenGL offers additional rendering capabilities not present in IBM's current GL offering, including texture mapping, stencil and accumulation buffers. See 9.1.4, "Technical Content of OpenGL" on page 131, for OpenGL rendering capabilities.

- Better integration with window systems:

  OpenGL does not provide the support for windowing, input, events, cursors, and pop-up menus that GL includes. Instead, OpenGL relies on the libraries for the window system which are better equipped to handle these functions on each platform and therefore is more integrated with the window system than GL. For example, such libraries are the X11 library (Xlib), Xt and OSF/Motif for AIXwindows.

- Open licensing:

  As described in 9.1.7, "OpenGL Licensing" on page 134, OpenGL is licensed by SGI to interested parties under its standard terms and conditions, whereas, GL remains proprietary.

- Technical control over the API under the OpenGL ARB:

  As described in 9.1.5, "The OpenGL Architecture Review Board (ARB)" on page 134, together they create detailed specifications and conformance test suite criteria. GL, on the other hand, is a proprietary API developed by SGI.

- Rigorous conformance test suite:

As described in 9.1.6, “Conformance Test Suite” on page 134, OpenGL requires test suite conformance.

- Available on low-end, low-cost adapters:

  OpenGL offers a software rendering version to provide 3D function on 2D adapters. See 9.2, “IBM Implementations” on page 135, for more information. IBM's offering of GL does not support this software-rendering capability.

9.6.1.3 Technical Differences of OpenGL and GL

The following segments describe the major differences between OpenGL and GL 3:

Window Management

As described before, OpenGL includes no window system commands. It is always supported as an extension to a window system. For example, the GLX includes about 10 commands for this purpose. GL 3.2 provides window management subroutines, too.

In OpenGL, an application window has static frame buffer configurations, but GL 3.2 subroutines, such as `gconfig` and `drawmode`, can change the configuration dynamically.

- Buffer Swapping:

  The GL 3.2 subroutine `swapbuffers()` requires no argument, but the GLX or OpenGL widget library requires target drawable. The `swapbuffers()` subroutine can be emulated as follows:

```
void swapbuffers( void )
{
      glFlush();
      GLwDrawingAreaSwapBuffers( current_drawable );
                  /* argement is a widget */
                  /* or glXSwapBuffers(display, current_drawable);
                  the second argument is a window */
}
```

  `current_drawable` can be controlled by the application, or can be obtained by `glXGetCurrentDrawable()`.

Clear

- Clear Color:

  OpenGL provides subroutines to set clear color, `glClearColor()` and `glClearIndex()`. In the GL 3.2 application, the color set by `color()`, in color index mode, is used as the clear color and the vertex color. For example,

the clear color for overlay planes might be set to the transparent pixel value once in the initialization subroutine.

- Clear Options:

  OpenGL clears buffers without applying the currently specified pixel operations such as blending and logicop, regardless of their modes. To clear using such features, you have to render a window-size polygon.

- `clear()` and `czclear()`:

  The `clear()` subroutine of GL 3.2 only clears color buffers. `czclear()` clears both color buffers and the depth buffer. In the OpenGL application, clear target buffer can be specified by `glClear()` as follows:

  ```
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ```

Transformation

- Matrix Mode:

  All OpenGL matrix operations operate on the current matrix, rather than on a particular matrix, as do the GL 3.2 ortho, ortho2, perspective, and window commands. All the OpenGL matrix operations except `glLoadIdentity` and `glLoadMatrix` multiply the current matrix rather than replacing it (as do ortho, ortho2, perspective, and window in the GL 3.2).

  GL 3.2 does not transform geometry by the modelview matrix while in projection matrix mode. OpenGL always transforms geometry by both the modelview and the projection matrix, regardless of the matrix mode.

- Conversion Samples:

  The following code shows samples of conversion from GL 3.2 to OpenGL:

```
void ortho2( GLfloat left, GLfloat right, GLfloat bottom, GLfloat top )
{
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
gluOrtho2D( left, right, bottom, top );
glMatrixMode( GL_MODELVIEW );
}

void ortho( GLfloat xmin, GLfloat xmax,
GLfloat ymin, GLfloat ymax, GLfloat zmin, GLfloat zmax )
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho( xmin, xmax, ymin, ymax, zmin, zmax );
glMatrixMode( GL_MODELVIEW );
}
```

```
void perspective( GLshort view_angle,
GLfloat aspect, GLfloat near_clip, GLfloat far_clip )
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective( 0.1 * view_angle, aspect, near_clip, far_clip );
glMatrixMode( GL_MODELVIEW );
}

void frustum( GLfloat xmin, GLfloat xmax,
GLfloat ymin, GLfloat ymax, GLfloat zmin, GLfloat zmax )
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum( xmin, xmax, ymin, ymax, zmin, zmax );
glMatrixMode( GL_MODELVIEW );
}

void rotate( GLshort angle, char axis )
{
glRotated( angle * 0.1,
axis == 'x' ? 1.0 : 0.0,
axis == 'y' ? 1.0 : 0.0,
axis == 'z' ? 1.0 : 0.0 );
}

void lookat( GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble rot )
{
GLdouble  vx = centerx - eyex;
GLdouble  vy = centery - eyey;
GLdouble  vz = centerz - eyez;

glLoadIdentity();
if( vx == 0.0 && vz == 0.0 ) /* Z-up */
gluLookAt( eyex, eyey, eyez, centerx, centery, centerz, 0., 0., -1.);
else
...
}
```

Color

- Lighting Equation:

  The OpenGL lighting equation differs from the GL 3.2 equation.

- Flat Shade Color:

GL 3.2 polygons take the color of the last vertex specified, but OpenGL polygons take the color of the first vertex specified.

- Packed Color:

  GL 3.2 accepts packed colors both for geometric and pixel rendering. OpenGL does not accept them for geometric rendering.

c3s()

The `c3s()` subroutine of GL 3.2 maps [0, 255] to [0.0, 1.0]; however, `glColor3sv()` of OpenGL maps [-32768, 32767] to [-1.0, 1.0], and `glColor3ubv()` maps [0, 255] to [0.0, 1.0]. Therefore, `c3s()` should be replaced as follows:

```
void c3s( GLshort v[3] )
{
    glColor3ub( (v)[0], (v)[1], (v)[2] );
}
```

Or, use GLubyte v[3] for GLshort v[3]; then `glColor3ubv(v)` is available.

Primitive Drawing

- Scalar and Vector Arguments:

  All OpenGL commands that are accepted between `glBegin` and `glEnd` have entry points that accept scalar and vector arguments, such as, `glColor4f(red,green,blue,alpha)` and `glColor4fv(v)`.

- Triangle Mesh:

  OpenGL provides GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN. GL 3.2 offers the triangle mesh drawing subroutine and the `swaptmesh()` subroutine to support triangle strips and triangle fans.

- Polygon Mode:

  GL 3.2 supports hollow polygons, but OpenGL does not support them. To render hollow polygons in an OpenGL application, the stencil capability is useful.

  OpenGL polygon modes are specified separately for front- and back-facing polygons, but GL 3.2 shares a single mode for all polygons.

- Polygon Vertex Count:

  GL 3.2 polygons are limited to a maximum of 255 vertices, but OpenGL primitives have no limitation to the number of vertices between `glBegin()` and `glEnd()`.

- Antialiased Line Stippling:

  OpenGL stipples antialiased lines, but GL 3.2 cannot do that.

- Draw and Read Pixels:

  `lrectwrite()` and `lrectread()` subroutines of GL 3.2 cannot be replaced with `glDrawPixels()` and `glReadPixels()` directly. The following code works.

```
void
lrectwrite( int left, int bottom, int right, int top, ulong *pixels )
{
  GLint  viewport[4];

  glGetIntegerv( GL_VIEWPORT, viewport );
  glPushMatrix();
  glLoadIdentity();
  glMatrixMode( GL_PROJECTION );
  glPushMatrix();
  glLoadIdentity();
  glOrtho( 0.0, viewport[2], 0.0, viewport[3], -100.0, 100.0 );

  if( left >= 0 && bottom >= 0 ) {
    glRasterPos2i( left, bottom );
    glDrawPixels( right - left + 1, top - bottom + 1, GL_ABGR_EXT,
    GL_UNSIGNED_BYTE, (GLubyte *)pixels );
  }
  else {
        int tempx, tempy, skipx, skipy, tempw, temph, orgw;
        int ii, ij;
        GLubyte *temppixels, *pt, *qt;

        tempx = ( left < 0 ) ? 0 : left;
        tempy = ( bottom < 0 ) ? 0 : bottom;
        skipx = tempx - left;
        skipy = tempy - bottom;
        orgw = right - left + 1;
        tempw = orgw - skipx;
        temph = top - bottom + 1 - skipy;

        temppixels
         = (GLubyte*)malloc( sizeof(GLubyte) * tempw * temph * 4 );

        pt = (GLubyte*)pixels + skipy * orgw * 4;
        qt = temppixels;
        for( ii = 0; ii < temph; ii++ ) {
              pt += skipx * 4;
              memcpy( qt, pt, tempw * 4 );
              qt += tempw * 4;
              pt += tempw * 4;
        }
        glRasterPos2i( tempx, tempy );
```

```
        glDrawPixels( tempw, temph,
                      GL_ABGR_EXT, GL_UNSIGNED_BYTE, temppixels );
        free( temppixels );
    }
    glPopMatrix();

    glMatrixMode( GL_MODELVIEW );
    glPopMatrix();
}

void
lrectread( int left, int bottom, int right, int top, ulong *pixels )
{
    glReadPixels( left, bottom, right - left + 1, top - bottom + 1,
    GL_ABGR_EXT, GL_UNSIGNED_BYTE, (GLbyte*)pixels );
}
```

Display List

- Defs and Binds:

  OpenGL does not have the concept of material or light objects, only of material and light properties. Use display lists to create material or light objects so that the display lists contain only material properties or light properties. GL 3.2 does not support texture mapping, but the SGI GL later version does and has the concept of texture objects. From OpenGL 1.1, texture objects are supported.

```
void
setlinestyle( int index )
{
    if( index == 0 )
      glDisable( GL_LINE_STIPPLE );
    else {
      glCallList( linestyle_binding_base + 2 );
      glEnable( GL_LINE_STIPPLE );
    }
}
void
setlinestyle( int index )
{
    if( index == 0 )
      glDisable( GL_LINE_STIPPLE );
    else {
      glCallList( linestyle_binding_base + 2 );
      glEnable( GL_LINE_STIPPLE );
    }
}
```

- Fonts and Strings:

  OpenGL expects the character glyphs to be manipulated as individual display lists. It provides a display list calling function that accepts a list of display list names, each name represented as 1, 2, or 4 bytes. `glCallLists()` adds a separately specified offset to each display list name before the call, allowing lists of display list names to be treated as strings.

  This mechanism provides all the functionality of GL 3.2 fonts, and considerably more. For example, characters comprised of triangles can be as easily manipulated.

  There is an OpenGL Character (GLC) library, and it is used to access particular fonts, but the IBM OpenGL implementation does not provide this library yet.

  The following code shows a sample of how to use fonts to draw strings (simulation of `charstr()`):

```
char useFontName[256] =
         "-misc-fixed-medium-r-normal--14-*-*-*-*-*-jisx0201.1976-0";

void makeRasterFont( int base )
{
  XFontStruct  *fontInfo;
  Font   id;
  unsigned int  first, last;

  fontInfo = XLoadQueryFont( display, useFontName );

  if (fontInfo == NULL) {
      fprintf( stderr, "no font found \n" );
      base = 0;
  }
  else {
    id = fontInfo->fid;
    first = fontInfo->min_char_or_byte2;
    last = fontInfo->max_char_or_byte2;

    glXUseXFont( id, first, last-first + 1, base + first );
  }
}

void charstr( char str[] )
{
  glListBase( raster_font_base );
  glShadeModel( GL_FLAT );
  glCallLists( strlen( str ), GL_UNSIGNED_BYTE, (GLubyte *)str );
```

```
  glShadeModel( GL_SMOOTH );
}
```

Others

- Error Checking:

  OpenGL checks for errors more carefully than GL 3.2 does. For example, all OpenGL commands that are not accepted between `glBegin` and `glEnd` are detected as errors, and have no other effect.

  When an OpenGL command results in an error, its only side effect is to update the error flag to the appropriate value. No other state changes are made. (The exception is the OUT_OF_MEMORY error; whether the offending command is ignored or not is undefined.) See the `glGetError()` manual page.

  When an OpenGL command that returns a value detects an error, it always returns zero. OpenGL commands that return data through pointers make no change to the array contents if an error is detected.

- Invariance:

  OpenGL guarantees certain invariances that GL 3.2 doesn't. For example, OpenGL guarantees that identical code sequences sent to the same system, differing only in the blending function specified, will generate the same pixel fragments. (The fragments may be different if blending is enabled and disabled, however.)

9.6.2 Comparison with graPHIGS

This section describes considerations of migration tasks and differences of functionalities between graPHIGS and OpenGL.

9.6.2.1 Consideration of Migration

It is possible that the graphics application currently using the graPHIGS API may be migrated to OpenGL in the future. Such applications may rely on the graPHIGS architecture and utilize its functionalities, and it is natural because the graPHIGS architecture and functionalities matched the needs for those applications. However, many of the functionalities that graPHIGS provides are not supported by OpenGL itself and should be implemented by software during the conversion process.

The migration is possible because, although OpenGL provides a low-level API, a very sophisticated graphics application can be built on top of it. The OpenGL API has been enhanced; therefore, the migration gets a little bit easier than it was before.

If you consider resource usage, graPHIGS initially requires much more memory than OpenGL because it always reserves space at least for one workstation description table, structure store, and so on. Add to that the fact that graPHIGS provides many more subroutines and predefined attributes, and the subroutine set for OpenGL is rather compact.

9.6.2.2 Differences in Functions

There are many differences between OpenGL and graPHIGS. The following descriptions list differences and some similarities. For graPHIGS terms, see 7.2, “Basic Terminology and Concepts” on page 92.

Viewing

- Number of viewports:

  The graPHIGS application can hold a number of views (or viewport) for each window (graPHIGS workstation) at a time, but OpenGL has only one viewport for each context. With graPHIGS, each view has its own view volume and other attributes, such as a drawing priority to the other views, a border color, a background (shielding) color, an HLHSR mode, an antialiasing mode, a transparency (blending) mode, and so on.

- Background Clear:

  There is no clear subroutine in the graPHIGS API. `GPUPWS()` – The update workstation subroutine implicitly clears the window background, then fills each view with its shielding color before primitive drawing if view shielding is enabled.

- Direction of Z:

  The Z axis of the world coordinates points to the viewer in graPHIGS, but it points to the opposite direction in OpenGL. In both, X and Y axes point to the right and top, respectively. The coordinate systems of graPHIGS and OpenGL are right-hand and left hand, respectively.

- Transformation Matrices:

  The graPHIGS view matrix corresponds to the OpenGL projection mode matrix. Combination of global and modeling transformation of graPHIGS corresponds to the OpenGL modelview mode transformation. OpenGL `glRotate*()` or `glTranslate*()` directly change the transformation matrix in a mode, where graPHIGS rotation or translation tool subroutines only produce a matrix from the specified rotation angle around an axis or translation values. The resulting matrix can be multiplied by another matrix. To set or update the global or local transformation matrix, the application passes such a matrix, or application-calculated matrix, to the corresponding transformation subroutine.

- Window Resize:

  Whenever the graPHIGS API window is resized, the device driver establishes a new display surface size within that window. This is the largest subarea of the resized X Window that maintains the aspect ratio provided by the application or user, or the root window by default. The current display contents are scaled uniformly to this new mapped display surface size. The OpenGL application can fit its viewport to the resized window.

Structure Hierarchy

The graPHIGS structure store can be simulated using OpenGL display lists. A graPHIGS structure can call other structures using the Execute Structure subroutine, `GPEXST()`. This mechanism creates the structure hierarchy. OpenGL display lists can also call other display lists. graPHIGS Structure Stores support not only structure hierarchy but also many ways of editing structures. Display lists can't be edited.

Attributes

- Current Color:

  OpenGL's current color (or color index) and materials are applied to all points, polylines and polygons. With the graPHIGS API, colors for polylines, polygon, polymarkers, edges, and text strings are held independently.

- Colormap Table:

  In color index mode with OpenGL, the application uses the X colormap directly. graPHIGS has its own colormap tables.

- Color Setting by RGB or Index:

  There are two types of subroutines to set the colors in OpenGL. `glColor*()` is used for windows with RGB visual, and `glIndex*()` is used for windows with color index visual. graPHIGS also provides two types of subroutines for color setting, and both are valid any time. For example, the Set Polyline Color Index subroutine, `GPPLCI()`, and the Set Polyline Color Direct subroutine, `GPPLCD()`, are available to set the color for polylines.

- Color Model:

  graPHIGS supports four color models: RGB, HSV, CMY, and CIELUV. OpenGL supports the RGB color model only.

- Edge Line Style:

  In the graPHIGS API, edges for polylines or polygons are two different primitives, and line style (line stipple) and edge style are applied to

polylines and edges, respectively. In OpenGL, the line loops and polygons in GL_LINE mode are similar, line stipple pattern, line width, and line antialiasing mode are applied to both, but polygons have front and back facets.

- Line Style:

  The graPHIGS API's line type representations can be roughly simulated in OpenGL with line stipple. But some detailed controls, like SCALED_TO_FIT_RENDERING, should be implemented by the OpenGL application.

- Line End Type:

  With wide lines in graPHIGS, three types of line end-shapes are available: FLAT, ROUND, and SQUARE. In OpenGL, such line representations cannot be implemented without using the polygon primitive.

- Line-on-Line Color:

  If you want to implement the line-on-line facility in an OpenGL application, you will have to use the logical operation with color buffers.

- Interior Style:

  graPHIGS provides five interior styles for polygon rendering:

  HOLLOW Transparent with interior not pickable

  SOLID Filled in with the designated color

  PATTERN Filled in with an application-defined pattern

  HATCH Filled in with a workstation-dependent hatch

  EMPTY Transparent with interior pickable

  HOLLOW and SOLID are similar to the OpenGL polygons in GL_LINE and GL_FILL modes, respectively. As for the pick facility, other than for the HOLLOW type, the OpenGL application should use GL_FILL mode for polygon rendering during the pick operation in GL_SELECT mode. (In this mode, no primitives are drawn in the color buffers.) graPHIGS HATCH interior style can be simulated in OpenGL using the polygon stipple or a combination of scissor test and 2D line drawings. For graPHIGS PATTERN interior style, polygon stipple or texture mapping is available in OpenGL, and the use of scissor test is convenient.

Primitives

- Primitive Drawing Subroutines:

  The graPHIGS drawing subroutines, such as Polyline3 or Polygon 3, are passed vertex data in arrays. OpenGL `glBegin()` and `glEnd()` drawing

needs many more subroutine calls between them, but it has the advantage of flexibility; that is, the color or normal vector, for example, can be changed for any coordinates. With one type of graPHIGS drawing subroutine, such as the Polygon3, you cannot change the color per vertex. With another type, such as Polygon3 With Data subroutine, you can do that, but even if you would like to change only one vertex color, colors should be supplied for all of the vertices.

- Markers:

  Only one of the graPHIGS polymarkers, the dot type, has a counterpart with the OpenGL polypoints. The other types are roughly simulated by a combination of OpenGL current raster position and bitmap drawings. As for graPHIGS polymarkers, their positions are transformed, but their shapes are not affected by the transformations. Their sizes are changed by their scale factors, but the other factors are invariable.

- Text:

  There are two types of text primitive in graPHIGS API, annotation text and geometric text. Annotation text is similar to polymarker, but the character shape of geometric text is affected by the transformations.

- Polygon Edges:

  The graPHIGS polygon edges are defined with a line type and a scale factor (line width) as well as color facilities. OpenGL polygon edge drawing can be simulated using polygon offset, depth test and polygon mode functions as well as line stipple and line width, but the effect of the edge line type may be different from the graPHIGS edge.

- Polygon Tessellation:

  The concave or multicontour (multisubarea) polygons supported by graPHIGS can be drawn using GLU polygon tessellation.

- OpenGL Unique Primitives:

  Among OpenGL primitives, GL_TRIANGLES (individual triangles), GL_QUADS (individual quadrilaterals) and GL_TRIANGLE_FAN are OpenGL unique. The vertex array version of these primitive drawings may be more effective than using graPHIGS drawing subroutines such as Polygon3 or Triangle Strip3.

Client and Server

The graPHIGS nucleus corresponds to the OpenGL server. The OpenGL extension must be loaded to the window system where the graphics windows are displayed. We don't need a specific extension for graPHIGS application if we use X or XSOFT graPHIGS workstation types. The graPHIGS nucleus can

display graphics drawables on the window system of a different machine or of an Xstation through the X protocol. Note that using graPHIGS without X server extensions results in some limitation of functionalities or interactive performance.

The OpenGL extension must be loaded when the window system starts. This server should give the client machines access permission (by xhost on the X Window System). On client machines, a display server should be specified (by the `DISPLAY` environment variable) before the client application is executed.

The graPHIGS nucleus can be started or terminated any time during the X server session. The nucleus should give client machines access permission by the `gPhost` command. From the client application, a nucleus should be specified as "hostname:nucleus_id" or "*" in the application program or PROFILE. "*" specifies a local nucleus for server and $DISPLAY for display.

9.6.3 Open Inventor

The Open Inventor product was originally shipped with the OpenGL and GL 3.2 for AIX Version 4.2 and 4.1. This product was developed by Portable Graphics, Inc. (PGI). However, PGI was acquired by Template Graphics Software, Inc. (TGS). TGS continues to support existing licenses of the PGI product, but does not provide new licenses for PGI's Open Inventor that is included in OpenGL and GL 3.2 for AIX Version 4.2 and 4.1. Now, TGS provides their own Open Inventor products for AIX. TGS encourages PGI customers to upgrade to this new release of Open Inventor from TGS because the PGI product will no longer be updated.

For questions or to place an order for Open Inventor, contact TGI:

`http://www.tgs.com/`

TGS U.S.	`info@tgs.com`	(619) 457-5359, fax (619) 452-2547
TGS Europe	`europe@tgs.com`	+33 (1) 42.37.66.66 fax +33 (1) 42.37.27.15
UNIX Sales	`diane@tgs.com`	(619) 457-5359 ext. 260
Hotline	`support@tgs.com`	(800) 428-7588
Europe Support	`maint@g5g.fr`	+33 (5) 56.13.37.70

OpenGL and GL 3.2 for AIX Version 4.3 do not include Open Inventor anymore.

The following is a short overview of Open Inventor:

Open Inventor is a 3D object-oriented developer's toolkit that assists software developers in developing high-performance 3D graphics applications; however, from an industry acceptance perspective, Open Inventor has not gained wide acceptance to date. One of the reasons is that they have a doubt about its chances of becoming a major standard in the future. Graphics application developers do not want to double their effort to learn a new API. Only a small number of 3D application vendors have selected Open Inventor for their applications.

Open Inventor for AIX exploits the capabilities of OpenGL. Open Inventor's programming model is based on a 3D hierarchical scene database which simplifies and accelerates graphics programming. This new approach replaces the traditional hardware-based drawing model with a more natural, powerful and flexible object model.

Open Inventor can be a cost-effective tool that saves programmers time and effort:

- Extensive functionality:

  The object model introduces numerous concepts that the traditional frame buffer approach cannot.

  For example, Open Inventor offers true geometric picking, 3D locate highlighting, animation, physical constraints, object names, level-of-detail, object culling, interactive direct manipulation, bounding box calculations, selection, automatic render caching, reusable viewers, a standard 3D file format for data exchange, instancing, and 3D cut and paste.

  In addition, its flexible, object-oriented architecture provides an elaborate group of preprogrammed building blocks the programmer can customize and extend.

- Completeness:

  Open Inventor provides the tools a programmer needs to develop high-performance 3D applications that feature animation and a high level of user interactivity.

- Ease-of-Use:

With the ease and modularity of a toolkit, Open Inventor gives an application programmer the functionality of OpenGL while operating at the peak rendering performance of the underlying graphics environment.

- Reduces programming time while extending 3D programming capabilities:

  Open Inventor includes a wide variety of geometry, property and group objects, as well as manipulators for user interaction and high-level viewers and editor components. Its rich set of objects includes cubes, polygons, text material, cameras, light, trackballs, handle boxes, and ready-to-use 3D viewers and editors — all designed to reduce the programming time required of the programmer and to extend their 3D programming capabilities.

- Increases programming productivity:

  Open Inventor can greatly increase programming productivity over programming with straight OpenGL code.

  For example, opening a window with OpenGL requires 14 lines of code. Opening a window with Open Inventor only requires two. Programming a texture map for a 3D object with OpenGL takes more than 200 lines of code, versus 11 with Open Inventor.

- Portable:

  Based on OpenGL and licensed to TGS, Open Inventor applications can easily be ported to other PC platforms and workstations.

9.7 References

This section introduces references for OpenGL and Open Inventor.

9.7.1 OpenGL References

The two main reference books of the Open ARB are part of the Addison-Wesley OpenGL Technical Library:

- *OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.1*, ISBN 0-201-46138-2

  Describes how to create OpenGL 1.1 programs, assuming only a knowledge of C programming. Sample programs from this book are located in /usr/lpp/OpenGL/samples/prog_guide and are also available through anonymous FTP (ftp://sgigate.sgi.com/pub/opengl/opengl.tar.Z)

- *OpenGL Reference Manual*, ISBN 0-201-46140-4

  Provides a technical view of how OpenGL operates on data that describes a geometric object or an image to produce an image on the screen. This

book also contains full descriptions of each set of related OpenGL commands, the parameters used by the commands, the default values for those parameters, and what the commands do.

The information in this book is also available on the AIX Version 4.3 Base Documentation CD.

Books specific to operating systems are also part of the Addison-Wesley OpenGL Technical Library:

- *OpenGL Programming for the X Window System*, ISBN 0-201-48359-9

  Describes how to tightly integrate OpenGL applications with the X Window System.

Also see the following Web sites

- `http://www.rs6000.ibm.com/software/OpenGL/`

  This page contains information about products and activities relating to the OpenGL 3D graphics API, both inside and outside of IBM. It also has many links to useful Web sites including:

- `http://www.opengl.org/`
- `http://www.opengl.org/Documentation/Specs.htm`
- `http://www.specbench.org/gpc/opc.static/`

9.7.2 Open Inventor References

The following reference books are part of the Addison-Wesley OpenGL Technical Library:

The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor, Release 2, ISBN 0-201-62495-8

The Inventor Toolmaker: Extending Open Inventor, Release 2, ISBN 0-201-62493-1

Open Inventor C++ Reference Manual: The Official Reference Document for Open Inventor, Release 2, ASIN 0-201-62491-5

Chapter 10. PEX

This chapter describes PEX and the IBM PEX product, which has not become a major 3D graphics API and is now only supported by AIX Version 4.2.1 systems. AIX Version 4.3 doesn't support the PEX API. Therefore descriptions in this chapter are provided only for migration from current PEX applications.

For applications that still require that interface, AIX Version 4.2.1 remains available. IBM encourages PEX customers to migrate to the preferred 3D API standard, OpenGL.

10.1 Definition

PEX is a protocol (like X11) and designed as a 3D extension to the X server, the PEX extension. The PEX protocol is generated when an application program accesses the PEX extension by programming in an API such as PHIGS or PEXlib.

PEX products provide interoperability for 3D graphics functions in networked heterogeneous environments. Target market segments include industrial design and modeling (CAD), engineering analysis, entertainment, and scientific visualization segments.

However, from an industry perspective, PEX has not gained wide acceptance to date, but OpenGL has gained lots of momentum in the industry. Only a small number of 3D application vendors have selected PEX as the basis for their applications.

10.1.1 PEX Extension to the X Server

The advantages of PEX include:

- Interoperability:

  Enables users to execute applications on one vendor's machine while simultaneously displaying the 3D graphical output on a second vendor's platform.

- Open API standards:

  Designed for portability and interoperability of applications across platforms.

- No licensing requirements:

Provides an inexpensive access to the standard. All specifications and materials available through the Open Group are free and available to the public, including the CGE PEX 5.1 extensions. (`http://www.opengroup.org/`).

PEX supports immediate mode as well as retained and mixed mode rendering through the structure store, a hierarchical graphical database manager.

10.1.2 PEXlib

The API provided with the PEX Sample Implementation, PEX-SI, is called PEXlib. PEXlib exposes the full power of PEX, including server-side implementation of lighting, shading, modeling and viewing transformations, and many of the other features found in PHIGS and PHIGS PLUS.

10.1.3 Graphics Environment PEX 5.1 Extensions (CGE PEX 5.1)

With the Common Graphics Environment (CGE) programming interface, the user can create highly portable 3D graphic applications on any platform that supports the Common Open Software Environment (COSE). CGE offers users increased portability and eliminates the need for vendor-specific code paths or drivers. It also provides access to some PEX 5.2 functionality:

- Texture Mapping:

  Enables the user to project pixmap-like images onto various PEX primitives.

- Antialiasing:

  Enables the user to create higher quality pictures.

- Transparency:

  Offers the user the ability to create objects that transmit light to varying degrees. Though defined in the PEX 5.1 specification, it was not a required feature.

- New Drafting Primitives:

  Maintains interoperability by making typical PEX 5.1 vendor extensions more standard. These primitives include circles, ellipses, and elliptical arcs.

CGE also requires several functions, which are optional with PEX 5.1. The CGE PEX 5.1 extensions conform to the rules and interoperability conventions defined by the X Consortium standard. This means that if a

customer upgrades to CGE PEX 5.1, all of their PEX 5.1 applications will still work without requiring modifications.

CGE is shipped with PEX and PHIGS Version 4.2.1 for AIX. It includes the CGE libraries, online documentation (a reference manual derived from the CGE PEX 5.1 specification and a CGE portability guide), and sample programs.

10.1.4 Technical Content of PEX

The PEX and PHIGS Version 4.2.1 for AIX product fully supports PEXlib 5.1 as the primary PEX programming interface. With PEXlib and CGE PEX 5.1 extensions, the user has access to graphics functions including:

- Basic primitives:
  - 2D and 3D text
  - Markers
  - Lines
  - Polygons
- Extended drafting primitives:
  - Circles and circular arcs
  - Ellipses and elliptical arcs
- Advanced primitives:
  - Triangle strips
  - Quadrilateral meshes
  - Fill area sets (SOFAS)
  - Concave and multi-contour polygons
  - Non-Uniform Rational B-Spline (NURBS) curves
  - Trimmed and untrimmed NURBS surfaces
- Hierarchical structure store including:
  - Object inheritance and attribute sharing
- A networking protocol including:
  - Server-side implementation of lighting, shading, modeling, and viewing
  - Transformations with retained structures
- User-defined clipping volumes
- Antialiasing of primitives

- Depth cueing and fog
- Transparency
- Texture mapping:
  - Texture creation from array or window
  - Texturing of with-data primitives
  - Replacing or blending with the primitive's intrinsic color
  - Interpolation based on perspective correction criteria
- Utilities for creating color maps, extended visuals, and windows

In addition, the PEX and PHIGS Version 4.2.1 for AIX offering includes PEX and CGE PEX 5.1 example programs, application development tools, an installation verification program, and online hypertext information with links to user-modifiable sample programs.

10.2 IBM Implementation

PEX and PHIGS Version 4.2.1 for AIX supports the X Consortium standard, PEXlib 5.1, and the Common Graphics Environment (CGE) PEX 5.1 Extensions. It follows the PEX protocol for distributed client/server support approved by the PEX-Interoperability Center (PEX-IC), a multivendor organization established for PEX vendors to test early releases and verify conformance within a distributed environment. CGE offers increased portability and access to some PEX 5.2 functionality, including some new drafting primitives.

The PEX server is implemented in two ways by IBM:

- IBM Softgraphics technology:

  Using the RS/6000 for all rendering operations, this provides full 3D function at lower cost. See 6.3, “Softgraphics” on page 84.

  The IBM PEX server provides functions such as culling, lighting, shading, and HLHSR on low-end, low-cost graphics adapters, including the 2D graphics adapters. Be aware that performance scales with the amount of CPU available.

- Hardware-accelerated graphics adapter:

  All or part of rendering to the PEX server is done by the adapter hardware and microcode. Section 1.2, “Different Classes of Graphics Adapters” on page 9, describes the role of class II and III graphics adapters.

Supported 2D and 3D graphics adapters:

- POWER GXT150 family
- POWER GXT250 family
- POWER GXT500 family
- POWER GXT800 family
- POWER GXT1000 family

10.3 Configuration

PEX and PHIGS Version 4.2.1 for AIX is offered as a Licensed Program Product for IBM AIX Version 4.2.

10.3.1 Filesets

There are four filesets included in this product :

PEX_PHIGS.PEX.rte	PEX Runtime Environment
PEX_PHIGS.PEX.dev	PEX Device Dependent Software
PEX_PHIGS.PEX.adt	PEX Application Development Toolkit
PEX_PHIGS.info	PEX Programming Guide

The first two filesets must be installed to run PEX applications on the system. To develop PEX programs on the system, it is necessary to install the PEX Application Development Toolkit fileset.

Be aware that directories and files have changed from previous releases.

10.3.1.1 PEX Runtime Environment

The fileset PEX_PHIGS.PEX.rte contains two modules:

PEX_PHIGS.PEX.rte.base PEX Base Runtime Environment

This provides:

- /usr/lpp/X11/README.PEX. See 10.3.3, “Other Information in /usr/lpp/X11/README.PEX” on page 200.
- Run time code under /usr/lpp/X11/bin/
- A library file /usr/lpp/X11/lib/libPEX5.a (/usr/lib/libPEX5.a links to it)
- Font files under /usr/lpp/X11/lib/X11/fonts/PEX/
- An installation verification program /usr/lpp/X11/Xamples/pex/ivp/cube. See 10.3.3.2, “Other Notes about PEX Server” on page 201.

PEX_PHIGS.PEX.rte.soft PEX Soft Runtime Environment

This provides soft graphics code for SoftGraphics PEX.

10.3.1.2 PEX/graPHIGS Device-Dependent Software

The fileset PEX_PHIGS.dev provides the following device-dependent software modules:

PEX_PHIGS.dev.pci.14103c00	GXT250P/GXT255P
PEX_PHIGS.dev.pci.14105400	GXT500P/GXT550P
PEX_PHIGS.dev.pci.14105e00	GXT800P
PEX_PHIGS.buc.00004002	GXT500
PEX_PHIGS.mca.8ee3	GT4
PEX_PHIGS.mca.8ffd	GTO
PEX_PHIGS.mca.8fbc	GXT1000

10.3.1.3 PEX Application Development Toolkit (ADT)

The fileset PEX_PHIGS.PEX.adt provides:

- PEX PEXlib Application Development Toolkit

  See 10.3.3.3, “PEXlib Programming Interface” on page 202, and 10.3.3.4, “CGE Extensions to PEX 5.1” on page 202.

- PEX SI-PHIGS Application Development Toolkit

  See 10.3.3.5, “PHIGS-SI Programming Interface” on page 203.

- PEX Application Development Toolkit Tools

The following is a more detailed view of what is included in these filesets:

PEX_PHIGS.PEX.adt.pexlib.include PEXlib ADT Include Files

This provides include files such as PEX.h or PEXlib.h under /usr/lpp/X11/include/X11/PEX5/.

PEX_PHIGS.PEX.adt.pexlib.clients PEXlib ADT Sample Clients

This provides source code of PEXLIB5.1 and CGE clients under/usr/lpp/X11/Xamples/pex/pexlib/5.1 or cge/clients/.

PEX_PHIGS.PEX.adt.pexlib.util PEXlib ADT Utilities

This provides source code of PEXLIB5.1 and CGE utilities under/usr/lpp/X11/Xamples/pex/pexlib/5.1 or cge/util/.

PEX_PHIGS.PEX.adt.si_phigs.include SI-PHIGS ADT Include Files

This provides include files for SI-PHIGS under /usr/lpp/X11/Xamples/pex/si-phigs/include/phigs/.

PEX_PHIGS.PEX.adt.si_phigs.clients SI-PHIGS ADT Clients

This provides source code of SI-PHIGS clients under /usr/lpp/X11/Xamples/pex/si-phigs/clients/.

PEX_PHIGS.PEX.adt.si_phigs.lib SI-PHIGS ADT Library

This provides /usr/lpp/X11/Xamples/pex/si-phigs/lib/libphigs.a and /usr/lpp/X11/Xamples/pex/si-phigs/lib/X11/PEX/phigsmon and so on.

PEX_PHIGS.PEX.adt.si_phigs.man SI-PHIGS ADT Man Pages

This provides README and man pages under /usr/lpp/X11/Xamples/pex/si-phigs/man/.*

PEX_PHIGS.PEX.adt.tools PEX ADT Tools

This provides a tool, pexstdcmap, and its source code under /usr/lpp/X11/Xamples/pex/tools/.

10.3.1.4 PEX Programming Guide

The fileset PEX_PHIGS.info.en_US provides the following module:

PEX_PHIGS.info.en_US.PEX PEX Programming Guide - U.S.

10.3.2 Installation

The PEX and PHIGS Version 4.2.1 for AIX is a fully integrated IBM product and is installed using the usual system administration tools, such as SMIT. Once this product is installed, you have to add extensions to the server in order to use it.

10.3.2.1 X Server Extensions

The commands used to start up the PEX Extension to X Window System automatically are in /usr/lpp/X11/defaults/xserverrc. To start-up automatically under `xinit`, edit the file and change the line reading:

```
EXTENSIONS=""
```

as follows:

```
For X11R5: EXTENSIONS="-x pex -x dbe"
For GXT1000: EXTENSIONS="-x pex -x abx -x dbe"
```

If the OpenGL is installed, `-x abx` and `-x dbe` are specified with -x GLX in another line in this file; so, you will just have to change the line to:

```
EXTENSIONS="-x pex"
```

Or, to start-up manually, use `xinit` command-line options as follows:

```
For X11R5: EXTENSIONS="-x pex -x dbe"
For GXT1000: EXTENSIONS="-x pex -x abx -x dbe"
```

Once again, if OpenGL is installed, type

```
xinit -- -x pex
```

On this command line, pex refers to the PEX Extension to X, abx refers to the X Windows Ancillary Buffer Extension, and dbe refers to the X Windows Double Buffer Extension (DBE). DBE is an official X standard and has replaced the non-standard X Multi-Buffer Extension (MBX).

Note

Starting the X server with -x mbx instead of -x dbe still works on an AIX Version 4.2.1 system, but not on an AIX Version 4.3.2 system.

If -x pex is not specified, you get the following error message when you run a PEX application.

```
Could not access PEX extension.
Message returned from PEXInitialize: Could not initialize the PEX
extension on the specified display
```

To obtain the list of the extensions currently loaded to the X server, run xdpyinfo (/usr/bin/X11/xdpyinfo) on your local system, or run it with the -display option from a remote system.

Other Useful X Server Options

Some PEX client programs, such as some older ones, create windows that inherit attributes from the root window. These clients perform better on 24-bit adapters if the root window is configured to a depth of 24-bits.

It is also often advantageous to configure the root window to use a TrueColor visual on 24-bit adapters to promote colormap sharing between all clients running on the same server.

To start X with the PEX extension, a 24-bit deep root window and a TrueColor visual, the xinit command is:

For X11R5: `xinit -- -x pex -x dbe -d 24 -cc 4`

For GXT1000: `xinit -- -x pex -x abx -x dbe -d 24 -cc 4`

10.3.2.2 /usr/bin/X11/xstdcmap

Many PEX clients search for a standard colormap on an X server and use it if one is found. This promotes colormap sharing and avoids executing the series of steps needed to create and initialize a colormap. If a client program starts and terminates almost immediately issuing a message involving a standard colormap that it couldn't find, run:

```
xstdcmap -all
```

This command creates all the standard colormaps and makes them available to the client programs. Run this command early in an X session so that all clients that seek and use standard colormaps can take advantage of them.

10.3.2.3 /usr/lpp/X11/Xamples/pex/tools/bin/pexstdcmap

The command `pexstdcmap` performs the same steps as `xstdcmap`, except that it can guarantee that the colormaps are usable by the PEX server, in terms of color approximation. Use the `-pex` option to force `pexstdcmap` to create these standard colormaps.

See /usr/lpp/X11/Xamples/pex/tools/src/pexstdcmap for the source files.

10.3.2.4 Environment Variable IBMPEXSOFT

With AIX Version 4.2.1, the PEX server extension operates on all IBM graphics adapters except the GrayScale Graphics Adapter and GXT3000P. The PEX server extension utilizes the graphics accelerator hardware on the GtO, Gt4, Gt4X, Gt4e, and Gtx1000 graphics adapters. On all other supported adapters, the server uses IBM SoftGraphics technology to generate the graphics with software.

Note

The IBM SoftGraphics PEX support is shipped as an optional Runtime Environment package, PEX_PHIGS.PEX.rte.soft, that must be installed to use PEX on an adapter that is not fully hardware accelerated. If you use hardware-accelerated adapters or if you do not use PEX, then you may decide not to install this fileset since it requires approximately 20 MB of disk space.

If the IBM SoftGraphics PEX support is not installed and you attempt to use this support, the PEX extension will not be available, but you can continue to use the X server without PEX.

If you wish to use the software-based PEX server, even when running on a hardware-accelerated adapter, you should set an environment variable before starting the server as follows:

For ksh: `export IBMPEXSOFT=1`

For csh: `setenv IBMPEXSOFT 1`

10.3.3 Other Information in /usr/lpp/X11/README.PEX

On an AIX Version 4.2.1 system, /usr/lpp/X11/README.PEX includes the other information as follows.

10.3.3.1 PEX Specification Deviations

The PEX server extension implements the PEX 5.1 specification with the following exceptions:

All Adapters:

PEXSearchNetwork Request is not implemented. A BadImplementation error is returned to the client if the client sends this request to the server.

Hardware Accelerated Adapters:

- Back Face Attribute InteriorStyle, InteriorStyleIndex, ReflectionAttrs, ReflectionModel and SurfaceInterpMethod Output Commands are not supported. The implementation uses the surface interior (front) attributes when rendering back-facing surfaces.
- The InteriorBundleLookupTable attributes ReflectionAttrs, ReflectionModel, InterpMethod, BFReflectionAttrs, BFReflectionModel, BFInterpMethod, BFStyle, BFStyleIndex, and BFColor are not supported and have no effect.
- The AnnotationText Output Commands (OC) support only one substring or mono encoding per OC on hardware-accelerated adapters. The server sends the client a PEXOutputCommandError if the client sends an AnnotationText OC with more than one substring.
- All Output Commands over 65532 bytes in length are not supported on hardware accelerated adapters. The server returns a PEXOutputCommandError to the client if the client sends an OC with a length greater than 65532 bytes. The SetOfFillAreaSets (SOFAS) OC is the only exception, as long as each individual FillAreaSet defined by the SOFAS is no longer than 65532 bytes.
- The server on hardware-accelerated adapters can only accept a subset of the possible entries in the Color Approximation Table. Invoke the Color

Approximation Query Escape to determine if the server can support a given entry.

- PEX clients should avoid associating multiple renderers to the same window on hardware-accelerated adapters.
- Clip lists are not supported and are ignored on hardware-accelerated adapters.
- The hardware-accelerated server cannot render to Pixmap Drawables. This condition is reflected in the information returned by the MatchRenderingTargets request.
- Only the POWER GTX1000 can render to Buffer Drawables using hardware acceleration. This condition is reflected in the information returned by the MatchRenderingTargets request.
- In addition to the standard marker types, extended marker types are also available and may be inquired using the `PEXGetEnumTypeInfo` PEXlib subroutine.

Softgraphics Adapters:
The software-based PEX server does not render primitives that contain color types other than RGBFloat and Indexed. The server renders primitives that contain color types other than RGBFloat or Indexed with the color stored in entry 1 of the color lookup table. If entry 1 of the color LUT is not defined, if or the color LUT does not exist, the server renders the primitive with the color white.

10.3.3.2 Other Notes about PEX Server

- The software-based PEX server performs better on 8-bit adapters when the PEX client uses a full 322 or best colormap.
- The virtual memory requirement for the PEX server is larger due to the size of the SoftGraphics rendering code. You may need more real memory or disk paging space to run the PEX server effectively.
- The software based PEX server does not normalize direction vectors that are found in the Light LUT Entries and in primitives that specify vertex or facet normals. The PEX specification requires that the client normalize these vectors for correct results. A PEXlib function is available for this purpose.
- The client program should accomplish double buffering by using the X DBE with the IBM SoftGraphics technology and the POWER GTX1000. You cannot use DBE for double buffering on the GtO, Gt4, Gt4x, and Gt4e adapters. In the case of these adapters, use the double buffering escapes

instead. See /usr/lpp/X11/include/X11/PEX5/extensions for more information.

- To access the color approximation query and double buffering escapes, use the PEXEscape PEXlib function. The header files defining the constants and data structures needed to use these escapes are:

  /usr/lpp/X11/include/X11/PEX5/extensions

- A sample test program is available in: /usr/lpp/X11/Xamples/pex/ivp/cube. You may find this program useful for verifying correct installation and operation of the PEX server.

10.3.3.3 PEXlib Programming Interface

PEXlib is the primary PEX API. This product meets the X Consortium PEXlib 5.1 standard.

- Include files: /usr/include/X11/PEX5
- Library file: /usr/lib/libPEX5.a
- Samples: /usr/lpp/X11/Xamples/pex/pexlib/5.1/clients

With the PEX program product, you have the option to install the PEX/PEXlib InfoExplorer documentation. This information should serve as the primary reference for PEXlib and can be viewed by invoking:

```
info -l pex
```

You are not required to install InfoExplorer Version 3.2.5 or later to view the PEX 4.1.0 InfoExplorer documentation.

The PEX documentation can be used with single-byte languages only.

10.3.3.4 CGE Extensions to PEX 5.1

The CGE extension operates on all IBM graphic adapters except the GrayScale Graphics Adapter and GXT3000P. IBM servers fully support CGE PEX 5.1 on all the adapters through the IBM SoftGraphics technology.

PEXlib

CGE PEX 5.1 is integrated into IBM's PEXlib library. To access the CGE functions, add the preprocessor directives to include the standard PEXlib 5.1 header file and the CGE PEX 5.1 header file:

```
#include <Xll/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h>
```

- Include files: /usr/include/X11/PEX5
- Library file: /usr/lib/PEXExtlib.h
- Samples: /usr/lpp/X11/Xamples/pex/pexlib/cge/clients/ptorus

For the InfoExplorer documentation about CGE PEX 5.1, invoke

```
info -l pex or
info -l pex -h cgetoc
```

to directly locate the CGE PEX 5.1 Portability Guide.

Utility Extensions
CGE offers utilities to assist you with visual and colormap organization. The header files defining the constants and data structures needed to use these utilities are in:

- Include files: /usr/lpp/X11/Xamples/pex/pexlib/cge/util/Cmap
- Library file: /usr/lpp/X11/Xamples/pex/pexlib/cge/util/libPEXUt.a
- Samples: /usr/lpp/X11/Xamples/pex/pexlib/cge/clients/ptorus

10.3.3.5 PHIGS-SI Programming Interface

This is the PHIGS Sample Implementation library from the X11R5 distribution. It implements a PHIGS/PHIGS PLUS style C binding and does not fully conform to the ISO PHIGS PLUS C bindings. The library requires the PEX Workstation Subset in the PEX server, which may not be implemented in some vendors' PEX servers.

- Include files: /usr/lpp/X11/Xamples/pex/si-phigs/include/phigs
- Library file:/usr/lpp/X11/Xamples/pex/si-phigs/lib/X11/PEX
- Samples: /usr/lpp/X11/Xamples/pex/si-phigs/clients
- Man pages: /usr/lpp/X11/Xamples/pex/si-phigs/man
- PHIGS monitor: /usr/lpp/X11/Xamples/pex/si-phigs/lib/X11/PEX/phigsmon

Use the following to set the environment variable for the PHIGS monitor:

(ksh:) `export PEXAPIDIR=/usr/lpp/X11/Xamples/pex/si-phigs/lib/X11/PEX`

(csh:) `setenv PEXAPIDIR /usr/lpp/X11/Xamples/pex/si-phigs/lib/X11/PEX`

The man pages replace the hardcopy manuals, use them as user’s guide and subroutines reference.

Please see the README file in the main directory for more information. The Text Formatting filesets are required to read the `man` command.

10.3.3.6 graPHIGS Programming Interface

If a graPHIGS application uses an XPEX workstation type, it automatically generates the PEX protocol.

10.4 PEX References

The following two books are published by O'Reilly & Associates.

- *PEXlib Programming Manual: 3D Programming in X. Gaskins, T.,* ISBN 1-56592-028-7, 1992
- *PEXlib Reference Manual: 3D Programming in X. Talbott, S.,* ISBN 1-56592-029-5, 1992

Chapter 11. Benchmarking

This chapter discusses some benchmarking issues. Since the performance of the systems is highly subject to changes, we focus more on describing the different benchmarks used in the graphic area and how to obtain the latest results rather than providing figures.

Just remember that, when it comes to standardized benchmarks, your mileage may vary, but, after reading this chapter, at least you will have a fair idea of the type of car you need to buy.

11.1 History

In the past, graphics performance was often measured in terms of vectors per second or polygons per second. Since the terms were ambiguous, it was difficult to compare one vendor's product with another's. Typical discrepancies were the length of the vectors and whether they were being drawn end to end or separately or what size of polygon was being drawn and how it was shaded or what type of lighting was being utilized.

This was not only frustrating for customers but also for vendors.

In late 1986, the major workstation vendors and users met to discuss this lack of standardized methods for measuring graphics performance. From this and subsequent meetings grew the Picture Level Benchmark (PLB) project. This benchmark, created with the assistance of Digital Equipment Corp. (now Compaq), Hewlett-Packard, IBM, and Sun Microsystems, is a software package that provides a practical comparison of graphics display performance for different hardware platforms.

The OpenGL Performance Characterization project was started in 1993, and it published the first numbers for the Viewperf benchmark in the fourth quarter of 1994. The GLperf benchmark was released in August 1997.

These benchmark projects are all part of the Graphics Performance Characterization (GPC) Group which joined the Standard Performance Evaluation Corp. (SPEC) in early 1996.

The latest addition to the GPC is the Application Performance Characterization project group that just announced its first benchmarks in 1998.

11.2 Which Benchmark to Use

The standard benchmark tests, that RS/6000 graphics workstation users should be concerned with, are the Picture Level Benchmark (PLB), the OpenGL Performance Characterization (OPC), and the X Performance Characterization (XPC).

The PLB is designed to measure the performance of CRT based display systems. This includes, but is not limited to, engineering workstations, personal computers, and special-purpose attached display systems.

The OPC is set up to characterize graphics performance for computer systems running applications, not overall graphics performance. The Viewperf benchmark measures 3D rendering performance of systems running under OpenGL. The GLperf benchmark is designed to measure optimal performance of 2D and 3D graphics primitives. Both of these benchmarks are available for download for UNIX, Windows (95 & NT), and for OS/2 from the GPC.

The APC is working to provide benchmarks that measure system performance based on how users typically interact with graphics-intensive applications. They have so far announced a benchmark for SolidWorks 98 CAD/CAM software and for the Quake II game. They are also working on a Pro/ENGINEER benchmark scheduled for release by the end of 1998. The downloadable versions of these benchmarks will only work on Windows platforms. You must have the actual applications installed on your system. They are not provided with these benchmarks.

The XPC project group created the Xmark93 benchmark, which uses the x11perf executable. IBM still reports the XPC benchmark results although the XPC project group has disbanded and has retired the Xmark93 benchmark.

Of course, even though these standard benchmarks give you a very good starting point in your search for the best graphics workstation for your money, the best benchmark is still running your own applications in a real world situation.

11.2.1 How to Run Benchmarks on Your System

The results for the above described benchmarks are published for a given hardware and software configuration. This section explains how you can reproduce these benchmarks on your own system.

11.2.1.1 The PLB Benchmark

This version is not available on the specbench ftp site. When IBM runs this benchmark, the graPHIGS API is used.

11.2.1.2 The OPC Benchmark

Download the GLperf code from the ftp.specbench.org site and view the README file for instructions on how to compile the benchmark test.

Once the GLperf code is compiled, you can issue the following command to start a test:

```
GLperf [-d] [-p] [-s] [-u] input_file [< input_file]
   where -d specifies delta output, default is no delta
   where -p specifies pixel metrics (e.g. pixels/sec), default is objects
   where -s specifies default state delta output, default is no default
   state delta
   where -u specifies microsecond timings, default is objects per second
   input_file is the name of the input file
```

GLperf will also take input from standard input if no input file is given.

You can download the Viewperf code and instructions for running the benchmark from the ftp.specbench.org site along with any of the five viewsets you wish to run. There are several precompiled versions of the Viewperf executable available as well. IBM has supplied one for the PowerPC platform as well as one for Windows NT on the x86 architecture.

Viewsets are designed by the vendors and not by the OPC project group although the OPC will assist the vendors. The five standard viewsets are:

1. Parametric Technology's *CDRS* (to be replaced soon by Parametric Technology Corporation's *Pro/DESIGNER*)
2. IBM's *Data Explorer*
3. Intergraph's *DesignReview*
4. Alias/Wavefront's *Advanced Visualizer*
5. Lightscape Technology's *Lightscape Visualization System*

Be aware that the viewsets have had some new versions released and ensure that you are comparing the same versions of benchmarks across all platforms under test.

The Viewperf benchmarks are run from shell scripts included with the individual viewsets, for example, ~/viewperf.5.1/Light-01/Light-01.sh.

11.2.1.3 The XPC Benchmark

The x11perf code is included in your shipment. It can be found in /usr/lpp/X11/Xamples. You may have to build the example code. If so, follow the instructions in the /usr/lpp/X11/Xamples/README file.

You will also have to ensure that the base Japanese fonts are installed (`lslpp -l bos.loc.pc.Ja_JP`).

Prior to starting the X server, you should execute:

```
export X_SHM_SIZE=252.
```

To start the benchmark, run the following command:

```
x11perf -all -rop GXcopy GXxor -repeat 2
```

Then direct the output to a temporary file.

11.2.2 How to Interpret Benchmarks Results

If you have followed the previous step, you should now have a resulting file. This section covers how to interpret these results.

11.2.2.1 The PLB Results

Currently, the PLB results are reported in terms of PLBwire93 and PLBsurf93.

Only one number is significant in each of these categories: PLBwire93 and PLBsurf93. These numbers represent the geometric mean of the PLBlit and PLBopt figures for the standard benchmark files in the two categories. The PLBlit number is obtained by running the benchmark with no optimization for the specific hardware. The PLBopt number is from running the benchmark with an optimization for the specific hardware.

The benchmark consists of six major components:

1. The Benchmark Interchange Format (BIF), the file format for specifying the geometry and actions that will be performed in a test.
2. The Benchmark Timing Methodology (BTM), which provides a standardized performance measurement.
3. The Benchmark Reporting Format (BRF), for standardized reporting of test results.
4. The Picture Level Benchmark (PLB) program, which implements BIF file processing and runs the test. This program is platform-dependent and must be ported to the device under test.
5. A suite of files for testing PLB implementation.

6. A suite of BIF standard benchmark files that are used for graphics performance tests.

The best method for running this benchmark is for the users to convert their applications into BIF files and run them directly on the vendors' ports of the PLB program. However, since few users may have the time or technical expertise to do this, the PLB project group has developed eight BIF files based on popular applications.

It is important to note that although the PLB allows buyers to compare performance, it does not address the issue of display quality. This subjective issue is left to the eyes of the beholder.

PLBwire93 is a composite of three tests: sys_chassis, race_car, and seafloor. These benchmarks are representative of entry level 3D wireframe applications. These benchmarks are rendered without Z buffering, depth cueing, anti-aliasing, or wide lines.

The file designated as sys_chassis is a 3D wireframe model of a computer chassis. During the course of the test, the chassis is rotated, panned, zoomed, and viewed from different perspectives. The chassis is composed of 6,107 3D solid polylines and 158 3-D dashed polylines. The number of vertices per polyline ranges from two to 11; there is a total of 19,064 vectors. Five hundred (500) frames are displayed in this test.

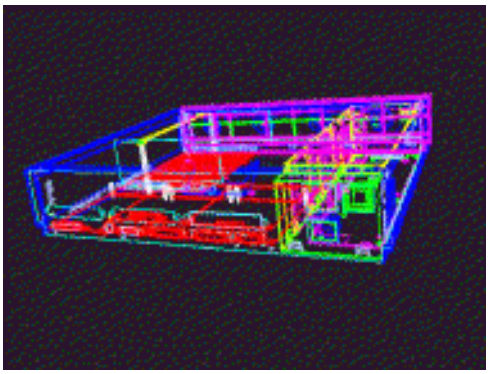


Figure 28. The sys_chassis Model

The race_car file is a data set representing a 3D wireframe model of a race car that is seen from different views. Unigraphics 3D modeling software was used to create the car model. The model is displayed and animated in four

different simultaneous views on the screen. There is an average of 17,917 3D solid polylines containing 141,934 vectors with 129 color attribute changes per frame. The animation sequence contains 600 frames.

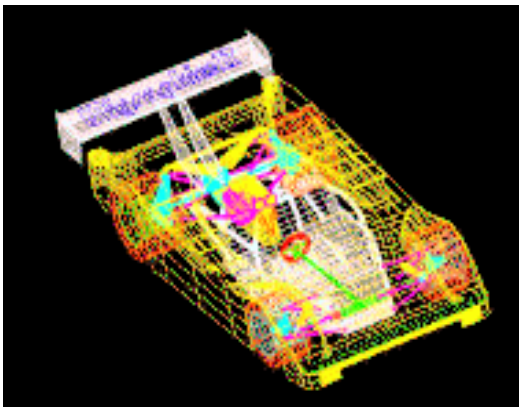


Figure 29. The race_car Model

The seafloor file is a 3D wireframe model representing the contour lines of the ocean floor and the islands above. This data is typical of applications that do mapping of seafloor terrain for exploration purposes. Color is used to depict elevations, with deep blue representing the deepest part of the ocea, and red the highest areas above sea level. The sea-level boundary is depicted as a white contour line. While the file is being run, the terrain data is rotated 360 degrees, followed by additional translations and rotations. The data set comprises 984 contour lines on 211 depth levels. Each contour line is represented as solid 3D polylines. There are 540 total frames with an average of 229,682 vectors per frame.

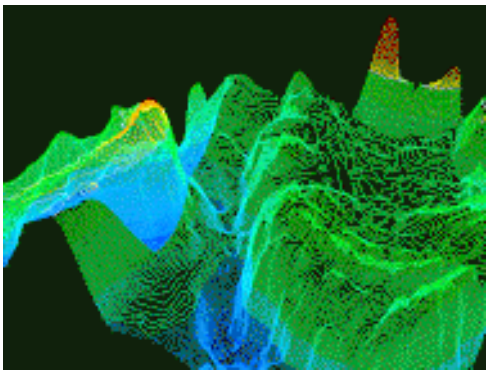


Figure 30. The Seafloor Model

The PLBsurf93 is a composite of four benchmarks: cyl_head, head, shuttle, and studio. These benchmarks are representative of applications which shade 3D surfaces. All of these benchmarks utilize Z-buffering and Gouraud shading, and some incorporate multiple light sources.

The cyl_head file contains a 3D solid model of an automobile engine's cylinder head. The test reflects a typical 3D mechanical CAD application. The cylinder head is rotated, translated and zoomed in on during the test. It is composed of 3,621 3D polygons and 32 3D fill-area sets. There is an average of 4.8 vertices per polygon.

The fill area sets have an average of 55.7 vertices and 2.7 contours per frame. There are 225 frames displayed in this test. All colors in this file are true color (RGB) values, and all polygons are Gouraud shaded. There are three light sources that use ambient, diffuse and specular light components.

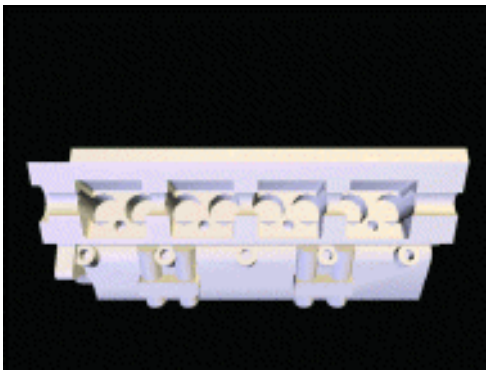


Figure 31. The cyl_head Model

The head file depicts a 3D human head modeled using data generated by a laser scanner. This type of file is typical of applications in animation, special effects, and biomedical areas. The data set consists of nearly 60,000 triangles that are rendered using multiple T-mesh strips. There are four directional light sources illuminating the object as it rotates three times around the Y-axis in 240 frames.

Figure 32. The Head Model

The shuttle file is an example of low-end simulation consisting of 560 frames. The scenario depicts the rendezvous of a space shuttle with a satellite in a low planet orbit. An astronaut uses the shuttle's robot arm to maneuver within repair distance of the satellite. The shuttle and satellite are modeled using

quadrilateral meshes, triangle strips, and polygons for a total of 3,355 facets. The planet model consists of quadrilateral meshes totalling 5,632 facets. Constellation data was derived from the BOSS star catalog and is modeled using 2,283 3D markers. Three light sources - ambient, directional and point - are used to light the scenes.

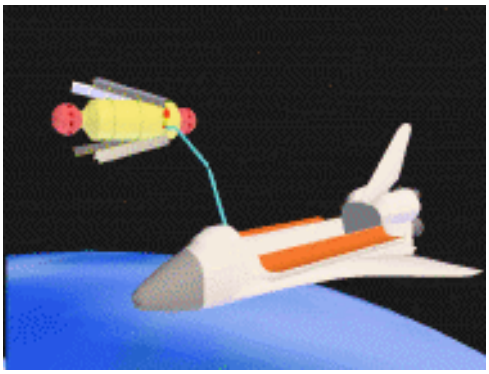


Figure 33. The Shuttle Model

The studio file is an architectural walk-through consisting of 300 frames. The viewer is given a tour of the interior of a photorealistic two-floor design studio, which features light coming from a variety of sources. Lighting conditions are generated using radiosity methods that were precomputed outside the PLB. The file contains 7,518 quadrilaterals in eight MULTI_POLYGON3's; the first seven MULTI_POLYGON3's have 1,000 quads, and the last has 518 quads. There are no light sources; shading is accomplished with the vertex colors present on every quad. No facet data is present in the file.

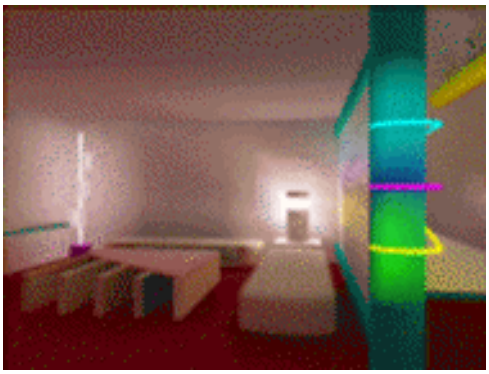


Figure 34. The Studio Model

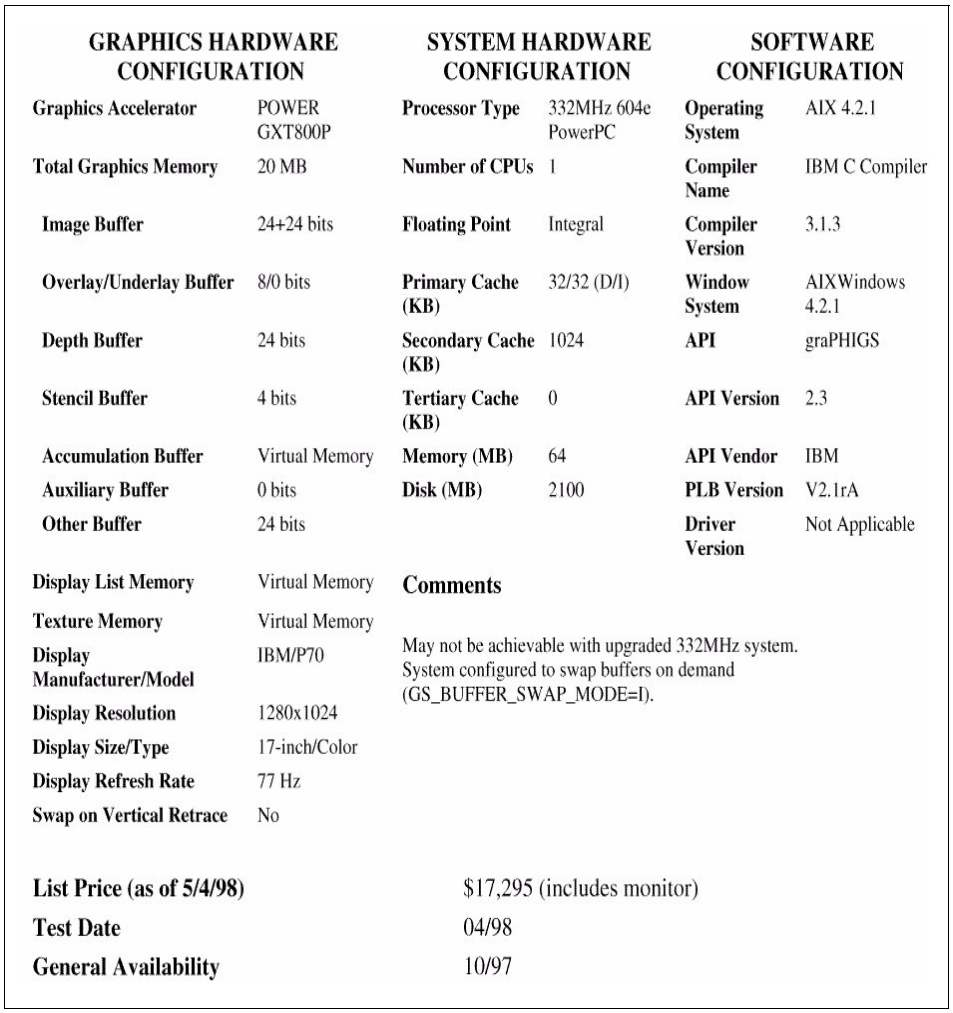
The oceantopo file is an extension of the database used to produce the current standard benchmark called seafloor. Like seafloor, the new file is a 3D wireframe model of the ocean floor contour lines and some islands above. The data is typical of applications that do mapping of seafloor terrain for exploration purposes. Color is used to depict elevations, with deep blue representing the deepest part of the ocean and red the highest areas above sea level. The sea-level boundary is depicted as a white contour line. While the model is being rotated and translated, the depth-cue mapping is changed several times to make the foreground data more visible. The data set comprises 991 contour lines on 211depth levels. Each contour line is represented as anti-aliased, depth-cued 3D polylines. The file contains a total of 600 frames with an average of 459,809 vectors per frame.

Once you have run the PLB benchmark an intermediate file, the .BRF file is created. An example of this file is shown in Appendix B.1, “A Sample BRF File” on page 253. This information is then processed to generate the PLB report as shown in the following figures.

IBM RS/6000 Model 43P-140 332Mhz POWER GXT800P graPHIGS			
Composite PLBmarks			
PLBwire93		156.8	
PLBsurf93		265.3	
Benchmark Name	**Normalizing Factor**	**PLBmarks**	
		PLB_{lit}	**PLB_{opt}**
3D Wireframe			
1. sys_chassis	1200	173.9	236.8
2. race_car	6800	165.7	NA
3. seafloor	6200	111.2	118.3
3D Surface			
4. cyl_head	1200	175.9	228.2
5. head	4800	281.4	NA
6. shuttle	4000	272.1	NA
7. studio	2500	322.7	NA
Other			
8. oceantopo	25000	123.7	166.8

Optimizations	Used in Benchmark #
Connect common endpoints	1
Sort lines by attribute	1
Mark triangulatable polgons as convex	4
Some primitives decomposed into convex polygons	4
Allow rotation of polygon vertices to search for triangulatable p	4
Facet normals provided where not provided by BIF	4
Limit lines to 100 vertices per line	3,8

Figure 35. PLB Report Page 1



GRAPHICS HARDWARE CONFIGURATION		SYSTEM HARDWARE CONFIGURATION		SOFTWARE CONFIGURATION	
Graphics Accelerator	POWER GXT800P	Processor Type	332MHz 604e PowerPC	Operating System	AIX 4.2.1
Total Graphics Memory	20 MB	Number of CPUs	1	Compiler Name	IBM C Compiler
Image Buffer	24+24 bits	Floating Point	Integral	Compiler Version	3.1.3
Overlay/Underlay Buffer	8/0 bits	Primary Cache (KB)	32/32 (D/I)	Window System	AIXWindows 4.2.1
Depth Buffer	24 bits	Secondary Cache (KB)	1024	API	graPHIGS
Stencil Buffer	4 bits	Tertiary Cache (KB)	0	API Version	2.3
Accumulation Buffer	Virtual Memory	Memory (MB)	64	API Vendor	IBM
Auxiliary Buffer	0 bits	Disk (MB)	2100	PLB Version	V2.1rA
Other Buffer	24 bits			Driver Version	Not Applicable
Display List Memory	Virtual Memory	Comments			
Texture Memory	Virtual Memory				
Display Manufacturer/Model	IBM/P70	May not be achievable with upgraded 332MHz system. System configured to swap buffers on demand (GS_BUFFER_SWAP_MODE=I).			
Display Resolution	1280x1024				
Display Size/Type	17-inch/Color				
Display Refresh Rate	77 Hz				
Swap on Vertical Retrace	No				

List Price (as of 5/4/98)	$17,295 (includes monitor)
Test Date	04/98
General Availability	10/97

Figure 36. PLB Report Page 2

11.2.2.2 The XPC Results

After you complete the x11perf benchmark, you should run the results file through the Xmark.sh script that is in the /usr/lpp/X11/Xamples/bin directory.

The output from the Xmark.sh script contains three numbers:

- The weighted x11perf number for the server under test
- The weighted x11perf number of a Sun SparcStation 1
- The Xmark - the ratio of the first two numbers

It will appear similar to this:

```
Weighted x11perf of International Business Machines server = 54025
Weighted x11perf of SparcStation 1 server = 2119
Xmark = 25.5012
```

The higher the Xmark value, the better it is.

The x11perf benchmark creates a series of standard X11 graphics patterns and measures the time it takes to draw them on the display.

11.2.2.3 The OPC Results

Viewperf measures performance for the following entities:

- 3D primitives, including points, lines, line_strip, line_loop, triangles, triangle_strip, triangle_fan, quads and polygons
- Attributes per vertex, per primitive and per frame
- Lighting
- Texture mapping
- Alpha blending
- Fogging
- Anti-aliasing
- Depth buffering

Viewperf is not a single-number benchmark. In order to use it to its fullest advantage, ISVs and users need to relate the benchmark to their actual applications. Here are the five steps recommended for using Viewperf effectively:

1. Identify software code paths that are important to the application.
2. Identify the primitives used within the application.
3. Select datasets that are most appropriate to the application. The datasets should reflect the level of geometry and rasterization found in the application.
4. Identify attributes and the level at which they are applied (per vertex, per primitive or per frame).
5. Assign a weight to each path based on the percentage of time in each path and the importance of the path to the application.

The Viewperf program is command line driven. The viewset's shell scripts may call the Viewperf executable multiple times with different options.

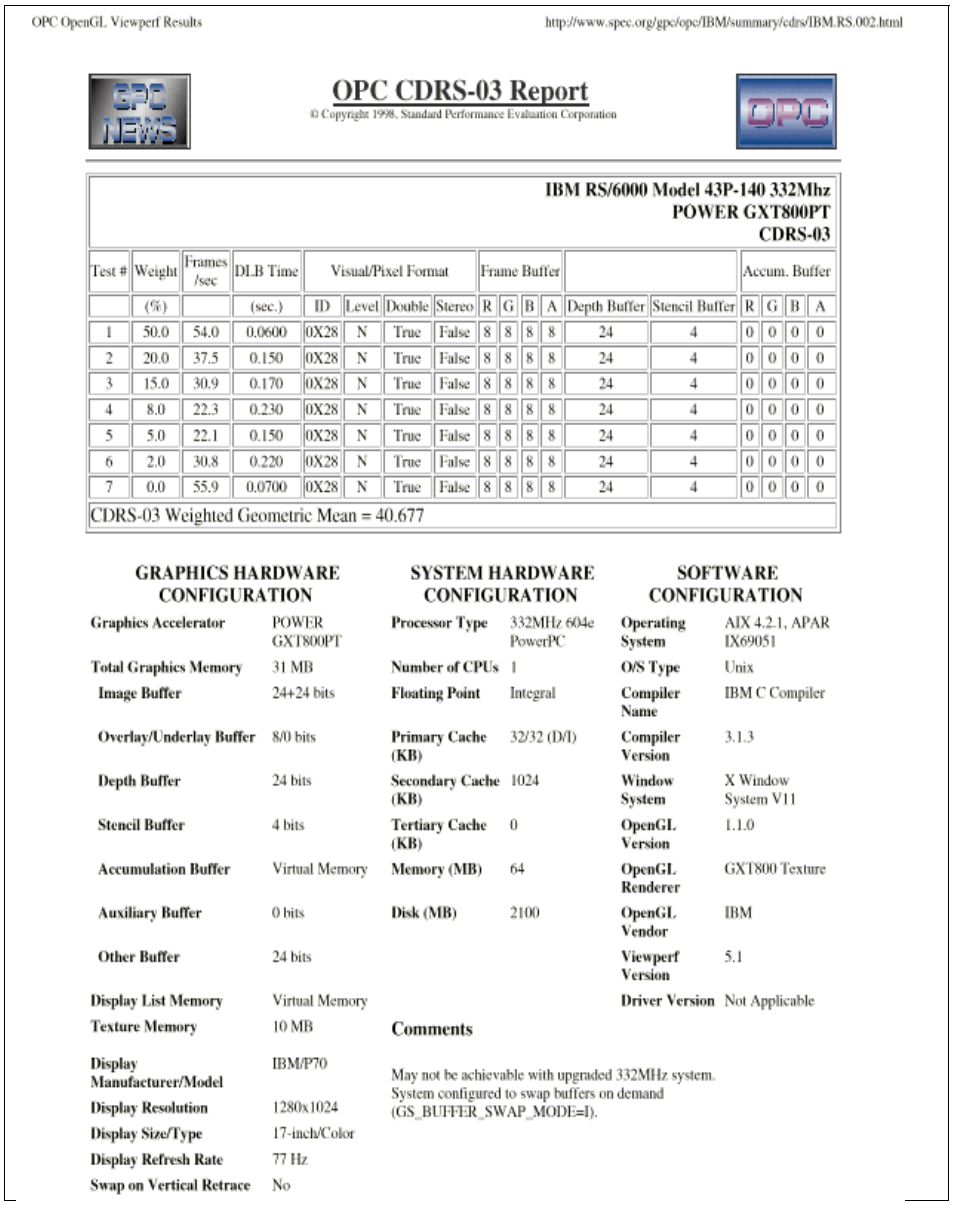
The possible options are:

```
-polygon -pg   <file>  : Viewpoint object to be used in the tests
```

```
-triangle -tr  <file>  : Viewpoint object to be used in the tests
-quad -qd      <file>  : Viewpoint object to be used in the tests
-mesh -mh      <file>  : Mesh object to be used in the tests
-rendermode -rm <mode> : POINT, VECTOR, LINE, POLYGON, TMESH, TFAN,
                         TRIANGLE, or QUAD - default LINE
-vcriteria -vcrit      : AUX Visual selection criteria - EXACT, MIN
                         - default MIN
-vid <id>              : Ask AUX for visual with ID = <id>
-vaccum -vac           : Ask AUX for an accumulation buffer visual
-valpha -val           : Ask AUX for an alpha buffer visual
-vdepthbuffer -vz      : Ask AUX for a depth buffer visual
-vstencil -vst         : Ask AUX for a stencil buffer visual
-indirectrender -ir    : Render indirect - default direct
-nodither -ndi         : Disable dithering
-ortho -or             : Parallel/Orthographic projection - default
Perspective
-displaylist -dl       : Render with display list mode
-colorper -cp  <mode>  : FRAME = Color per Frame,
                       : PRIMITIVE = Color per Primitive,
                       : VERTEX = Color per Vertex - default FRAME
-texture -tx   <file>  : MTV image for texturing
-texgen -txg <file> <mode> : <file> is MTV image for environment mapping
                         <mode> is SPHERE_MAP, OBJECT_LINEAR, EYE_LINEAR
                         - default EYE_LINEAR
-magfilter -magf <flt> : NEAREST, LINEAR - default NEAREST
-minfilter -minf <flt> : NEAREST, LINEAR, NEAREST_MIPMAP_NEAREST,
                         LINEAR_MIPMAP_NEAREST, NEAREST_MIPMAP_LINEAR,
                         LINEAR_MIPMAP_LINEAR - default NEAREST
-texenv -te <env>      : Texture environment, MODULATE, DECAL, BLEND
                         - default DECAL
-texcomp -tc <num>     : Texture components where <num> is 1,2,3, or 4
                       : -default 3
-blend -bl             : Enable Blending
-srcblendfunc -sbf     : ZERO, ONE, DST_COLOR, ONE_MINUS_DST_COLOR,
SRC_ALPHA, ONE_MINUS_SRC_ALPHA, DST_ALPHA, ONE_MINUS_DST_ALPHA,
SRC_ALPHA_SATURATE - default SRC_ALPHA
-dstblendfunc -dbf     : ZERO, ONE, SRC_COLOR, ONE_MINUS_SRC_COLOR,
SRC_ALPHA, ONE_MINUS_SRC_ALPHA, DST_ALPHA, ONE_MINUS_DST_ALPHA, - default
ONE_MINUS_SRC_ALPHA
-linewidth -lw <width> : Linewidth for wire/vector tests - default 1.0
-xwinsize -xws <side>  : Size of test windows X dimension - default 700
-ywinsize -yws <side>  : Size of test windows Y dimension - default 700
-numframes -nf <num>   : Number of frames to be rendered during measurement
                         Takes priority over -mp
-numilights -nil <num> : Turns on <num> infinite lights - default 0
-numllights -nll <num> : Turns on <num> local lights - default 0
-colormaterial -cm <side> <mode> :
```

```
                        <side> is FRONT, BACK, FRONT_AND_BACK - default
FRONT
                        <mode> is AMBIENT, DIFFUSE, EMISSION, SPECULAR,
                        AMBIENT_AND_DIFFUSE - default AMBIENT_AND_DIFFUSE
-backface -bf          : Cull Backfacing primitives - default off
-frontface -ff         : Cull Frontfacing primitives - default off
-singlebuffer -sb      : Single buffer mode
-fog -fg               : Enable fog
-linesmooth -ls        : Enable line antialiasing
-polysmooth -ps        : Enable polygon antialiasing
-facetnormal -fn       : Use facet normals when lighting
-linestipple -lp       : Enable line stipple
-polystipple -pp       : Enable polygon stipple
-toggle -tg <cap>      : Toggle per primitive - BLEND, DEPTH_TEST, DITHER,
                       LIGHTING, LINE_WIDTH, LINE_STIPPLE, POLYGON_STIPPLE,
                        or MATRIX - multmatrix
-batch -bt <num>       : Batch <num> primitives together per glBegin/glEnd
                        Valid with POINT, VECTOR, TRIANGLE, and QUADS
-polymodefront -pmf    : POINT, LINE, or FILL - default FILL
-polymodeback  -pmb    : POINT, LINE, or FILL - default FILL
-flat -f               : Set shademodel to FLAT - default GOURAUD
-zbuffer -zb           : Enable zbuffer for tests - default off
-clip -c               : Align object on 3D clip boundary
-lighttwoside -l2s     : Light both sides of model
-localview -lv         : Define local viewer for lit tests
-minperiod -mp <num>   : Set minimum testing period in seconds
-mblur <num>           : Use motion blur with num being amount of decay
-aa_multi <x> <r>      : Full scene antialiasing rendered x times at an
                       : offset of r.  r should be tuned to the viewset
-walkthru -wt          : Walkthru mode
-threads -th <num>     : Sets number of threads (no arg means 1 per
processor)
```

Appendix B.2, “Viewperf Output from CDRS 03 Test 3” on page 257, shows the Viewperf output from CDRS-03 Test #3. The last line of the output provides the number of frames per second for that portion of the test. This result, along with the rest of the test’s results, is printed in the report page that is submitted to the OPC and is published in the GPC News. A single composite number is also reported that is derived using a weighted geometric mean methodology.



OPC OpenGL Viewperf Results http://www.spec.org/gpc/opc/IBM/summary/cdrs/IBM.RS.002.html

OPC CDRS-03 Report

IBM RS/6000 Model 43P-140 332Mhz
POWER GXT800PT
CDRS-03

Test #	Weight	Frames /sec	DLB Time	Visual/Pixel Format				Frame Buffer						Accum. Buffer			
	(%)		(sec.)	ID	Level	Double	Stereo	R	G	B	A	Depth Buffer	Stencil Buffer	R	G	B	A
1	50.0	54.0	0.0600	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
2	20.0	37.5	0.150	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
3	15.0	30.9	0.170	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
4	8.0	22.3	0.230	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
5	5.0	22.1	0.150	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
6	2.0	30.8	0.220	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0
7	0.0	55.9	0.0700	0X28	N	True	False	8	8	8	8	24	4	0	0	0	0

CDRS-03 Weighted Geometric Mean = 40.677

GRAPHICS HARDWARE CONFIGURATION

Graphics Accelerator	POWER GXT800PT
Total Graphics Memory	31 MB
Image Buffer	24+24 bits
Overlay/Underlay Buffer	8/0 bits
Depth Buffer	24 bits
Stencil Buffer	4 bits
Accumulation Buffer	Virtual Memory
Auxiliary Buffer	0 bits
Other Buffer	24 bits
Display List Memory	Virtual Memory
Texture Memory	10 MB
Display Manufacturer/Model	IBM/P70
Display Resolution	1280x1024
Display Size/Type	17-inch/Color
Display Refresh Rate	77 Hz
Swap on Vertical Retrace	No

SYSTEM HARDWARE CONFIGURATION

Processor Type	332MHz 604e PowerPC
Number of CPUs	1
Floating Point	Integral
Primary Cache (KB)	32/32 (D/I)
Secondary Cache (KB)	1024
Tertiary Cache (KB)	0
Memory (MB)	64
Disk (MB)	2100

Comments

May not be achievable with upgraded 332MHz system. System configured to swap buffers on demand (GS_BUFFER_SWAP_MODE=I).

SOFTWARE CONFIGURATION

Operating System	AIX 4.2.1, APAR IX69051
O/S Type	Unix
Compiler Name	IBM C Compiler
Compiler Version	3.1.3
Window System	X Window System V11
OpenGL Version	1.1.0
OpenGL Renderer	GXT800 Texture
OpenGL Vendor	IBM
Viewperf Version	5.1
Driver Version	Not Applicable

Figure 37. An OPC CDRS-03 Report

There is a very good paper explaining the reasoning for and method of calculating the weighted geometric mean. It is available for viewing or download at `http://www.spec.org/gpc/opc.static/geometric_mean.html`.

11.2.2.4 The GLperf Result

The OPC project group has approved a set of 13 GLperf scripts for reporting results within its Web publication, the GPC News. These are split into 10 RGB scripts and three color-index scripts. The scripts are further divided by functionality. The OPClist scripts (RGB and color index) contain a number of tests for a variety of graphics primitives and other operations (such as window-clears). These tests are probably the closest parallel to primitive-level results available from most vendors today. Other scripts feature specific graphics operations, such as CopyPixl.rgb, DrawPixl.rgb, ReadPixl.rgb, TexImage. rgb measure glCopyPixels, glDrawPixels, glReadPixels, and glTexImage2D RGB operations. DrawPixl.ndx and ReadPixl.ndx are the color index analogs to DrawPixl.rgb and ReadPixl.rgb.

Remaining scripts address underlying graphics concepts that affect OpenGL performance. BgnEnd.rgb measures performance as it varies with the number of primitives batched together (in a glBegin/glEnd pair). FillRate.rgb measures how fast rasterization operations are performed (how many pixels are drawn per second). Light.rgb measures the effect of the number of enabled light sources on drawing a particular primitive, and LineFill.rgb and TriFill.rgb measure the effect of increasing primitive size on the drawing rates of lines and triangles, respectively.

An example of an input file format is shown in Appendix , “The input file scripts look something like the following:” on page 260. For an in-depth explanation for the format, attributes, and the properties, please read the tutorial available at:: `http://www.spec.org/gpc/opc/glperf_publish/GLperf.htm`.

For a reasonable interpretation of what all these numbers mean, check out the Java-based tool at
`http://www.spec.org/gpc/opc/glperf_publish/index.htm`.

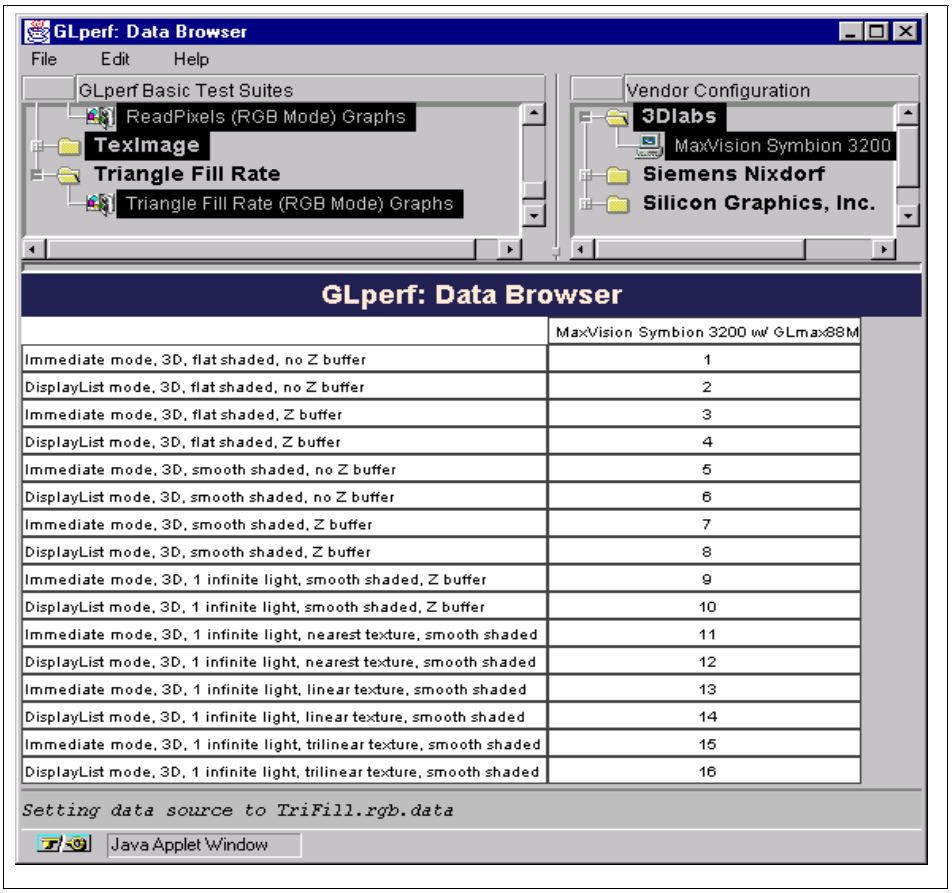


Figure 38. The GLperf Data Browser

Both Viewperf and GLperf measure the graphics performance of a system through the OpenGL API. They were designed, however, with different goals in mind. While Viewperf draws an entire model with differing sizes of primitives (as you would see in an actual application), GLperf artificially assigns a specific size to every primitive drawn within a test. While Viewperf attempts to emulate what an application would do graphically and measure it, GLperf makes no such attempt. Instead, GLperf provides a more controlled environment within which to extract and measure the highest performance or upper bound of a particular system.

Another difference is that Viewperf reports results in frames drawn per second, whereas GLperf measures its results in primitives drawn per second, whether the primitive is pixels, points, lines, triangles, or some other object.

11.3 Latest Results

After studying the various benchmarks available, here are the latest results available for the different combinations of RS/6000 systems and graphics adapters:

Table 18. Graphics Performance

Adapter	System	Clock Rate	Op. Sys.	L2 cache	GPC/XPC Xmark93
GXT120P	43P-140	233	AIX430	1 MB	12.77
GXT120P	43P-140	3323	AIX430	1 MB	14.13
GXT150M	595		AIX421	NA	
GXT150M	3CT		AIX421	NA	
GXT250P	43P-133	166	AIX421	512 KB	
GXT250P	43P-140	233	AIX421	1 MB	24.27
GXT250P	43P-240(1)	166	AIX421	512 KB	18.93
GXT250P	43P-240(1)	233	AIX421	1 MB	21.07
GXT250P	43P-240(2)	166	AIX421	512 KB	18.16
GXT250P	43P-240(2)	233	AIX421	1 MB	19.80
GXT250P	F40(1)	166	AIX421	512 KB	18.93
GXT250P	F40(1)	166	AIX421	1 MB	21.07
GXT250P	F40(2)	233	AIX421	512 KB	18.16
GXT250P	F40(2)	166	AIX421	1 MB	19.80
GXT255P	43P-140	233	AIX421	1 MB	21.27
GXT255P	43P-140	332	AIX431	1 MB	24.27
GXT255P	43P-150	---	AIX432	1 MB	29.21
GXT255P	43P-240(1)	166	AIX421	512 KB	19.55
GXT255P	43P-240(1)	233	AIX421	1 MB	19.73
GXT255P	43P-240(2)	166	AIX421	512 KB	18.96
GXT255P	43P-240(2)	233	AIX421	1 MB	18.93
GXT255P	43P-260(1)	---	AIX432	4 MB	25.79

Adapter	System	Clock Rate	Op. Sys.	L2 cache	GPC/XPC Xmark93
GXT255P	43P-260(2)	---	AIX432	4 MB	25.56
GXT255P	F40(1)	166	AIX421	512 KB	19.55
GXT255P	F40(1)	233	AIX421	1 MB	19.73
GXT255P	F40(2)	166	AIX421	512 KB	18.96
GXT255P	F40(2)	233	AIX421	1 MB	18.93
GXT255P	F50	166	AIX431	256 KB	24.62
GXT550P	43P-140	233	AIX421	1 MB	15.36
GXT550P	43P-140	332	AIX431	1 MB	16.73
GXT550P	43P-140	332	AIX432	1 MB	---
GXT550P	43P-150	---	AIX432	1 MB	18.10
GXT550P	43P-240(1)	166	AIX421	512 KB	12.17
GXT550P	43P-240(1)	233	AIX421	1 MB	13.41
GXT550P	43P-240(2)	166	AIX421	1 MB	11.96
GXT550P	43P-240(2)	233	AIX421	1 MB	113.00
GXT550P	F40(1)	166	AIX421	512 KB	12.17
GXT550P	F40(1)	233	AIX421	1 MB	13.41
GXT550P	F40(2)	166	AIX421	512 KB	11.96
GXT550P	F40(2)	233	AIX421	1 MB	13.00
GXT800M	397	160	AIX430	NA	---
GXT800P	43P-140	233	AIX421	1 MB	18.80
GXT800P	43P-240(1)	166	AIX421	512 KB	14.20
GXT800P	43P-240(1)	233	AIX421	1 MB	15.71
GXT800P	43P-240(2)	166	AIX421	512 KB	14.03
GXT800P	43P-240(2)	233	AIX421	1 MB	15.31
GXT800P	F40(1)	166	AIX421	512 KB	14.20
GXT800P	F40(1)	233	AIX421	1 MB	15.71

Adapter	System	Clock Rate	Op. Sys.	L2 cache	GPC/XPC Xmark93
GXT800P	F40(2)	166	AIX421	512 KB	14.03
GXT800P	F40(2)	233	AIX421	1 MB	15.31
GXT800PT	43P-140	233	AIX421	1 MB	18.39
GXT800PT	43P-240(1)	166	AIX421	512 KB	14.59
GXT800PT	43P-240(1)	233	AIX421	1 MB	16.35
GXT800PT	43P-240(2)	166	AIX421	512 KB	14.17
GXT800PT	43P-240(2)	233	AIX421	1 MB	15.49
GXT800PT	F40(1)	166	AIX421	512 KB	14.59
GXT800PT	F40(1)	233	AIX421	1 MB	16.35
GXT800PT	F40(2)	166	AIX421	512 KB	14.17
GXT800PT	F40(2)	233	AIX421	1 MB	15.49
GXT3000P	43P-150	---	AIX421	1 MB	37.17
GXT3000P	43P-260(1)	---	AIX432	4 MB	37.35
GXT3000P	43P-260(2)	---	AIX432	4 MB	37.09

GPC/PLB Results (PLBwire and PLBsurf)

Adapter	System	Clock Rate	Op. Sys.	L2 cache	PLBwire	PLBsurf
GXT250P	43P-140	233	AIX421	1 MB	62.90	38.50
GXT250P	43P-240(1)	166	AIX421	512 KB	41.60	29.50
GXT250P	43P-240(1)	233	AIX421	1 MB	50.80	35.50
GXT250P	43P-240(2)	166	AIX421	512 KB	56.90	41.40
GXT250P	43P-240(2)	233	AIX421	1 MB	62.10	46.90
GXT250P	F40(1)	166	AIX421	512 KB	41.60	29.50
GXT250P	F40(1)	166	AIX421	1 MB	50.80	35.50
GXT250P	F40(2)	233	AIX421	512 KB	56.90	41.40
GXT250P	F40(2)	166	AIX421	1 MB	32.10	46.90

Adapter	System	Clock Rate	Op. Sys.	L2 cache	PLBwire	PLBsurf
GXT255P	43P-140	233	AIX421	1 MB	127.10	45.90
GXT255P	43P-140	332	AIX431	1 MB	152.70	51.50
GXT255P	43P-150	---	AIX432	1 MB	178.60	75.40
GXT255P	43P-240(1)	166	AIX421	512 KB	83.80	35.60
GXT255P	43P-240(1)	233	AIX421	1 MB	96.20	40.00
GXT255P	43P-240(2)	166	AIX421	512 KB	97.60	49.00
GXT255P	43P-240(2)	233	AIX421	1 MB	100.40	52.20
GXT255P	43P-260(1)	---	AIX432	4 MB	158.60	78.10
GXT255P	43P-260(2)	---	AIX432	4 MB	213.30	106.00
GXT255P	F40(1)	166	AIX421	512 KB	83.80	35.60
GXT255P	F40(1)	233	AIX421	1 MB	96.20	40.00
GXT255P	F40(2)	166	AIX421	512 KB	97.60	49.00
GXT255P	F40(2)	233	AIX421	1 MB	100.40	52.20
GXT255P	F50	166	AIX431	256 KB	121.70	80.60
GXT550P	43P-140	233	AIX421	1 MB	134.30	133.90
GXT550P	43P-140	332	AIX431	1 MB	149.70	144.20
GXT550P	43P-140	332	AIX432	1 MB	149.90	144.00
GXT550P	43P-150	---	AIX432	1 MB	152.80	160.40
GXT550P	43P-240(1)	166	AIX421	512 KB	114.30	108.80
GXT550P	43P-240(1)	233	AIX421	1 MB	135.30	123.70
GXT550P	43P-240(2)	166	AIX421	1 MB	141.30	129.10
GXT550P	43P-240(2)	233	AIX421	1 MB	146.30	153.00
GXT550P	F40(1)	166	AIX421	512 KB	114.30	108.80
GXT550P	F40(1)	233	AIX421	1 MB	135.30	123.70
GXT550P	F40(2)	166	AIX421	512 KB	141.30	139.10
GXT550P	F40(2)	233	AIX421	1 MB	146.30	153.00

Adapter	System	Clock Rate	Op. Sys.	L2 cache	PLBwire	PLBsurf
GXT800M	397	160	AIX430	NA	158.70	257.80
GXT800P	43P-140	233	AIX421	1 MB	131.40	202.40
GXT800P	43P-240(1)	166	AIX421	512 KB	113.60	167.30
GXT800P	43P-240(1)	233	AIX421	1 MB	134.00	198.40
GXT800P	43P-240(2)	166	AIX421	512 KB	140.00	223.80
GXT800P	43P-240(2)	233	AIX421	1 MB	148.40	269.40
GXT800P	F40(1)	166	AIX421	512 KB	113.60	167.30
GXT800P	F40(1)	233	AIX421	1 MB	134.00	198.40
GXT800P	F40(2)	166	AIX421	512 KB	140.00	223.80
GXT800P	F40(2)	233	AIX421	1 MB	148.40	269.40
GXT800PT	43P-140	233	AIX421	1 MB	134.00	207.40
GXT800PT	43P-240(1)	166	AIX421	512 KB	114.50	168.70
GXT800PT	43P-240(1)	233	AIX421	1 MB	137.70	202.50
GXT800PT	43P-240(2)	166	AIX421	512 KB	140.80	228.90
GXT800PT	43P-240(2)	233	AIX421	1 MB	154.80	277.20
GXT800PT	F40(1)	166	AIX421	512 KB	114.50	168.70
GXT800PT	F40(1)	233	AIX421	1 MB	137.70	202.50
GXT800PT	F40(2)	166	AIX421	512 KB	140.80	228.90
GXT800PT	F40(2)	233	AIX421	1 MB	154.80	277.20
GXT3000P	43P-150	---	AIX421	1 MB	257.30	468.90
GXT3000P	43P-260(1)	---	AIX432	4 MB	436.90	610.80
GXT3000P	43P-260(2)	---	AIX432	4 MB	627.40	866.20

GPC/OPC Results (CDRS-03, DX-03 and DRV-04)

Adapter	System	Clock Rate	Op. Sys.	L2 cache	CDRS-03	DX-03	DRV-04
GXT150M	595		421	NA	7.85	1.54	0.91

Adapter	System	Clock Rate	Op. Sys.	L2 cache	CDRS-03	DX-03	DRV-04
GXT150M	3CT		421	NA	4.26	0.79	0.47
GXT250P	43P-133	166	421	512 KB	3.26	0.80	0.56
GXT250P	43P-140	233	421	1 MB	4.75	1.16	0.79
GXT250P	43P-240 (1)	166	421	512 KB	3.07	0.81	0.59
GXT250P	43P-240 (1)	233	421	1 MB	4.63	1.11	0.77
GXT250P	F40(1)	166	421	512 KB	3.07	0.81	0.59
GXT250P	F40(1)	166	421	1 MB	4.63	1.11	0.77
GXT255P	43P-133	166	421	512 KB	5.81	2.17	1.12
GXT255P	43P-140	233	421	1 MB	7.90	3.13	1.62
GXT255P	43P-140	332	431	1 MB	9.02	4.06	2.04
GXT255P	43P-150	---	432	1 MB	10.05	5.02	2.52
GXT255P	43P-240 (1)	166	421	512 KB	5.73	2.15	1.09
GXT255P	43P-240 (1)	233	421	1 MB	6.69	2.80	1.41
GXT255P	43P-260 (1)	---	432	4 MB	8.06	4.35	2.08
GXT255P	43P-260 (2)	---	432	4 MB	8.06	4.35	2.08
GXT255P	F40(1)	166	421	512 KB	5.73	2.15	1.09
GXT255P	F40(1)	233	421	1 MB	6.69	2.80	1.41
GXT255P	F50	166	431	256 KB	8.18	4.75	2.18
GXT550P	43P-140	233	421	1 MB	30.53	6.50	2.67
GXT550P	43P-140	332	431	1 MB	32.70	8.21	3.28
GXT550P	43P-140	332	432	1 MB	32.51	8.22	3.27
GXT550P	43P-150	---	432	1 MB	33.77	10.30	3.96

Adapter	System	Clock Rate	Op. Sys.	L2 cache	CDRS-03	DX-03	DRV-04
GXT550P	43P-240 (1)	166	421	512 KB	24.49	4.84	2.02
GXT550P	43P-240 (1)	233	421	1 MB	30.03	6.50	2.68
GXT550P	43P-240 (2)	233	421	1 MB	31.51	8.12	3.00
GXT550P	F40(1)	166	421	512 KB	24.49	4.84	2.02
GXT550P	F40(1)	233	421	1 MB	30.00	6.01	2.65
GXT550P	F40(2)	166	421	512 KB	30.39	6.93	2.44
GXT550P	F40(2)	233	421	1 MB	31.51	8.12	3.00
GXT800M	397	160	430	NA	41.67	7.70	3.79
GXT800P	43P-140	233	421	1 MB	33.47	6.47	2.84
GXT800P	43P-240 (1)	166	421	512 KB	29.03	5.02	2.17
GXT800P	43P-240 (1)	233	421	1 MB	33.92	6.52	2.89
GXT800P	43P-240 (2)	166	421	512 KB	34.09	6.91	2.47
GXT800P	43P-240 (2)	233	421	1 MB	35.75	8.29	3.02
GXT800P	F40(1)	166	421	512 KB	29.03	5.02	2.17
GXT800P	F40(1)	233	421	1 MB	33.92	6.52	2.89
GXT800P	F40(2)	166	421	512 KB	34.09	6.91	14.03
GXT800P	F40(2)	233	421	1 MB	35.75	8.29	3.02
GXT800PT	43P-140	233	421	1 MB	38.51	6.43	3.27
GXT800PT	43P-240 (1)	166	421	512 KB	33.76	4.96	2.56
GXT800PT	43P-240 (1)	233	421	1 MB	39.16	6.49	3.36
GXT800PT	43P-240 (2)	166	421	512 KB	39.47	6.83	3.00

Adapter	System	Clock Rate	Op. Sys.	L2 cache	CDRS-03	DX-03	DRV-04
GXT800PT	43P-240 (2)	233	421	1 MB	41.54	8.25	3.68
GXT800PT	F40(1)	166	421	512 KB	33.76	4.96	2.56
GXT800PT	F40(1)	233	421	1 MB	39.16	6.49	3.36
GXT800PT	F40(2)	166	421	512 KB	39.47	6.83	3.00
GXT800PT	F40(2)	233	421	1 MB	41.54	8.25	3.68
GXT3000P	43P-150	---	421	1 MB	94.76	11.16	6.17
GXT3000P	43P-260 (1)	---	432	4 MB	218.17	16.37	7.32
GXT3000P	43P-260 (2)	---	432	4 MB	218.17	16.37	7.32

GPC/OPC Results (light-01 and Awads-01)

Adapter	System	Clock rate	Op sys	L2cache	light-01	Awadvs-01
GXT800M	397	160	430	NA	0.69	7.27
GXT800PT	43P-140	233	421	1 MB	0.66	7.46
GXT800PT	43P-240(1)	166	421	512 KB	0.48	5.40
GXT800PT	43P-240(1)	233	421	1 MB	0.65	7.34
GXT800PT	43P-240(2)	166	421	512 KB	0.63	7.43
GXT800PT	43P-240(2)	233	421	1 MB	0.73	9.69
GXT800PT	F40(1)	166	421	512 KB	0.48	5.40
GXT800PT	F40(1)	233	421	1 MB	0.64	7.34
GXT800PT	F40(2)	166	421	512 KB	0.63	7.43
GXT800PT	F40(2)	233	421	1 MB	0.73	9.69
GXT3000P	43P-150	---	421	1 MB	1.22	13.77

Appendix A. 3D Graphics API Additional Information

This appendix contains information that is too long or of a minor interest to be included in the main chapters discussing GL or OpenGL. You'll find here, for example, the illustration of complex program written with the GL 3.2 API, an example of output for the `xglinfo` command, a discussion about Easy-MP, and a detailed list of the OpenGL extensions supported on the graphics adapter.

A.1 GL 3.2 Sample Code

The following section is a light introduction to GL3.2 programming. It shows several easy programs and provides a step-by-step explanation for every line of code.

A.1.1 Sample Program 2 - Animation Using Double Buffering

The following example program demonstrates how to create an animated scene. When this example program executes, the "Hello, World!" text moves around in a circle. This is done by clearing and redrawing the text again and again, each time at a new location.

This action is complicated by image flicker, which occurs because the system draws each image quickly, but perceptibly; that is, you do not see each individual character being drawn, only an irregular flashing and flickering. The flashing can be mild to severe, depending on what else is happening in the system or on the screen at that time.

To avoid this flashing, double buffering is used. The frame buffer is partitioned into two pieces, front and back. The front buffer contains data for the pixels that are visible. The back buffer, which also contains pixel data, is invisible, but identical to the front buffer in other respects. To get smooth animation, never draw to the front buffer; instead, limit all drawing to the back buffer. When drawing is complete, the front and back buffers are swapped, and what was previously hidden is now visible.

The result is a smooth, flicker-free animation. The actual, step-by-step drawing process is not visible, only the final result. The following example shows how to create an animated scene by using double buffering:

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

void main( void )
```

```
{
   int i, ix, iy;

   prefsize( 200, 100 );
   winopen( "HI THERE" );
   doublebuffer();
   gconfig();

   for( i = 1; i < 1800; i++ ) {
       color( BLACK );
         clear();
         color( GREEN );
         ix = (int)( 40.0 * cos ( ((double) i) / 20.0 ) );
         iy = (int)( 40.0 * sin ( ((double) i) / 20.0 ) );
         cmov2( 50 + ix, 50 + iy );
         charstr( "Hello, World!" );
         swapbuffers();
     }
 }
```

The detail for each line is:

```
prefsize( 200, 100 );
winopen( "HI THERE" );
```

First, a window is created exactly as before:

```
doublebuffer();
gconfig();
```

Next, the system is told to convert this window into a double-buffered window. The program does this in two steps:

1. Alerts the system with the `doublebuffer()` subroutine.
2. Sets double buffering into operation with the `gconfig()` subroutine.

The process requires two steps because there are, in fact, a number of configurations into which a window can be placed.

```
for( i = 1; i < 1800; i++ ) {
     color( BLACK );
     clear();
     color( GREEN );
     ix = (int)( 40.0 * cos ( ((double) i) / 20.0 ) );
     iy = (int)( 40.0 * sin ( ((double) i) / 20.0 ) );
     cmov2( 50 + ix, 50 + iy );
     charstr( "Hello, World!" );
     swapbuffers();
```

```
}
```

Next, the program goes into a loop that is repeated 1800 times. Inside this loop, we clear the screen and draw the text as before.

The `sin()` and `cos()` subroutines are AIX system calls that return the sine and cosine of an angle. They are useful for drawing circular primitives.

The program uses the loop counter as an angle and moves the current character position accordingly.

Finally, when drawing is complete, the `swapbuffers()` subroutine exchanges the front and the back buffers.

After looping 1800 times, the program exits, and the window disappears.

Note: Not all adapters support double buffering.

A.1.2 Sample Program 3 - Event Loop

The next program demonstrates how to obtain input and illustrates the concept of an event loop.

The event loop is critical to writing applications for a windowing system. An event loop allows a program to respond to events occurring in the system that are beyond the control of the application program, such as when a user picks up a window and moves it, has a window obscured and then unobscured by other windows, or resizes a window.

At this point, return to the first program (Sample Program 1), change the sleep time to 50 seconds, then recompile and rerun the program. While it is running, pick up another window (for instance, the xclock), drop it on the HI THERE window, pick it up again, and remove it. Notice that the original "Hello, World!" display was destroyed. This occurred because the other window overwrote the pixel data in the HI THERE window, and the overwritten data was not saved (GL does not support backing store or save-under).

When the contents of a window are destroyed that way, the application itself must redraw the window. GL provides an event that indicates that a window may have to be redrawn. The application must test for this event and redraw the window whenever the event is received. The discussion after the following program explains how the testing and redrawing is done:

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

```
/* This subroutine draws stuff */
drawstuff( int xxx, int yyy )
{
   color( BLACK );
   clear();
   color( GREEN );
   cmov2i( xxx, yyy );
   charstr( "Hello, World!" );
   swapbuffers();
}

void main( void )
{
   int ox, oy;
   int ix = 150, iy = 200;
   int update = TRUE;

   /* Create and configure window */
   prefsize( 400, 400 );
   winopen( "HI THERE" );
   doublebuffer();
   gconfig();

   /* get window origin */
   getorigin( &ox, &oy );

   /* queue up input devices */
   qdevice( REDRAW ); /* window needs to be redrawn */
   qdevice( WINQUIT ); /* user selected "close" from window menu */
   qdevice( MOUSEX ); /* mouse x position, in pixels */
   qdevice( MOUSEY ); /* mouse y position, in pixels */
   qdevice( ESCKEY ); /* user pressed escape key */
   qdevice( RIGHTMOUSE );/* user pressed right mouse button */

   /* enter event loop */
   while( TRUE ) {
         long dev;
         short value;

         /* if there aren't any events, and data has changed, redraw */
         if( !qtest() & update ) {
           drawstuff( ix, iy );
           update = FALSE;
         }

         /* get the next event */
```

```
        dev = qread( &value );

        /* dispatch the next event */
        switch( dev ) {
          case MOUSEX:
                    ix = value - ox;  /* update x location */
                    update = TRUE;
                    break;
          case MOUSEY:
                    iy = value - oy; /* update y location */
                    update = TRUE;
                    break;
          case REDRAW: /* redraw it */
                    getorigin( &ox, &oy ); /* get window origin */
                    update = TRUE;
                    break;
          case ESCKEY: /* if user presses escape key, quit */
          case RIGHTMOUSE: /* if user presses right mouse button, quit */
          case WINQUIT: /* if "close" selected from window menu */
                    exit();
                    default:
          break;
       }
   }
}
```

When you run this program, you will find that the character string "Hello, World!" follows the cursor around. You can pick up the window, move it, obscure it, and uncover it, but it will always appear correctly. The following discussion examines the operation of this complicated program in detail.

The detail for each line is:

```
/* This subroutine draws stuff */
drawstuff( int xxx, int yyy )
{
color( BLACK );
clear();
color( GREEN );
cmov2i( xxx, yyy );
charstr( "Hello, World!" );
swapbuffers();
}
```

To make the program easier to read, the drawing section has been put into its own subroutine, `drawstuff`, which contains a set of GL subroutines. When you want the program to draw, call this subroutine.

```
/* Create and configure window */
prefsize( 400, 400 );
winopen( "HI THERE" );
doublebuffer();
gconfig();
```

The program begins as before. The first step opens a window.

```
/* get window origin */
getorigin( &ox, &oy );
```

Next, the program obtains the window origin with the `getorigin()` subroutine, which will be needed later.

```
/* queue up input devices */
qdevice( REDRAW ); /* window needs to be redrawn */
qdevice( WINQUIT ); /* user selected "close" from window menu */
qdevice( MOUSEX ); /* mouse x position, in pixels */
qdevice( MOUSEY ); /* mouse y position, in pixels */
qdevice( ESCKEY ); /* user pressed escape key */
qdevice( RIGHTMOUSE );/* user pressed right mouse button */
```

Then, a number of devices are queued up. These devices, the REDRAW device, the MOUSEX device, and so on, generate events and place them on a queue. The program reads events from the bottom of the queue and processes them according to what came in. The `qdevice()` subroutine itself does not generate or process events, but only initializes these devices and readies them for use.

```
/* enter event loop */
while( TRUE ) {
...
}
```

Next, the program enters the event loop.

```
switch( dev ) {
...
case ESCKEY: /* if user presses escape key, quit */
case RIGHTMOUSE: /* if user presses right mouse button, quit */
case WINQUIT: /* if "close" selected from window menu */
exit();
...
}
```

Although this looks like an infinite loop, if the user presses either the escape key or the right mouse button, or else chooses the Close option from the window menu, the program ends.

```
long dev;
short value;

/* if there aren't any events, and data has changed, redraw */
if( !qtest() & update ) {
drawstuff( ix, iy );
update = FALSE;
}

/* get the next event */
dev = qread( &value );
```

In the loop, the program tests to see if there are any events on the event queue. If the queue is empty and the picture needs to be redrawn, the program begins to draw at this time.

If the queue is not empty, then the program reads the next event and processes it. The `qread()` subroutine returns the device that generated the event and a value associated with that event.

```
/* dispatch the next event */
switch( dev ) {
...
}
```

Depending on the device, the switch statement branches to the correct code to handle the event.

```
case MOUSEX:
ix = value - ox;    /* update x location */
update = TRUE;
break;
case MOUSEY:
iy = value - oy;    /* update y location */
update = TRUE;
break;
```

If, for instance, the event is a mouse-motion event, the new x or y coordinate (or both) is recorded.

```
case ESCKEY: /* if user presses escape key, quit */
case RIGHTMOUSE: /* if user presses right mouse button, quit */
case WINQUIT: /* if "close" selected from window menu */
exit();
```

If the event is an Escape key press, then the program is ended.

```
case REDRAW: /* redraw it */
```

```
getorigin( &ox, &oy ); /* get window origin */
update = TRUE;
break;
```

If the user moves the window, a REDRAW event is generated. In this case, the window origin is obtained, so that the character string can be drawn in the right place later.

After a nonterminating event is processed, the program returns to the beginning of the event loop and processes the next event.

A.1.3 Begin-End Style Drawing

Begin-end style drawing subroutines draw primitive graphical figures. In these subroutines, all points, lines, and polygons are described in terms of vertices (sets of coordinates that identify points in space).

- A point is described by a single vertex.
- A line segment is described by two vertices indicating its end points.
- A polygon is described by a set of three or more vertices indicating its corners.

To draw a graphical figure, use a series of vertex subroutines surrounded by a pair of begin and end subroutines, which mark the beginning and end of the figure. For example, the code to draw a set of five points A, B, C, D and E takes the following form:

<beginning of point vertices>

<vertex A>

<vertex B>

<vertex C>

<vertex D>

<vertex E>

<end of point vertices>

To draw a polygon whose corners are the same five points, the code takes the form:

<beginning of polygon vertices>

<vertex A>

<vertex B>

<vertex C>

<vertex D>

<vertex E>

<end of polygon vertices>

A.1.3.1 Lines

This code draws a pair of lines connecting its opposite corners:

```
Int32 vert1[2] = { 100, 100 }; /* lower left corner */
Int32 vert2[2] = { 100, 500 }; /* upper left corner */
Int32 vert3[2] = { 500, 500 }; /* upper right corner */
Int32 vert4[2] = { 500, 100 }; /* lower right corner */

bgnline();
   v2i( vert1 );
   v2i( vert3 );
endline();
bgnline();
   v2i( vert2 );
   v2i( vert4 );
endline();
```

In this example, four long arrays are declared, vert1, vert2, vert3, and vert4. Values are assigned to all the elements of each array.

The next four lines of code draw a line from (100, 100) to (500, 500) — the lower-left corner to the upper-right corner. The `bgnline()` subroutine tells the system to prepare to draw a line using the following vertices. Then the `v2i()` subroutine takes an array of coordinates as its parameter and creates a vertex at those coordinates.

The first `v2i()` subroutine call after the `bgnline()` subroutine creates the first end point of the line segment. The second `v2i()` subroutine call after the bgnline() subroutine creates the end point of the line segment, and the system draws a line. The `endline()` subroutine call tells the system that it has all the vertices for the line. The next four lines of code draw a line from (100, 500) to (500, 100), the lower-right corner to the upper-left corner.

A.1.3.2 Polylines

If more than two points are listed between the `bgnline()` and `endline()` subroutines, each point is connected to the next by a line:

```
bgnline();
     v2i( vert1 );
     v2i( vert2 );
     v2i( vert3 );
```

```
    v2i( vert4 );
    v2i( vert1 );
endline();
```

Note: The first vertex, `v2i(vert1)`, is repeated to close the series of line segments.

A series of connected line segments is called a polyline. GL cannot draw polylines with more than 256 vertices. Other than the number of vertices, there are no restrictions on a polyline. The segments can cross each other, vertices can be reused, and if the vertices are defined in terms of three dimensions, you can place them anywhere within 3D space. In a 3D space, the vertices need not all lie in the same plane.

A.1.3.3 Closed Lines

In the previous subsection, the code draws a closed polyline — a line segment connecting the last point in the polyline to the first point in the polyline. Because this is a fairly common operation, there is a pair of subroutines to do it: the `bgnclosedline()` and `endclosedline()` subroutines.

This code generates the same figure as the previous code does.

```
bgnclosedline();
   v2i( vert1 );
   v2i( vert2 );
   v2i( vert3 );
   v2i( vert4 );
endclosedline();
```

A.1.3.4 Vertex Subroutine

The code in the above examples only one form of the vertex subroutine: a 2D version with 32-bit integer coordinates. GL contains 12 forms of vertex (v) subroutines. The coordinates can be short integers (16-bits), long integers (32-bits), single-precision floating-point values (32-bits), and double-precision floating-point values (64-bits).

For each of these types, there is a 2D version, a 3D version, and a version that expects vertices expressed in homogeneous coordinates.

The vertex subroutines are illustrated in the following table.

Table 19. Different Forms of the Vertex Subroutines

Elements type	2D vectors	3D vectors	4D vectors
16-bit integer	v2s	v3s	v4s

Elements type	2D vectors	3D vectors	4D vectors
32-bit integer	v2i	v3i	v4i
32-bit floating point	v2f	v 3f	v4f
64-bit floating point	v2d	v3d	v4d

All forms of the vertex subroutine begin with the letter v. The second character is 2, 3, or 4, indicating the number of dimensions, and the final character is s for short integer, i for long integer, f for single-precision floating-point, and d for double-precision floating-point. For example, the 2D syntaxes are as follows:

```
void v2s( Int16 vector[2] )
void v2i( Int32 vector[2] )
void v2f( Float32 vector[2] )
void v2d( Float64 vector[2] )
```

The following code illustrates the use of some of the different vertex subroutines. It draws exactly the same picture as the previous code does, but uses different versions of the vertex subroutine.

```
Int16 vert1[3] = { 200, 200, 0 };
Int32 vert2[2] = { 200, 400 };
Float32 vert3[2] = { 400.0, 400.0 };
Float64 vert4[3] = { 400.0, 200.0, 0.0 };

bgnline();
   v3s( vert1 );
   v2i( vert2 );
   v2f( vert3 );
   v3d( vert4 );
   v3s( vert1 );
endline
```

This code illustrates two things:

- Within one geometric figure (in this case, a polyline), you can mix different kinds of vertices together. In a typical application, all the vertices tend to have the same dimension and form.
- GL treats all geometric figures as 3D figures. 2D versions of the vertex subroutines are actually shorthand for an equivalent 3D subroutine with the z coordinate set to zero.

A.1.3.5 Points

To draw a set of unconnected points in GL, enter a set of vertices specified between the `bgnpoint()` and `endpoint()` subroutines. The system draws each vertex as a one-pixel point on the screen.

```
bgnpoint();
   v2i( vert_first ); /* draw first point */
   ...
   v2i( vert_last ); /* draw first point */
endpoint();
```

As for the line-drawing subroutines, you can have no more than 256 vertices between calls to the `bgnpoint()` and `endpoint()` subroutines.

Polypoints Subroutine

The points that are drawn by the `bgnpoint()` and `endpoint()` subroutines are precisely one pixel in size. This size cannot be changed. Although GL does not have any explicit `bgnpolymarker()` or `endpolymarker()` subroutines, there are several methods you can use to get polypoints that are larger than one pixel.

Font Subroutines — defrasterfont, font

If you want polypoints in the shape of raster patterns, use the font subroutines. That is, the set of rasters to use should be associated with letters of the alphabet with the `defrasterfont()` subroutine. This font is then made current with the font subroutine. The raster patterns, which do not have to look like letters, can be positioned and drawn with the `cmov()` and `charstr()` subroutines, respectively.

Nonraster Polymarker Primitives Using Display Lists

Nonraster polymarker primitives can be created with display lists. For instance, display list line drawings in the shape of boxes, stars, crosses, asterisks, and so forth, can be created by using the `makeobj()` subroutine, followed by the drawing, followed by a `closeobj()` subroutine. To draw one of these items, position with the `translate()` subroutine and draw it with the `callobj()` subroutine.

A.1.3.6 Other Primitives

Other aspects of begin-end style drawing include Points, Polygons, Point-Sampled Polygons, Polygonal Shading, and Triangular Meshes.

A.2 The OpenGL API

This section provides several additional information about the OpenGL API, such as the output of the `xglinfo` program (that provides information about the environment available to display your OpenGL program), a description of Easy MP, and a list of all the OpenGL extensions supported on the RS/6000 graphics adapters.

A.2.1 Output of xglinfo

```
================================ Display :0.0 ================================
name of display:        :0.0
version number:         11.0
vendor string:          International Business Machines
vendor release:         6100
max request size:       262140 bytes
motion buffer size:     0
bitmap:                 unit = 32, bit order = MSBFirst, padding = 32
image byte order:       MSBFirst
keycode range:          minimum 8, maximum 254
focus window:           0x100000a, revert to RevertToPointerRoot
default screen num:     0
number of screens:      1
pixmap formats:         4 total
                        depth  1, bits_per_pixel  1, scanline_pad 32
                        depth  8, bits_per_pixel  8, scanline_pad 32
                        depth 12, bits_per_pixel 16, scanline_pad 32
                        depth 24, bits_per_pixel 32, scanline_pad 32
Server Extensions:      15 total
                        XTestExtension1, SHAPE, MIT-SHM, XInputExtension,
                        XTEST, BIG-REQUESTS, SCREEN-SAVER, XC-MISC, RECORD,
                        aixCursorExtension, xColormapExtension,
                        xDirectAccessExtension, xAncillaryBufferExtension,
                        DOUBLE-BUFFER, GLX,
GLX Extension:          error base = 137, event base = 84, Version 1.2
GLX Library:            Vendor = IBM
                        Version = 1.2
                        Extensions = GLX_EXT_visual_info GLX_EXT_visual_rating
                          GLX_EXT_import_context
GLU Library:            Version = 1.2.3 AIX
                        Extensions = GLU_EXT_nurbs_tesselator
                          GLU_EXT_object_space_tess
================================== Screen 0 ==================================
screen:                 0
dimensions:             1280x1024 pixels (356x284 millimeters)
resolution:             91x92 dots per inch
depths:                 (4 total):       8, 8, 12, 24
root window id:         0x32
depth of root window:   8 planes
default visual id:      0x21
number colormaps:       minimum 7, maximum 29
default colormap:       0x30, number colormap cells 256
preallocated pixels:    black 0x0, white 0x1
options:                backing-store NO, save-unders NO
input event mask:       0x30003c
                        ButtonPress          ButtonRelease        EnterWindow
                        LeaveWindow          SubstructureRedirect FocusChange

GLX Server:             Vendor = IBM
```

```
                        Version = 1.2
                        Extensions = GLX_EXT_visual_info GLX_EXT_visual_rating
                          GLX_EXT_import_context
                        Useable Extensions = GLX_EXT_visual_info
                          GLX_EXT_visual_rating GLX_EXT_import_context
GL Extension:           Vendor = IBM
                        Renderer = GXT3000
                        Version = 1.2.0
                        Extensions = GL_EXT_texture_object GL_EXT_vertex_array
                          GL_EXT_rescale_normal GL_IBM_rasterpos_clip
                          GL_IBM_cull_vertex GL_EXT_multi_draw_arrays
                          GL_EXT_abgr GL_EXT_bgra GL_EXT_blend_color
                          GL_EXT_blend_logic_op GL_EXT_polygon_offset
                          GL_EXT_subtexture GL_EXT_texture3D
                          GL_EXT_texture_edge_clamp GL_EXT_texture_lod
                          GL_IBM_texture_mirrored_repeat
================== 15 Visuals for Screen 0 (default = 0x21) ==================
PseudoColor visual: ID = 0x21 (hex) 33 (decimal) screen = 0
  OVERLAY(1) SINGLE buffered MONO COLOR INDEX visual   GL Sizes: ColorIndex=8,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=8, colormapSize=256

PseudoColor visual: ID = 0x22 (hex) 34 (decimal) screen = 0
  OVERLAY(1) SINGLE buffered MONO COLOR INDEX visual   GL Sizes: ColorIndex=8,
  Extensions: visualCaveat=None, TRANSPARENT INDEX, index value=255
  Core X: depth=8, colormapSize=255

PseudoColor visual: ID = 0x23 (hex) 35 (decimal) screen = 0
  DOUBLE buffered MONO COLOR INDEX visual with (Z Stencil)
  GL Sizes: ColorIndex=8, Z=24, Stencil=8
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=8, colormapSize=256

PseudoColor visual: ID = 0x24 (hex) 36 (decimal) screen = 0
  DOUBLE buffered MONO COLOR INDEX visual with (Z)
  GL Sizes: ColorIndex=8, Z=24,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=8, colormapSize=256

TrueColor visual: ID = 0x25 (hex) 37 (decimal) screen = 0
  DOUBLE buffered STEREO RGB visual with (Alpha Z Stencil Accum)
  GL Sizes: RGBA=(4,4,4,4), Z=24, Stencil=8, Accum=(16,16,16,16)
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=12, colormapSize=16 RGB: masks=(0xf00,0xf0,0xf) bits=4

TrueColor visual: ID = 0x26 (hex) 38 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Z)
  GL Sizes: RGBA=(8,8,8,0), Z=24,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

DirectColor visual: ID = 0x27 (hex) 39 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Z)
  GL Sizes: RGBA=(8,8,8,0), Z=24,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

TrueColor visual: ID = 0x28 (hex) 40 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Z Stencil)
  GL Sizes: RGBA=(8,8,8,0), Z=24, Stencil=8,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8
```

```
DirectColor visual: ID = 0x29 (hex) 41 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Z Stencil)
  GL Sizes: RGBA=(8,8,8,0), Z=24, Stencil=8,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

TrueColor visual: ID = 0x2a (hex) 42 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z)
  GL Sizes: RGBA=(8,8,8,8), Z=24,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

DirectColor visual: ID = 0x2b (hex) 43 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z)
  GL Sizes: RGBA=(8,8,8,8), Z=24,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

TrueColor visual: ID = 0x2c (hex) 44 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z Stencil)
  GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

DirectColor visual: ID = 0x2d (hex) 45 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z Stencil)
  GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8,
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

TrueColor visual: ID = 0x2e (hex) 46 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z Stencil Accum)
  GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8, Accum=(16,16,16,16)
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8

DirectColor visual: ID = 0x2f (hex) 47 (decimal) screen = 0
  DOUBLE buffered MONO RGB visual with (Alpha Z Stencil Accum)
  GL Sizes: RGBA=(8,8,8,8), Z=24, Stencil=8, Accum=(16,16,16,16)
  Extensions: visualCaveat=None, OPAQUE
  Core X: depth=24, colormapSize=256 RGB: masks=(0xff0000,0xff00,0xff) bits=8
```

A.2.2 Using Easy MP

This section discusses the use of Easy MP.

To successfully use Easy MP, you must meet the following requirements (as of November 1998):

- Your workstation must have multiple processors.
- Your workstation must contain one of the following adapters:

  GXT500P, GXT550P, GXT800P (with or without texture option)

  and you must run your OpenGL application on that adapter.

  Note that Easy MP is not supported on the GXT3000P.
- your application must request a "Direct" context at the time you call `glXCreateContext()`.

A.2.2.1 Preparation

No coding changes in the application are needed. Effective with the release of AIX Version 4.3.2, it is no longer necessary to relink your applications to use Easy MP, but with the previous release of AIX, your application must be recompiled using the thread-enabling compiler whose name is cc_r (instead of cc).

The application automatically selects the uni-processor version of OpenGL if your system has only one processor, you are running on an adapter that Easy MP does not support, or a direct context is not available for your application (for example, if you are using accumulation buffers, you are not permitted to run direct on current MP-capable adapters).

Note, that the target machine must NOT be down level to the relinking machine, because AIX does not guarantee backward binary compatibility. As long as your application is intended for use on AIX Version 4.1.5, 4.2.1, or 4.3.0, you can relink your application for use by Easy MP according to the following rules:

Table 20. Relinking Rules for Easy MP

Target Machine	Re Linking Machine	Compiler to Use
AIX Version 4.1.5	AIX Version 4.1.5	cc_r
4.2.1	4.1.5	cc_r
4.2.1	4.2.1	cc_r
4.3.2	4.1.5	cc_r
4.3.2	4.2.1	cc_r
4.3.2	4.3.2	cc

A.2.2.2 Use of Easy MP

You can select whether you would rather have it run UP (run in uniprocessor mode, using the traditional OpenGL libraries). The default is that you get the UP libraries. To actually run the Easy MP code, you must set one environment variable, as follows:

For ksh: `export _OGL_MP_ENABLE=1`

For csh: `setenv _OGL_MP_ENABLE 1`

Once this is done, you can start up your application and enjoy the performance advantages that Easy MP has to offer. If you are unsure of whether your application is successfully loading our Easy MP code, a

different environment variable can be used to show one line message in command line console at the time the application is coming up:

for ksh: `export _OGL_OUTPUT_LIB_INFO=1`

for csh: `setenv _OGL_OUTPUT_LIB_INFO 1`

The message will tell you whether you are running "MP" or "UP".

A.2.2.3 Caveats

Many factors go into the performance results that your specific application will experience:

1. Some applications already feed the hardware as fast as it can run. These hardware-bound applications will not see any performance improvements using Easy MP. In fact, under certain circumstances, users may actually see performance degradations.
2. Some applications don't really spend a lot of time in the EasyMP libraries. If your application spends 20 percent of its time in the libraries, and 80 percent elsewhere (in the application layer, in the X Server, in the kernel), the time required for some long rendering step might shrink from 10 seconds to 9 seconds (as a theoretical best case).
3. Applications which spend a large percentage of their cycles tieing up the memory bus are going to find that Easy MP does not help them significantly. This includes applications that do lots of BLTs (`glDrawPixels()`, `glBitmap()`) or applications that move large quantities of data from one place to another.
4. We have seen the best improvements in those applications which make heavy use of large display lists. This is understandable, given the architecture of Easy MP.

A.2.3 OpenGL Extensions Supported on AIX

The following table lists the OpenGL extensions currently supported on AIX. The "OpenGL Version" entry indicates what version of OpenGL incorporated the feature. The entries "1.2" and "1.2 Imaging" refer to extensions that are

incorporated in OpenGL 1.2 and the official OpenGL Imaging Subset, respectively.

Table 21. OpenGL Extensions

Extension Name	OpenGL Version	Description
GL_EXT_abgr	-	Accepts pixels in an ABGR format compatible with SGI GL.
GL_EXT_bgra	1.2	Accepts pixels in an BGR and BGRA order format compatible with typical Windows component ordering. Is usually used with the EXT_packed_pixels extension.
GL_EXT_blend_color	1.2	Extends the blending equation by defining a constant color that can be include in the blending equations.
GL_EXT_blend_logic_op	1.1	Performs bitwise operations between RGB colors of incoming pixels and those in the frame buffer. Useful for rubberbanding in RGB mode. The 1.1 Version has a different interface than the extension.
GL_EXT_blend_minmax	1.2 Imaging	Performs min/max operations between the color components of an incoming pixel and the stored pixel, leaving the minimum or maximum in the frame buffer.
GL_EXT_blend_subtract	1.2 Imaging	When blending a multiplied source and destination color, subtract one from the other instead of adding.
GL_EXT_compiled_vertex_array	-	Allows the results of transformation and lighting of static vertex array data to be compiled and reused for glDrawElements calls.
GL_EXT_multi_draw_arrays	-	Accommodates the bundling of multiple vertex array calls in a single OpenGL call.
GL_EXT_packed_pixels	-	Provides for non byte-aligned packing of pixels in a format that may more closely matches that of the physical frame buffer.

Extension Name	OpenGL Version	Description
GL_EXT_polygon_offset	1.1	Displaces polygons in the Z buffer to avoid Z fighting artifacts and allows for edges to be drawn on polygons. Because of interpolation artifacts on the GXT500D 500P / 550P / 800P / 1000 rasterizer, you will have to set the factor and/or bias to values larger than strictly necessary. The 1.1 version has a different interface than the extension.
GL_EXT_rescale_normal	1.2	Rescales previously normalized normals to compensate for a uniform scale matrix. Produces correct lighting effects without having to compute normal length and a reciprocal square root.
GL_EXT_subtexture	1.1	Allows an application to define (replace) a subset of a texture image. Allows the rendering library to download only the defined subset of an image, when a NULL pointer is passed to `glTexImage()`. This can improve download time when using images that don't exactly match a power of two as texture maps.
GL_EXT_texture3D	1.2	Allows the application to do texture mapping using a 3D image. Hardware acceleration is not present in all products - only in the GXT3000P.
GL_EXT_texture_edge_clamp	1.2 Imaging	Defines a new texture wrap method that clamps texture coordinates and samples clamped coordinate texels only from the edge of the image - never sampling border texels.

Extension Name	OpenGL Version	Description
GL_EXT_texture_object	1.1	Allows an application to encapsulate a number of texture attributes (including images) into a named object. That named object can then be bound when rendering. For applications with multiple textures, using texture objects allows all textures to be cached and frees the application from having to respecify texture images each time they are used. Superior to putting TexImage calls into display lists (the officially-sanctioned 1.0 version); this was accelerated only on GXT1000
GL_EXT_vertex_array	1.1	Allows an application to pass pointers to application data and render using indices or ranges of vertices relative to the base pointers. Produces transformation performance comparable to display lists without having to keep an extra copy of the user's data around. The interface was extended (adding glDrawElements) in OpenGL 1.1.
GL_IBM_rasterpos_clip	-	Allows an application to specify a raster position without having it culled if it is out of bounds. Useful for geometry-aligned markers near the edge of the window where portions of the marker would be visible if the raster position were not culled.
GL_IBM_static_data	-	Allows implementations to defer dereferencing of vertex array data which is promised by the application to be static. Useful ONLY for EasyMP to avoid making copies of vertex data for use by other threads.

Extension Name	OpenGL Version	Description
GL_IBM_texture_mirrored_repeat	-	Extends the texture wrap modes to include a mode (GL_MIRRORED_REPEAT_IBM) that effectively uses a texture map twice as large at the original image in which the additional half of the new image is a mirror image of the original image.
GL_IBM_vertex_cull	-	Provides and alternate method of culling in which dot product of the normal and the eye direction determine the face direction of a polygon. Normally the face direction is determined by the geometric order of the vertices. Can be used to improve software culling performance.

A.2.4 Extensions Support

The following table documents under which release an extension was first supported and by which adapters.

Table 22. Support of OpenGL Extensions

Extension Name (first implemented)	VFB	250P 255P	500 500D	500P 550P	800P 800M	1000	3000P
GL_EXT_abgr (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_bgra (4.3.2)							Y
GL_EXT_blend_color (4.3.2)							Y
GL_EXT_blend_logic_op (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_blend_minmax (4.1.4)	Y	Y	Y	Y	Y	Y	
GL_EXT_blend_subtract (4.1.4)	Y	Y	Y	Y	Y	Y	
GL_EXT_compiled_vertex_array (4.3.0)	Y	Y	Y	Y	Y		

Extension Name (first implemented)	VFB	250P 255P	500 500D	500P 550P	800P 800M	1000	3000P
GL_EXT_multi_draw_arrays (4.3.2)	Y	Y	Y	Y (up only)	Y (up only)		Y
GL_EXT_packed_pixels (4.3.2)							Y
GL_EXT_polygon_offset (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_rescale_normal (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_subtexture (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_texture3D (4.2.0)			Y	Y	Y		Y
GL_EXT_texture_edge_clamp (4.3.2)							Y
GL_EXT_texture_object (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_EXT_vertex_array (4.1.4)	Y	Y	Y	Y	Y	Y	Y
GL_IBM_rasterpos_clip (4.2.1)	Y	Y	Y	Y	Y	Y	Y
GL_IBM_static_data (4.2.1 PTF)				Y (mp only)	Y (mp only)		
GL_IBM_texture_mirrored_repeat (4.3.2)							Y
GL_IBM_vertex_cull (4.3.0)	Y	Y	Y	Y	Y		Y

Appendix B. Benchmarks Files

This appendix includes several files that are too big to be included in the chapter dedicated to benchmarks. You will find here an example of an intermediate output for the GPC/PLB benchmark, a Viewperf output for the CDRS 03 test, and an input file for the GLperf benchmark.

B.1 A Sample BRF File

Here is a sample of what a BRF file will look like. It is made of several sections. The title for each of these sections is written in bold.

```
          Graphics Performance Characterization Committee
                Standard Graphics System Benchmarks

Date                    :   Mon Nov 18 16:54:31 1991


  Benchmark Information

Benchmark Title         :   Level 2: Picture Level Benchmark
Benchmark Authors       :   SimGraphics Engineering Corporation


  Implementation Information

Implementation Title    : Acme Computer Corporation Port of PLB
Implementation Version  :   V1.2 Rev A
Implementation Date     :   Mon Oct 14 16:05:56 PDT 1991
Implementation Authors  :   ACC Engineering

Graphics Library        :   YAGL
Graphics Library Version:   2.0


  System Configuration

 System Make and Model   :   Acme Model 111 with TurboGraphics
 Hardware Configuration  :   16MB RAM
                         :   8 color bit planes

 Operating System        :   Acme OS 2.5
 Windowing System        :   Louver Windows 2.5
 PLB Window Size         :   900 x 720 pixels (not including borders)
 Stopwatch Accuracy      :   10 ms


  Invocation Information

 plb brftest.vrb
 LITERAL TEST
```

```
    Beginning Of Test Loop 1
```

Graphics Lighting/Shading Summary Table

Average Times Called Per Frame		Average Ambient Lights Per Frame	Average Directional Lights Per Frame	Average Positional Lights Per Frame	Average Spot Lights Per Frame
1.0	LIGHT_STATE:.	1.0	1.0	1.0	1.0
2.0	INTERIOR_SHADING: GOURAUD:				

Graphics Primitives: Polygon Summary Table

Average Times Called Per Frame		Average Vertices Per Frame	Average Facets Per Frame	Average Edges Per Frame	Average contours Per Frame
3.0	POLYGON:.	14.0	3.0	14.0	3.0
3.0	POLYGON3:	14.0	3.0	14.0	3.0
2.0	FILL_AREA_SET:. . . .	21.0	2.0	21.0	5.0
2.0	FILL_AREA_SET3: . . .	22.0	2.0	22.0	5.0
2.0	TRIANGLE3:.	13.0	9.0	13.0	9.0
2.0	QUAD_MESH3:	25.0	13.0	25.0	13.0
1.0	INDEX_POLYGONS3:. . .	6.0	2.0	8.0	2.0

Graphics Primitives: Polygon Optional Data Summary Table

Average Times Called Per Frame		Average Vertex Colors Per Frame	Average Vertex Normals Per Frame	Average Facet Color Per Frame	Average Facet Normals Per Frame
3.0	POLYGON3:	4.0	0.0	0.0	0.0
2.0	FILL_AREA_SET3: . . .	13.0	9.0	0.0	0.0
2.0	TRIANGLE3:.	0.0	0.0	0.0	0.0
2.0	QUAD_MESH3:	0.0	0.0	0.0	0.0
1.0	INDEX_POLYGONS3:. . .	6.0	0.0	0.0	0.0

Graphics Primitives: Polygon Edge Visibility Table

Average Times Called Per Frame	Average Edges Visible Per Frame

```
   2.0  FILL_AREA_SET3: . . .    14.0
   1.0  INDEX_POLYGONS3:. . .     6.0
```

Graphics Primitives: Non Uniform Bspline Curves/Surfaces

```
Average                        Average      Average    Average
Times                           number      Rational  Non Rational
Called                  Control Points       Splines    Splines
Per Frame                    Per Frame     Per Frame  Per Frame

   2.0  NURBS_CURVE:. . . . .    11.0       1.0       1.0
   2.0  NURBS_SURFACE:. . . .    10.0       1.0       1.0
   2.0  TRIMMING_CURVE: . . .     6.0       0.0       2.0
```

Graphics Primitives: Non Uniform Bspline Curves/Surfaces

```
Average                        Average    Average    Average
Times                           Uorder     Vorder    TrimCurves
Called                           per        per        per
Per Frame                     Primitive  Primitive   Primitive

   2.0  NURBS_CURVE:. . . . .     2.5
   2.0  NURBS_SURFACE:. . . .     2.5       2.0       1.0
   2.0  TRIMMING_CURVE: . . .     2.0
```

Graphics Primitives: Line, Marker, and Text Summary Table

```
Average                        Average
Times                         Individual
Called                        Primitives
Per Frame                      Per Frame

   4.0  MARKER: . . . . . .    11.0  markers
   8.0  MARKER3:. . . . . .    25.0  markers
   5.0  LINE: . . . . . . .    15.0  vectors
  12.0  LINE3:. . . . . . .    22.0  vectors
   4.0  TEXT: . . . . . . .    37.0  characters
   8.0  TEXT3:. . . . . . .    82.0  characters
   5.0  ANNOTATION_TEXT3: . .  37.0  characters
```

Graphics Attributes: Summary Table

```
Average
Times
Called
Per Frame

   8.0  MARKER_TYPE:
```

```
   4.0  MARKER_SIZE:
   9.0  MARKER_COLOR:
   4.0  LINE_WIDTH:
  12.0  LINE_COLOR:
   2.0  INTERIOR_STYLE:
  10.0  INTERIOR_COLOR:
   1.0  SURFACE_PROPERTIES:
   3.0  EDGE_TYPE:
   2.0  EDGE_COLOR:
   1.0  TEXT_COLOR:
   3.0  CHAR_HEIGHT:
   1.0  CHAR_EXP:
   1.0  CHAR_SPACE:
  11.0  LINE_TYPE:
   1.0  INTERIOR_LIGHTING:
   2.0  BACKFACE_PROCESSING:
   8.0  EDGE_FLAG:
   1.0  TEXT_PREC:
```

Matrix Operations: Summary Table

```
Average
Times
Called
Per Frame

   1.0  ACTIVE_VIEW:
```

Structure Calls: Summary Table

```
Average
Times
Called
Per Frame

   1.0  EXECUTE_STRUCTURE:
   1.0  CALL_STRUCTURE:
   1.0  INVOKE_AT_FRAME TOTAL:
   1.0  INVOKE_AT_FRAME CALL:
   0.0  INVOKE_AT_FRAME EXECUTE:
```

Global Exception: Summary Table

```
This test cannot be double buffered on this hardware.
Dot and Phong shading not supported; Gouraud substituted.
GEN_SPHERE3 not currently supported by this PLB version.
```

Port Notes

```
Note: True color approximated on this system by a 5x9x5 color cube.
```

Conversion Time Information

```
Conversion Time (total) :    21.61
```

Test Loop Timing Information

Test Loop of 2 frames from File brftest.vrb

```
Number of Frames        :   2
Elapsed Time (sec)      :     4.03 (literal test)
Transport Delay         :     0.02
Avg. Frames per Second  :     0.50
Avg. Time per Frame     :     2.02
Timing Merit Mthd 1     :    94.47%
Timing Merit Mthd 2     :    21.16%
```

B.2 Viewperf Output from CDRS 03 Test 3

Here is this example of output from the Viewperf benchmark. The long first part is a description of the environment in which the benchmark took place. The very interesting part is the very end of this file where the real results are located, with the information Number of Frames.

```
Viewperf Version                                    5.0
    Viewperf Arguments                                -mh mower-ts  -dl  -rm
TMESH  -vz  -bl  -zb  -nil 1  -cp PRIMITIVE  -mp 10
    Month                                              12
    Day                                                12
    Year                                               1996
    Host                                               mayhem
    Operating System                                   AIX
    Operating System Version                           4.1
    Host Vendor                                        IBM
    Host Model                                        unknown (CPU_id=0X4C)
    Host CPU                                           PowerPC 604
    Host CPU Count                                     1
    Host Memory Size (MB)                              256
    Host Primary Cache Size (KB)                       32/32 (D/I)
    Host Secondary Cache Size (KB)                     1024
```

```
    Window System                                   X Window System V11
    Driver Version                                  NA
    OpenGL Vendor                                   IBM
    OpenGL Version                                  1.0.0
    OpenGL Extensions                              GL_EXT_texture_object
GL_EXT_vertex_array GL_EXT_rescale_normal GL_EXT_abgr
GL_EXT_blend_logic_op GL_EXT_blend_minmax GL_EXT_blend_subtract
GL_EXT_polygon_offset GL_EXT_subtexture GL_EXT_texture3D
    OpenGL Renderer                                 GXT800 Texture
    OpenGL Client Vendor                            IBM
    OpenGL Client Version                           1.1
    OpenGL Client Extensions                        GLX_EXT_visual_info
GLX_EXT_visual_rating GLX_EXT_import_context
    GLU Version                                     1.2.1 AIX
    GLU Extensions
    Direct Rendering                                False
    Double Buffer                                   True
    Stereo                                          False
    RGBA                                            True
    Color Index Size                                 1
    Red Size                                        8
    Green Size                                      8
    Blue Size                                       8
    Alpha Size                                      8
    Accum Red Size                                  0
    Accum Green Size                                0
    Accum Blue Size                                 0
    Accum Alpha Size                                0
    Depth Size                                      24
    Stencil Size                                    4
    Auxiliary Buffer Count                          0
    Frame BufferLevel                               0
    Visual ID                                       0X28
    Visual Class                                    TrueColor
    Window Width (pixels)                           700
    Window Height (pixels)                          700
    Screen Width (pixels)                           1280
    Screen Height (pixels)                          1024
    Display                                         :0.0
    OpenGL Server Vendor                            IBM
    OpenGL Server Version                           1.1
    OpenGL Server Extensions                        GLX_EXT_visual_info
GLX_EXT_visual_rating GLX_EXT_import_context
    GLX Server Version                              1.1
    GLX Server Extensions                           GLX_EXT_visual_info
GLX_EXT_visual_rating GLX_EXT_import_context
    Screen Number                                   0
```

Shared Memory Connection	True
Visual Selection Criteria	MINIMUM
Number of Execution Threads	1
Geometry File	./mower-ts
Input Mode	-mh
Minimum Test Period	10.000000
Number of Frames	0
Number of Primitives	71
Number of Vertices per Frame	31380
Number of Vertices per Primitive	441.971831
Toggle Mode	NONE
Batching Count	0
Render Mode	TMESH
Color per	COLOR_PER_PRIMITIVE
Orthographic Projection	FALSE
Display List	TRUE
Clip Geometry	FALSE
Walkthrough Mode	FALSE
Back Face Cull	FALSE
Front Face Cull	FALSE
Front Polygon Mode	FILL
Back Polygon Mode	FILL
Polygon Stipple Enable	FALSE
Polygon Antialiasing Enable	FALSE
Line Width	1.000000
Line Stipple Enable	FALSE
Line Antialiasing Enable	FALSE
Number of Infinite Lights	1
Number of Local Lights	0
Color Material Enable	TRUE
Color Material Face	FRONT
Color Material Mode	AMBIENT_AND_DIFFUSE
Facet Normals	FALSE
Two Sided Lighting Enable	FALSE
Local Viewer Enable	FALSE
Flat Shading	FALSE
Fog Enable	FALSE
Texture Enable	FALSE
Texture Generation Mode	NO_TEXTURE_GENERATION
Texture File	NONE
Texture Minification Filter	NEAREST
Texture Magnification Filter	NEAREST
Texture Environment Mode	DECAL
Texture Components	3
Depth Test Enable	TRUE
Blend Enable	TRUE
Source Blend Function	SRC_ALPHA

```
   Destination Blend Function                          ONE_MINUS_SRC_ALPHA
   Dithering Enable                                    TRUE
   Motion Blur Amount                                  0.000000
   Full Scene Antialiasing Redraws                     0
   Full Scene Antialiasing Jitter Amount               0.000000
  0.140       sec (DL Build)          -mh mower-ts -rm TMESH -cp PRIMITIVE
-dl -zb -nil 1 -bl -vz

               Number of frames run: 174, Test period: 9.980000 (sec)
   17.4       frames/sec              -mh mower-ts -rm TMESH -cp PRIMITIVE
-dl -zb -nil 1 -bl -vz
```

B.3 Input File for GLPerf

The file format to run the GLperf benchmark can be described as follows:

```
Suite  > GlobalProperty Suite | TestDescription Suite |TestDescription
TestDescription  > TestName | TestName { LocalPropertyList }
LocalPropertyList  > Property | Property LocalPropertyList
GlobalProperty  > Property
Property  > ( PropertyName AttributeValue )
AttributeValue  > Range | List | Wildcard
Range  > from int to int | from int to int step int | from int to int
step int % | from float to float | from float to float step float |
from float to
float step float %
List  > Value | Value List
Value  > Enumerated | float | int | 0xhex
Wildcard  > ALL | *
```

The input file scripts look something like the following:

```
//
// OPC GLperf ReadPixl.rgb script (version 0.3)
//

(FileName "ReadPixl.rgb 0.3")

// This script will generate data for the following 2 bar charts:
//  * ReadPixels (RGB, ubyte)
//  * ReadPixels (RGBA, ubyte)
//
```

```
// All tests are run in Immediate Mode.
// In each case above, the width and height of the ReadPixels
// image (ImageWidth, ImageHeight) will be incremented from
// 16 to 512, stepping by powers of 2. (16, 32, 64, 128, 256, 512)

// Put all your "discretionary" global property definitions here.
// These are the following properties:
//
/*
AccumAlphaSize
AccumBlueSize
AccumGreenSize
AccumRedSize
AlphaSize
AuxBuffers
BlueSize
ClearColor
DataAlignment
DepthSize
DirectRender
DoubleBuffer
DrawBuffer
GreenSize
IndexSize
LoopFuncPtrs
LoopUnroll
RedSize
StencilSize
Stereo
VisualClass
VisualId
*/
//
// For example:
// (LoopUnroll 8)
// (LoopFuncPtrs True)

//
// These are all the "default" values for this script that will
// remain constant unless otherwise specified in a particular test.

(AcceptObjs             1.0)
(AlphaBias             0.0)
(AlphaRef             0.0)
(AlphaScale             1.0)
(AlphaTest             Off)
(Antialias             Off)
```

```
(Aspect                    1.0)
(AtoAMapSize              1)
(Blend                     Off)
(BlueBias               0.0)
(BlueScale               1.0)
(BtoBMapSize              1)
(CharFont               f9x15)
(CharsPerString             16)
(ClearAccumBuffer        False)
(ClearColor               White)
(ClearColorBuffer        True)
(ClearDepthBuffer        False)
(ClearIndex               0.0)
(ClearStencilBuffer       False)
(ClipAmount               .5)
(ClipMode               Random)
(ClipObjs               0.0)
(ColorData               None)
(ColorDim               3)
(ColorMask               TTTT)
(ColorMaterialMode        GL_AMBIENT_AND_DIFFUSE)
(ColorMaterialSide        GL_FRONT)
(CopyPixelsType             GL_COLOR)
(CullFace               Off)
(DataAlignment              0)
(DepthBias               0.0)
(DepthMask               On)
(DepthOrder               BackToFront)
(DepthScale               1.0)
(DepthTest               Off)
(DirectRender               True)
(Dither                    On)
(DrawBuffer               GL_FRONT)
(DrawOrder               Spaced)
(DrawableType               WindowDraw)
(DstBlendFunc               GL_ONE)
(ExecuteMode              Immediate)
(FacingBack               0.0)
(FacingFront               1.0)
(Fog                      Off)
(GreenBias               0.0)
(GreenScale               1.0)
(GtoGMapSize              1)
(ImageAlignment              4)
(ImageFormat              GL_RGBA)
(ImageLSBFirst              False)
(ImageSwapBytes             False)
```

```
(ImageType              GL_UNSIGNED_BYTE)
(IndexMask              0xfff)
(IndexOffset              0)
(IndexShift              0)
(InfiniteLights                0)
(ItoAMapSize              1)
(ItoBMapSize              1)
(ItoGMapSize              1)
(ItoIMapSize              1)
(ItoRMapSize              1)
(LineStipple              Off)
(LineWidth              1.0)
(LocalLights              0)
(LocalViewer              Off)
(LogicOp             Off)
(LoopFuncPtrs                False)
(LoopUnroll              1)
(MapColor             Off)
(MapStencil              Off)
(MinimumTime              5)
(NormalData              None)
(Objs                    1)
(ObjsPerBeginEnd       1)
(Orientation               Random)
(PixelZoomX              1.0)
(PixelZoomY              1.0)
(PointDraw             Off)
(PolygonModeBack       GL_FILL)
(PolygonModeFront       GL_FILL)
(PolygonSides              4)
(PolygonStipple                Off)
(PrintModeDelta                Off)
(PrintModeMicrosec       Off)
(PrintModePixels       On)
(PrintModeStateDelta          Off)
(Projection              Perspective)
(RasterPosDim              2)
(ReadBuffer              GL_FRONT)
(ReadOrder              Spaced)
(RedBias             0.0)
(RedScale             1.0)
(RejectObjs              0.0)
(RenderMode              GL_RENDER)
(Reps                    1)
(Rgba                    True)
(RtoRMapSize             1)
(Scissor             Off)
```

```
(ShadeModel             GL_SMOOTH)
(Shininess             10.0)
(Size                    10.0)
(SpecularComponent      On)
(SrcBlendFunc            GL_ONE)
(StencilTest            Off)
(StoSMapSize            1)
(TexBorder             0)
(TexComps             3)
(TexData             None)
(TexFunc             GL_DECAL)
(TexGen                    Off)
(TexHeight             64)
(TexImageBorder             0)
(TexImageComps             3)
(TexImageLevel             0)
(TexImageMipmap             None)
(TexImageSrc            SystemMemory)
(TexImageTarget            GL_TEXTURE_2D)
(TexLOD                   0.0)
(TexMagFilter            GL_NEAREST)
(TexMinFilter            GL_NEAREST)
(TexTarget             Off)
(TexWidth             64)
(TexWrapS             GL_REPEAT)
(TexWrapT             GL_REPEAT)
(TransformType             Rotate)
(TwistsPerStrip             0)
(TwoSided             Off)
(VertexDim             3)
(WindowHeight             600)
(WindowWidth            600)

//
// Pixel Read Tests
//

ReadPixelsTest {
    (UserString printf("ReadPixels (Immediate, RGB, ubyte, %dx%d)",
ImageWidth, ImageHeight))
    (ImageFormat GL_RGB)
    (ImageType GL_UNSIGNED_BYTE)
    ([ImageWidth ImageHeight] from 16 to 512 step 100%)
}

ReadPixelsTest {
    (UserString printf("ReadPixels (Immediate, RGBA, ubyte, %dx%d)",
```

```
ImageWidth, ImageHeight))
    (ImageFormat GL_RGBA)
    (ImageType GL_UNSIGNED_BYTE)
    ([ImageWidth ImageHeight] from 16 to 512 step 100%)
}
```

Appendix C. Special Notices

This publication is intended to help RISC SYSTEM/6000 sales people to understand the variety of solutions provided for the IBM platform and for support people to quickly locate information needed to solve problems regarding the configuration of graphics adapters or their functions. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM or the OpenGL Architecture Review Board. See the PUBLICATIONS section of the IBM Programming Announcement for graPHIGS, PEX, GL, and OpenGL for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been

reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

400
AIX
AIXwindows
AT
GXT150L
GXT150M
GXT1000
IBM®
Micro Channel
OS/2
PC/XT
POWER Gt1
POWER Gt3
POWER Gt3i
POWER Gt4
POWER Gt4e
POWER Gt4x
POWER Gt4xi
POWER GTO
PowerPC 604
RS/6000
SP
XT

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 273.

- *AIX Version 4 Desktop Handbook,* GG24-4451
- AIX Version 4.2 Differences Guide, SG24-4807

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

D.3 Other Publications

- *IBM 6094-020 LPFK Options Instructions,* GA23-2403
- *IBM 6094-010 Dials Option Instructions,* GA23-2404
- *7250 POWER GXT1000 Graphics Accelerator User's Guide*, SA23-2070
- *PCI Adapter Placement Reference,* SA38-0538
- *Introducing the graPHIGS Programming Interface*, SC33-8190

- *The graPHIGS API: Understanding Concepts*, SC33-8191
- *The graPHIGS API: Technical Reference,* SC33-8193
- *The graPHIGS API: Subroutine Reference,* SC33-8194
- *Personal graPHIGS Customization and Problem Diagnosis,* SC33-8130
- *The graPHIGS API: Messages and Codes,* SC33-8196
- *The graPHIGS API: Getting Started V2R2.2,* SC33-8198
- *The graPHIGS API: Quick Reference,* SC33-8195
- *ISO PHIGS Subroutine Reference*, SC33-8140
- *ISO PHIGS Quick Reference*, SC28-2705
- *GL 3.2 Version 4 for AIX: Programming Concepts,* SC23-2612
- *GL 3.2 Version 4 for AIX: Graphics Library (GL) Technical Reference*, SC23-2630
- *The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor, Release 2,* ISBN 0-201-62495-8
- *The Inventor Toolmaker: Extending Open Inventor, Release 2,* ISBN 0-201-62493-1
- *Open Inventor C++ Reference Manual: The Official Reference Document for Open Inventor, Release 2*, ASIN 0-201-62491-5
- *OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.1,* ISBN 0-201-46138-2
- *OpenGL Reference Manual: The Official Reference Document for OpenGL Version 1.1*, ISBN 0-201-46140-4
- *OpenGL Programming for the X Window System,* ISBN 0-201-48359-9
- *PEXlib Programming Manual: 3D Programming in X. Gaskins, T.,* ISBN 1-56592-028-7, 1992
- *PEXlib Reference Manual: 3D Programming in X. Talbott, S.,* ISBN 1-56592-029-5, 1992

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at http://www.redbooks.ibm.com/ and for IBM employees at http://w3.itso.ibm.com/.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at http://inews.ibm.com/.

IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
ABX	Ancillary Buffer Extension
ACM	Association of Computing Machinery
AIX	Advanced Interactive eXecutive
ALU	Arithmetic and Logic Unit
ANSI	American National Standards Institute
APA	All Points Addressable
APAR	Authorized Program Analysis Report
APC	Application Performance Characterization
API	Application Programming Interface
ARB	Architectural Review Board
BIF	Benchmark Interchange Format
BLAST	Bit Line and Span Transform
BRF	Benchmark Reporting Format
BTM	Benchmark Timing Methodology
CAD	Computer Aided Design
CAM	Computer AIded Manufacturing
CBGA	Ceramic Ball Grid Array
CDE	Common Desktop Environment
CGE	Common Graphics Environment
CMY	Cyan Magenta Yellow
COSE	Common Open Software Environment
CPU	Central Processing Unit
CRT	Cathodic Ray Tube
DBE	Double Buffer Extension
DDC	Display Data Channel
DDX	Device Dependent X
DMA	Direct Memory Access
DSO	DirectSoft OpenGL
DSP	Digital Signal Processor
EDF	External Defaults File
ETC	Explicit Traversal Control
FIPS	Federal Information Processing Standards
GKS	Graphical Kernel System
GKX-CO	Graphical Kernel System Compatibility Option
GL	Graphic Library
GLU	OpenGL Utility
GLUT	OpenGL Utility Toolkit
GLX	OpenGL Extension to X
GPC	Graphics Performance Characterization
GUI	Graphical User Interface
HLHSR	Hidden Lines Hidden Surfaces Removal

HSB	Hue Saturation Brightness
HSV	Hue Saturation Value
IBM	International Business Machines Corporation
IOCA	Image Object Content Architecture
ISO	International Standards Organization
ITSO	International Technical Support Organization
LPFK	Lighted Program Function Keyboard
MBX	Multi-Buffer Extension
MPI	Message Passing Interface
NURBS	Non-Uniform Rational B Splines
OPC	OpenGL Performance Characterization
PAC	Peripheral Adapter Connector
PCA	Peripheral Connector Assembly
PEX-IC	PEX-Interoperability Center
PEX-SI	PEX Sample Implementation
PGI	Portable Graphics, Inc.
PHIGS	Programmer's Hierarchical Interactive Graphics System
PLB	Picture Level Benchmark
RAM	Random Access Memory
RAMDAC	Random Access Memory Digital Analog Converter
RGB	Red Green Blue
ROM	Read-Only Memory
SGI	Silicon Graphics, Incorporated
SIGGRAPH	Special Interest Group on Graphics
SIMD	Single Instruction, Multiple Data
SMP	Symmetric Multi Processing
SPEC	Standard Performance Evaluation Corporation
TGS	Template Graphics Software, Inc.
UDS	User Defined Specifications
VFB	Virtual Frame Buffer
VOO	Video Output Option
X	X Windows
XPC	X Performance Characterization
XVFB	X Virtual Frame Buffer
Xlib	X Windows Library

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

ITSO Redbook Evaluation

RS/6000 Graphics Handbook
SG24-5130-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5130-00
Printed in the U.S.A.

RS/6000 Graphics Handbook

SG24-5130-00