HP PEX Implementation and Programming Supplement

HP9000 Series 700 Color Workstations

HEWLETT PACKARD

HP Part No. B2355-90118
Printed in USA June 1996

E0696

FINAL TRIM SIZE : 7.5 in x 9.0 in

Notices

FINAL TRIM SIZE : 7.5 in x 9.0 in

Printing History

New editions of this manual will incorporate all material updated since the previous edition.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

June 1996 . . . Edition 1. This manual is valid for HP PEX 5.1v4 on all HP9000 Series 700 Computers running HP-UX 10.20.

0

About This Book

This manual is intended primarily for programmers of graphics applications and assumes familiarity with PEXlib and the installation and setup of graphics workstations. We also assume a knowledge of the C programming language and the X Window System .

Important information for system administrators is also included to aid installation, system maintenance, and troubleshooting.

While this book is not intended to teach operation of PEXlib, tutorial and other learning documentation are provided with the HP PEX 3D Developer's Environment for this purpose.

What's New in This Book

Product changes reflecting added functionality in this new edition of the *HP PEX Implementation and Programming Supplement* for HP PEX 5.1v4 include:

- Functionality
  - Triangle Primitives
    - `PEXOCCTriangleFan`
    - `PEXOCCTriangles`
  - Indexed Primitives
    - `PEXHPOCCIndexedMarkers`
    - `PEXHPOCCIndexedPolylines`
    - `PEXHPOCCIndexedTriangleFan`
    - `PEXHPOCCIndexedTriangleStrip`
    - `PEXOCCIndexedTriangles`
  - User-Defined Line Types and Marker Glyphs
    - `PEXHPOCCSetUserLinetype`
    - `PEXHPOCCSetUserMarkerGlyph`
  - User-Defined Highlight Color
    - `PEXHPOCCSetHighlightColor`
  - Face Lighting Control
    - `PEXHPOCCSetFaceLightingMode`
  - Stereo Viewing
    - `PEXHPSetStereoMode`
  - Wide Line Rendering Control
    - `PEXHPChangeRenderer`
  - Polygon Offset Rendering Control
    - `PEXHPChangeRenderer`
- New Device Support
  - HP VISUALIZE-EG
  - HP VISUALIZE-48XP

HP CDE and HP VUE

Hewlett-Packard is in the process of moving its users to a standard user environment. Two user environments will be shipped with HP-UX 10.20: HP VUE and HP CDE (Common Desktop Environment). Starting with HP-UX 10.20, HP CDE will be the default user environment. HP VUE will be available with HP-UX 10.20, but will not be available in future HP-UX releases.

From a 3D graphics point of view, the change in user environments should be transparent. See the *Common Desktop Environment User's Guide* for more information on HP CDE.

Although most examples in this manual only discuss HP CDE, HP PEX 5.1v4 supports both HP CDE version 1.0 and HP VUE version 3.0.

HP PEX Learning Products

Information about HP PEX and/or PEXlib is found in these books:

- *PEXlib Programming Manual*—This manual provides detailed instructions for learning and using PEX lib. Written by Tom Gaskins and published by O'Reilly and Associates, it provides beginners with the basics for getting started in PEXlib. Experienced programmers will use this book as a reference for more detailed information—beyond the basics. ISBN 1-56592-028-7.
- *PEXlib Programming Reference*—This book, also published by O'Reilly and Associates, is a reference to the PEXlib procedures based on the MIT document. The listings of each call and file include syntax, semantics, and a discussion of functionality. ISBN 1-56592-029-5.
- *HP PEX On-Line Information System*—Here is extremely fast access to desired information—whether it's PEXlib reference information or program code utilities, learning to get started using PEXlib, or even for running example programs or animation routines. Included is an on-line adaptation of the "Getting Started" chapter from the O'Reilly and Associates' *PEXlib Programming Manual*.
- *Portable Programming with CGE PEX 5.1*—This document gives information useful to those who want to create highly portable 3D graphics applications using the Common Graphics Environment (CGE) PEX 5.1 Extensions.
- *Read Me Before Using HP PEX Runtime*—Provides important information about running the HP PEX Runtime Environment product.
- *Read Me Before Installing HP PEX Development*—Provides important information about the installation and use of the HP PEX Development Environment for this product release.
- *Graphics Administration Guide*—This document, while not dealing solely with PEX, addresses many issues common to HP's 3D APIs—PEX, PHIGS, and Starbase. Issues include pathnames, compilation, and the operation of X Windows.
- *Using the X Window System*—Familiarizes users with the X Window System, beginning with the basic concepts and ending with a reference of the X Window commands. Includes information for system administrators.
- *HP Help System Developer's Guide*—Documents a complete system for developing online help for application software. Programmers can write online help that includes graphics and text formatting, hyperlinks, and communication with the application. HP Help also provides a programmer's toolkit for integrating the help facilities into applications.

Documentation Published by O'Reilly and Associates

O'Reilly and Associates are no longer printing the following manuals:

- *PEXlib Programming Manual*
- *PEXlib Programming Reference*

Although they are no longer orderable from O'Reilly and Associates, you may still find these manuals in bookstores that carry technical documentation.

Document Conventions

`verbatim`	This book makes extensive reference to PEXlib programming commands. When a reference is made, the function name is given in a typewriter-like font that indicates the verbatim entries you will make as program text or on the command line with file and directory names, routine names, parameters, and arguments. For example: `PEXEndStructure`
<*italic*>	Italic type enclosed in angle brackets indicates conceptual parameters (not verbatim parameters). That is, you "fill in the blank" with a value appropriate for the context.
OC	By PEX convention, this means "output command"
PEX	Technically, PEX is a protocol, a 3D extension to the X protocol that adds new functions to the X protocol for rendering 3D graphical objects.
PEXlib	A set of subroutines that follow the rules defined in the PEX protocol, PEXlib creates and sends PEX protocol. PEXlib provides an application's interface to the PEX protocol.

FINAL TRIM SIZE : 7.5 in x 9.0 in

Contents

Figures

Tables

1

An Overview of the HP PEX Product

The HP PEX Product

The HP PEX product brings high-performance, full-featured 3D graphics through the X server to the HP 9000 Series 700 workstations. PEX is composed of two parts:

- *PEX*: Technically, PEX is only a protocol—a set of rules that an implementation follows.
- *PEXlib*: This part is a set of subroutines that follow the rules defined in the PEX protocol.

The name "PEX" is often used to refer to the above two entities as a unit. In this document, this convention is followed: "HP PEX" refers to the whole product, and if only the PEX protocol is being discussed, it will be stated as such. When the implementation itself—the subroutine library—is being discussed, it is referred to as "HP PEXlib."

HP PEX is a powerful combination of Application Programmer's Interface (API) and workstation technology that provides integrated, distributed graphics—from low-cost workstations for simple 2D drawings or 3D wireframe to advanced 3D workstations with sophisticated lighting, shading, and texture-mapping for complex 3D models.

The industry-standard PEX developed by the X Consortium is a 3D Protocol Extension to the X Window System. Hewlett-Packard's PEXlib is a low-level 3D API or library to the PEX protocol that conforms to the PEXlib 5.1 standard. Hewlett-Packard's implementation of PEXlib can use Direct Hardware Access (DHA) to provide full performance, advanced 3D graphics in a local X window.

For remote rendering, HP's PEXlib emits PEX protocol requests over the network, to be finally rendered by a remote X server that supports PEX, or by a PEX terminal. To enable HP PEXlib client applications to run to an X server or

X terminal that does not support the PEX extension, HP PEXlib can generate X protocol. HP PEXlib does not support raw-mode graphics (no-windows mode).

For displaying pictures, HP PEX supports immediate-mode, structure-mode, and mixed-mode rendering to local and remote displays.

Several capabilities serve animation and visualization applications; for example, MBX (multi-buffering extension), texture mapping capabilities, alpha transparency, and others.

Structure mode calculates structures once, saving a database of information in memory on the server. When a user manipulates the 3D object, the client formulates instructions to set up graphical operations such as rotating, clipping, and zooming. The client sends instructions to the server to perform the calculations and redisplay the object with minimum changes to the database. Mixed mode offers users a combination of the two modes.

HP PEX implements the 5.1 PEXlib Specification with the CGE extensions (see below), and ANSI-C Language Binding. You may compile programs using either the ANSI C or Kernighan and Ritchie C compiler options.

In the HP PEX releases 5.1v3, HP released new functionality defined in 5.2 PEXlib. Specific key features were selected to enhance performance and ease of use in the HP PEX product. In the HP PEX release 5.1v4, HP releases more new functionality defined in 5.2 PEXlib, including new features and supported devices. HP PEX 5.1v4 also includes some extensions added by HP that are not part of the 5.2 PEXlib standard.

Supporting The Common Graphics Environment (CGE)

Hewlett-Packard encourages programmers in programming practices that assure the greatest portability and interoperability of programs and minimize the need for vendor-specific drivers or code paths. Several workstation vendors, including HP, provide Common Graphics Environment (CGE PEX 5.1), a library of functions with utilities that will assist you in establishing the uniformity necessary for programs to compile and run seamlessly regardless of the platform vendor.

One of the most exciting benefits about CGE is that it provides early availability of many functions slated for future releases of PEX. The early availability of these functions with the CGE PEX 5.1 Extensions can help you put more powerful and more widely usable applications into the hands of your customers.

Supporting Selected PEX 5.2 Functionality and HP Extensions

HP PEX releases 5.1v3 and 5.1v4 include some functionality and interfaces from the future PEX/PEXlib 5.2 standards, as well as extensions designed by HP. These features are being implemented in advance of the final standards, because HP believes that they will have significant value in many PEXlib applications. In some cases, minor differences between the HP implementation and the final PEX 5.2 standard may occur, but none should require more than very minor adjustments to make your application 5.2 conformant. It is important to note that the 5.1v3 and 5.1v4 releases are *not* a complete PEX 5.2 implementation; instead, as the release name implies, it is PEX 5.1 plus CGE PEX 1.0 extensions, plus certain selected items from the PEX 5.2 *draft* standard. Release 5.1v4 also includes HP extensions. Some of these 5.2 features may be available only from HP for some time to come, so use of them is a consideration for portability and interoperability. Nevertheless, you may find them very valuable in the interest of performance, functionality, and experimentation with some important features of PEX/PEXlib 5.2.

Product Structure

Hewlett-Packard makes two PEX products available—a developer's environment, bundled with the HP-UX Developer's Toolkit, and a runtime-only environment, bundled with the operating system. If you ordered the Instant Ignition option, the installation of HP PEXlib is greatly simplified because the operating files have been pre-loaded for you and the system is ready to run the moment you set up your workstation. This run-time product is required for each workstation that runs HP PEXlib applications.

Supported Workstation Configurations

This release of HP PEX is supported on the HP 9000 Series 700 3D Workstations running HP-UX 10.20 and HP CDE 1.0 (or HP VUE 3.0). However, you can choose not to use HP CDE and use only the X Windows environment instead.

The PowerShade product (B2156B/C) provides full lighting and shading functionality. HP work stations without PowerShade provide wireframe rendering and flat-shaded polygons.

Some devices enable you to create transparent overlay planes that are useful, for example, to add annotations that “float” over images rendered in the image planes.

As mentioned, the HP PEX Developer’s environment is bundled with the HP-UX Developer’s Toolkit or the ANSI/C HP-UX Developer’s Toolkit for the header files and development tools for the X Window System Version 11, Release 6, that are included.

Supported Environments

When running to a Hewlett-Packard server, an HP PEX client not using the X protocol mode supports only the visual types shown below (as returned by the Xlib call `XGetVisualInfo`). These are image visuals *and* overlay plane visuals, not pixmap drawables.

Table 1-1. Visual Types Supported by HP PEX/PEXlib

Visual Depth	Visual Class	Double Buffer	Supported Devices
8	`PseudoColor`	SW 8/8	Series 700 Color Workstations, HP VISUALIZE-EG
8	`PseudoColor`	HW 8/8	CRX, CRX-24, CRX-24Z, CRX-48Z, HCRX-8, HCRX-24, HP VISUALIZE-8/-24/-48
8	`TrueColor`	SW 8/8	Model 712, HP VISUALIZE-EG
8	`TrueColor`	HW 8/8	HCRX-8 HCRX-24, HP VISUALIZE-8/-24/-48/-48XP
12	`DirectColor`	HW 12/12	CRX-24, CRX-24Z, HCRX-24, HP VISUALIZE-24/-48/-48XP
12	`TrueColor`	HW 12/12	CRX-24, CRX-24Z, HCRX-24, HP VISUALIZE-24/-48/-48XP
24	`DirectColor`	None	CRX-24, CRX-24Z, HCRX-24, HP VISUALIZE-24/-48/-48XP
24	`DirectColor`	HW 24/24	CRX-48Z, HP VISUALIZE-48, HP VISUALIZE-48XP
24	`TrueColor`	None	CRX-24, CRX-24Z, HCRX-24, HP VISUALIZE-24/-48/-48XP
24	`TrueColor`	HW 24/24	CRX-48Z, HP VISUALIZE-48/-48XP

When running to a Hewlett-Packard server, the X protocol method of producing PEX graphics supports only the visual types shown below (as returned by the Xlib call `XGetVisualInfo`). These are image visuals and overlay plane windows, not pixmap drawables.

Table 1-2.
Visual Types Supported in the X Protocol Mode (VMX Driver)

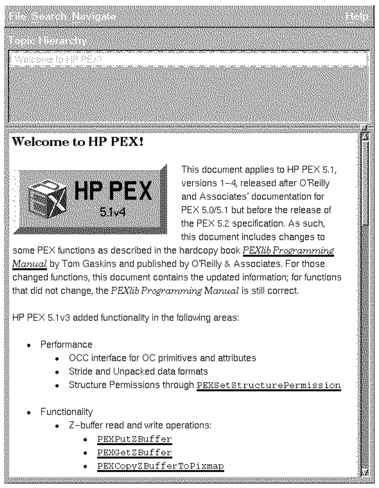
Visual Depth	Visual Class	Double-Buffer Support
8	`PseudoColor`	SW 8/8
8	`DirectColor`	SW 8/8
8	`TrueColor`	SW 8/8
24	`DirectColor`	SW 24/24
24	`TrueColor`	SW 24/24

Mixing Graphics APIs

The combinations of graphics APIs that are supported are these: Calls to Xlib and PEXlib can be mixed within an application, while calls to Starbase, PHIGS, and GKS cannot be made from a PEXlib application.

Information At The Speed Of Sight!

The HP PEX product lets you learn about the industry standard as well as access reference information so you work quickly and easily—displaying it all on your workstation:



Welcome to HP PEX!

This document applies to HP PEX 5.1, versions 1–4, released after O'Reilly and Associates' documentation for PEX 5.0/5.1 but before the release of the PEX 5.2 specification. As such, this document includes changes to some PEX functions as described in the hardcopy book *PEXlib Programming Manual* by Tom Gaskins and published by O'Reilly & Associates. For those changed functions, this document contains the updated information; for functions that did not change, the *PEXlib Programming Manual* is still correct.

HP PEX 5.1v3 added functionality in the following areas:

- Performance
  - OCC interface for OC primitives and attributes
  - Stride and Unpacked data formats
  - Structure Permissions through `PEXSetStructurePermission`
- Functionality
  - Z–buffer read and write operations:
    - `PEXPutZBuffer`
    - `PEXGetZBuffer`
    - `PEXCopyZBufferToPixmap`

Figure 1-1. HP PEX Online Information System's Front Page

The on-line PEX-related documentation on your system is for the PEX 5.1v4 release; it includes information on standard PEX, as well as HP's extensions to PEX.

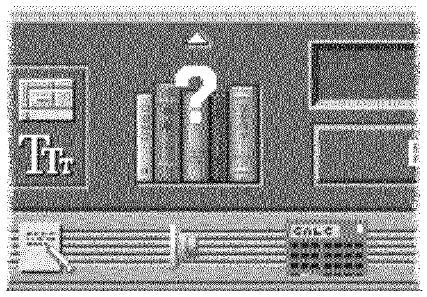
HP PEX Documentation: Tutorial, Reference, and Help

Loaded into the *HP PEX On-Line Information System* is a selected portion of Chapter 3, "Getting Started," a learning tutorial, from the *PEXlib Programming Manual*. This book, as well as the *PEXlib Programming Reference*, are published by O'Reilly & Associates—publishers widely acclaimed for their highly informative and easy-to-use books on the UNIX operating system, the X Window System, and PEX.

The *HP PEX On-Line Information System* is based on HP CDE Help, an application that provides this integrated, hypertext learning environment.

How to Access the HP PEX On-Line Information System

1. To access the *HP PEX On-Line Information System*, simply click on the Help Manager on the HP CDE Control Panel:



2. When the help window appears, scroll down unto "HP PEX" is visible. Click on the PEX Cube, then the underlined "Online Documentation" text. The PEX Online documentation will appear.

You can also bring up the on-line documentation by running the script ⟨*vhelp*⟩[1]`/bin/pexman`, with no arguments, from the command-line prompt. If you want to see a *particular* reference page for a PEXlib function, you can also type ⟨*vhelp*⟩`/bin/pexman` ⟨*command_name*⟩ (note that ⟨*command_name*⟩ is case-insensitive). For example:

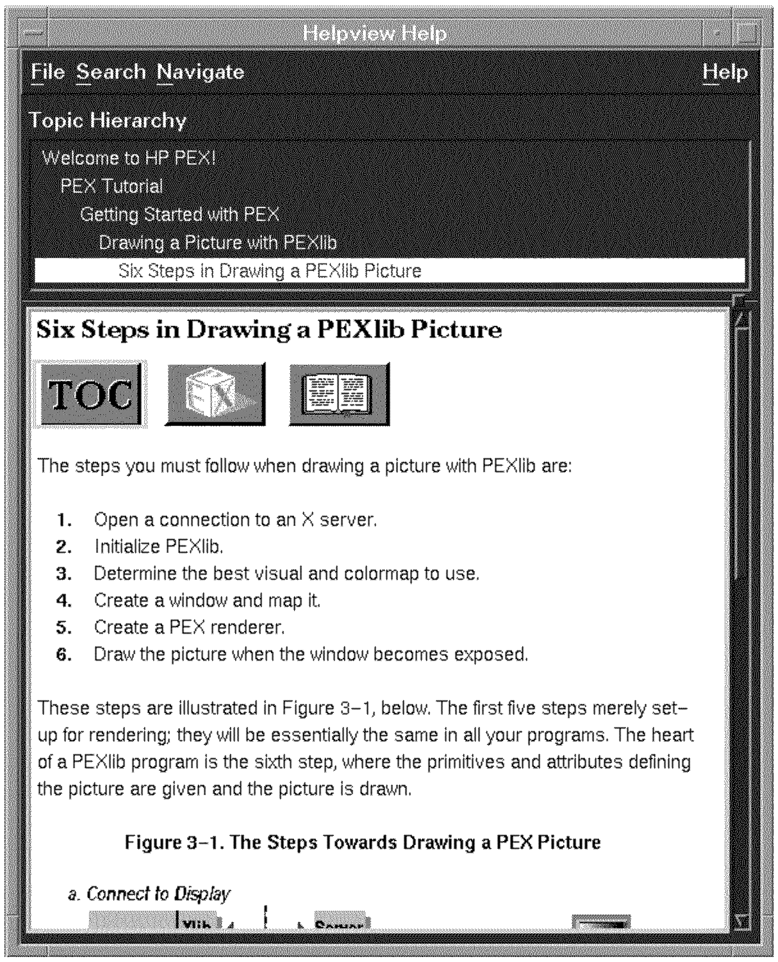
```
⟨vhelp⟩/bin/pexman pexinitialize  Return
```

If you need help in understanding how to operate the help system, you will find it by clicking the left mouse button on the "Help" menu in the upper right corner of the HP CDE Help window.

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

The Tutorial Gets You Started . . .

The on-line tutorial portion of the *HP PEX On-Line Information System* is designed to let you learn and even practice the essential steps that PEXlib applications must take to draw pictures:



Six Steps in Drawing a PEXlib Picture

The steps you must follow when drawing a picture with PEXlib are:

1. Open a connection to an X server.
2. Initialize PEXlib.
3. Determine the best visual and colormap to use.
4. Create a window and map it.
5. Create a PEX renderer.
6. Draw the picture when the window becomes exposed.

These steps are illustrated in Figure 3–1, below. The first five steps merely set-up for rendering; they will be essentially the same in all your programs. The heart of a PEXlib program is the sixth step, where the primitives and attributes defining the picture are given and the picture is drawn.

Figure 3–1. The Steps Towards Drawing a PEX Picture

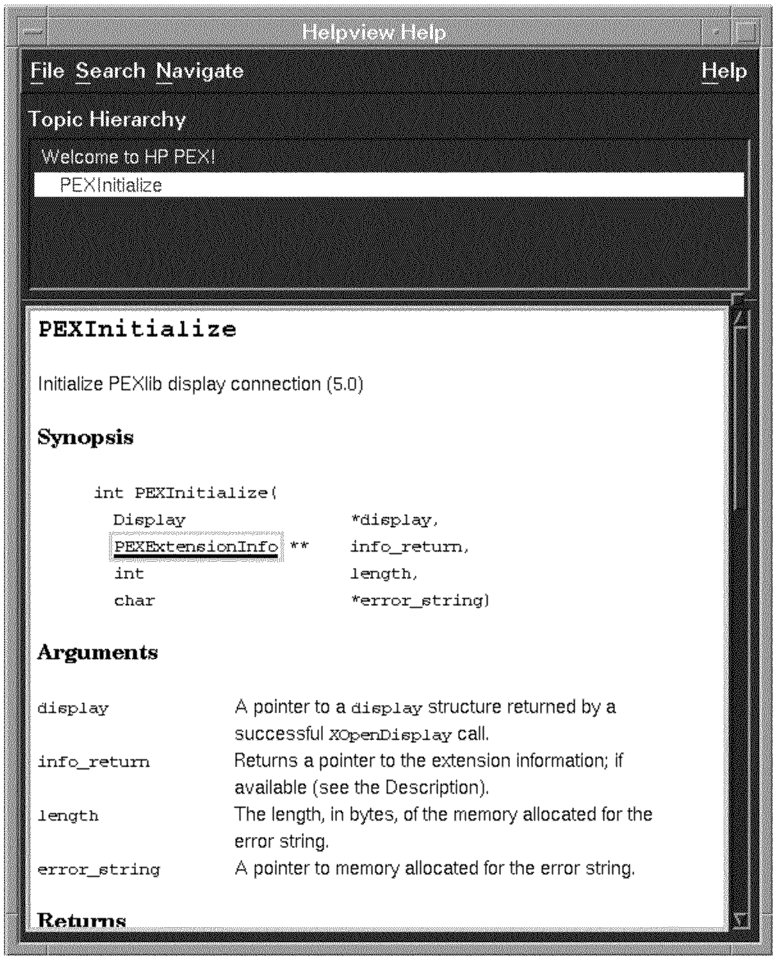
a. Connect to Display

Figure 1-2. The Tutorial Lets You Practice and Experiment

The tutorial enables you to run all the examples from *PEXlib Programming Manual*. Code from these examples can also be copied and pasted into your own programs to speed development. Many other examples and utilities are also provided, including documentation about the new functionality released in HP PEX 5.1v4, and tutorial information on texture mapping.

Reference Information Fast . . .

The *HP PEX On-Line Information System* also provides convenient access to reference information. You can select a PEX function, a `typedef`, or a `#define`, use the keyword search, or simply click on the name of the command you need, and the reference page is displayed.



PEXInitialize

Initialize PEXlib display connection (5.0)

Synopsis

```
int PEXInitialize(
  Display                *display,
  PEXExtensionInfo **    info_return,
  int                    length,
  char                   *error_string)
```

Arguments

display	A pointer to a display structure returned by a successful XOpenDisplay call.
info_return	Returns a pointer to the extension information; if available (see the Description).
length	The length, in bytes, of the memory allocated for the error string.
error_string	A pointer to memory allocated for the error string.

Returns

Figure 1-3. On-Line Reference Pages Feature Hyperlink Navigation

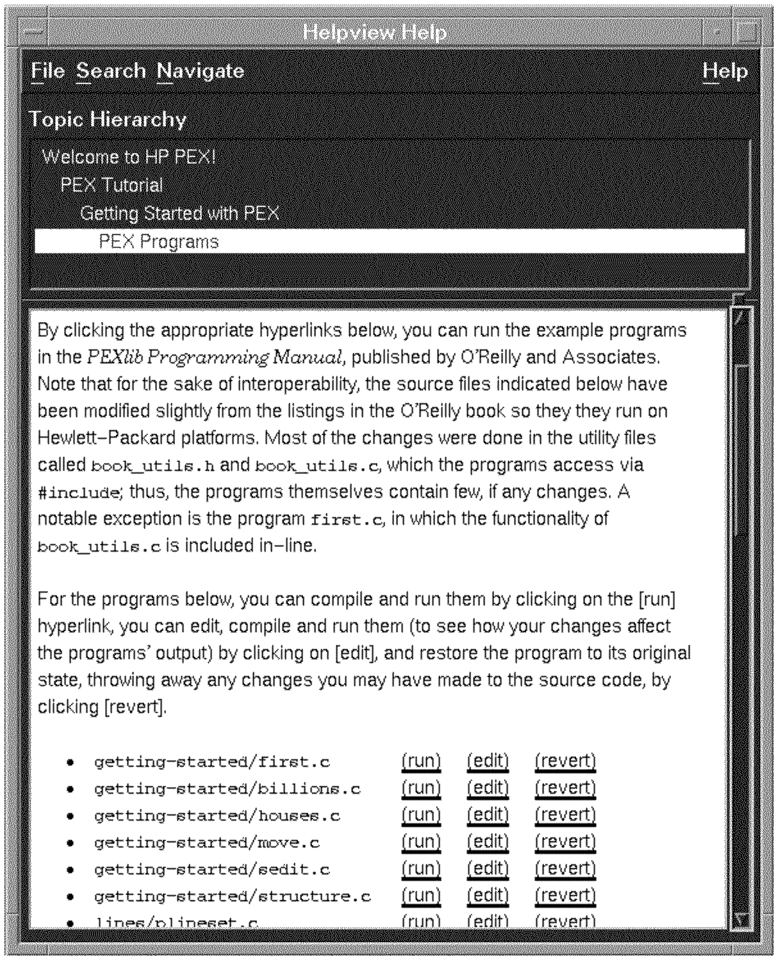
Access to Performance Hints

From the “Welcome to HP PEX!” window, select Performance Hints from the list of available options.

Performance tuning hints have been added to on-line documentation. This documentation is intended to be used by application developers who don’t have access to HP support channels to tune applications independently. The purpose of this feature is to add sufficient detail, and make it accessible to customers so that they can tune applications themselves.

How to Run Examples and Demos

The on-line system enables you to run the examples from the *PEXlib Programming Manual*. From the topic entitled "Getting Started with PEX" (under "PEX Tutorial"), click on the underlined hypertext link at the bottom of the page to run PEX programs: "(Click here to run PEX programs)." You are presented with a list of the example programs from Chapter 3 of the *PEXlib Programming Manual*:



By clicking the appropriate hyperlinks below, you can run the example programs in the *PEXlib Programming Manual*, published by O'Reilly and Associates. Note that for the sake of interoperability, the source files indicated below have been modified slightly from the listings in the O'Reilly book so they they run on Hewlett–Packard platforms. Most of the changes were done in the utility files called `book_utils.h` and `book_utils.c`, which the programs access via `#include`; thus, the programs themselves contain few, if any changes. A notable exception is the program `first.c`, in which the functionality of `book_utils.c` is included in-line.

For the programs below, you can compile and run them by clicking on the [run] hyperlink, you can edit, compile and run them (to see how your changes affect the programs' output) by clicking on [edit], and restore the program to its original state, throwing away any changes you may have made to the source code, by clicking [revert].

- `getting-started/first.c` (run) (edit) (revert)
- `getting-started/billions.c` (run) (edit) (revert)
- `getting-started/houses.c` (run) (edit) (revert)
- `getting-started/move.c` (run) (edit) (revert)
- `getting-started/sedit.c` (run) (edit) (revert)
- `getting-started/structure.c` (run) (edit) (revert)
- `lines/plineset.c` (run) (edit) (revert)

Figure 1-4. Easily Experiment with Programs

This enables you to experiment with the programs; running and editing them to see the effect of settings as you change them and revert the programs to their original state—*all without having to manually copy the programs and recompile them.*

These examples can also be copied and pasted (used as utilities) in your own PEXlib programs to speed development. Many other examples are also available in the ⟨*pex*⟩ directory[2] to use as a source of utilities from HP.

Similarly, there are texture-mapping programs that you can edit and run. You can find these in the "Texture Mapping Tutorial."

How to Print HP PEX Images

Hardcopy printing of HP PEX images is supported via `screenpr(1)`.

The `screenpr` command supports all the visuals and colormaps used by PEX on the supported HP graphics devices, with a variety of colormap setups including both `PEXColorSpace` and `PEXColorRange`.

How to Access PEXlib 5.2 Standard Specification Draft on the World Wide Web (WWW)

The PEXlib 5.2 specification is available on the World Wide Web. The URL is `http://www.x.org/pexlib/PEXlib52main.nographx.html`.

The HTML version of the PEXlib 5.2 draft may be out of date with the newest working document. The latest document can be accessed from the X Consortium FTP site at `ftp://ftp.X.org`.

Note that current HP PEX releases do not implement the PEXlib 5.2 specification. Instead, the current revisions of HP PEX implement the PEXlib 5.1 specification with some of the PEXlib 5.2 features and other extensions.

[2] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

2 Installation And Setup

Introduction

If you are setting up a new workstation, all software may be preloaded for you with the Instant Ignition option and you need simply verify this. If, however, you did not order Instant Ignition, then you will need to install the PEX filesets from the HP-UX Developer's Toolkit.

Special Considerations for the HP-UX 10.0 Release

HP-UX 10.0 is the first release of HP-UX to support the UNIX V.4 file system. Functionality-wise, little has changed; mostly just the file system organization.

Special Consideration for the HP-UX 9.0X Releases

You should consider the following information before you run an older HP-UX 9.0*x* release, such as an 9.01/9.03/9.05 PEX application on a more recent HP-UX 9.0*x* system, such as an HP-UX 9.07 system; before you run a more recent HP-UX 9.0*x* PEX application on a later HP-UX system; or before you run a more recent HP-UX X Windows application on an older HP-UX system.

HP-UX and Older HP PEX Applications Running on a More Recent HP-UX 9.07 System

The following issue should be considered before you run an HP-UX 9.01, 9.03, or 9.05 HP PEX application on an HP-UX 9.07 system.

- If your PEX application was created prior to the HP-UX 9.03 release, the application may exercise new paths through its source when using the `TrueColor` visual. To avoid compatibility problems, an environment variable has been included in X Windows for the *HP-UX 9.03 and 9.05 release only*. The name of the environment variable is:

  ```
  HP_SUPPRESS_TRUECOLOR_VISUAL
  ```

  This environment variable turns off the `TrueColor` visual.

HP-UX New HP PEX Applications Running on Older HP-UX Systems

The following list of items should be considered before you run a new HP-UX 9.07 HP PEX application on an older HP-UX system, for example, 9.03.

- If you create an HP-UX 9.05 or 9.07 HP PEX application that uses the Multi-Buffered X extension (MBX) and try to run it on an HP-UX 9.01 or 9.03 system, the application will not work. MBX is supported on HP-UX 9.05 or later systems.
- If your HP-UX 9.07 PEX application was compiled using archived libraries it will require the HP-UX or 9.07 X server when executing in the *direct hardware access* (DHA) mode (that is, rendering in a local window on the same system as the application). The HP-UX 9.07 X server is required because of graphics' dependencies on the HP-UX 9.07 X server.
- If your HP-UX 9.05 or 9.07 PEX application links `libXext.l`, it will not run on an HP-UX 9.01 or 9.03 system because this library is not available on either of these systems. To get the application to work on these systems, you need to explicitly re-link your application using the archive library `libXext.a`.

HP-UX 9.05 or 9.07 X Windows Applications Running on HP-UX 9.01 or 9.03

The following list of items should be considered before you run an HP-UX 9.05 or 9.07 X Windows application on an HP-UX 9.01 or 9.03 system.

- If you create an HP-UX 9.05 or 9.07 X Windows application that uses the Multi-Buffered X extension (MBX) and try to run it on an HP-UX 9.01 or 9.03 system, the application will not work.
- If your HP-UX 9.05 or 9.07 X Windows application links `libXext.l`, it will not run on an HP-UX 9.01 or 9.03 system because this library is not available on either of these systems. To get the application to work on these systems, you need to explicitly re-link your application using the archive library `libXext.a`.

Installation/Verification Instructions

Is Your System Software Preloaded with Instant Ignition?

Your workstation is preloaded with software, which may include HP PEX, if it was ordered with the Instant Ignition option. A yellow label attached to the workstation in its shipping carton confirms the workstation is preloaded:

Important

This product contains preloaded software.

Do not initialize internal hard disk drive.

Verify HP PEXlib on Your Workstation

To verify that PEX in installed correctly on your system, execute the program <*pex*>;`/demos/verify_install` to run the verification program (make sure that `verify_install`'s path[1] is in your `PATH` variable first):

```
verify_install Return
```

This program draws a cube with letter-shaped holes drilled through it: one shaped like a "P", one like an "E", and one like an "X", one letter for each of the three dimensions. You can iconify, maximize, and resize the window at will. Close the window to terminate the program.

You can also verify that PEX is installed and learn quite a lot about your particular system and the various extensions enabled on your workstation, such as the MBX and CGE extensions, with `/usr/contrib/bin/X11/xdpyinfo` (see also Appendix A).

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Note

Depending upon your hardware and colormap settings, you may experience "color flashing," something that is described in "Using `SB_X_SHARED_CMAP`" in Chapter 3. If there are other problems running this test graphic, subsequent error messages will point you to a solution.

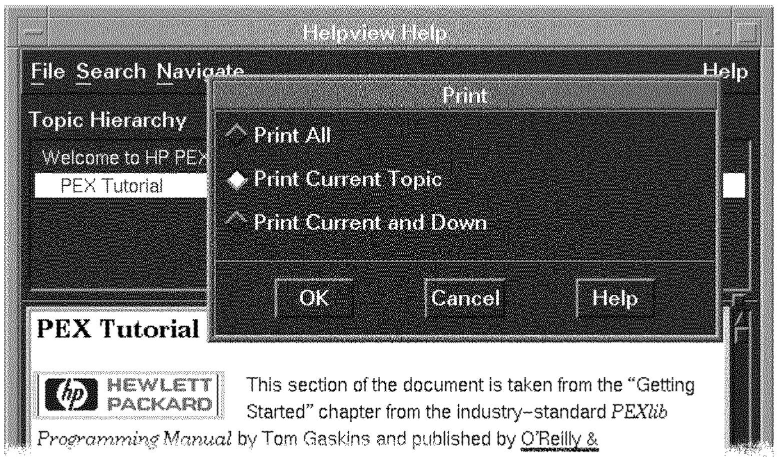
If HP PEXlib is *not* preloaded for you, skip to the section "Color" in Chapter 6, and the instructions "To Load HP PEX" in this chapter.

Setting Up the On-line Information System

If you are using the CDE environment, no additional effort is needed to install the *HP PEX On-Line Information System*. However, you may wish, for example, to change the appearance or color of the on-line information windows, or setup a Postscript printer.

How to Print On-Line Information

Printing of the on-line pages will be to your default printer. If your printer is Postscript-capable or has the fonts resident, pages are printed in the fonts as displayed on-screen.



Print All [✓ Print Current Topic] Print Current and Down

This section of the document is taken from the "Getting Started" chapter from the industry–standard *PEXlib Programming Manual* by Tom Gaskins and published by O'Reilly &

Figure 2-1. The Print Menu Display

The files that control the default printer settings of the on-line information with CDE Help are contained in the <*app-defaults*>; directory[2]. List the application default files, `Help*`. The file `Helpprint` enables you to select a printer model.

[2] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

How to Change the Appearance of your On-Line Display

The files that control the appearance of the on-line information with CDE Help are contained in the <*app-defaults*&> directory. List the application default files, `Help*`. The `Helpview` file specifies the size, color, and fonts used in displaying the on-line information.

Using a Font Other Than the Default

A default font is provided. However, because HP CDE saves and restores the X font path across multiple sessions, HP CDE users will need to explicitly modify the X font path to access fonts other than the default. This is described in "Text and Fonts" in Chapter 6.

Using the Command Line Prompt to Start the On-Line System

To be able to run the on-line information system from a command line prompt (with or without CDE), run the script <*vhelp*>[3]`/bin/pexman`. This allows you to access the tutorial and reference information from the command line.

On-Line Learning Product File Structure

The *HP PEX On-Line Information System* comprises a set of files under the ⟨*vhelp*⟩ directory. The entirety of the documentation is accessible via `pexman`; it starts with the HP PEX 5.1v4 on-line tutorial/reference and displays the top-level topic on your screen.

[3] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Information for HP-UX 9.0x System Administrators

To Load HP PEX

If you have a 9.0*x* release of HP-UX, and your workstation is not preloaded with Instant Ignition, you must install HP PEX (usually from a CD-ROM or DDS tape or over your company network), which requires superuser capability. (If you have HP-UX 10.0, PEX is bundled with other products, so no explicit PEX installation is necessary.)

1. Check that the workstation is running the HP-UX operating system, 9.05 or later; enter `uname -a` to display the operating system version information.
2. Verify long-file-name capability, which is required by HP PEX. If your system does not already support long file names, then, as `root`, use `sam` (1M), the menu-driven System Administration Manager program to convert the file system before you begin installation with `update`. Alternatively, you can use the `convertfs` (1M) command to convert your file system.
3. If you are using a CD-ROM or DAT, insert the source media into its device. If you are using a CD-ROM you will also mount the device onto the system. If you are installing HP PEX over your company network, skip this step.
4. As `root`, use `update` and follow the instructions. This update program is interactive, it provides messages, menus, prompts, and help screens to guide you through the procedure of selecting partitions `PEX` and `SHLIBS`.

   The `update` program checks fileset dependencies and that required disk space is available. It reports inadequate space, if necessary, so that you can take corrective action.

   Many types of errors, if they occur during the `update` program, will result in a message describing what happened and directions to remedy the error. See the next section "If you experience difficulty during installation".
5. Read the file `update.log`, which notes any installation problems, and reports other information from the installation process.
6. When your installation is complete, you must restart the X server.

   If you are using the HP CDE environment, log out of the session. Then at the welcome display, click on `Options`, then click on `Restart Server`. When the login display reappears, log in.

If you are using X Windows with some other window manager, stop the server (usually this is done by typing Shift CTRL Reset) and then restart it.

If You Experience Difficulty During Installation

Check for any error messages that were recorded in the file `/tmp/update.log` (make sure you check the dates and times printed in the file so you know which `update` session the error messages refer to). Also check the messages that may have been displayed from running `verify_install`.

Also check the device files. Devices specified in X server screen configuration files must correspond to existing device files with appropriate permission. If you don't already have the appropriate device file, you must create it using the `mknod` command. For information on `mknod`, see the *HP-UX Reference Manual* and/or *Using the X Window System*.

You may need to review following sections in this chapter in order to set up device files or use error messages to troubleshoot the installation before you compile and run programs.

If You Reinstall The X Window System

The file, `libXhpPEX.1`, a shared library file which is used by the PEX server, is installed with the X Window System with `update`. When HP PEX is installed, this file is overwritten to activate the PEX capabilities within the server.

It is important to note that if, for some reason, the X Window System is reinstalled after HP PEX has been installed, `libXhpPEX.1` is overwritten and HP PEX will fail to function. To correct this and reinstall the complete HP PEX version of `libXhpPEX.1`, type the following as `root`:

```
cd /
/system/PEX5-RUN/customize HP-PA
```

Also note that if the fileset `PEX5-RUN` is removed using `rmfn`, the original version of `libXhpPEX.1` from the fileset `X11-SERV` is restored.

If You Reinstall the HP-UX Operating System

If, for some reason, you must reinstall the HP-UX operating system, you must also reinstall HP PEX.

Caution HP PEX replaces some HP-UX 9.0/9.01 graphics filesets, so in those cases where it is necessary to reinstall your operating system, you will need to reinstall the HP PEX product filesets as well.

The HP PEX File Structure

This section contains information about some of the files shipped with the PEX product that is important for you to know before you begin working with HP PEX.

The PEXlib Filesets

The PEXlib fileset lists are found in the `/etc/filesets` directory under HP-UX 9.0*x* and are accessible via the `swlist` command under HP-UX 10.*x*. Listing the files in this directory that begin with `PEX5` will help you understand the structure of HP PEX.

HP PEX depends upon filesets of other software products in order to install and operate correctly. These filesets must be installed *before* you install PEXlib in your system.

The HP PEX Developer's Environment requires the header files for the X Window System Version X11R6. These are included with the HP-UX Developer's Toolkit (B2356A) or the ANSI/C HP-UX Developer's Toolkit (B2354A).

For a list of these products, please see the accompanying sheet, *Read Me Before Installing HP PEX Development*.

Server Files

There are a number of files that specify or affect the operation of the HP PEX server processes and are located in various directories[4]:

Table 2-1. Server Files

File	Contains
`libXhpPEX.1`	This file, in `/usr/lib/X11/extensions`, contains the server libraries.
`pexd`	This file, in ⟨*pexd*⟩, contains the PEX server program.
`XErrorDB`	This file, in `/usr/lib/X11`, contains the basic and standard X error messages.
`fp.PEX`	This file, in `/usr/lib/X11/extensions`, contains directory names that are added to font paths when the X server is started.
`PEXErrorHelp`	This file, in ⟨*err-help*⟩ directory, contains additional explanation of HP-specific error messages that require more than 80 characters in order to provide useful information.
`PEX.cat`	This file, in ⟨*nls*⟩ directory, contains the text strings for HP PEXlib-specific information included with errors to help users.

[4] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

3

Running HP PEXlib Programs

Introduction

This chapter describes the characteristics of HP PEXlib programs as they run on HP workstations. The PEX protocol defines the way information is exchanged between clients and servers. Clients send encoded requests to the PEX server and the server generates events, replies, and errors.

One of the key enhancements that HP has made in the area of client/server communications is enabling HP PEXlib to control and select the protocol method of operation that provides the best rendering performance. The selection and use of one particular method, and consequently the performance of your system, is dependent upon criteria that are described in the next section of this chapter.

This chapter also includes information about how you can customize the HP workstation clients by setting an operating method, particular colormap, or the use of color recovery all through setting **environment variables**.

Note that for some environment variables, the implementation details you'll need for using them are found in later chapters because they provide added graphics functionality.

Note If you are installing HP PEX, it is essential that you pass along this chapter to those who will be programming with HP PEXlib or using the PEXlib application.

Before you begin using the HP PEX product, it is important to check on-line files. You will find time-critical information in the HP PEX product Release Notes file `ReleaseNotes` (`10.0_Rel_Notes` on HP-UX 10.0) in the ⟨*rel-notes*⟩[1] directory.

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

This file contains important information about the HP PEX product that was not available in time for printing the hardcopy documentation. This includes information about utility programs that conveniently perform common operations and which aid interoperability of programs. Also look in the `README` files that may be found in the various directories and subdirectories of the HP PEX product.

Operating Methods

This section describes differences in how the client application can control the way rendering is performed through the HP PEXlib interface. There are three basic methods that PEXlib can assume: DHA, PEX, or X. You make the selection via the type of X display connection (normally specified with the environment variable `DISPLAY`) and the environment variable `HPPEX_CLIENT_PROTOCOL`.

Instructions for setting and controlling these methods using the `DISPLAY` and `HPPEX_CLIENT_PROTOCOL` as one of the environment variables are covered in this chapter. (The other environment variables that affect PEXlib behavior are also covered in a later section of this chapter).

In the accompanying table, you see the possible values of the environment variable `HPPEX_CLIENT_PROTOCOL` against the possible values of the X `DISPLAY` variable. Each cell in the table shows the order in which PEXlib tries to initialize the client connection based on the two environment variables. The progression is shown by "$\rightarrow$".

If the `HPPEX_CLIENT_PROTOCOL` environment variable is not set, or is set to an unrecognized value, this indicates the client wants PEXlib to select the best possible connection method to the server based on rendering performance and the relation between client and server systems. If a specific connection type is asked for, but the connection cannot be made, PEXlib does not try any other options and initialization fails.

A maximum of 16 display connections per HP-UX process can be initialized for PEXlib at the same time (see `PEXInitialize`).

Definitions for the terms that appear in the following table are explained here:

Definitions Used In Table 3-1

$\langle local\_host \rangle$: *n* . *n*	Indicates the client has connected to a server on the local machine, the same machine the client is running on.
$\langle remote\_host \rangle$: *n* . *n*	Indicates the client has connected to a server on the remote machine, a machine other than one the client is running on.
$\langle default \rangle$	If `HPPEX_CLIENT_PROTOCOL` environment variable is not set, or is set to an unrecognized value, this indicates the client wants PEXlib to select the best possible connection method to the server based on rendering performance and the relation between client and server systems.
DHA	Indicates the client will try to connect to the server using DHA (Direct Hardware Access). DHA does maintain a connection to the server, but not for rendering. All PEX rendering is done directly to the hardware, in cooperation with the X server.
PEX	Indicates the client will try to connect to the server using the PEX protocol extension.
X	Indicates the client will try to connect to the server using the X protocol.

Table 3-1. Progression of Protocol Selection

HP PEX Client Protocol	**X Display Variable**	
	`unix:`*n*`.`*n* *or* `local:`*n*`.`*n* *or* `shmlink:`*n*`.`*n* *or* ⟨*local_host*⟩`:`*n*`.`*n*	⟨*remote_host*⟩`:`*n*`.`*n*
⟨*default*⟩[1]	DHA→PEX→X→Error	PEX→X→Error
DHA	DHA→Error	Error
PEX	PEX→Error	PEX→Error
X	X→Error	X→Error

1 If the `HPPEX_CLIENT_PROTOCOL` environment variable is not set (or is set to an unrecognized value), PEXlib selects the method for best rendering performance.

Direct Hardware Access Method (DHA)

The DHA protocol method, a Hewlett-Packard feature, provides the highest graphics performance for local connections, its chief advantage. However, DHA is available only when the client and server are on the same workstation; that is, the client is running on the local server.

Visible Behavioral Differences of DHA Method

A client may want to use DHA method (and will get DHA method by default on a local machine) for performance reasons. However, since DHA method does not generate any PEX protocol requests, some minor behavioral differences may be observed. The differences are listed here along with a brief explanation:

- *Request Sequence Numbers Reported by Errors*—Normally, each X/PEX protocol request is tagged with a sequence number when it is sent to the X server. Both the client and the server keep track of these numbers independently. If an error occurs, the client reports an error for the request along with the sequence number of the request.

  In the DHA method, the sequence numbers reported for errors generated by PEX requests are meaningless. In the place of the appropriate sequence number, the application receives a sequence number for the most recent Xlib request, as if no PEX requests were being generated. This is natural because DHA method doesn't generate protocol, but it is a difference the application may see if any errors are generated.
- *Floating Point Exceptions*—Clients can receive floating-point exceptions due to bad data (see table "Data Values That Cause Problems," at the end of the chapter) or some divide-by-zero operations when running DHA.

It is often helpful to debug your applications using the DHA Protocol Method, before attempting to run them using the X Protocol Method. This is because errors are asynchronous in the X Protocol Method and because DHA rendering does additional error checking.

PEX Protocol Method

PEX protocol method provides the best performance in distributed graphics environments when a PEX-capable server is available, i.e., when the client and server are on different systems and the remote X server has initialized the PEX extension. In this method, PEX protocol is transmitted over the network to the remote server. Performance is dependent upon factors such as network loading, bandwidth, and the type of network.

There is little that can be assumed about protocol performance when sending PEX protocol to other vendors' PEX servers. Performance issues relating to the network type and total bandwidth apply, but the point-to-point capacities of the various vendors' network interface cards vary. In addition, each particular implementation of the PEX standard will possess its own performance characteristics.

X Protocol Method

This method is provided so that an HP PEXlib client application can run to an X server or X terminal that does not support the PEX extension. This method is characterized by local rendering into virtual memory, followed by `XPutImage` requests to the server. Since this method also uses the network for protocol transmission, it is subject to the same performance considerations as the PEX Protocol method. This method can be "forced," even if the server *does* support the PEX extension.

Visible Behavioral Differences of X Protocol Method

All the comments pertaining to the DHA method are also applicable to X Protocol method, plus the following additional differences:

- *PEX Extension Initialization*—Normally, a PEX extension is supported by the server and a corresponding structure is created on the client side when the PEX extension is initialized. In X protocol method, either there is no PEX extension on the server, or, the application is forcing rendering using the X protocol even though PEX is supported. The first case is the one that exhibits a difference.

  Since there is no PEX extension on the server, HP PEXlib creates a client-side extension structure anyway in order to function properly. However, the server knows nothing about a PEX client. In essence, the client thinks there is a PEX extension, the server thinks there is not.

  One result of this is that if the client calls `XListExtensions` or `XQueryExtension`, the PEX extension will not show up. (For this reason, applications that check PEX support and quit if there is no support in the server cannot use the X protocol method without some changes.) However, the `PEXInitialize` call will succeed.

  Another result of this is that error reporting and extension-event handling may collide with another valid initialized extension. The client may interpret a PEX error or event to be from another initialized extension. This aliasing behavior will only show up if enough valid extensions are initialized by the client so that the error- and event spaces are filled up—overlapping the PEX "fake" extension.
- *PEX Fonts*—The only font directories that are searched are the directories named in the file ⟨*extensions*⟩`/fp.PEX`[2] on the system where the *client* is running. All PEX fonts that are needed must be on the client's system.

[2] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Setting and Using Environment Variables

This section provides instructions for setting and controlling the environment variables that you can use to configure the PEXlib client.

To assure interoperability, your program should look for the existence of environment variables, since these variables are implementation-dependent, in order to use them as program parameters. If these do not exist, then the program uses default values.

Environment variables supported on HP workstations are summarized in the following table. Additional information you'll need for implementing them is provided either later in this chapter or in Chapter 6, as appropriate.

Table 3-2. Environment Variable Summary

Environment Variable Name	Range of Values	Description
`HPPEX_CLIENT_PROTOCOL`	⟨*default*⟩1, DHA, PEX, X	Desired rendering method. If method other than default is specified, initialization fails if selected method cannot be used.
`HPPEX_DHA_ECHO_COLOR`	`#`*xxx*2	Default colors echoed for primitives in a newly-created renderer. Note that echoing in HP PEX is done using "exclusive-or" drawing mode. This means that the actual echo color rendered will vary in different image locations, based upon the frame buffer contents prior to rendering.
`HPPEX_DHA_HIGHLIGHT_COLOR`	`#`*xxx*2	Colors highlighted for primitives.

1 DHA is the default if not set.

2 Color resources are set using color names from the X11 color database **rgb.txt** (e.g., "`White`"), or using the syntax `#`*rrggbb*. See "How To Set Environment Variables."

Table 3-2. Environment Variable Summary (continued)

Environment Variable Name	Range of Values	Description
`HPPEX_DHA_COMPLIANCE_MODE`	⟨any_value⟩; e.g., `True`.	This variable enables complete PEXlib compliance with the official standard specification. Because this variable must be set in order to cause complete compliance, HP PEXlib runs in the highest-performance mode by default. If your application cannot tolerate any differences from the standard, then set this variable. It is recommended this variable be set during development of an application to enable more robust error-checking. Because users must explicitly set `HPPEX_DHA_COMPLIANCE_MODE`, HP PEXlib's default behavior will exhibit minor differences from the standard.
`HPPEX_DHA_AUTO_COLOR_APPROX`	⟨*any_value*⟩; e.g., `True`.	When this variable is *not* set, HP PEX provides standard behavior with respect to the setting of the color approximation table. When this variable is set to any value, HP PEX enables some clients to run successfully that would otherwise abort, attempting to set an unsupported color approximation entry (which is not standard PEX behavior).
`SB_X_SHARED_CMAP`	⟨*any_value*⟩; e.g., `True`.	If you are using a low-end graphics device with only one hardware colormap, you can avoid color flashing through the use of this variable, setting it to any value. Use of `SB_X_SHARED_CMAP` and achieving generally satisfactory behavior requires some explanation; please see additional information in "Using `SB_X_SHARED_CMAP`".

Table 3-2. Environment Variable Summary (continued)

Environment Variable Name	Range of Values	Description
`HP_DISABLE_COLOR_RECOVERY`	⟨*any_value*⟩; e.g., `True`.	When this variable is set to any value before `PEXBeginRendering` or other similar entrypoint that binds the renderer to the window, the color recovery feature is disabled. For more on the use of this environment variable, see "Environment Variable—Color Recovery".
`HP_ENABLE_TRANSPARENT_MODE`	⟨*any_value*⟩; e.g., `True`	When this variable is set to any value, before starting the X11 server, the overlay planes become transparent. For more on the use of overlay planes and this environment variable, see "2: Determine Use of Transparent Overlay Planes" in Chapter 6.
`HP_COUNT_ TRANSPARENT_IN_ OVERLAY_VISUAL`	⟨*any_value*⟩; e.g., `True`	Determines whether or not you want to count the "transparent color" as a real color in the overlay visual. (Formerly named `CRX24_COUNT_TRANSPARENT_IN_ OVERLAY`).
`HPPEX_TXTR_SHMEM_THRESHOLD`		If an application wants to adjust the threshold to a lower limit, this variable can be exported. This variable will set the decimal number of bytes for a shared memory segment. The current threshold for texel maps is greater than or equal to a 1024 x 1024 x 3 byte (3MB) size. The system will attempt to allocate a shared memory segment.

How To Set Environment Variables

There are two ways to set environment variables. The choice depends on whether you are using HP CDE or if you are simply using the X environment.

Both methods are illustrated using the `HP_ENABLE_TRANSPARENT_MODE` environment variable:

- *Setting Environment Variables In HP CDE*

  If you are using HP CDE (Common Desktop Environment) add the following line to your `Xconfig` file:

  ```
  Dtlogin*environment:HP_ENABLE_TRANSPARENT_MODE=TRUE
  ```

  The `Xconfig` file may contain commented out entries for some of the more popular resources, including "`environment`". You need to find the line containing "`environment`", add the appropriate value, and uncomment the line. To eliminate the overlay plane, remove the line.
- *Setting Environment Variables In X Windows*

  If you are using `x11start`, make sure you have the environment variable `HP_ENABLE_TRANSPARENT_MODE` set before you execute `x11start`:

  ```
  export HP_ENABLE_TRANSPARENT_MODE=TRUE
  ```

  The best way to do this is to include it in your `$HOME/.profile`. To eliminate the overlay plane, this environment variable is unset by typing:

  ```
  unset HP_ENABLE_TRANSPARENT_MODE
  ```

  ... and restarting the X11 server.

Using Environment Variables

Environment Variable—To Specify Color

Color resources are set using color names either from the X11 color database `rgb.txt`, or using the syntax `#`*RedGreenBlue* where *Red*, *Green*, and *Blue* are hexadecimal numbers containing 1, 2, 3, or 4 digits (that is, `#`*rgb*, `#`*rrggbb*, `#`*rrrgggbbb*, and `#`*rrrrggggbbbb* are all legal syntaxes). These hexadecimal numbers indicate the amount used of that primary color. There must be the same number of digits for each of the primary colors.

For example, color names from `rgb.txt` or a color specified by one of these can take the following form (where *r*, *g*, and *b* are hexadecimal digits):

`#`*rgb*	*4 bits per color*
`#`*rrggbb*	*8 bits per color*
`#`*rrrgggbbb*	*12 bits per color*
`#`*rrrrggggbbbb*	*16 bits per color*

To set an environment variable for color, you may use this syntax:

```
export HPPEX_DHA_ECHO_COLOR=red
```
or
```
export HPPEX_DHA_ECHO_COLOR="#ffff00000000"
```

Colors are set, using the syntax shown above, for these HP environment variables:

```
HPPEX_DHA_ECHO_COLOR
HPPEX_DHA_HIGHLIGHT_COLOR
```

The default colors are echoed for primitives in a newly-created Renderer. Note that echoing in HP PEX is done using "exclusive-or" drawing mode. This means that the actual echo color rendered will vary in different image locations based upon the frame buffer contents prior to rendering.

More information about application resources, including color, is found in the book *Using the X Window System*.

Environment Variable—PEX Protocol Method

When using the PEX Protocol Method of connecting to the server, (that is, you have set `HPPEX_CLIENT_PROTOCOL` to `PEX`), you must reset the following environment variables, if you wish to use them, *before* the X server is started on the PEX server's system for the variables to have an effect:

```
HPPEX_DHA_ECHO_COLOR
HPPEX_DHA_HIGHLIGHT_COLOR
HPPEX_DHA_COMPLIANCE_MODE
HPPEX_DHA_AUTO_COLOR_APPROX
```

Setting these variables before starting the client process will not affect the PEX server which, ultimately, does the final rendering.

Environment Variables—Compliance Mode

The environment variable `HPPEX_DHA_COMPLIANCE_MODE` allows users to specify whether HP PEXlib strictly adheres to the standard, or to maximize HP PEX's performance, allowing some minor behavioral differences. In most PEXlib applications, these differences should be acceptable and preferable for the performance edge. For this reason this is the default behavior; that is, the variable is *not* set.

If your application cannot tolerate any differences from the standard, then set this variable. It is also recommended this variable be set during development of an application to enable more robust error checking. Because users must explicitly set `HPPEX_DHA_COMPLIANCE_MODE`, this default behavior will exhibit minor differences from the standard.

Here are the differences from the standard when `HPPEX_DHA_COMPLIANCE_MODE` is off:

- *Specular Reflections*—A directional eyepoint is used in lighting calculations; this is manifested as subtle changes in specular reflections.
- *Disabled Clamping*—Clamping of ambient, diffuse, and specular reflection attribute values is disabled.
- *Error Checking*—Comprehensive parameter error checking is not performed. The burden of transmitting good data to PEXlib procedures is placed upon the application.
- *Higher Performance*—The fastest transformation and rasterization paths are enabled to give applications a significant performance boost.

Because invalid data can cause programs to fail, Hewlett-Packard suggests the compliance mode variable be set during development of the application, and unset once the application is defect-free. The precise performance differences with the mode set and unset vary with the application. The application documentation should specify whether or not this variable should be set.

Using HPPEX_DHA_AUTO_COLOR_APPROX

When the variable `HPPEX_DHA_AUTO_COLOR_APPROX` is *not* set, HP PEX checks the values in a color approximation table entry and reports an error if any of the values do not match what is supported on the particular device and visual.

If the variable is set to any value, HP PEX recognizes unsupported values as the color approximation entry values are set. But rather than reporting an error, it creates and initializes a new X color map with a supported content for any window on which the invalid color approximation entry is used at the time it is installed. While this is not standard PEX behavior, the advantage is that it allows some clients to run that would otherwise fail.

Environment Variable—Colormaps

Note

If you are installing HP PEX, it is essential that you pass along the information about resetting the environment variable, `SB_X_SHARED_CMAP` to those who will be programming with HP PEXlib or using the PEXlib application.

Using `SB_X_SHARED_CMAP`

On graphics devices with only one hardware colormap, PEX applications may experience "color flashing." Color flashing is the condition where colors displayed on the workstation screen change as the focus moves from X window to X window and where the PEX image looks correct only when the focus is in its window.

This is common on low-end graphics devices because most PEX applications are demanding in their color usage, more demanding than many other X clients and window managers. Therefore, the PEX default color sampling requires a different colormap setup than many X clients and window managers. When the graphics device only supports one hardware colormap, both colormaps cannot be installed at the same time, and the switch from one colormap to another in hardware causes color flashing.

HP offers a method for avoiding color flashing on low-end graphics devices through the use of the environment variable `SB_X_SHARED_CMAP`. The use of this environment variable is explained in further detail below. However, you need to be aware that this may result in anomalous color behavior with some color-demanding X clients and window managers such as HP CDE. (An example of aberrant color behavior would be the minute hand on the clock in the HP CDE front panel leaving behind a different color as it moves around the face of the clock.)

If you are using a low-end graphics device with only one hardware colormap, you can avoid color flashing through the use of the `SB_X_SHARED_CMAP` environment variable. This is the best choice if you do not use color-demanding X clients and window managers such as HP CDE in "High Color" mode (the default HP CDE color mode). If you must avoid aberrant color behavior by color-demanding X clients, then this method is not available to you.

In general, if you attempt to share X colormaps among color-demanding X and PEX applications on a low-end graphics device, you may experience unexpected color behavior.

The setting of the `SB_X_SHARED_CMAP` environment variable determines the supported `PEXColorSpace` color approximation entry on the display. This environment variable has effect only on older simple HP displays that only support depth 8 visuals and exerts its effect at the time that the X/PEX server is started. (Changing the value after the server is running has no effect, except when X protocol mode is being used with a non-HP server. In this case, the local value of the variable at the time the PEX application is started controls the color approximation support.)

`SB_X_SHARED_CMAP` is especially important if you have PEX applications that need to share a colormap (perhaps the default colormap) with other X clients. In such a case, it is common for the X clients and window managers to be using pixel values in the low end of the colormap, for X rendering and for borders and backgrounds. On all of the single-visual devices, HP PEX supports a colormap in which the lowest forty cells are available for X clients, and the upper 216 cells are set up in an RGB color space sampling that HP PEX supports via `PEXColorSpace` approximation. This colormap configuration is often called a "6|6|6" colormap, because the color sampling includes six levels each of red, green, and blue.

The 6|6|6 default colormap setup and PEX color approximation support are enabled by setting `SB_X_SHARED_CMAP` to any value before the X/PEX server is started. If `SB_X_SHARED_CMAP` is not set, HP PEX supports a color sampling using 8 values of red, 8 of green, and 4 of blue. This colormap configuration is called "8|8|4." (It is also sometimes called "3:3:2", for the number of planes allocated to each of red, green, and blue. Note that "|" is used to delimit color levels and ":" is used to delimit frame buffer planes.) Since 8|8|4 mode consumes all 256 cells in the colormap, a PEX application must use a colormap other than the default.

Regardless of which protocol method of connecting to the server is used, you must reset the environment variable, `SB_X_SHARED_CMAP`, if you wish to use it, *before* the X server is started on the PEX server's system for the variable to have an effect.

On devices with only one hardware colormap, both the 6|6|6 colormap being used by the other clients, and the 8|8|4 colormap being used by PEX, cannot be installed at the same time. This results in "color flashing", since the PEX image will not look correct except when the 8|8|4 colormap is installed.

Note

If your PEX application does not require the higher 8|8|4 color resolution, set your `SB_X_SHARED_CMAP` environment variable to "True" on these lower-end graphics devices. Be aware that this may result in aberrant color behavior on the part of other color-demanding X applications as discussed above. However, setting the environment variable will allow many PEX applications to share a colormap with other clients, avoiding the "color flashing" problem.

The CRX Device And Color Support

The CRX device (only) exhibits one exception to standard X/PEX color support. For best performance, it requires a color sampling (either 6|6|6 or 8|8|4) in the colormap that contains the same number and values of cells as the other single-visual devices, but the cells are not in the "canonical" order described by X standard colormap properties or PEX color approximation entries. HP provides a program, `xhpcmap`, to transform a canonical color sampling into the required "shuffled" setup.

If your PEX applications all use this utility or equivalent code, you can set `SB_X_SHARED_CMAP` to "True" as described above, and obtain the best performance possible on the CRX. If, however, you have applications that do not incorporate code to set up the "shuffled" colormap, or there are other reasons why the special colormap is unacceptable, you can set `SB_X_SHARED_CMAP` to the special value `XA_RGB_DEFAULT` before starting the X/PEX server. HP PEX will render correctly using a canonical 6|6|6 colormap, but rendering performance on CRX may be noticeably impacted. The only mechanism to cause rendering to a canonical 8|8|4 colormap on CRX is to set `HPPEX_CLIENT_PROTOCOL` to "X" though, again, performance may be affected.

Environment Variable—Color Recovery

In order to provide higher-quality shaded images on low-cost 3D graphics systems like the Model 712, as well as the HCRX-8 devices, you can take advantage of a new feature called *color recovery*. Color recovery provides better pictures on low-cost workstations than are possible using only dithering by attempting to eliminate the apparent graininess caused by dithering.

Use of the color recovery feature attaches a different colormap to the X window than the application originally attaches. The colormap substitution occurs on the first `PEXBeginRendering` or other similar entrypoint that binds the renderer to the window. However, since it only occurs in `TrueColor` visuals, color flashing due to this change should not be objectionable. Applications that attempt to free the colormap they created will succeed; it is recommended that they not assume they can free whatever colormap is currently attached to the window, since the substitution may have occurred and they will get a permissions error.

Color recovery requires a different dither cell size when rendering shaded polygons and a digital filter is used when displaying the contents of the frame buffer to the screen. For this reason, color recovery can occasionally produce undesirable artifacts in the image. Some applications that read or write PEX images as raster images may be affected by the different dither cell.

To disable color recovery, you'll need to set and export the environment variable `HP_DISABLE_COLOR_RECOVERY` in the environment in which the X/PEX server is started before running your application. This disables the colormap substitution and color recovery. However, if this environment variable is not set in the server's environment, but is set in the DHA client's environment, color recovery is disabled for the client only.

The color recovery colormap is a read-only colormap. Attempts to change it are ignored and errors are not reported.

Environment Variable—Turning Off the TrueColor Visual

`TrueColor` visuals are supported with 9.05 release and later releases of HP-UX. Note that applications created prior to the 9.03 release of HP-UX may exercise new paths through their source when using this visual. To avoid compatibility problems, an environment variable has been included in X Windows for the *HP-UX 9.03 and 9.05 release only.* The name of this environment variable is:

```
HP_SUPPRESS_TRUECOLOR_VISUAL
```

Note The `HP_SUPPRESS_TRUECOLOR_VISUAL` environment variable is supported on HP-UX 9.03 and 9.05 only. It will not be supported in future releases of HP-UX.

The existence (not the value) of the `HP_SUPPRESS_TRUECOLOR_VISUAL` environment variable before starting the X11 server disables the TrueColor visual. If you set this environment variable after starting X11 server, it will be ignored.

To set the `HP_SUPPRESS_TRUECOLOR_VISUAL` environment variable before the X11 server is started, use one of the methods given below.

For Generic X Windows. If you are using `x11start`, make sure you have the environment variable `HP_SUPPRESS_TRUECOLOR_VISUAL` set before you execute `x11start`:

```
export HP_SUPPRESS_TRUECOLOR_VISUAL=TRUE
```

The best way to do this is to include it in your `$HOME/.profile`.

To unset the environment variable, type:

```
unset HP_SUPPRESS_TRUECOLOR_VISUAL
```

and restart the X11 server.

For HP CDE. If you are using HP CDE (Common Desktop Environment) add the following line to your `Xconfig` file:

```
Dtlogin*environment:HP_SUPPRESS_TRUECOLOR_VISUAL=TRUE
```

The `Xconfig` file may contain commented out entries for some of the more popular resources, including “environment.” Simply find the line containing “environment,” add the appropriate value, and uncomment the line.

To unset the environment variable, remove this line:

```
Dtlogin*environment:HP_SUPPRESS_TRUECOLOR_VISUAL=TRUE
```

from your `Xconfig` file and restart the X11 server.

PEX Fonts

PEX fonts are separate from X fonts in the HP implementation. The PEX fonts are located in the ⟨*pex-fonts*⟩ directory[3]. The way the server finds PEX fonts is through the X font-path mechanism. The application must be careful when manipulating the X font path; it is possible to leave PEXlib client applications without fonts. Just as with Xlib, the burden is on the client application to be a good X font-path citizen.

In order that the HP PEX fonts be configured into the default font path, the X server will look for file ⟨*extensions*⟩`/fp.PEX` on startup. If it exists, and if the default font path is not overridden on the server command line, the paths listed in this file (full paths ending in `/`, one path per line) will be part of the server's default font path. HP PEXlib installation will create this file to include paths for all the shipped PEX fonts.

Users of HP CDE should know that CDE saves the font path from each session and uses it for the next session. This means that the path to PEX fonts won't be properly set when using HP PEX for the first time after it has been installed. When you first use HP PEX following installation, you'll need to use `xset` to put the PEX fonts into the X font path. See "Text and Fonts" in Chapter 6.

[3] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Parameter Error Checking and Reporting

The HP PEXlib client, when emitting PEX protocol, does no parameter error checking. The standard PEX method is for the server to detect parameter errors in incoming protocol requests and report them to the client.

If DHA rendering is being used, or the X protocol method is in effect, parameter error checking will be done in the client process. Error reporting is handled differently when in the DHA and client process methods versus the PEX protocol method. In DHA and X methods, errors are synchronous so that special PEX sequence numbers are not reported.

The standard PEX error types and messages documented in the PEX Protocol Specification are generated by HP PEXlib. When running remotely to an HP PEX server or when running DHA, additional error data is included with the standard PEX error messages to help users better under stand the sources of the errors. An NLS-compatible error catalog, ⟨*nls*⟩`/PEX.cat`[4], contains the additional error messages for DHA PEXlib or when communicating with an HP PEX server. Some of these error messages cannot be adequately explained in 80 characters, the limit on error messages, so the file ⟨*err-help*⟩`/PEXErrorHelp` provides the additional HP-specific information to which users are referred.

The NLS error catalog resides in the directories ⟨*nls*⟩`/msg/C` and ⟨*nls*⟩`/american`. This additional information is handled by special functions installed in the Xlib extension hooks that enable the default X error handler to detect and print the values. The default X error handler, as of X11R6, is able to call these value-printing functions.

The basic and standard PEX error messages reside in the usual X11 error message catalog, ⟨*x11*⟩`/XErrorDB`, which is not NLS-compatible.

Certain errors, such as those related to calling a PEXlib routine before PEXlib itself has been initialized, and some graphics pipeline errors, cannot be handled as regular PEX errors. These types of errors are printed to `stderr` with PEX context information to aid troubleshooting.

[4] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

The Effects of Client Failures

The most common situation in which one PEXlib client can adversely affect another occurs when a client passes bad values in its data. This may cause a server to dump core, thus affecting all the clients connected to that server. However, the effects of this vary according the connection method of the PEXlib client—whether DHA, PEX, or X.

In the PEX method, a client can send bad floating-point data values to the PEX server, causing it to abort and affecting all other PEX clients. The HP PEX server supplies reasonable default values as results for operations involving bad data. The resulting images may not appear as you expected them, but the server will not fail. The effects of bad data on non-HP servers is unknown.

For clients using the X protocol method to render to a non-PEX-capable server, an abort will normally occur only on the client side—without affecting any other X or PEX clients. However, if for some reason the client aborts the X server itself, all other X/PEX clients also abort just as if any other X client aborts the X server.

Benefits gained by operating clients in the DHA protocol method are speed, isolation from other clients, and if a DHA client aborts, it is not likely to affect other PEXlib clients.

The following table describes the most common data values causing these problems and should be avoided:

Table 3-3. Data Values That Cause Problems

`NaN` ("Not a Number")	There is a reserved value in floating-point bit space that is called `NaN`. It is generated, for example, when 0/0 is evaluated.
`+Infinity`	There is a reserved value in floating-point bit space for positive infinity.
`-Infinity`	There is a reserved value in floating-point bit space for negative infinity.

It is important to know of these three conditions because many applications generate their graphics data as part of some application-specific calculations—and generated data is prone to producing these “toxic numbers.” If this occurs, your displayed images can appear with unexpected results. To avoid this situation, your application must not generate these three conditions or must filter these conditions from their data.

4
Utilities, Compiling And Linking, Examples and Demos

Introduction

The *PEXlib Programming Manual*, Chapter 3, “Getting Started”, and the on-line version of Chapter 3, *HP PEX On-Line Information System*, both illustrate the general steps for creating and running PEXlib programs. Specifics are included in the on-line documentation, where source code is compiled; see the `make` files for details. Instructions in this chapter are supplemental and necessary in order to develop and run programs on Hewlett-Packard workstations.

Listing the contents of the ⟨*pex*⟩ directory[1], you will see a number of important subdirectories, including:

Table 4-1. Demos, Utilities, and Program Examples

Subdirectory	**Description**
`demos`	Contains programs that demonstrate 3D capabilities of HP PEX. The PEX verification program `verify_install`, for example, is included here.
⟨*hp-examples*⟩	The program examples in this directory illustrate ways to achieve special graphics effects using PEXlib calls.
⟨*ora-examples*⟩	Contains source files for all the programs and utilities described in the *PEXlib Programming Manual*.
⟨*cge-examples*⟩	Contains program examples that demonstrate some of the portability and functionality of the CGE PEX 5.1 extensions.
⟨*pex-utils*⟩	Contains general information and a number of important utilities dealing with colormaps and visuals, double-buffering, gamma correction, and use of Motif widgets.
⟨*cge-utils*⟩	Contains important, highly recommended utilities that will assist you in creating highly portable applications.
`README`	Read this file to learn about obtaining and using the source files.

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Including Header Files In Your Applications

HP PEX uses standard C and X11 header files, providing definitions and declarations shared among program files (list the ⟨*pex_incl*⟩ directory[2]). Most, if not all programs, will require that at least these header files must be included at the beginning of your programs. Your program may require others in addition to these.

```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

Most HP PEXlib programs and applications that only use the standard PEXlib data types, definitions, and function declarations, need only include the header file `PEXlib.h` under the ⟨*include*⟩ directory. Use the following syntax:

```
#include <X11/PEX5/PEXlib.h>
```

Notice that `PEXlib.h` includes several key Xlib header files such as `Xlib.h`. It also includes `PEX.h`, which is a separate file in the same directory, and contains the definitions and types defined by the PEX protocol.

Still other header files may be needed by your program, depending on your application. For example, in order to provide access to the additional functionality of multi-buffering extension and the CGE extensions (among varied workstation platforms from workstation vendors, including Hewlett-Packard) you must include `PEXExtlib.h`, as shown below. Among these header files are `PEXExtlib.h` and `PEXHPlib.h`, which must be included in your program *after* `PEXlib.h`, as the final include example shows.

```
#include <X11/PEX5/PEXExtlib.h>
#include <X11/PEX5/PEXHPlib.h>
```

For 5.1v3 and later HP PEX releases, a new header file, `PEXHPlibint.h` is included by `PEXlib.h`.

[2] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Another example is the colormap and visual utilities that require you to include `PEXUtCmap.h` from the ⟨*cge_utils*⟩[3] directory. This header file defines the constants and structure types for use with these utilities.

Instructions for including various additional header files are usually provided with the `README` file that accompanies the utility or function. The `README` also includes instructions for using or operating the utilities.

Table 4-2. Header Files for Advanced Functionality

File[1]	Description
`PEXHPlib.h`	Contains the data types, declarations, and function declarations required by extensions to the PEXlib API that are supported by Hewlett-Packard.
`PEXExtlib.h`	Contains CGE extensions for portability and interoperability.
`multibuf.h`	Multi-buffering extension from `libXext.sl`.[2]

1 See list below for complete pathnames

2 See the O'Reilly *PEXlib Programming Manual*, 14.2: "The Multi-buffering Extension" for further information.

Now your header file declarations at the beginning of your program should appear:

```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h>
#include <X11/PEX5/PEXHPlib.h>
#include <X11/extensions/multibuf.h>
```

[3] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Using the Utility Programs

A number of utilities have been added to the HP PEXlib product that simplify and speed programming or which enable you to display sophisticated images.

Utilities For The Common Graphics Environment

To encourage and assist you in the development of applications based on the common graphics interoperability conventions, a number of highly recommended utilities are made available in the directory ⟨*cge-utils*⟩[4]:

```
PEXUtCmap.c
PEXUtCmap.h
PEXUtCmapint.c
PEXUtCmapint.h
PEXUtExt.h
PEXUtdbint.h
```

A shared library providing these utilities is shipped as ⟨*pex-lib*⟩[4]`/libPEXUt.sl`.

Utilities From the O'Reilly Manual

The directory ⟨*ora_examples*⟩, which contains the examples from *PEXlib Programming Manual*, also contains the `book_utils.c` utility. O'Reilly developed this utility to set up the workstation. However, it is important to notice that HP has modified `book_utils.c` shipped with the HP PEX product to provide improved interoperability and properly set up HP workstations to run the O'Reilly programs.

Hewlett-Packard recommends that you use the HP-modified utility instead, especially if you intend to use programs from other workstations which use O'Reilly examples, or if you obtain the programs again directly from O'Reilly & Associates per the instructions in the Preface of *PEXlib Programming Manual*, "Obtaining the Example Programs."

```
#include "book_utils.h"
```

[4] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Utilities from Hewlett-Packard

Hewlett-Packard ships a number of additional utilities with the HP PEX product. These utilities are in the ⟨*pex_utils*⟩ directory[5]. See the `README` file in this directory to learn about these utilities and how they are useful.

Table 4-3. HP Utilities in Utilities Directory

Subdirectory	Description
`PEXSimple.c`	A basic Motif widget for a PEX drawing area.
`hpgamma.c`	An HP utility to enable gamma-correction for anti-aliasing.
`pexutcmap.c` `pexutcmaphp.c` `pexutcmapint.c`	Source code for HP-originated utilities for visual selection and colormap creation.
`pexutdb.c` `pexutdbint.c`	Source code for HP-originated utilities for portable double buffering.

[5] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Examples, Utilities and Demo Programs

The HP PEX product contains many examples and programs, in addition to those supplied with the O'Reilly *PEXlib Programming Manual*, that demonstrate use of various functions and utilities that can make you more productive or to display more sophisticated images.

The ⟨*pex*⟩ directory contains the examples and demos. This directory and subdirectories for each of the examples and programs also contains a `README` file to explain the contents and, importantly, instructions for using the programs. This table notes some of these examples; see the directories themselves for a complete list, and the `README` files in the directories for explanations.

- ⟨*hp-examples*⟩`/SubsetAAModeling`—Interactive demos showing the differences between subset mode, mixed mode and immediate mode rendering. These examples also demonstrate antialiasing capabilities.
- ⟨*pex*⟩`/demos/drive/PEXdrive`—HP's networked driving simulator.
- ⟨*hp-examples*⟩`/TexMap/boundary.c`, ... `/composition.c`, ... `/orientation.c`, ... `/param.c`, and ... `/texture.c`—Texture-mapping examples (see Chapter 9)
- ⟨*hp-examples*⟩`/pexdpyinfo`—A utility for developers; displays PEX extension information, plus information on the enumerated types, implementation-dependent constants, lookup table entries, and sup ported PEX visuals for a particular display. See "`pexdpyinfo`" in Appendix A.
- ⟨*hp-examples*⟩`/dblbuffer_pexut`—Rotates a cube using the `PEXUt` double-buffering and color utilities.
- ⟨*hp-examples*⟩`/wireframe.c`—Uses two supplied graphic object data files to display both wire frame and solid surface image with shading.
- ⟨*hp-examples*⟩`/overlay.c`—Simple example of overlay and image plane use.
- ⟨*hp-examples*⟩`/screen_dump.c`—Example of programmatic invocation of the `screenpr`(1) command to generate a screen-resolution dump to a PCL printer.
- ⟨*hp-examples*⟩`/alpha_blend.c`, `alpha/alpha_twopass.c`—Alpha blending examples demonstrate the use of the HP alpha blending extensions to PEX functionality.
- ⟨*pex*⟩`/demos/verify_install`—Rotates the PEX cube.
- ⟨*hp-examples*⟩`/hp_example_utils.c`—Utility procedures for the examples
- ⟨*hp-examples*⟩`/cge_simplewin.c` and ⟨*hp-examples*⟩`/cge_makewin.c`—Programs that illustrate the use of the two most powerful colormap/visual utilities in CGE.

- ⟨*hp-examples*⟩`/widgetdemo`—Demonstrates creation of `XgPEXSimple` Motif widget, displays dotted line in it, and provides a pulldown menu to close it.
- ⟨*hp-examples*⟩`/wideline_ctl.c`—Demonstrates the control of stroked versus filled widelines.
- ⟨*hp-examples*⟩`/polyoff/polyoff_ctl.c`—Demonstrates use of the polygon offset performance feature.

Using the HP Examples

The file[6] ⟨*hp-examples*⟩`/README` describes how `Makefile` is used to build executable programs of the various examples included in the directory.

How To Link To Shared Libraries

HP PEXlib is supported on the Series 700 workstations using shared libraries that must be linked with the application program.

When you compile your PEXlib programs, you must link the application with the PEXlib library `libPEX5` just as described in the *PEXlib Programming Manual* and on-line system. Notice that the PEX library is dependent on the math library.

A compile line will typically appear:

```
cc program.c -I/usr/include/X11R6 -I⟨pex-incl⟩/X11R6 -L⟨x11r6⟩ \
  -L⟨pex-lib⟩ -lPEX5 -lXext -lX11 -lm
```

See the *Graphics Administration Guide* for more information on compiling.

[6] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

This table summarizes the shared libraries and X11 directories that are linked on the command line example above.

Table 4-4. Shared Libraries and X11 Directories

Library	Description
`libX11`	X11 routines
`libXext`	X11 extensions
`libPEX5`	PEXlib routines
`libm`	math functions

5

Performance Hints

Steps to Getting Good 3D Graphics Performance

As an application developer, one of your primary concerns is designing and tuning your application so that it runs at full performance on supported platforms. On HP graphics systems, a simple technique can be applied to graphics-intensive applications that will help you accomplish this goal.

The first step in this process is to choose appropriate hardware and software for the application that you will be running. Once the application has been written or ported to this platform, determine if you are reaching expected performance levels. If not, determine where the bottlenecks in the application are, or how the application is using system resources.

If the problem is in the application's interaction with the graphics hardware or software, some straightforward techniques can be used to help identify the source of the problem, and some simple guidelines can be followed in order to improve performance. If the problem is not related to the application's use of graphics, other options may be available to improve application performance.

Although many of these techniques apply to all of the graphics libraries supported by Hewlett-Packard, the specific tuning graphics guidelines discussed here focus on PEXlib.

Identify SPU and Graphics Hardware Suited For the Application

Your choice of a hardware platform will depend on the type of application you are planning to support. For example, will end users spend only a small amount of time creating a model, but spend most of their time rotating objects and viewing them from different directions? If so, the graphics hardware may be the more important consideration. On the other hand, if the application is going to solve complicated equations while rendering, the choice of CPU may be more important to your application performance.

Where to Get Information About HP Systems

Benchmarks and technical information about HP systems is published in the Product Data Sheet for that system, which is available from your sales representative. If you have access to World Wide Web, you can also find much of this information by opening Hewlett Packard's home page, at `http://www.hp.com`. From the home page, you can access information about HP computers and peripherals, support services, hints for troubleshooting problems on HP systems and other timely information. Another source of news about HP products is *The Hewlett-Packard Journal*.

System Level Benchmarks

Several benchmarks are published about HP graphics systems that should help you to determine if the system is an appropriate choice for your application. Typical system data includes `SPECint` and `SPECfp` ratings, and `Linpack` and `Dhrystone` benchmark results.

Graphics Benchmarks

The Graphics Performance Characterization committee (GPC) provides a set of Picture Level Benchmarks (PLBs), which are a standardized, vendor-independent measure of graphics performance. The benchmarks are run from the PLB interpreter program, which executes a series of graphics calls. A number of data sets, which contain graphics calls that a typical application might make, are provided by the GPC committee. For example, PLBWire93 is a good indicator of wireframe application performance. PLBSurf94 is a good indicator of 3D shaded surface performance. Xmark93 is a good indicator of how the user interface will perform.

Other published graphics data includes triangles/second, vectors/second and quadrilaterals/second. This type of performance number is not always available for a given device. Beware of these specifications, because they rarely reflect actual performance of an application. These benchmarks and PLBs are usually available on the Product Data Sheet. Other information about GPCs is available in the *GPC Quarterly*. You can receive copies of that publication from university libraries, by subscription, or from a sales representative. The *GPC Quarterly* is published by the National Computer Graphics Association (NCGA), 2722 Merrilee Drive, Suite 200, Fairfax, VA 22031.

Other Considerations

Other performance considerations for 3D graphics users include:

- Does the graphics system include a hardware Z buffer? A hardware Z buffer is used to accelerate hidden surface removal. If your application renders 3D solids or surfaces, a hardware Z buffer will accelerate rendering and animation of complex models.
- Is hardware double-buffering supported at the depth your application needs? Double-buffering allows smooth movement of dynamic images. To the human eye, double-buffered animation sequences appear to run faster.
- Does the system include overlay planes? Running the graphical user interface (GUI) in the overlay planes and the graphics in the image planes can result in a substantial performance improvement for some 3D applications. This is because exposure events, caused by GUI interactions like pop-up menus, can force expensive redrawing of the graphics images when the pop-up menus disappear.

- Is rasterization of primitives accelerated? If it isn't, rendering times are strongly impacted by window sizes.
- How does the graphics system interact with the CPU? If the graphics computations are done completely by the graphics device, graphics performance will not scale with a faster CPU. If the mathematical computations are done in the CPU, instead of by specialized graphics hardware, graphics performance will scale with CPU performance.
- Is texture mapping supported in hardware? Texture mapping gives an application the ability to map a 2D image onto a 3D surface for a more realistic rendering.

Software double-buffering, hidden surface removal and texture mapping are supported on all HP graphics devices. However, hardware support significantly improves performance for this functionality.

Choosing a 3D Graphics Application Programmer Interface

HP supports several graphics application programmer interfaces (APIs). Your choice of API will depend on the features of the API, as well as performance.

PEX is a vendor-independent extension to the X Window System that is supported by major workstation vendors including HP, Sun, IBM, and DEC. PEXlib is the corresponding API that generates PEX protocol. PEX provides client/server graphics and the broadest set of functionality supported on HP platforms. PEXlib provides full performance graphics on HP systems.

Starbase is a low-level, proprietary API that has been used by HP customers for many years. Starbase is a feasible option for many applications that do not need client/server technology. Starbase runs at full performance on all HP-designed graphics devices now, and will continue to be supported in the future.

HP-PHIGS is a high-performance implementation of the industry standards, PHIGS and PHIGS PLUS. The current release of HP-PHIGS (Version 3.0) is the last major release of the product. Future systems may not run HP-PHIGS at full performance. Figaro and GPHIGS are commercial products. Figaro is available from Template Graphics Software. GPHIGS is supplied by G5G.

OpenGL is supported on HP-UX 9.07 for application developers that use OpenGL. The OpenGL software distributed by Hewlett-Packard is an Evans & Sutherland product for Freedom systems, and is supported by Evans & Sutherland. OpenGL runs at full performance on Freedom systems. Evans & Sutherland plans to enhance OpenGL over time.

Determining How the Application is Using System Resources

Choosing an Effective Benchmark

To accurately verify whether or not your application is reaching maximum performance levels, you will need to run some benchmarks. An effective benchmark focuses on the critical functionality of the application. What tasks does a typical end-user repeat most often while using your application? Is the data used in the application typical of the complexity of the data that your end users work with?

In addition, it is important that the benchmark does not spend a lot of time in startup activities, but does spend enough time on other tasks to give you an accurate understanding of performance issues. Benchmark performance should scale on different systems according to the published specifications. If it doesn't, your benchmark is probably not spending enough time on the critical application tasks, in this case, rendering graphics.

Finally, the benchmark results should be repeatable, with approximately the same timing measurements for each time a given task is run.

Identify the Bottlenecks

If the use of system resources is not balanced by the application, performance can slow down considerably. Performance bottlenecks can occur in many different places in your applications, including: graphics, the CPU, memory, the network, and I/O systems. For example, an application that does extensive mathematical computations on data before displaying that data may be CPU-bound. In other words, it will spend a long time processing data and using the CPU resources before sending any information down for graphics processing, although it might appear as if graphics is slow.

It is critical that you understand how the system resources are being used before beginning to tune your code. Otherwise, the time spent tuning code may have little impact on your overall application performance. For example, if graphics is the bottleneck and you are already getting maximum performance from the graphics hardware, no amount of change to the interactions between the graphics and your application will improve your application's performance. Similarly, if the network is the bottleneck, no amount of graphics tuning will improve application performance.

Performance Analysis Tools

Several HP-UX tools are available to help you determine how the system resources are being used.

`/bin/time` is a UNIX command that can be used to run a program and determine what percentage of time is being spent in user code and what percentage is being spent in the system.

For example, running a demo program from the PEXlib Programming Manual produced these timing results.

```
$ /bin/time night-time
real 15.2
user 11.4
sys 0.4
```

The first time is the time elapsed while running the program. The `user` time shows how much time was spent in the `night-time` code, plus how much time was spent in all of the libraries linked with the code (including `libPEX5.sl`). The `sys` time shows how much time was spent in the HP-UX kernel.

GlancePlus is an interactive performance diagnostic tool for HP-UX systems. It provides general data on system resources and active processes. You can use Glance to view information about the current use of system resources and active resources. Specific data is available about the current CPU, memory, disk I/O, LAN, NFS, and swap usage. Glance provides both global system information and specific process information.

HP-PerfRX is a tool that continually logs global performance data about processes running on your system and prints tables and graphs showing global system resource usage. It provides summaries of system usage metrics over time. These tools would probably not be useful for initial performance tuning. However, the information might provide insight into how different processes are affecting overall application performance on a particular system.

If you order a new system with instant ignition, you will automatically get trial copies of Glance and HP-PerfRX. For ordering information, or access to trial copies of Glance, GlancePlus, or HP-PerfRX, call 1-800-237-3990 in North America.

HP/PAK, the HP Performance Analysis Kit, consists of three tools that help you analyze the performance of your applications. Each of the tools examines program performance at a different level of detail. XPS looks at the relative use of system resources by all processes at the system level. DPAT is an interactive tool that looks at the performance of a process at the procedure level. HPC looks at the performance of compute-bound procedures at the statement/instruction level. In HP-UX 10.0, HP/PAK is bundled with compilers.

Profiling Your Code

Profiling tools are available on HP-UX. Execution profiles provide information about where your application spends most of its time to help you to identify performance bottlenecks.

The `gprof`(1) command can be used to give you some raw data about the amount of time spent in each of your application's procedure calls. `gprof` requires recompilation of your application. `gprof` also provides cumulative timing information about the execution time of each procedure and the subroutines it calls. Shared libraries are not profilable, so gprof will not provide information about the graphics calls your application makes unless you follow the instructions described in the next section; only the information about the procedures in your graphics libraries will be profiled.

If you have purchased SoftBench, you also have access to the SoftBench Performance Analyzer. Softbench provides user-friendly access to profiling information similar to the information produced by `gprof`, and some other features as well. More details on SoftBench Performance profiling can be found in the *SoftBench User's Guide*.

Profiled PEXlib

If you have HP-UX 9.07 or 10.10, and have installed HP-PEX 5.1, Version 3.0 or later, a profiled archive library containing the highest level of PEXlib calls is shipped in ⟨*profile*⟩[1]. To get additional information about time spent in PEXlib calls, add `libPEX5_prof.a` to your link line, as shown in the example below.

```
cc -DHPPEX_PROCEDURES program.c -I/usr/include/X11R6 \
  -L/usr/lib/X11R6 \
  -W, -L /usr/contrib/PEX5/lib \
  -lPEX5_prof -l PEX5 -lXext -lX11 -lm
```

This profiled library is only useful for performance tuning. It should not be used to build an actual product, since it is not a supported part of the HP-PEXlib product.

Other Tools

Many of the techniques described in this document are somewhat invasive. In other words, you need to be able to modify and recompile code to perform some experiments. You may also be able to get access to less invasive tools by contacting your sales representative. Included in this set are tools that extract graphics calls from your application, producing compilable code. By using that extracted code, you can effectively duplicate the interactions of your library with the graphics library, without rebuilding your entire application.

There are several reasons to do this. First, you can easily find out what percentage of time is spent in your application outside of the graphics library. To do this, run your application benchmark and time the results. Next, run your application benchmark and extract graphics calls. Create a compilable program of just graphics calls, run it, and time the results. By comparing the two timings you can see exactly how much time is spent in application overhead compared to graphics calls.

Another reason for extracting calls from your program is to see how efficiently graphics calls are being used. For example, you can look for redundant attribute setting calls. You can also see the effects of different sequences of calls on graphics performances. For example, polylines might be rendered very fast when preceded

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

by one sequence of attribute calls, but not rendered according to published specifications when preceded by another set of attribute setting calls. Extracting the calls gives you an easy way to view the sequence of graphics operations as performed by your application.

Interpreting Published Performance Data

One way to determine if your application's graphics performance is reaching acceptable levels is to compare your benchmark results with the published figures from the Product Data Sheet for your system.

The performance level that you achieve may vary from the published benchmark numbers. For example, the size of vectors in your application might differ from the size of vectors quoted in the Product Data Sheet. If your vectors are longer than the vectors described in the benchmark, and more pixels need to be drawn, your application will not draw as many vectors/second as were drawn in the benchmark. It is best to use the GPC benchmarks to get an indication of the type of application performance you can expect to see.

If all conditions are the same as the Product Data Sheet benchmarks, you should be able to achieve performance comparable to the numbers listed on the Product Data Sheet. Otherwise, it is possible that your application is not executing the most optimized paths of the graphics libraries. If so, it is worth trying some of the experiments described next.

Examining Graphics Interactions

If after profiling your benchmarks you determine that graphics is the bottleneck, and, furthermore, that you are not achieving maximum performance on a graphics system, you need to look at your application's interaction with the graphics libraries.

HP's Graphics Library Optimizations

All HP-UX graphics products are tuned for maximum performance based on typical application usage. In other words, there are some combinations of primitives and attributes that HP graphics libraries will execute faster than other combinations of primitives and attributes. In order to determine what those paths are, you need to study the available documentation for each release. If this documentation does not help you understand the performance problems you are experiencing, then you will need to perform some simple experiments.

Documentation Sources

In most cases, the combinations of primitives and attributes that are optimized do not vary from one API to another, since optimizations are focused on typical application usage. However, there may be some performance issues specific to the graphics products that you are using. HP tunes its graphics libraries for each release. For most graphics APIs, some documentation is shipped with the product about performance tuning. In order to be aware of all performance improvements in the graphics library, read the Release Notes and `PERF_NOTES` whenever you plan to support a new release of any of HP's graphics APIs.

Most vendors ship similar documentation. When designing your application, it is good practice to read the documentation from multiple vendors, in order to determine which primitives and attributes work best across all of the platforms you plan to support.

Online Documentation

Tips for improving application performance on PEXlib are published in the Release Notes document. These files are found in the `/etc/newconfig` directory on HP-UX 9.01, 9.03, 9.05 and 9.07 releases, or, on HP-UX 10.0 and later releases, in `/opt/graphics/PEX5`.

Starbase tips are available in `/usr/lib/starbase/PERF_NOTES` on HP-UX 9.07 or earlier releases, and in `/opt/graphics/PERF_NOTES` for systems running HP-UX 10.0 or later releases.

Studying Optimizations Shown in the GPC Quarterly

In general, PLB benchmarks are most useful for comparing systems before purchase. However, there is one piece of information in the GPC Quarterly that is helpful to application developers for performance tuning. Published with a summary of the GPC results is a description of the optimizations made by the vendor to achieve maximum performance for the benchmark. By applying the optimizations used in the benchmarks to your program, you should be able to improve performance in your application.

Systematically Tuning Your Graphics Application

Attributes

The settings of attributes, the number of times attributes are called, and the types of attributes used in your program can all affect graphics performance. Some attribute settings simply involve more work than others. For example, for each light turned on in a `PEXSetLightSourceState` call, a set of mathematical computations must be done to light the primitives in the scene. The more light sources turned on, the more expensive the call.

Redundant attribute settings (for example, attribute calls that are made more than once but don't change values of the current settings) can be very expensive in some implementations. Although the HP graphics libraries do a lot of redundancy checking, certain redundant attribute calls will cause primitives to be drawn using non-optimized paths in the graphics libraries. This can slow graphics performance considerably. Always avoid making duplicate calls to attribute setting routines. If you must set attributes frequently to different values, consider grouping primitives that share similar attributes. For example, sort the primitives according to reflection characteristics, and render all primitives with the same reflection characteristics at once.

Finally, some attribute settings are not optimized by the implementation.

All three of these factors can affect your application performance. The next section describes an experimentation process that will help you determine which attribute calls are having the most impact on graphics performance.

Attribute Suppression Experiments

It is relatively simple to determine whether or not you are setting attributes correctly in order to execute the optimized paths in the graphics libraries. Run your benchmark on your application as it is currently written and record the timing results. Then, experimenting with one attribute call at a time, suppress the attribute calls (that is, comment out the function calls) in your benchmark and rerun it. Compare the timing results. This will change the appearance of the rendering, but that is acceptable for this kind of experimentation. If you get significantly better results with a reduction in attribute calls, look for redundancy in attribute calls.

A single attribute call may not affect whether or not your application hits the optimized paths. Sometimes you need to experiment with sets of related attributes. If attributes are changed in sets, you need to experiment with the entire set. For example, in PEXlib, both the view orientation matrix and the view mapping matrix might be modified to change the view. Commenting out these calls one at a time would have no effect, since the view orientation matrix and the view mapping matrix are concatenated each time one of them changes; but commenting both calls out at the same time might show a significant improvement.

For example, in PEXlib, you might experiment with the following attribute calls:

PEXlib Lighting and Shading Calls:

- `PEXSetLightSourceState`
- `PEXSetReflectionModel`
- `PEXSetSurfaceInterpMethod`
- `PEXSetTableEntries` (for lighting table setup)
- `PEXSetDepthCueIndex`
- `PEXSetTableEntries` (for depth cueing table)
- `PEXSetPolylineInterpMethod`

PEXlib Viewing Calls:

- `PEXViewOrientationMatrix`
- `PEXViewMappingMatrix`
- `PEXSetTableEntries` (for view matrix)
- `PEXSetViewIndex`

PEXlib Surface Attributes:

- `PEXSetReflectionAttributes`
- `PEXSetInteriorStyle`

PEXlib Color Attributes:

- `PEXSetLineColor`
- `PEXSetLineColorIndex`
- `PEXSetMarkerColor`
- `PEXSetMarkerColorIndex`
- `PEXSetSurfaceColor`
- `PEXSetSurfaceColorIndex`
- `PEXSetTextColor`

Other PEXlib Calls You Should Experiment With

- `PEXSetFacetCullingMode`
- `PEXSetFacetDistinguishFlag`
- `PEXSetLineWidth`

If your application does not set attributes redundantly, then it might be that your application is setting attributes in a way that is not optimized in the libraries. You need to look at the attribute calls and determine if an optimized path might work for your application instead. While it is not always possible to reduce the number of attribute calls, you may want to make some appropriate tradeoffs between appearance and performance. For example, in a preview operation, it may not be necessary to turn depth cueing on or use wide lines.

If you are confused about which paths are optimized, use the published documentation that is shipped with your libraries, or study the optimizations described in the *GPC Quarterly*.

Data Formatting Experiments

Just as redundant attribute changes can impact performance, frequent changes in the data formats can also affect application performance. In this case, data format refers to whether or not normals and colors are passed to PEXlib with the vertices. In primitive calls like `PEXFillAreaWithData`, this information is passed to PEXlib in the `vertex_attributes` mask. If you are using the OCC interface, vertex attributes are described in the `PEXOCC` structure.

Determining how changes in data formats are affecting your overall application performance is more difficult than the attribute experiments described above. You will need either to sort the data passed to PEXlib or to perform multipass rendering using your data. To sort data, you would need to group all of the geometry with identical vertex attributes and render it all at once. In a multipass rendering, you would need to traverse the data several times. The first pass might only render primitives without vertex normals. The second pass might render only primitives with vertex normals, etc., until you have rendered all of the primitives in the model. Timing results can be a little confusing, though, because the traversal time needs to be accounted for in a multipass rendering.

Translation between the application's native data format and a packed data format can also have an impact on performance. See the section called Data Formats below for PEX-specific information on data formats.

Window System Interactions

Window size may be a factor in rendering performance. Larger windows can be slower, especially when rasterization is not done in hardware. On HP systems, this is usually not a problem. In HP-PEXlib, window size might be a factor in texture mapping performance, since hardware acceleration is not available for texture mapping on all devices.

Window system interactions can affect performance if the user interactions cause the graphics to be redrawn frequently. This can happen when the application generates a lot of exposure events, and when menus and other user interface items are drawn in the image planes.

Geometry Suppression

Often the amount of detail in the geometric model is greater than the amount of detail needed to render an object realistically. By experimenting with geometry suppression you may also be able to improve application performance. In this case, your application does not render all of the geometry available.

Two general techniques can be applied to many graphics applications. In the first, multiple variations of the geometry are used. Depending on the level of resolution required for the user's task, different geometry is rendered. In some cases, a very coarse resolution is acceptable. Multiple primitives can be combined into a single primitive. Objects that are too small to be seen can be removed altogether, and replaced with an alternate representation.

Another technique uses bounding boxes to trivially reject all offscreen geometry. This technique is useful when you have a "world" scene, and the viewer can only look in one direction.

PEX Specifics

DHA, Protocol Mode and VMX Mode

On HP systems, there are three fundamental ways to communicate with the graphics libraries and render 3D graphics in PEXlib. Direct Hardware Access (DHA) is the fastest method available when both client and server are on the same workstation. In DHA mode, graphics commands are sent directly to the graphics rendering libraries by PEXlib. In contrast, in protocol mode, graphics requests are sent over the network to the graphics server, where they are decoded and translated into graphics commands that are then sent to the graphics libraries on the server. In the X protocol method, PEXlib commands are translated into X protocol requests, which in turn travel over the network to be decoded by the X server and rendered.

Whenever both the client and server are available on the same system and performance is important, you should run in DHA mode. This is the default, but you can explicitly control the mode by setting the environment variable `HPPEX_CLIENT_PROTOCOL` to DHA.

Structure Mode, Immediate Mode

In PEXlib, there are two ways to draw a scene. If you are using immediate mode, you can pass all of the primitives and attributes in the scene to PEXlib one at a time, each time you want to draw the picture. In structure mode, you store all of the primitives and attributes in a graphics database, then tell PEX to render the contents of that database. Immediate mode rendering is best suited for applications that need to modify model data frequently, or need to reduce memory usage. Structure mode can be used when the data is somewhat static throughout the application.

When running locally, structure mode reduces procedure call overhead and parameter processing times. Most of the cost is incurred at the time the model is built, not when it is rendered. For example, error checking of parameters can be done when the data is stored in the structure, and does not have to be repeated each time the model is rendered.

Structure mode is even more useful in a distributed environment. Since the network is frequently the bottleneck in distributed application performance, it is important to try to minimize network traffic. Storing the data in structures is one way to do that.

Many applications combine the two modes. Non-changing data is stored in structures, but other data that changes frequently is sent to the graphics server in immediate mode. For example, the geometry of a model might be stored in a structure, but viewing calls are made each time the scene is redrawn.

Structure Permissions

Structure permissions control the access to structures by applications. By calling `PEXSetStructurePermission`, an application can set the permission of a structure to either `PEXStructureWriteOnly` or `PEXStructureLocked`. Write-only structures cannot be read by `PEXFetchElements`, and locked structures cannot be edited.

By setting structure permissions appropriately, you permit PEXlib to use its internal knowledge about the best performance paths, and pack primitives in the most efficient way for the hardware on which it is running. For example, if a locked structure contains multiple `PEXPolyline` primitives, PEXlib can pack those primitives into a single `PEXPolylineSetWithData` call, reducing procedure call overhead and resulting in faster execution of those polylines. This example is only possible when the structure is locked.

Other optimizations are even possible in write-only structures. For example, decomposition of polygons can be done only once per write-only structure, instead of every time the contents of the structure are rendered.

Whenever an application needs to continually rerender unchanging models, storing data in structures with write-only or locked permissions should be considered. If you need to edit your structure, set the structure permissions to `PEXStructureWriteOnly`. If you don't need to edit, best performance can be achieved by setting permissions to `PEXStructureLocked`.

Using Structures Efficiently

The `ExecuteStructure` output command that is used to create a structure network can be expensive, because attributes' values are saved when a child structure is executed, and are restored when the traversal returns to the parent structure. Consequently, it is good practice to avoid excessively deep structure networks and avoid creating structures that have very few elements.

Stride and OCC vs. PEX 5.1 Interface

PEXlib offers two major argument interfaces for output commands (primitives and attributes): an explicit interface and an output command context interface (OCC). The explicit interface requires that you specify the display, resource ID (renderer or structure), and request type for every output command function call that you make. The explicit interface is the only interface available on PEXlib 5.1 (including HP-PEXlib, Versions 1.0 and 2.0).

The OCC interface is currently available in HP-PEXlib5.1, Version 3, and will be available in the PEXlib 5.2 implementation. The OCC interface generates the same protocol as it generated using the explicit interface, so that PEXlib programs using the OCC interface can communicate with earlier 5.1 servers.

Output commands using the OCC interface replace the first three arguments, and other frequently used primitive descriptions, like `vertex_attributes`, with a single OC context. The OC context is an opaque structure that contains many of the arguments that are commonly found in the explicit interface output commands.

In addition to providing a reduced argument count for output commands, the OCC interface supports different data formats. The packed form is the same form that was used in earlier releases of PEXlib. It requires you to format data into packed data structures defined by PEXlib. The stride form allows you to supply data formatted in application-defined structured arrays without the need to copy the data into the PEXlib-defined structures before invoking the PEXlib functions. The unpacked form allows you to supply the data in separate lists for each data type. Vertex coordinates, normals, and colors are stored in separate lists in the unpacked form.

Using the OCC Interface

In general, the OCC interface uses far fewer arguments than the explicit interface, making coding easier and improving performance. The OCC interface is recommended for applications that are supported on HP-PEXlib, Version 3.0 or later. Best performance is achieved by minimizing the number of calls that modify the OCC context, and not intermixing calls that use different OCC contexts.

Data Formats

If your application is running in DHA mode on HP, the selection of data interface may have a significant impact on performance of your application performance. The OCC interface is implemented at the PEXlib level; the protocol generated by the OCC interface is identical to the protocol generated using the 5.1 PEXlib interface. Consequently, performance is only affected when running in DHA mode.

HP-PEXlib is optimized to use the packed and stride data interfaces most efficiently. The unpacked data interface is executed significantly slower. However, whether or not to use the packed interface depends on the size and nature of your data. Using the unpacked form may be more efficient than converting large amounts of data to the packed form, if the packed form is different from the application's native data format, and if conversion routines are more time-consuming than the difference in performance you will get by calling HP-PEXlib using the packed data interface. In order to determine which interface to use for your application, write some simple benchmarks and experiment with the data formats and conversion routines.

One advantage of the stride interface is that it is possible to change vertex and facet attributes without copying data. If your application is going to rerender the same geometric data with different attributes, the stride interface is an appropriate choice.

Shape Hints

All PEXlib `FillArea` calls accept a `shape_hint` parameter. By providing a value other than `PEXShapeUnknown`, you can bypass some unnecessary processing in some cases. For example, on HP hardware, convex shapes can be passed to the graphics hardware immediately. Non-convex shapes may require some preprocessing by PEXlib.

In early releases of HP-PEXlib, shape hints were ignored. In PEXlib 5.1, Version 3.0, shape hints make significant performance differences in many cases. The use of shape hints for potentially non-convex polygons (that is, polygons with more than four sides) is strongly recommended.

Remember that the shape hints must be accurate. Incorrect shape hints can result in the wrong picture being drawn, or in slower performance.

Use of Complex Primitives

Best performance on the newer graphics products can be achieved by reducing the CPU "non-graphics" overhead, such as procedure call overhead. Applications can reduce overhead by packing more primitives into library calls. In HP-PEXlib 5.1, Version 3, a number of complex primitives are supported. However, just using those primitives is not enough. Applications must send enough primitives per call to amortize the overhead. Best performance is achieved when at least eight primitives are packed per call, with performance levelling out at some point above fifty primitives per call. However, some applications have achieved significant performance improvements simply by changing the number of triangles per strip from two or three to five or six.

Compound primitives optimized for PEXlib include:

- `PEXTriangleStrip`
- `PEXPolylineSetWithData`
- `PEXFillAreaSet`
- `PEXSetOfFillAreaSets`
- `PEXQuadrilateralMesh`
- `PEXFillAreaSetWithData`

If the Bottleneck is Not Graphics

By profiling your code, or running `/bin/time`, you may have determined that graphics is not the bottleneck for your application. Several of the more promising options for tuning your code are described below.

Build Environments

Compilers are continually tuned by Hewlett-Packard. Simply updating to newer revisions of compilers and rebuilding your application may improve performance significantly.

Compiler and Linker Options

If your application is CPU bound, you may be able to substantially improve performance just by compiling and linking with some optimization options. For example, compiler options can automatically remove dead code, make better use of registers, optimize loops, generate in-line code, optimize for a particular architecture (for example, PA RISC, Version 1.0; or PA RISC, Version 1.1), and optimize your application based on a run-time profile. For C programs, this process is described in Optimizing HP C Programs. Linker optimization options are described in Programming on HP-UX.

Archive Math Libraries

If graphics applications are spending significant time in the math libraries, linking with the archive version instead of the shared version might help. This is because the symbol resolution overhead is reduced with archive libraries.

Cache and TLB Misses

Cache and Translation Lookaside Buffer (TLB) misses can cause a CPU bottleneck. A cache holds frequently accessed data and instructions in "local" memory that is faster for the process to access than main memory. A cache miss occurs when the processor needs to reference memory, and a copy of the memory is not stored in cache. Because it is very expensive for the processor to access main memory, frequent cache misses will slow down application performance considerably.

The Translation Lookaside Buffer is used to map physical memory to virtual memory. It contains translations for recently addressed virtual pages. A TLB miss occurs if your application tries to access a page of virtual memory that has not been mapped to physical memory.

Both cache misses and TLB misses can be avoided by improving locality in your application. For example, loops can be written to sequentially access contiguous memory addresses, as opposed to accessing data that is scattered throughout memory. Code routines that frequently call each other should be included in the same source files, or the files containing those routines should be listed next to each other in the `ld` command. You can also use the `ld(1)` options for profile-based optimization to reposition code so that better locality is achieved. Note that profile-based optimization will not improve the locality of data references. Profile-based optimization is described in Optimizing HP C Programs.

Memory Bottlenecks

If memory is the bottleneck and your program is thrashing (that is, pages of virtual memory are excessively swapped into physical memory), you may be able to improve application performance just by increasing the amount of physical memory in your system. You can also tune your application's memory usage. To do this, consider the following, all of which will require some code changes in your application:

- Improve locality within your program. Frequently accessed items that are relatively small and reside in different pages will increase the size of memory needed to swap in pages containing those items.
- Reduce heap fragmentation. Fragmentation occurs when memory is allocated and freed in patterns that leave unused holes in the heap.
- Eliminate memory leaks. Memory leaks occur when an allocated piece of memory is no longer needed but is not freed. Several high-quality commercial products are available on Hewlett Packard systems to help you identify and eliminate memory leaks within your program. You should be able to get information about commercial tools from your sales representative.
- Reduce the size of code and data structures. For example, if a structure contains 32-bit values for each of several boolean values, consider using a group of 1-bit fields instead. Infrequently used code and data can be separated from frequently used code and data.
- Re-use memory. You can consider using buffers that are allocated once to store temporary items, instead of allocating and freeing memory at different times throughout the execution of your program. This will help reduce fragmentation of the heap, and avoid calls to `malloc` and `free`, which are expensive procedures.
- Consider using primitives that reuse data, like `PEXTriangleStrip` or `PEXSetOfFillAreaSets`. `PEXSetOfFillAreaSets` uses a single "database" of vertices. A set of connectivity lists describe how to connect those vertices to make a polygon.

If Disk Access is the Bottleneck

If the problem is disk access, you might consider modifying your program to access the disk more efficiently. For example, make some tradeoffs between memory usage and disk usage. Blocks of frequently used data could be read in all at once and stored in memory, instead of accessing the disk each time an item is accessed.

Summary

Good graphics application performance depends on a number of factors, including the raw performance capabilities of the graphics hardware and software used by the application; the efficient, balanced use of system resources; and calling sequences that use the most optimized paths through the graphics libraries. Each application is different. No single set of rules will provide optimal performance for a specific application. Good design for efficient use of system resources, an under standing of the performance-critical tasks, and some amount of experimentation are the keys to achieving the best graphics application performance possible.

6

Writing HP PEXlib Programs

Introduction

This chapter provides you with recommendations and specific details of the HP PEX implementation that affect how you will write your programs.

As a PEXlib programmer, if it is among your objectives to write programs that are portable and interoperable on a variety of workstations and graphics devices—you will want to do so without sacrificing performance. To succeed at this, you may want, for example, to add PEXlib inquiries to your program that will enable it to determine which protocol version is supported by the server, attribute values that are supported, the visuals that you may use with PEX, as well as other details of implementation such as the number of supported line widths.

The information and examples that you need to accomplish this is described in this and following chapters in this book as well as a companion publication, *Portable Programming with CGE PEX 5.1*. (More information and examples of specific PEXlib inquiries are provided for you in Chapter 24, "Determining a Server's Features" in *PEXlib Programming Manual*.)

The *Portable Programming with CGE PEX 5.1* was written specifically to assist you in creating highly portable 3D graphics applications on platforms supporting the Common Graphics Environment, including HP. It contains tools and utilities that you can use to develop portable and interoperable applications more easily.

Determining A Server's Features

You can learn quite a lot about your particular system, server, and the supported X and PEX extensions with the `xdpyinfo` command. On Hewlett-Packard workstations this command is in the directory[1] ⟨*contrib*⟩`/bin/X11`. See sample output in Appendix A.

An HP PEX example program, `pexdpyinfo`, displays detailed information on the PEX extensions on your work station; information such as the enumerated types, implementation-dependent constants, lookup table entries, and supported PEX visuals for a particular display. See sample output in "`pexdpyinfo`" in Appendix A.

In order to use this utility, you'll first need to use the `Makefile` to build the executable. See the `README` file in ⟨*hp-examples*⟩ for instructions. See the file `pexdpyinfo.spec` for usage details. The file `pexdpyinfo.design` contains design information and `pexdpyinfo.c`, the source code, for developers interested in extending the capabilities of the utility.

PEX Extension Information

As described in the *PEXlib Programming Manual*, Chapter 24, the extension information returned from `PEXInitialize` and `PEXGetExtensionInfo` verifies that a PEX extension exists within the PEX server in order to set PEXlib variables and establish communication with the server.

HP PEX supports the Immediate Mode subset. HP PEX also supports the Structure Mode rendering subset of PEX functionality with all but the Search Context requests. This means that calls to `SearchContext` entrypoints (except to issue protocol to another server) will report a `BadImplementation` error. Calls to `Workstation` or `PickMeasure` entrypoints will report a `BadRequest` error. The `Workstation` and `PickMeasure` entry points do not emit protocol. Search context request protocol is generated and can be sent to servers that do support those requests. Errors are detected and reported by the client.

Your program can determine which subsets are supported by a PEX server in an interoperable way with `PEXGetExtensionInfo`:

```
   PEXExtensionInfo *PEXGetExtensionInfo(Display *display)
```

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

This also tells what version of protocol is supported by the PEX server as well as other information that can be used for error handling.

PEXlib functions that have a display parameter are not allowed to be called before calling `PEXInitialize`. If the application calls a PEXlib function that is not allowed to be called before `PEXInitialize` is called, HP PEXlib will ignore the call.

The following table shows the HP-specific error codes that can be returned from the `PEXInitialize` function. We recommend that you print the error string which is returned in your program for more information needed to determine the cause of the error.

Table 6-1. `PEXInitialize` Error Codes

Return Value	Standard PEX Error Strings
4	`PEXlib client-side memory allocation failed during initialization`
3	`The PEX extension does not support a compatible floating-point format`
2	`The PEX extension does not support a compatible protocol version`
1	`The PEX extension does not exist or could not be initialized`
0	`successful initialization`

Table 6-2. `PEXInitialize` Error Codes

Return Value	HP-Specific Error Strings
−1	*(The error string returned here varies according to the error condition. Print the error string which is returned in your program to learn about the actual error condition.)*
−2	`The maximum number of displays have already been initialized`
−3	`Something is wrong with the X display name`
−4	*(The −4 error string is not used)*
−5	`Attempt to initialize a DHA connection failed`
−6	`Attempt to initialize a PEX connection failed`
−7	`Attempt to initialize a X connection failed`
−8	`Something is wrong with the X11R6 libraries` (application may be linked with pre-X11R6 libraries)

The PEX extension information pointer returned from `PEXInitialize` will be valid only if `PEXInitialize` succeeds or the error `PEXBadProtocolVersion` is generated. Any other error that occurs in `PEXInitialize` will cause the PEX extension information pointer to be `NULL`.

Enumerated Types

Enumerated types define values used to define attributes such as marker types (dot, asterisk, circle, or "×"). Other examples include line type, color type, and interior style. Because a PEX server may not support all enumerated types, PEX provides the `PEXGetEnumTypeInfo` information inquiry for determining which enumerated types are supported by the particular server.

The table below lists the enumerated types supported by HP PEX on supported graphics devices. For writing interoperable programs it is best to inquire which values are supported by using the `PEXGetEnumTypeInfo` inquiry rather than relying on documentation from vendors of vendor-specific support.

```
PEXGetEnumTypeInfo(
    Display             *display,
    Drawable            drawable,
    unsigned long       count,
    int                 *enum_types,
    unsigned long       item_mask,
    unsigned long       **info_count_return,
    PEXEnumTypeDesc     **enum_info_return)
```

Table 6-3. Enumerated Type Inquiry Parameters

Value	Description
`display`	A valid display pointer.
`drawable`	An example drawable indicating the screen and depth of the window for which the values will be used.
`count`	The number of enumerated types.
`enum_types`	A list of enumerated types for which information is to be returned.
`item_mask`	A mask indicating the data to be returned for each type.
`info_count_return`	Returns an array of counts. For each enumerated type, there is an entry specifying the number of descriptors in the return value array.
`enum_info_return`	Returns an array of enumerated type descriptors containing the enumerated type information.

For example, you may use the enumerated type descriptors to inquire which double-buffering escapes are supported. This information is covered earlier in this chapter in "Inquiring Supported Escapes".

Enumerated Types List

Information about HP-supported enumerated types is shown here. There are standard PEXlib inquiries which return HP-supported enumerated types.

CGE PEX 1.0 also defines additional enumerated types to list extension features for which PEXlib 5.1 does not define a mechanism of inquiry. For example, `PEXExtETOC` can be inquired to list extension OCs (Output Commands) beyond the PEXlib standard. See the `PEXExt.h` and `PEXHPlib.h` files for lists of extension enumerated types.

Table 6-4. Enumerated Types

Enumerated Type	HP PEX 5.1v4

`MarkerType`	All PEX types, plus the following:
	`PEXHPMarkerTriangle`
	`PEXHPMarkerSquare`
	`PEXHPMarkerDiamond`
	`PEXHPMarkerCrossSquare`

`PEXATextStyle`	`PEXATextNotConnected` `PEXATextConnected`
`InteriorStyle`	All styles but `Pattern`
`HatchStyle`	45 degrees, 135 degrees, plus CGE and HP types
`LineType`	All PEX, CGE, and HP types
`SurfaceEdgeType`	All PEX types
`PickDeviceType`	`DCHitbox`, `NPCHitVolume`
`PolylineInterpMethod`	`None`, `Color` (only with PowerShade)

`CurveApproxMethod`	Implementation-dependent (`AdaptiveDC`)
	`WCSRelative`
	`NPCRelative`
	`DCRelative`

`ReflectionModel`	All PEX types
`SurfaceInterpMethod`	`None`, `Color` (only with PowerShade)

`SurfaceApproxMethod`	Implementation-dependent (`AdaptiveDC`)
	`WCSRelative`
	`NPCRelative`
	`DCRelative`

`TrimCurveApproxMethod`	Implementation-dependent (adapt to surface criteria)
`ModelClipOperator`	All PEX types
`LightType`	All PEX types

Table 6-4. Enumerated Types (continued)

Enumerated Type	HP PEX 5.1v4
`ColorType`	`Index`, `RGBFloat`, `HPRGBA`
`FloatFormat`	`IEEE_754_32`
`HLHSRMode`	`Off`,`PEXHPHLHSRZBuffer` (only with PowerShade); `PEXHPHLHSRZBufferID` (only with PowerShade); `PEXHPHLHSRZBufferReadOnly` (only with PowerShade); `PEXHPHLHSRZBufferIDReadOnly` (only with PowerShade)
`PromptEchoType`	Not Applicable
`DisplayUpdateMethod`	Not Applicable
`ColorApproxType`	`PEXColorSpace` `PEXColorRange` `PEXHPColorApproxTypeIndexed`
`ColorApproxModel`	RGB
`GDP`	No supported GDPs
`GDP3`	No supported GDP3s
`GSE`	`HP_GSE_SET_ANTIALIAS_MODE`
`RenderingColorModel`	RGB
`ParametricSurface-Characteristics`	None (default); Implementation-dependent (interior edging)
`PickOneMethod`	`PEXPickLast`
`PickAllMethod`	`PEXPickAllAll`

Table 6-4. Enumerated Types (continued)

Enumerated Type	HP PEX 5.1v4
Escape	ES_ESCAPE_DBLBUFFER ES_ESCAPE_ET_DBLBUFFER ES_ESCAPE_ET_SWAPBUFFER ES_ESCAPE_ET_SWAPBUFFERCONTENT ES_ESCAPE_SWAPBUFFER ES_ESCAPE_SWAPBUFFERCONTENT HP_ESCAPE_DFRONT HP_ESCAPE_ET_DFRONT HP_ESCAPE_ET_SET_GAMMA_CORRECTION HP_ESCAPE_SET_GAMMA_CORRECTION PEXEscapeQueryColorApprox PEXEscapeSetEchoColor PEXExtEscapeChangePipelineContext PEXExtEscapeChangeRenderer PEXExtEscapeCreateTM PEXExtEscapeCreateTMDescription PEXExtEscapeCreateTMFromResources PEXExtEscapeFetchElements PEXExtEscapeFreeTM PEXExtEscapeFreeTMDescription PEXExtEscapeGetPipelineContext PEXExtEscapeGetRendererAttributes PEXExtEscapeGetTableEntries PEXExtEscapeGetTableEntry PEXExtEscapeOpcodeChangePipelineContext PEXExtEscapeOpcodeChangeRenderer PEXExtEscapeOpcodeCreateTM PEXExtEscapeOpcodeCreateTMDescription PEXExtEscapeOpcodeCreateTMFromResources PEXExtEscapeOpcodeFetchElements PEXExtEscapeOpcodeFreeTM PEXExtEscapeOpcodeFreeTMDescription PEXExtEscapeOpcodeGetPipelineContext PEXExtEscapeOpcodeGetRendererAttributes PEXExtEscapeOpcodeGetTableEntries PEXExtEscapeOpcodeGetTableEntry PEXExtEscapeOpcodeQueryColorApprox

Table 6-4. Enumerated Types (continued)

Enumerated Type	HP PEX 5.1v4
Escape (continued)	PEXExtEscapeOpcodeSetTableEntries PEXExtEscapeQueryColorApprox PEXExtEscapeSetTableEntries PEXHPEscapeOpcodeStereoMode PEXHPEscapeChangePipelineContext PEXHPEscapeChangeRenderer PEXHPEscapeDfront PEXHPEscapeGetPipelineContext PEXHPEscapeGetRendererAttributes PEXHPEscapeGetZBuffer PEXHPEscapeOpcodeChangePipelineContext PEXHPEscapeOpcodeChangeRenderer PEXHPEscapeOpcodeDfront PEXHPEscapeOpcodeGetPipelineContext PEXHPEscapeOpcodeGetRendererAttributes PEXHPEscapeOpcodeGetZBuffer PEXHPEscapeOpcodePutZBuffer PEXHPEscapeOpcodeSetGammaCorrection PEXHPEscapeOpcodeSetZBuffer PEXHPEscapePutZBuffer PEXHPEscapeSetGammaCorrection PEXHPEscapeSetZBuffer PEXHPEscapeEVEInformation

Implementation-Dependent Constants

There are other PEX values in addition to enumerated types that vary among implementations, as allowed by the PEX standard. Implementation-dependent constants define things like the maximum value of a name or the number of line widths. The table "Implementation-Dependent Constants" below lists the constants that are supported in HP PEX.

As with other implementation-dependent features, it is best to inquire about supported implementation-dependent constants using the standard PEXlib inquiry, `PEXGetImpDepConstants`.

```
PEXGetImpDepConstants(
    Display             *display,
    Drawable            drawable,
    unsigned long       count,
    unsigned short      *names,
    PEXImpDepConstant   **constants_return)
```

Table 6-5.
Implementation Dependent Constants Inquiry Parameters

Value	Description
`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`drawable`	The resource identifier of a drawable.
`count`	The number of implementation-dependent constants.
`names`	An array of names of implementation-dependent constants to be returned.
`constants_return`	Returns an array of implementation-dependent constants.

This table lists some of the implementation-dependent constants that are supported in HP PEX. For an exhaustive list, see the include files `PEX.h`, `PEXExt.h`, `PEXlib.h`, and `PEXHPlib.h`.

Table 6-6. Implementation-Dependent Constants

Implementation-Dependent Constant	**HP PEX 5.1v4**
`NumSupportedLineWidths`	No limit
`MinLineWidth`	1
`MaxLineWidth`	16383
`NominalLineWidth`	1
`NumSupportedEdgeWidth`	1
`MinEdgeWidth`	1
`MaxEdgeWidth`	1
`NominalEdgeWidth`	1
`NumSupportedMarkerSizes`	No limit
`MinMarkerSize`	1
`MaxMarkerSize`	No limit
`NominalMarkerSize`	3
CIELUV values (approximate, true values are monitor-dependent)	`red.u` = 0.450 `red.v` = 0.522 `red.l` = 1.0 `green.u` = 0.120 `green.v` = 0.561 `green.l` = 1.0 `blue.u` = 0.175 `blue.v` = 0.157 `blue.l` = 1.0 `white.u` = 0.188 `white.v` = 0.466 `white.l` = 1.0

Table 6-6. Implementation-Dependent Constants (continued)

Implementation-Dependent Constant	HP PEX 5.1v4
`MaxNameSetNames`	`MAXINT`
`MaxModelClipPlanes`	With PowerShade, 6; without, `False`
`TransparencySupported`	With PowerShade, `True` without, `False`
`DitheringSupported`	`True` or `False`
`MaxNonAmbientLights`	With PowerShade, 15;
`MaxNurbOrder`	6
`MaxTrimCurveOrder`	6
`BestColorApproxValues`	0 (`PEXColorApproxAnyValues`)
`DoubleBufferingSupported`	`True`
`PEXHPIDDeformationSupported`	`True` or `False`
`PEXHPIDCappingPlanesSupported`	`True` or `False`
`PEXHPIDInterferenceSupported`	`True` or `False`
`PEXHPIDPolygonOffsetSupported`	`True` or `False`

PEX Extensions

Generalized Structure Elements (GSEs)

HP PEX supports one GSE, used to enable or disable line and edge antialiasing (see "Line Types"). This is not a standard feature (there are no standard GSEs specified by PEX), but may have value for your application. The constants and data structure for the PEXlib interface are defined in the header file `PEXHPlib.h`.

Escapes

Extensions to PEX are provided by HP for capabilities beyond the standard. These are implemented, according to provisions in the PEX standard, in a way that makes for a common interface to the extensions which does not negatively affect the portability of applications.

The Evans & Sutherland escape requests are described in the section "Animation", later in this chapter. The section also shows an example of the syntax and return information. Other escapes are documented in other relevant sections of this manuals.

PEX Subset Lists

Immediate Mode Subset

HP PEX supports the Immediate Mode subset.

Structure Subset

HP PEX also supports the Structure Mode rendering subset of PEX functionality with all but the Search Context requests.

Search Context Requests

These functions emit protocol but the HP PEX server does not process them and will generate an error upon receiving them.

`PEXChangeSearchContext`	`PEXFreeSearchContext`
`PEXCopySearchContext`	`PEXGetSearchContext`
`PEXCreateSearchContext`	`PEXSearchNetwork`

PHIGS Workstation Resources

HP PEX does not support the PHIGS Workstation subset because it is expected to be removed from PEX at a future version. These functions will not emit protocol and will generate a `BadRequest`.

This table lists the PEX functions in subsets that HP PEX does not support as allowed by the standard.

Table 6-7. Unsupported Subset Entrypoints

`PEXCreateWorkstation`	`PEXSetWorkstationHLHSRMode`
`PEXExecuteDeferredActions`	`PEXSetWorkstationViewport`
`PEXFreeWorkstation`	`PEXSetWorkstationViewPriority`
`PEXFreeWorkstationInfo`	`PEXSetWorkstationViewRep`
`PEXGetWorkstationDynamics`	`PEXSetWorkstationWindow`
`PEXGetWorkstationInfo`	`PEXUnpostAllStructures`
`PEXGetWorkstationPostings`	`PEXUnpostStructure`
`PEXGetWorkstationViewRep`	`PEXUpdateWorkstation`
`PEXMapDCToWC`	`PEXSetPWAttributeMask`
`PEXMapWDToDC`	`PEXGetPickDevice`
`PEXPostStructure`	`PEXChangePickDevice`
`PEXRedrawAllStructures`	`PEXCreatePickMeasure`
`PEXRedrawClipRegion`	`PEXFreePickMeasure`
`PEXSetWorkstationBufferMode`	`PEXGetPickMeasure`
`PEXSetWorkstationDisplayUpdateMode`	`PEXUpdatePickMeasure`

HP Implementation Details for Writing Programs

Supported PEX Subsets

The O'Reilly *PEXlib Programming Manual*, Chapter 3, "Getting Started," is a good starting point for learning to use features of PEXlib as well as inquiring PEX extension information, enumerated types, and specific implementation-dependent constants. You'll also need to learn HP implementation details that are covered in this and the following chapters.

Resource Sharing

Because X resources are global in the server, X window applications have been able to share resources between processes by passing the resource ID through some inter-process communications mechanism. HP PEX does not support sharing of PEX resources.

Synchronization

All the normal requirements for achieving proper ordering of rendering and windowing operations in X programs also apply to PEXlib programs. In particular, note that some kinds of requests are re-routed to the window manager (for example, `XConfigureWindow`). For these requests, it is good practice to wait for the proper type of X event to confirm that the operation has been completed before rendering further in the window.

HP PEXlib Programming

The remaining sections of this chapter are arranged to supplement the information in the same order that it is presented in the O'Reilly *PEXlib Programming Manual*. The sections are in order, beginning with the following section, "Color".

Color

This section begins with a general discussion, a quick primer, on the basics of PEX color support, and follows with a more detailed discussion of the specifics of the HP implementation. A companion publication, *Portable Programming with CGE PEX 5.1*, covers issues and general interoperability programming recommendations.

The application of color in the X Window System and interactions with PEX are often complex so that an understanding of background information makes it easier to accomplish color portability among a wide variety of X servers and display devices. In some areas, conventions for interoperable X color programming have already been developed, particularly in the area of allocating colormap cells.

If you're not familiar with X visuals, colormaps, and other basics, a good place to begin is the book *Xlib Programming Manual*, from O'Reilly. In addition, you'll find good program examples to guide you in dealing with colormaps in the examples directories under ⟨*pex*⟩[2]. Various utilities that help you deal with color support are also available and are explained later in this chapter, as well as in the *Portable Programming with CGE PEX 5.1*.

PEX Color Support Basics—Four Steps

PEX itself does not create any *new* issues in managing X colors. It requires the same basic series of steps that X applications require:

- Choose a visual in which the window will be created.
- Create a colormap or find one to share with other similar clients in that visual.
- Load colors into the colormap.
- Create a window in the chosen visual with the proper colormap.

[2] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Color is an important element in shaded images such as those typically drawn via PEX. A PEX application may be more "demanding" in each of these steps than a simple X application. The colormap that is used by your HP PEXlib program must be consistent with the color approximation specified in your program. The colormap's interpolation ramp and the color approximation describing that ramp are limited by the visual selected, which in turn are limited by individual vendors or graphics devices. PEX may not be supported in all visuals.

With this in mind, let's take a closer look at these steps.

1: Choose a Visual in which the Window Will be Created

Many simple X applications don't take explicit actions to evaluate the color capabilities of a visual for this step. That's because they simply use the default visual for the X server. The default visual is often a `PseudoColor` visual of eight or fewer planes and is occasionally located in the overlay planes if the device has them.

A PEX application, on the other hand, is often more sophisticated about visual selection. PEX rendering may not be supported in the default visual for the server, either because the default visual is not capable of the kinds of color ranges that PEX images require, or because the graphics rendering pipeline needed for PEX cannot support that visual. It may also be because it has other limitations that make using it difficult for PEX. The bottom line is, an application should not simply create a window in the default visual and expect PEX to successfully create a renderer and draw in it. Instead, the application should select a visual type based on application needs and one which PEX supports.

You'll find that the *PEXlib Programming Manual* features a utility procedure that chooses a PEX-capable visual, `ora_find_best_visual()`. Since PEX usually requires a lot of color cells in order to display a shaded image well, this utility searches for the visual with the most colors. It also prefers read-only visuals such as `TrueColor`.

While this procedure can be used on many servers, it may not work on some others. Your application may need to use a more involved method to choose a PEX visual. For example, if your application needs to use double-buffering, it must choose a visual that supports that operation.

It is wise to use `PEXMatchRenderingTargets` when using a PEX 5.1 server, because this routine lets the application know positively whether or not a visual is supported by PEX.

Ready-made solutions to the visual selection problem and other problems are available in the utilities directories under ⟨*pex*⟩[3]. They contain a number of utilities to help programs select visuals as well as resolve colormap issues, and create windows for HP workstations. One such utility of interest here is `PEXUtSelectVisual`. Programmers are encouraged to use the `cge_utilities` in order to advance portability and interoperability.

2: Determine Use of Transparent Overlay Planes

In the *PEXlib Programming Manual*, 5.7, "Color Approximation," mention is made of "window memory," or "frame buffer," and its role in rendering and making pixels on the display take on desired colors. In Hewlett-Packard terms, all the rows and columns of the frame buffer array are mapped directly onto the rows and columns of pixels on the display. And as you'd expect, the display refreshes images by scanning through the frame buffer, line by line, to re-display the image.

To display color and intensity on the screen, each place in the frame buffer must have more than one bit. This is described in literature from Hewlett-Packard as the depth of the frame buffer. (Also see *PEXlib Programming Manual*, 5.7.1.1, for its discussion of the number of bits in the pixel segments determining the number of colors displayed.) Frame buffer depth, of course, is determined by the hardware or software with which it can be implemented.

Simple monochrome systems only require the frame buffer to record whether or not individual pixels are on or off, so the frame buffer has a depth of one. A simple color system will provide at least four bits to describe each pixel and is termed a 4-plane system. Most graphics systems provide eight-bit pixels, termed 8-plane systems, and so on, up to 12-, 24-, and 48-plane systems.

To this system of planes devoted to the color approximation and display of images, Hewlett-Packard adds a capability for planes that "overlay" the image planes. Rendering to the overlay plane visuals generally does not affect the contents of the image plane visuals that appear underneath. This makes it possible to render

[3] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

text or other graphics to the overlay window without re-rendering the image window graphics. This is especially useful for user-interface objects (menus and such) or annotation text that “floats” over image-plane objects. These overlay planes provide lesser functionality than the image planes.

The overlay planes’ transparency feature enables you to render opaque objects (for example, menus and text) to a transparent overlay plane and at the same time, view rendered objects in the image planes. For example, you may want to show a map of the United States without all of its state boundaries, and then add the state boundaries as you need them. This can be done by creating two X windows: one in the overlay planes and one in the images planes. The United States map would be draw in the image planes window and the state boundaries in a transparent overlay planes window.

The overlay and image planes, then, are accessed by using different visuals to create X windows and colormaps.

3: Create a Colormap or Find One to Share With Other Similar Clients in that Visual

Once you’ve chosen a visual, it may be necessary to find or create an X colormap for it before any windows can be created in that visual. If the chosen visual is not the default, this is necessary be cause the X server often creates only a colormap for the default visual.

Even in the default visual, it can be beneficial for an application to share a colormap rather than create one of its own. This will, for example, reduce color flashing—the distracting effect of displayed objects changing colors as colormaps with different contents are installed. (Colormap installation is typically under control of a window manager, and is often triggered by moving the pointer from one window to another.) If multiple windows can be rendered using the same color map, the chances of this flashing can be reduced.

Convenience and efficiency are other reasons to share a colormap. If your application will create four windows and put similar information in all of them, why create and set up four colormaps? This is time-consuming and wastes server resources. On the other hand, your application may actually need different colormaps for the four windows—but be ready for the color flashing.

These same principles apply to concurrent PEX clients. In fact, it is likely that all PEX clients coexisting in a particular visual can be satisfied sharing the same colormap (see the next item in this list that discusses the colormap contents). Even if each PEX client has its own colormap resource, it is likely that the colormap contents will be very similar, and the severity of color flashing among them will be much reduced.

X standard colormap properties are elements of an established convention that allows description and sharing between X clients of the kind of colormaps that are appropriate for PEX. The book *Xlib Programming Manual* offers information on this convention.

4: Load Colors into the Colormap

Once a colormap is found or created, it may be necessary to load colors into the cells. Typically, if a colormap is being shared, the contents of most of the cells have already been agreed upon. The convention in sharing is the first client that needs a standard colormap may set it up. By putting the resource ID into the standard colormap property, it makes the colormap available for sharing. Later clients simply use it without changing it.

PEX never deals directly with the contents of the colormap, it only generates pixel values as directed by the color approximation table entry that is in effect during rendering. To get a correct image on the screen, the colormap must be set up to match the color approximation table entry. Color approximation is the "translation step" between the RGBs (or other colors) that are the output of the PEX rendering pipeline, and the X colormap attached to the window. Color approximation, and the corresponding setup of the colormap, are the most critical elements in achieving correct appearance of PEX rendering.

Most PEX applications will find that color approximation type `PEXColorSpace` is the most natural method to use. This is because `PEXColorSpace` attempts to reproduce the RGB values produced by the PEX rendering pipeline as faithfully as it can, given a limited number of color levels for each of red, green, and blue. In other words, the colors that are displayed are as true to the colors of the primitives, lights, etc., as possible.

The colors in the X colormap for `PEXColorSpace` are expected to represent a "sampling" of the color space, also called a **color ramp**. For a set number of red levels, green levels, and blue levels, all the combinations are expected to be represented in the colormap, either by separate cells (in the case of `PseudoColor` visuals) or by combinations of red, green, and blue components (in the case of

`DirectColor` and `TrueColor` visuals). Again, the ⟨*cge_utils*⟩ directory[4] provides utilities that aid in setting up a colormap to contain a color sampling.

Ideally, every PEX implementation could support any color approximation setup that is "legal." However, this is not likely to be the case and even if it were, rendering performance would be impaired for setups that were unnatural to the device. You should expect that each implementation will only support a small set of possible color approximation entries well—so your application must choose one of those. The HP server implements an escape that can be used to decide whether or not a color approximation entry is supported. This escape is part of a recently established convention for PEX color interoperability. Other PEX servers will also be implementing this escape; it is discussed in more detail later in this chapter "Color Approximation—Utilities And Escapes".

`PEXColorRange` is used primarily for applications using the RGB channels in the PEX rendering pipeline to represent data other than color. For example, a finite-element analysis might use the red channel to represent temperature, and the green and blue channels to represent other data. `PEXColorRange` gives the application a way to apply a simple function to the data (after the pipeline may have interpolated it across surfaces) to convert it to colors that represent the data combinations.

There is one other interesting application of `PEXColorRange`. On a `GrayScale` or `StaticGray` visual, many colormap cells can be saved by using `PEXColorRange` to convert the RGB coming from the rendering pipeline to a gray intensity level before the lookup in the X colormap.

One more step may be needed in order to load the colormap correctly for PEX rendering. Some graphics hardware or software may not be able to render to a colormap setup conforming to the PEX color approximation scheme, perhaps because the hardware or driver software was developed before the advent of PEX. Rather than not supporting PEX on such hardware, the vendor may furnish instructions (or a utility routine) to adjust the colormap contents to the behavior of the hard ware rendering.

[4] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

5: Create a Window in the Chosen Visual, with the Colormap

PEX adds no new tasks to this step. Assuming the visual that has been chosen is supported by PEX, and the colormap has been set up to match the color approximation, your application can create the window normally, and should then be able to create Lookup Tables and Renderers using the window as an example drawable, and/or bind a Renderer to the window for drawing or picking.

PEX Color Support Basics—Portability and Interoperability

A basic PEXlib programming goal is to enable your application to run, without change, across all the X platforms from one vendor. It may also be a goal that your application port, without significant change, to PEXlib from other vendors, and/or to run via PEX protocol to servers from other vendors. PEX and PEXlib are intended to support these similar and valuable objectives.

When writing the part of your application dealing with X and PEX color issues, you are dealing in one area where these device-specific and vendor-specific capabilities and issues are exposed. Therefore, it is important to anticipate the wide range of color capabilities your application may encounter when operating across the network or during a port to another vendor's PEX implementation.

For example, not all workstations support all the visual classes. This means that on some servers, there may be several PEX-capable visuals to choose from and you must write your application so that it can find one that exactly meets its color requirements. However, there are often advantages in choosing a visual other than the default, if you can. Advantages include these:

- Non-default visuals may access separate hardware colormap resources, which may allow both your application and other concurrently operating clients to "look correct" at the same time.
- Because your application's color needs may not match those of most of the simple concurrent X clients, choosing a visual other than the default means your application will not interfere with or compete against other X clients for colormap cells.

On other servers, there may only be one relatively limited visual. In such a situation, your application has two choices:

- It may be able to operate successfully with a limited range of colors that fit into a colormap that can be shared.
- It may need to create a separate colormap and you (and your end user) will have to tolerate the color flashing if there are insufficient hardware colormaps to keep them all installed.

Some servers may have several hardware colormaps and color flashing may not be much of a problem when running on those servers. If you have not incorporated any code to attempt to share colormaps with other clients, and then you run the application on a server that has only one hardware colormap resource, the flashing could be severe. (The flashing could be severe anyway; it can only be avoided when *all* the clients that are running concurrently cooperate in the sharing of limited color resources.)

The implementation limitations on color-approximation support mentioned earlier are another example of the kind of configuration dependency that must be dealt with in order to make an application truly interoperable and portable.

Programming for this kind of portability and interoperability requires some decided effort. You should note, however, that with X and PEX, the same solutions that give portability across a single vendor's platforms can also contribute to portability and interoperability across vendors and networks—so time spent on design for interoperability is seldom wasted.

PEX Color Support Basics—One Last Note ...

Although at this time there are no completely-established interoperability conventions specific to PEX color issues, these conventions are being developed through the efforts of software application developers via the PEX Interoperability Consortium. Staying up to date on these conventions, and even participating in their development, is of value to you as a PEXlib programmer.

Color Support in HP PEX

Now, apply these basic strategies (that your application might use for selecting a visual and setting up a compatible color approximation table and colormap contents) to Hewlett-Packard graphics workstations:

- If you definitely need to coexist in the same visual with many other X clients (this applies to low-end devices that have only one visual), you can use the `DefaultVisual` and `DefaultColormap` macros (defined in `Xlib.h`) to acquire the visual information and colormap ID. This is the simplest method, but is not recommended for high-end systems because it does not use the full color capabilities of the graphics device.

  HP PEX is capable of rendering to the default colormap on the supported low-end devices, but only if the `SB_X_SHARED_CMAP` variable is set when the X/PEX server is started. See Chapter 3 for more information on this and other environment variables.
- A more sophisticated method of visual selection, one that shares the colormap with other clients in order to avoid color flashing, is to have your application search for standard colormap properties on the server and use one of the visuals described there. In order to share the colormap, first check to see if the ID for a colormap is already present in the property. If so, use it without modifying its contents. If there is no ID in the property (its value is "`None`"), then create a colormap and initialize it according to the description in the property.

  Note that conventions regarding the use of colormap properties in conjunction with PEX are still being developed and that not all servers create properties. Therefore, we recommend that your application should have an alternative strategy in case properties are not present.

Utilities To Help You Deal With Color

HP PEX helps you resolve PEX color issues through the definition of X standard colormap properties, predefined color-approximation LUT entries that are appropriate for the HP graphics devices on which PEX programs run, and a complete set of utilities for selecting visuals, creating and loading colormaps and creating windows in the visuals.

These utilities are in one of the utilities directories under the $\langle pex \rangle$ directories[5].

Instructions for using the utilities are documented in `README` files in the same directories.

Choosing A Visual In Which the Window Will Be Created

Several of the devices supported by HP PEX have only a single visual; several others have two or more visuals that are PEX-capable. The visuals range from 8-bit `PseudoColor` all the way up to 24-plane `DirectColor`. Because interactivity is important to most PEX applications, HP PEX largely supports visuals that are capable of double-buffering (see the appropriate section of this manual for more information on double-buffering).

In the table "Visual Types Capable of Multi-Buffering," you see a list of supported visuals on HP PEX graphics devices and whether or not double-buffering is supported. For maximum application portability, it is best to use the `PEXMatchRenderingTargets` call and the `PEXGetEnumTypeInfo` call to determine whether or not the visual of interest to your application is supported on HP.

Many programs have similar requirements in choosing a visual. The HP PEX server defines a standard X colormap property, `PEX_BEST_MAP`, that defines what HP considers to be the "best" color approximation on each visual on a particular device for the widest range of applications.

The first entry in the property describes the visual that provides the most color capabilities but can still be double-buffered. If these are your criteria for selecting a visual in your application, you can fetch the value of the property using `XGetWindowProperty` (an Xlib entry point) and use the visual named in the first entry.

[5] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

To Get A Colormap That Supports Transparency

If you need an overlay colormap that supports transparency, create the color map using the visual that includes transparency in its `SERVER_OVERLAY_VISUALS` property. If this property exists on the root window of a particular screen, that screen has overlay planes. This property also allows a program be able to determine which visuals are in the overlay planes and which are not, and to access the transparency information to determine if there is a "transparent color" which can be used to "see through" the overlay planes to the image planes.

This property is accessed by the visual-selection utility `PEXUtSelectVisual` in ⟨*cge_utils*⟩[6], then the layer criterion (image or overlay) is applied as a selection factor.

Creating a Colormap or Finding One to Share

HP PEX changes nothing in the process of creating a colormap in X—colormaps are created using the `XCreateColormap` call. `XCreateColormap` returns a resource ID that can be used for the new colormap.

Shareable colormaps can be found using the colormap properties. If the colormap ID in the `XStandardColormap` structure returned by `XGetRGBColormaps` or `XGetWindowProperty` is "`None`", (a constant defined by Xlib), you'll need to create a colormap with `XCreateColormap`. This ID can be put into a standard colormap property if you wish to share the resource with other clients.

[6] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

Color Approximation—Utilities And Escapes

How can your application determine the particular color sampling that HP PEX supports? As discussed in Chapter 25 of the *PEXlib Programming Manual*, escapes provide for features not defined by the standard PEX specification. In this case, the escape `PEXEscapeQueryColorApprox` enables applications to inquire whether or not a particular color approximation entry is supported by the PEX server and its (or that of the very similar CGE extension escape `PEXExtQueryColorApprox`) use is recommended to assure portability.

To illustrate a basic color approximation inquiry, see the file `pexutcmap.c` in the ⟨*pex-utils*⟩ directory. A basic color-approximation inquiry is implemented by `PEXUtVerifyColorApproximation`.

6

The syntax for PEXEscapeWithReply is:

```
char* PEXEscapeWithReply(
  Display              *display,
  unsigned long        escape_id,
  int                  length,
  char                 *escape_data,
  unsigned long        *reply_length_return)
```

Table 6-8. PEX Escape With Reply Parameters

Value	Description
display	A pointer to a display structure returned by a successful XOpenDisplay call.
escape_id	Set to PEXEscapeQueryColorApprox.
length	The length, in bytes, of data for the escape request.
*escape_data	Set to the address of a structure of type PEXEscapeQueryColorApproxData
*reply_length_return	Returns the length, in bytes, of the reply data.
return	Interpret the return value to be a block of storage beginning with a structure of type PEXEscapeQueryColorApproxReplyData, followed by zero or more structures of type PEXColorApproxEntry.

As with all PEX escapes, the technique which most contributes to interoperability is to inquire whether or not the PEX server supports an escape_id before attempting to use it by calling PEXGetEnumTypeInfo for the PEXETEscape enumeration. The values returned for index and mnemonic fields for this escape are PEX-ETEscapeQueryColorApproxData and PEXETMEscapeQueryColorApproxData respectively. If you send an escape_id to a server that does not support it, a BadValue error is reported.

The input data structure (*escape_data) for PEXEscapeWithReply for this particular opcode is defined as:

```
typedef struct {
    Drawable            drawable;
    PEXColorApproxEntry capx;
} PEXEscapeQueryColorApprox;
```

Table 6-9. Data Structure Parameters

Value	Description
drawable	The identifier of an example drawable in the correct screen and visual (similar to the drawable parameter of other PEXlib inquiries such as PEXGetEnumTypeInfo).
capx	A color approximation table entry for which you wish to check support by the server. A typical source for the information in such an entry would be a standard colormap property, but your application could acquire or generate this information by other means.

Notice that the capx field is not a pointer. You must actually copy the information into this embedded structure (by using a C structure copy).

The function of the PEXEscapeQueryColorApprox opcode is to allow the server to verify whether or not the supplied color entry can be supported by the PEX server. If the entry is supported, the return data indicates such. If the entry is not supported, the server returns one or more supported entries from which your application may choose.

The PEXEscapeQueryColorApproxReplyData is defined as:

```
typedef struct {
    char                capx_is_supported;
    char                all_capx;
    char                reserved1[2];
    unsigned long       count;
    unsigned int        reserved3[3];
} PEXEscapeQueryColorApproxReplyData;
```

Table 6-10. Return Data

Value	Description
capx_is_supported	True indicates that the color approximation entry you sent is supported "as is." False indicates otherwise.
all_capxs	True indicates that all supported (alternative) color-approximation entries have been returned in the list that follows the PEXEscapeQueryColorApproxReplyData structure. False indicates otherwise.
count	The number of PEXColorApproxReplyData structures returned in the storage following the reply structure. This is zero if capx_is_supported is False.

The set of alternative color approximation entries can be accessed by computing the appropriate pointer and using array indexing, as in the following example:

```
Drawable                             my_example_drawable;
PEXColorApproxEntry                  my_candidate_color_approx;
PEXColorApproxEntry                  my_chosen_color_approx;
PEXEscapeQueryColorApproxData        query_in;
char                                 *return_ptr;
unsigned long                        return_size;
PEXEscapeQueryColorApproxReplyData   *query_out;
PEXColorApproxEntry                  *alternative_entries;

(example drawable ID and candidate color approximation entry omitted)
query_in.drawable = my_example_drawable;
query_in.capx = my_candidate_color_approx;
return_ptr = PEXEscapeWithReply(display, PEXEscapeQueryColorApprox,
                                sizeof (PEXEscapeQueryColorApproxData)),
                                (char *) &query_in,
                                &return_size);
if (return_ptr == NULL) {
    (the request failed, and some kind of error is reported)
}
else {
    query_out = (PEXEscapeQueryColorApproxReplyData *) return_ptr;
    alternative_entries = (PEXColorApproxEntry *) (return_ptr
                        + sizeof(PEXEscapeQueryColorApproxReplyData));
    if (query_out->capx_is_supported) {
        (the candidate is supported)
        my_chosen_color_approx = my_candidate_color_approx;
    }
    else {
        (use the various fields in "query_out" as needed)
        (select from the alternative entries)
        my_chosen_color_approx = alternative_entries[0];
        my_chosen_color_approx = alternative_entries[query_out->count-1];
    }
    (always free the returned storage when done)
    XFree(return_ptr);
}
```

This escape can also report a `BadDrawable` error (if the example drawable ID is not valid) or `BadValue` (due to an illegal value in one of the color approximation entry fields).

Table 6-11. Encoding of HP-Supported Color Escape Extensions

Extension/size	Type/value	Explanation
`PEXEscapeWithReply` (`PEXEscapeQueryColorApprox`$-\langle request \rangle$)		
4	`0x80010001`	vendor ID = `MIT`
2	`INT16`	Floating point format
2		unused
4	`CARD32`	example: `drawable_id`
2	`INT16`	Color approximation type
2	`INT16`	Color approximation model
2	`CARD16`	max1
2	`CARD16`	max2
2	`CARD16`	max3
1	`SWITCH`	Dither: $0 \rightarrow$ Off; $1 \rightarrow$ On
1		Unused
4	`CARD32`	multiplier 1
4	`CARD32`	multiplier 2
4	`CARD32`	multiplier 3
$\langle fp \rangle$	`FLOAT`	weight 1
$\langle fp \rangle$	`FLOAT`	weight 2
$\langle fp \rangle$	`FLOAT`	weight 3
4	`CARD32`	base pixel
`PEXEscapeWithReply` (`PEXEscapeQueryColorApprox` $- \langle reply \rangle$)		
4	`0x80010001`	Escape ID
1	`BOOLEAN`	Given color approximation is supported
1	`BOOLEAN`	Exhaustive list of color approximations
2		Unused
4	`CARD32`	Number n of alternatives is supported color approximations
12		Unused
12 $(3\times\langle fp \rangle+28)\times\langle n \rangle$	`LISTofCOLOR_APPROX`	Alternative supported color approximations (same format as request encoding, from color approximation type to end of request)

Making Color Approximation Inquiries

The Hewlett-Packard implementation of PEX does not support arbitrary color samplings. In fact, for any given device and environment, only one color sampling (and therefore, only one particular `PEXColorSpace` color approximation setup) is supported.

You need access to the color sampling information—you must use the escape—at the time that you load the colormap. `PEXEscapeWithReply` cannot be called before `PEXInitialize`; therefore, it is convenient to call `PEXInitialize` before window and colormap creation, not afterwards.

The predefined entry support in the HP PEX color approximation table means that if you plan to use `PEXColorSpace` approximation, you don't even need to call `PEXSetTableEntries` to set up a table entry, although in general, this is an important practice to assure interoperability. It is very important that you do create a color approximation table, because HP PEX cannot set up the correct predefined entry (which may vary from one device to the next) until you indicate what screen and visual to use, by passing an example drawable to `PEXCreateLookupTable`. We recommend that you always set the table entry explicitly because other PEX implementations may not have predefined entries.

Here is another implication of the single color-sampling support per configuration in HP PEX: If you call `PEXSetTableEntries` to set a `PEXColorSpace` color approximation table entry to values that do not match the one supported setting, the server reports an error.

Some applications may need to use `PEXColorRange` color approximation rather than `PEXColorSpace`. While `PEXColorRange` is supported on HP devices on `PseudoColor` and `StaticColor` visuals, it is *not* supported on `DirectColor` or `TrueColor` visuals. The only restriction on the color approximation entry values, beyond what the PEX standard prescribes, is that the `mult1`, `mult2`, and `mult3` values in the color approximation entry must be 1.0, 0.0, and 0.0, respectively.

In cases where the device depends on colormap interpolation ramps that can be described in an `XStandardColormap` structure, you can use some of the utilities described in the *PEXlib Programming Manual*, but in other cases, you will need to use the utility code provided by Hewlett-Packard to set the colormap properly.

If your graphics device is a CRX, you will need to use the special utilities for adjusting the colormap for that hardware. Source code for these utilities is provided online and described in the `README` file in the ⟨*pex-utils*⟩ directory[7].

Effect of Dithering Control on Color

For some applications, it is desirable to disable dithering—such as finite-element analysis or data visualization that benefit from the resulting color banding effects. On most devices, this is accomplished in the color approximation table entry by setting the dither hint to `PEXOff`. However, dithering cannot be disabled on CRX devices when using `PEXColorSpace`. Dithering is not performed in 24-bit visuals so the dither hint has no effect for those targets.

Using Indexed Colors

PEX supports specification of color attributes for primitives and light sources and the renderer's background color via color indices as well as via RGB triples. In fact, the PEX standard specifies that the default value for all colors is index 1 (except for the renderer background color, which defaults to index 0).

By default, however, the PEX standard does not specify what color is selected by index 1. HP PEX defines that default color to be white. This means that if you don't create a table of type `PEXLUTColor` and associate it with the renderer, all primitives will be drawn in white by default.

For many applications, it is natural to specify colors in terms of RGB. These applications do not need to create a color lookup table, but do need to set the renderer background color and the primitive colors to RGB values before they cause PEX to use those attributes, otherwise everything will be drawn in white.

For applications that need to use indexed colors, it is important to note that the HP PEX color table only has eight entries predefined to a set of simple primary colors. If you use many color indices but don't load the entries in the table, you'll still end up with a lot of things drawn in white.

Remember, the color table is at the "front end" of the PEX rendering pipeline. Indexed colors are always converted to RGBs as primitives enter the pipeline (unless using `PEXHPColorApproxTypeIndexed`). This means that there is no

[7] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

correspondence between colors you load into the PEX color lookup table, and the colors you must put in the X colormap. The rendering pipeline always operates in RGB. For this reason, `PEXColorSpace` is the most natural color approximation method, even for applications that use indexed colors.

Alpha Blending and Transparency

Alpha blending is a frame buffer operation that blends a source color with whatever color is already in the frame buffer, for each pixel in an image. Alpha transparency allows per-pixel blending of surface colors with other objects in the scene to produce high-quality transparency effects. The alpha transparency method results in a much smoother transparency than that achieved via the screen-door transparency method that has been available to date. See the example programs in ⟨*hp-examples*⟩[8] and the `README` for instructions.

Screen-Door Transparency

Screen-door transparency (available in previous releases of HP PEX) is computationally very fast. Currently, the only control over screen-door transparency is the "transmission coefficient" in the front- and back-surface reflection attributes. A coefficient of 0.0 indicates a completely opaque surface; a coefficient of 1.0 means the surface is completely transparent and does not contribute to the image. Screen-door transparency is accomplished by mapping the coefficient value into one of a discrete number of levels defined by a "screen door pattern cell". On most HP devices, this is a 4×4 cell, so there are 17 possible levels, from $\frac{0}{16}$, completely opaque (all pixels in the cell are drawn), to $\frac{16}{16}$, completely transparent (no pixels are drawn).

Screen-door transparency is very fast because it requires no frame buffer reads and can be built into the rasterizers or the frame buffer data path. This allows transparent objects to be animated with little or no performance impact.

The fixed-raster nature of the screen-door cell, and the fact that it is typically tiled either in screen coordinates or window coordinates, sometimes creates unpleasant visual artifacts. For example, two equally-transparent surfaces, one in front of the other, result in only the top surface being visible, because the screen-door cells overlay exactly. The image gets no color contribution from the surface that

[8] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

is farther away, and the user cannot see it at all. There also may be interference patterns between the screen-door cell and any dithering pattern that is in use.

Alpha Transparency

"Alpha" is an extra channel carried along with colors (or vertex data from which colors can be computed; for example, by lighting equations) that specifies the opacity of the color. Typically, an alpha of 1.0 indicates complete opacity, and 0.0 indicates complete transparency (i.e., the inverse of the transmission coefficient).

Alpha transparency blends the color of a transparent surface with the colors of other primitives that are "behind" it for every pixel of the surface. This eliminates the aliasing artifacts due to screen-door cells, but requires that the current color of each pixel be read out of the frame buffer and that a blending algorithm be applied to mix the new surface color with the existing color. The blending functions make use of the alpha channel to do the mixing. These extra operations can have significant performance impacts if not built into the graphics hardware, and have some even when the hardware *does* offer support. This part of the necessary functionality is called **alpha blending**, and it has uses in other techniques besides alpha transparency; for example, antialiasing and texture mapping both require some alpha blending.

In addition to alpha blending, to render a correct picture, alpha transparency requires two or more passes to render the primitives in the image. The first pass renders the background color and all "opaque" primitives. This creates the "background image" to be used in blending with transparent primitives. Then, the transparent primitives are rendered in one or more passes, blending their colors with the image already rendered. Ideally, the primitives are rendered in sorted order from most-distant in Z to nearest, for each pixel in the image. Compromises for better performance are possible, but introduce various sorts of visual artifacts.

HP PEX provides extensions to support simple alpha blending and to use the Z-buffer in a read-only mode—to test the Z-buffer value without changing it. These features can be utilized in an application to implement various alpha transparency effects. Additional work would be required in the application to do view-dependent Z-buffer sorting even at the level of HP PEX primitives.

Implementation of Alpha Transparency

Values are shown here; mnemonic strings have names derived from the value identifiers in the usual fashion.

Table 6-12.
Encoding of HP-Supported Alpha Transparency Extensions

Extension/New Values	Numeric Value	Explanation
`PEXETColorType`		
`PEXHPColorTypeRGBA`	`0x8700`	An addition to existing enumerated type
`PEXETHLHSRMode`		
`PEXHPHLHSRZBufferReadOnly`	`0x8700`	Additions to existing enumerated type
`PEXHPHLHSRZBufferIDReadOnly`	`0x8701`	
`PEXHPETTransparencyMethod`	`0x8700`	New enumerated type
`PEXHPTransparencyMethodScreenDoor`	`0x8700`	No alpha transparency is in effect
`PEXHPTransparencyMethodAlphaBlend`	`0x8701`	Alpha blending is in effect
`PEXHPETAlphaBlendFunction`	`0x8701`	New enumerated type
`PEXHPAlphaBlendFunctionSrcColor`	`0x8700`	Use only the source color
`PEXHPAlphaBlendFunctionSimpleAlpha`	`0x8701`	Blend the source color with the frame buffer color according to the formula $\langle src\rangle \times \alpha + \langle dest\rangle \times (1-\alpha)$

6

Renderer Attributes

The `transparency_method` HP extension Renderer attribute controls the type of transparency algorithm supported by the renderer. Its default is `PEXHPTrans-parencyMethodScreenDoor`. See `PEXHPChangeRenderer` in the on-line documentation for information on how to set this attribute.

Pipeline Context Attributes

There is one new Pipeline Context attribute supported for alpha blending. The `alpha_blend_function` default is `PEXHPAlphaBlendFunctionSrcColor`. Please see `PEXHPChangePipelineContext` in the on-line documentation for information on how to change it.

Color Types

An additional color type is defined, which carries alpha as a fourth channel. This color type is currently accepted only by primitives that can include color data per facet or per vertex. Since HP PEX only supports floating-point color values, only one type, `PEXHPColorTypeRGB`, need be supported (added to the `PEXETColorType` enumerated type). The encoding for the new color type is straightforward; its size is four words.

With-Data Primitives

In order to pass alpha values per-vertex or -facet into the CGE PEX extended with-data primitive entrypoints, additional vertex and facet data types are defined that use the new color types. There was no need to define new bits for the `vertex_attributes` and `facet_attributes` masks, because the `color_type` attribute in the existing parameter lists carries all the information that is necessary.

There are three unions that appear in with-data primitive parameter lists that might support RGBA color: `PEXFacetData`, `PEXArrayOfFacetData`, and `PEXArrayOfVertex`. `PEXListOfVertex` is also indirectly affected since it contains `PEXArrayOfVertex` as a field. Also affected are the vertex data structures, `PEXExtArrayOfVertex` and `PEXExtListOfVertex`.

The PEXFacetData union is most affected since its size would be changed, though it is only passed to primitives by reference and in all cases only a single facet's worth of data is passed. However, it is embedded directly inside members of the PEXOCData union, so changing it would create an incompatibility for old programs using the PEXlib OC encoding routines. Therefore, the union has not been changed and typecasting of the pointer is necessary when calling the entry points that use this union. A typical example of the typecasting technique would be:

```
        ⋮
    PEXHPColorRGBA facet_color;
        ⋮
    facet_color.red   = 1.0;
    facet_color.green = 0.5;
    facet_color.blue  = 1.0;
    facet_color.alpha = 0.5;
        ⋮
    PEXFillAreaSetWithData(..., PEXHPColorTypeRGBA, ...,
      ((PEXFacetData *) &(facet_color), ...);
        ⋮
```

In a similar vein, no change to union PEX[Ext]ArrayOfFacetData is implemented. Instead, the following technique can be used to pass a pointer to the data.

```
        ⋮
    PEXHPColorRGBA          facet_color;
    PEXArrayOfFacetData     facet_data;
        ⋮
    facet_color.red   = 1.0;
    facet_color.green = 0.5;
    facet_color.blue  = 1.0;
    facet_color.alpha = 0.5;
        ⋮
    facet_data.rgb = (PEXColorRGB *) &facet_color;
        ⋮
```

A technique similar to that shown above for PEXArrayOfFacetData can be used to pass this color data via PEX[Ext]ArrayOfVertexData or, indirectly, PEX[Ext]ListOfVertex.

Output Command

The function `PEXHPSetAlphaBlendFunction` creates an output primitive attribute that sets the blend functions for alpha blending and alpha transparency. Unsupported values will default to `PEXHPAlphaBlendFunctionSrcColor`, which results in source color rendering. Also see `PEXHPChangeRenderer`.

```
PEXHPSetAlphaBlendFunction(
    Display     *display,
    XID         resource_id,
    PEXOCType   OCtype,
    int         blend_function)
```

Using this routine, parameters are as follows:

Table 6-13. Alpha Blending Reply Parameters

Value	Description
`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`resource_id`	The resource ID of the structure or renderer.
`req_type`	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle`, or `PEXOCStoreSingle`).
`blend_function`	The alpha blending function to apply.

This entrypoint generates an extended OC with an output command number of `0x8700`. The encoding for the output command is:

Table 6-14. Encoding of HP-Supported Alpha Blending Extension

Extension/size	Type/value	Explanation
2	`0x8700`	Output command number
2		Output command length
2	`INT16`	Alpha blend function
2	unused	

Behaviors

The default value of all attributes results in alpha transparency being off; screen-door transparency is in effect. Alpha values are not interpolated across primitives when in screen door mode; only the front and back surface transmission coefficients induces screen-door transparency.

The transmission coefficient in the surface reflection attributes is directly mapped to an alpha value for surfaces as $1-\langle \mathit{coeff} \rangle$. This alpha value is used for surfaces that do not have per-facet or per-vertex alpha data, when alpha transparency is enabled.

Texture mapping can produce textures with alpha values. Depending on the compositing rule in use, alpha values from the surface color or alpha-per-vertex may or may not be used. For example, a texture map with alpha values and a "replace" composition rule does not use any color channels (including alpha) from the surface, but the "modulate" rule does. When surface alpha is used, it is applied as part of the surface color, during the first texture mapping compositing operation. For example, an opaque surface may become transparent due to alpha in a texture.

Alpha is not applied to vector/edge antialiasing. It is not effective for interior style "hollow".

When depth cueing is enabled, it is applied *after* a source alpha value is derived for the pixel, but it does not modify the alpha channel of the pixel color as it does the red, green, and blue channels—that is, during depth-cue modification of the color, the source alpha value is carried through unchanged. After depth cueing, the alpha value is used to blend with the pre-existing frame buffer contents.

On targets that do not support alpha blending (e.g., 8-plane visuals and CRX-24), when the transparency method calls for alpha blending, HP PEX does not do transparency. Specifically, screen-door transparency is not substituted for alpha blending. (In fact, the `PEXHPETTransparencyMethod` enumerated type does not list any alpha method as a supported value on such platforms, so a `BadValue` error is reported if an attempt is made to put the renderer in such a mode.)

Setting Up An Alpha Blending Program

Two sequences to set up alpha blending are illustrated in pseudocode below.

1. A typical sequence to set up simple alpha blending:
   - `PEXHPChangeRenderer( ... );` (⟨*transparency_method*⟩= `PEXHPTransparencyMethodAlphaBlend`)
   - `PEXHPChangePipelineContext( ... );` (⟨*alpha_blend_function*⟩= `PEXHPAlphaBlendFunctionSimpleAlpha`)
2. And then:
   - `PEXRenderNetwork();` *(for Structure Mode)*
     or
   - `PEXBeginRendering();` *(for Immediate Mode)*
   - *(render the OCs)*
   - `PEXEndRendering();`
3. A simple two-pass transparency can be implemented in the client using the following mixed-mode sequence:

```
PEXBeginRendering();
  (render opaque OCs in immediate mode or via PEXExecuteStructure)
  PEXHPChangeRenderer(...);
  ⟨transparency_method⟩=PEXHPTransparencyMethodAlphaBlend;
  ⟨hlhsr_mode⟩=PEXHPHLHSRZBufferReadOnly);
  PEXHPSetAlphaBlendFunction(...);
  ⟨alpha_blend_function⟩=PEXHPAlphaBlendFunctionSimpleAlpha);
  (render transparent OCs in immediate mode or via PEXExecuteStructure)
PEXEndRendering();
```

6

Anti-aliasing

HP supports two methods of anti-aliasing for producing high-quality images, most noticeable as smooth lines and polygon edges. In the first, which was provided with HP PEX 5.1v1, anti-aliasing is provided through a GSE (see "Line Types"). However, this GSE is not the preferred method and is retained primarily for compatibility reasons.

The second and preferred method, added for HP PEX 5.1v2 is an OC routine `PEXExtOCSetPrimitiveAA`, which selects the primitives that are to be anti-aliased. A blending operation specifies the anti-aliased method to be used.

Note that there is no mention of gamma correction made in the PEX 5.2 or CGE PEX 1.0 specifications. Thus, it is not addressed in the specification. However, HP PEX supports the Gamma Correction Escape in order to maintain compatibility with older programs. Other tools for setting up a gamma corrected colormap should be considered by application developers.

The `PEXExtSetPrimitiveAA` entry point controls anti-aliasing. The `blend_op` parameter indicates the blend operation to be used. For HP PEX, `PEXExtPrimAABlendOpImpDep` and `PEXExtPrimAABlendOpSimpleAlpha` have the same result. The calculation for `PEXExtPrimAABlendOpSimpleAlpha` is $\alpha \times \langle \mathit{src\_color} \rangle + (1-\alpha) \times \langle \mathit{dest\_color} \rangle$.

The HP/CGE PEX 1.0 supported/unsupported anti-aliasing methods are:

Table 6-15.

`PEXExtPrimAANone`	HP/CGE PEX 1.0 supported (default)
`PEXExtPrimAAPoint`	unsupported by HP
`PEXExtPrimAAVector`	HP/CGE PEX 1.0 supported
`PEXExtPrimAAPointAndVector`	unsupported by HP
`PEXExtPrimAAPolygon`	unsupported by HP
`PEXExtPrimAAPointAndPolygon`	unsupported by HP
`PEXExtPrimAAVectorAndPolygon`	unsupported by HP
`PEXExtPrimAAPointVectorAndPolygon`	unsupported by HP

Setting the mode to an unsupported index will cause the mode to default to `PEXExtPrimAANone`.

Enumerated type `PEXExtETPrimitiveAABlendOp` indicates the supported blend operations. `PEXExtPrimAABlendOpImpDep` and `PEXExtPrimAABlendOpSimpleAlpha` are both supported on HP. Both methods use the same algorithm on HP.

Note that anti-aliasing is not supported on some unaccelerated devices and on all 8-bit visuals. Use `PEXGetEnumTypeInfo` to determine anti-aliasing capability for a particular drawable. If anti-aliasing is not supported attempts to enable it will be silently ignored.

Line Primitives and Attributes

Line Types

These line types are supported:

- `PEXLineTypeSolid`
- `PEXLineTypeDashed`
- `PEXLineTypeDotted`
- `PEXLineTypeDashDot`
- `PEXExtLineTypeCenter` *(CGE extension)*
- `PEXExtLineTypePhantom` *(CGE extension)*
- `PEXHPOCCSetUserLinetype`

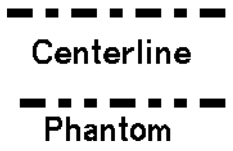


Figure 6-1. ET Line Types

Additional information and recommendations about extended line types, the `PEXExtETMLineType` enumerated type lists, and the use of antialiasing to enhance the appearance of lines and edges are also covered in the *Portable Programming with CGE PEX 5.1*.

HP PEX supports only an edgewidth scale factor of 1.0. Any OC that attempts to set it is silently mapped to 1.0.

To improve the quality of images, of lines and polygon edges in particular, antialiasing functionality is available.

Antialiased images, however, do not look their best unless gamma correction has been performed on the X colormap contents. Therefore, it is recommended that if you use the HP GSE for anti-aliasing that you also apply gamma correction to your colormap.

Utilities in the ⟨*pex-utils*⟩ directory[9] include a utility procedure that you can call to create a gamma-corrected colormap. Note that gamma correction may cause some vectors to be drawn without antialiasing to appear dimmed. You should be aware of this effect, although it is unusual for an application to enable and disable antialiasing during traversal.

The `PEXGSE` entrypoint has the following interface:

```
void PEXGSE(
    Display             *display,
    XID                 resource_id,
    PEXOCRequestType    req_type,
    long                id,
    int                 length
    char                *data)
```

Table 6-16. `PEXGSE` Parameters

Value	Description
`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call
`resource_id`	The resource identifier of the renderer or structure
`req_type`	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`)
`id`	The identifier of the GSE
`length`	The length, in bytes, of the GSE data
`data`	A pointer to the GSE data

[9] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

To use the HP antialias mode GSE, the ID parameter should be set to `HP_GSE_SET_ANTIALIAS_MODE`. The data structure type name needed for the data parameter is `hpGSESetAntialiasMode`. The `antialias_mode` field in this structure can take one of the following values:

Table 6-17. Valid `antialias_mode` Values

`HP_ANTIALIAS_MODE_OFF`	Disables antialiasing for line primitives and for fill-area edges.
`HP_ANTIALIAS_MODE_BEST`	Enables antialiasing for both line primitives and for fill area edges using a filtering method that is device-dependent, but which gives the best visual results for the device.

Table 6-18.
Encoding HP-Supported Antialiasing and Gamma Correction Extensions

Extension/Size	Type/Value	Explanation
`PEXGSE` (`HP_GSE_SET_ANTIALIAS_MODE`)		
4	`0x80070001`	HP opcode (`decimal1`)
4	`INT32`	`antialias_mode`; `HP_ANTIALIAS_MODE_OFF` (0) *or* `HP_ANTIALIAS_MODE_BEST` (1)
`PEXEscape`(`HP_ESCAPE_SET_GAMMA_CORRECTION` – ⟨*request*⟩)		
4	`0x80070002`	HP opcode (`decimal2`)
4	`CARD32`	`drawable_id`
4	`INT32`	`gamma_correction mode`; `HP_GAMMA_CORRECTION_OFF` (0) *or* `HP_GAMMA_CORRECTION_ON` (1)

Wide-Line End Styles

Cap and join style support is provided through two CGE extension OCs: `PEXExtOCSetLineCapStyle` and `PEXExtOCSetLineJoinStyle`. Defined values (not all necessarily supported by HP PEX) for these additional enumerated types are:

- `PEXExtETLineCapStyle`
  - `PEXExtETLineCapStyleButt`
  - `PEXExtETLineCapStyleRound`
  - `PEXExtETLineCapStyleProject`
- `PEXExtETLineJoinStyle`
  - `PEXExtETLineJoinStyleImpDep`
  - `PEXExtETLineJoinStyleMiter`
  - `PEXExtETLineJoinStyleRound`
  - `PEXExtETLineJoinStyleBevel`

Area Primitives and Attributes

The PEX standard makes interior style `PEXInteriorStylePattern` optional and HP PEXlib does not support "pattern" interior styles, defaulting to `PEXInteriorStyleHollow`. Attempts to set an unsupported style results in the default `PEXInteriorStyleHollow`. Attempts to create a pattern LUT result in a `BadPEXLookupTable` error.

NURBS Approximation

Setting the curve approximation criteria to a particular value for NURBS surface trim curves has no effect in HP PEX. Trimming curves are automatically computed with a resolution compatible with the surface approximation criteria.

The parametric surface characteristics attribute can be set to either of two values: `PEXNone` or `PEXPSCImpDep`. The setting `PEXPSCImpDep` is intended to implement a method by which all edges of all fill areas generated by NURBS surface tessellation are made visible. In the HP PEX 5.1 release, the parametric surface characteristics attribute has no effect. The appearance of NURBS surfaces is governed entirely by interior attributes. When fill area edging is enabled, NURBS interior edges are visible. This behavior is avoided by disabling edging around NURBS. Hewlett-Packard recommends that applications set the parameter surface characteristics attribute for portability and because HP PEX behavior may change in future releases.

Antialiasing

HP PEX supports a Generalized Structure Element (GSE) to enable or disable line- and edge antialiasing. This is not a standard feature (there are no standard GSEs specified by PEX), but may have value for your application. The constants and data structure for the PEXlib interface are defined in the header file `PEXHPlib.h`. Please see the section “Line Types” for more information.

Capping and Interference Checking

These two visualization techniques are suited to the modeling of solid objects common to MCAD applications. Capping is an adjunct to model clipping that re-closes a capped volume where it has been clipped. The result appears as though a section has been cut from a solid object. Interference checking can detect interpenetrating solids by highlighting overlapping caps within a clip plane.

HP PEX supports the formation and rendering of capping facets to indicate how a volume-enclosing set of surfaces is intersected by a model clipping plane. It also allows capping facets for different enclosed volumes to be collected and their intersection to be rendered, usually using a distinguishing set of attributes. This is called “interference checking,” since it can be used to show intersection of two or more volumes (solids) in a model clipping plane.

PEX defines model clipping, but does not address capping or interference checking. Responding to customer request, HP developed an interface to provide access to these features through PEXlib. This interface supports the definition of volumes independently of the primitives or structures used to define them. The interface is implemented with the procedure `PEXHPSetCappingPlanes` (see the reference page in the on-line documentation). This entrypoint is implemented as an Extended Output Command (extended-OC).

Deformation

Another technique useful for modeling of solid objects and MCAD applications is **deformation**. Deformation modifies geometric coordinates in the "with-data" primitives before modeling transformations are applied. There are four steps required to control deformation:

1. Set the values in a "global" deformation factor. The deformation factor is a complex number, meaning that it has real and imaginary components. This factor is multiplied by the deformation values, which are defined for each individual vertex. Depending on the current deformation mode, the real or imaginary portion of the product is then added to the geometric coordinates of the vertex.
2. Set the deformation values supplied with each vertex in a with-data primitive. For each vertex, the individual deformation value is multiplied by the global deformation factor. Depending on the current deformation mode, the real or imaginary portion of the product is then added to the geometric coordinates of the vertex.
3. Set the deformation value location.
4. Set the deformation mode. The deformation mode is an attribute that turns deformation calculations on and off, and (if deformation calculations are turned on) determines which portions of the product of the deformation factor and deformation values are added to the geometric coordinates of a vertex.

In the HP PEX 5.1v2 implementation, a new attribute command, `PEXHPSetDeformationMode` is used to set the deformation mode, and the value of the global complex deformation factor. The initial deformation mode and factors can be set in the pipeline context.

The following CGE PEX 1.0 extended area primitives will accept deformation values in the vertex data:

- `PEXExtFillAreaSetWithData`
- `PEXExtTriangleStrip`
- `PEXExtQuadrilateralMesh`
- `PEXExtSetOfFillAreaSets`

These primitives are defined in the CGE PEX 1.0 specification. Existing 5.1 area primitives (`PEXFillAreaSetWithData`, `PEXTriangleStrip`, `PEXSetOfFillAreaSets`, etc.) are unaffected by the deformation mode.

The HP extended primitives `PEXHPPolylineSetWithData` and `PEXHPMarkersWithData` will also accept deformation data.

Deformation values are stored in a list of floats. There may be one list of floats containing "extra data" like texture mapping data or deformation values per vertex. These values are passed to PEXlib for each vertex, following the vertex normals, colors and edges flags. Extra vertex data, like deformation values and texture mapping coordinates, can appear anywhere in this list of floats. `PEXHPSetDeformationValueLocation` indicates where in the list of extra data deformation values can be found.

The deformation value location is also an attribute whose initial value can be set in the pipeline context, modifiable via `PEXHPChangePipelineContext`.

It is not likely that deformation will be incorporated in a future revision of the PEX standard.

There is no (inquirable) enumerated type for the deformation mode since HP is the only vendor that supports deformation.

These are the HP extended OCs required to support deformation, inquirable by enumerated type `PEXETExtOC`.

- `SetDeformationMode`
- `SetDeformationValueLocation`
- Extended Polyline Set With Data
- Extended Polymarker With Data

Extended Pipeline Context Attributes

These values are part of the list returned by an inquiry of enumerated type `PEXETExtPC`.

- `PEXHPPCDeformationMode`
- `PEXHPPCDeformationValueLocation`

Pipeline Context Attributes

Deformation mode may be set in the pipeline context. The default value for deformation mode is `PEXHPDeformationOff`. `PEXHPChangePipelineContext` is used to modify the extended pipeline context. See Chapter 6 for details on changing the HP-only attributes in the pipeline context.

Deformation values are stored in a list of floats associated with each vertex in a with-data primitive. The deformation value location is an index into that list of floats. The default value for the deformation value location is index 0. The deformation value location can also be changed using the `PEXHPChangePipelineContext` routine defined in Chapter 6.

This value is returned by an inquiry of enumerated type `PEXExtETID`:

```
PEXHPIDDeformationSupported
```

Text and Fonts

The fonts supported by HP PEXlib are accessed using the X Logical Font Description (XLFD) conventions. PEX stroke fonts can be regarded as infinitely scalable and rotatable, although, unlike scalable bitmap fonts, no process of font generation occurs when a font is opened.

The values shown in the following tables are supported for XLFD name fields to access HP fonts that are returned by `PEXListFonts` and `PEXListFontswithInfo` and are accepted by `PEXLoadFont`.

Table 6-19. Text and Fonts

Field Name	HP Value(s) and Explanation
`FOUNDRY`	hp
`FAMILY_NAME`	Stick, simplex sans serif, polygonal sans serif, polygonal serif
`WEIGHT_NAME`	medium, bold
`SLANT`	r
`SETWIDTH_NAME`	normal
`ADD_STYLE_NAME`	normal, accel
`PIXEL_SIZE`	0 ⟨convention for scalable fonts⟩
`POINT_SIZE`	0 ⟨convention for scalable fonts⟩
`RESOLUTION_X`	0 ⟨convention for scalable fonts⟩
`RESOLUTION_Y`	0 ⟨convention for scalable fonts⟩
`SPACING`	p, m
`AVERAGE_WIDTH`	0 ⟨convention for scalable fonts⟩
`CHARSET_REGISTRY` `CHARSET_ENCODING`	`hp-roman8`, `iso8859-1`, `hp-japaneseeuc`, `jisx0208.1983-0`

These properties are defined for HP PEX fonts: `FOUNDRY`, `FAMILY_NAME`, `WEIGHT_NAME`, `ADD_STYLE_NAME`, `SPACING`, `CHARSET_REGISTRY`, and `CHARSET_ENCODING`.

Font Naming and Files

The default font used by PEX is a monospaced, stroke, `hproman8` font (i.e., non-proportional, `hproman8` glyph layout, one byte per character). This applies to both DHA applications and to the PEX server. Unlike other PEX fonts that you can inquire for font information, additional font information is not available for the default font.

The pattern of the `FAMILY_NAME` field starting with `PEX` is an HP convention for stroke fonts, it is not yet an interoperability convention.

HP PEX font files, used by both the HP PEX server and DHA PEXlib, are organized in the following directory structure that parallels the X11 font structure.

Table 6-20. HP PEX Font File Structure

⟨*fonts*⟩ **directory**			
`font_info/stroke` fonts style information	`usascii/stroke` `usascii` and `hproman` fonts	`usascii/stroke` `ascii` and `latin1` fonts	`hp_japanese/stroke` `jisascii`, `katakana`, `kanji` fonts

The fonts are represented in an HP-specific format (the same used by Starbase and HP-PHIGS). All PEX fonts have the file name suffix "`.pht`" and their file names also indicate the character set and font style. The specific fonts supported by Hewlett-Packard, including both the `XLFD` and file names, are listed at the end of this section in the three "Fonts" tables.

The X server font path is used by the PEX server to gain access to the PEX fonts. At server startup, whenever the font path is changed, and whenever the font directories are explicitly rescanned using `xset -fp` or `xset -rehash`, the X server (and font server in X11R6) searches the new directories for `fonts.dir` files. It will merge new files into a hash table so it can quickly find all fonts without searching each of the directories named in the font path.

For PEX font directories there is a corresponding `phonts.dir` file that is ignored by the X server, but is read by the PEX extension. This keeps the sets of X11 fonts and PEX fonts disjoint so `XListFonts` never returns PEX fonts and `PEXListFonts` never return X bitmap fonts.

X11R6 supports the X font server. However, the font server supports only X fonts, not PEX fonts.

Another file, ⟨*extensions*⟩[10]`/fp.PEX`, contains a list of directories to be added to the X font path during server initialization; it should always contain at least one of those directories.

Since HP CDE saves and restores the X font path across multiple sessions, HP CDE users will need to explicitly modify the X font path to access fonts other than the default. This can often be done by resetting the font path to the default, including the directories in ⟨*extensions*⟩`/fp.PEX`.

To reset the font path, type:

```
xset fp default  Return
```

This is not an appropriate solution for users who have customized their font paths.

[10] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

List of Fonts Supported by Hewlett-Packard

The specific fonts supported by Hewlett-Packard, including both the `XLFD` and file names are:

Table 6-21. ⟨*fonts*⟩`/usascii/stroke` **fonts**

Font File Name	Supported `XLFD` Names
`usascii.1.pht`	`-hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-hp-roman8`
`usascii.2.pht`	`-hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-hp-roman8`
`usascii.-2.pht`	`-hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-hp-roman8`
`usascii.-4.pht`	`-hp-PEX polygonal sans serif-bold-r-normal-normal-0-0-0-0-p-0-hp-roman8`
`usascii-6.pht`	`-hp-PEX polygonal serif-bold-r-normal-normal-0-0-0-0-p-0-hp-roman8`
`usascii.-8.pht`	`-hp-PEX polygonal serif-bold-r-accel-0-0-0-0-p-0-hp-roman8`

Table 6-22. ⟨*fonts*⟩`/usascii/stroke` **fonts**

Font File Name	Supported `XLFD` Names
`ascii.1.pht`	`-hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-iso8859-1`
`ascii.2.pht`	`-hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-iso8859-1`
`ascii.-1.pht`	`-hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-m-0-iso8859-1`
`ascii.-2.pht`	`-hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-iso8859-1`

Table 6-23. ⟨*fonts*⟩/`hp_japanese/stroke` **fonts**

Font File Name"	Supported XLFD Names
jisasc.1.pht	-hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-hp-japaneseeuc
jisasc.2.pht	-hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
jisasc.-2.pht	-hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
jisasc.-4.pht	-hp-PEX simplex sans serif-bold-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
jisasc.-6.pht	-hp-PEX polygonal serif-bold-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
kanjeuc.2.pht	-hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-jisx0208.1983-0
kanjeuc.-2.pht	-hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-jisx0208.1983-0
kanjeuc.-4.pht	-hp-PEX polygonal sans serif-bold-r-normal-normal-0-0-0-0-p-0-jisx0208.1983-0
kanjeuc.-6.pht	-hp-PEX polygonal serif-bold-r-normal-normal-0-0-0-0-p-0-jisx0208.1983-0

With the HP PEX 5.1v2 release and later, HP supports both annotation text styles `PEXATextNotConnected` (default) and `PEXATextConnected`.

Internationalized Text

The `kanjeuc`* fonts are two-byte fonts which support two-byte-encoded text strings.

HP PEX supports rendering two-byte PEX-encoded text and two-byte PEX-encoded annotation strings and two-byte encoded fonts. `PEXCSByte`, `PEXCSShort` and `PEXCSLong` are supported lengths for PEX-encoded text strings. Since HP PEX does not support three-byte fonts, using `PEXCSLong`-encoded text is not recommended at this time.

Marker and Cell Array

Hewlett-Packard adds the following extension marker types:

Table 6-24. Marker Type Additions

Extension/Size	Type/value
`PEXHPMarkerTriangle`	`0x8700`
`PEXHPMarkerSquare`	`0x8701`
`PEXHPMarkerDiamond`	`0x8702`
`PEXHPMarkerCrossSquare`	`0x8703`

The HP PEXlib implementation meets the PEX "cell array" requirements by simulating the cell array: drawing its outline with polylines.

B-spline Curves and Surfaces

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib.

Bundled Attributes

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib.

Modelling

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib.

Viewing

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib.

Animation

Animation, or imparting the appearance of motion to objects, is accomplished with any of several methods. Each method offers specific advantages that you'll need to consider when selecting the most appropriate method for the circumstances.

HP PEXlib supports rendering to X multi-buffer (MBX) as an efficient and portable method for creating animated images.

However, the option to resort to double-buffering with a pixmap, when MBX and E&S escapes are *not* available, is not supported.

The Double-buffering Extension (DBE)

HP PEXlib 5.1v3 and later releases support the use of the Double-Buffering Extension (DBE) version 1.0, as a means for double-buffering PEX (and X) graphics. DBE is a newer, simpler standard than MBX, which was supported in HP PEXlib 5.1v2 and which is still recommended for applications that wish to be CGE PEX 1.0 compliant.

Background Information

In the 5.1v2 release, HP PEXlib supported the Multi-buffering Extension (MBX) to accomplish double-buffering. In other words, MBX buffers were accepted as valid targets for the Renderer, and could be used in any inquiry or procedure call that required an example drawable. The application program would need to make MBX calls directly (`Xmbuf( ... )`) to create and swap buffers, but could pass the ID of an MBX buffer to the Renderer to cause drawing to that buffer. This allowed mixed X and PEX rendering in the back buffer, something that the earlier Evans & Sutherland escapes did not allow.

Since that release, a simpler double-buffering extension (DBE) has been brought through a rapid review process to become a new standard. DBE is very similar to MBX in many respects, but is simpler in that it only supports basic double-buffering (MBX supports creation of more than two buffers for a window) and allows for API-dependent interaction during clearing and swapping. Again, the application must directly make DBE calls (`Xdbe ... ()`) in order to create and swap buffers.

HP PEXlib 5.1v3 and later releases support the use of a DBE back buffer name in a manner very similar to the MBX support.

Summary of DBE Client Entrypoints

Here is a brief summary of the DBE client side entrypoints. To use these, the application program must include header file `X11/extensions/Xdbe.h`. For more detail, please see the reference pages in the appropriate X library manuals.

Table 6-25. DBE Entry Points

Entry Point	Description
`XdbeQueryExtension`	Verifies that an X server supports the extension, and, if so, what version it supports (currently the only known version is 1.0).
`XdbeGetVisualInfo`	Returns information about DBE support for Visuals on one or more screens of a server.
`XdbeFreeVisualInfo`	Frees the storage allocated by `XdbeGetVisualInfo`.
`XdbeAllocateBackBufferName`	Allocates a Drawable ID for the back buffer for a particular window. If this is the first name being created for the back buffer, resources may be allocated as part of this call. It is important to note that DBE only supports one back buffer for a window; calling this entrypoint multiple times simply creates multiple names for the same buffer. A back buffer name always addresses the back buffer, regardless of how many times swapping has occurred. If the window's Visual does not support double-buffering, this procedure results in a BadMatch error.
`XdbeDeallocateBackBufferName`	Releases a DBE back-buffer name. If the last existing name for the back buffer is being released, resources may be freed as part of this call.
`XdbeSwapBuffers`	Swaps the front and back buffers for one or more windows. A swap action can be given for each window, specifying what should be done to the back buffer after the swap.

Table 6-25. DBE Entry Points (continued)

Entry Point	Description
`XdbeBeginIdiom`	Specifies the beginning of a mixed sequence of calls, typically to accomplish swapping and clearing, that may or may not be optimized. In this release, the HP implementation does not optimize any sequences.
`XdbeEndIdiom`	Specifies the end of a sequence of calls.
`XdbeGetBackBufferAttributes`	Returns the ID of the window to which a back-buffer name is assigned.

Changes to Existing Functionality

`PEXMatchRenderingTargets`

Visuals that support creation of a DBE back buffer name match the `PEXBufferDrawable` target type. For practical purposes, the set of visuals on each device that support DBE is the same set that supports at least two MBX buffers. An application can directly inquire what Visuals support DBE via the `XdbeGetVisualInfo` entrypoint.

It is important to note that in HP's implementation, some visuals that cannot support double-buffering will be listed as supporting creation of one MBX buffer (see `XmbufGetScreenInfo`), in accordance with CGE PEX 1.0 requirements. However, these visuals will not be listed as supporting DBE (via the function `XdbeGetVisualInfo`) because no back buffer can be created for them.

Procedures Using Example Drawables

The following procedures that have an "example Drawable" entrypoint will all accept a DBE back buffer name as a valid example drawable:

- `PEXGetEnumTypeInfo`
- `PEXGetImpDepConstants`
- `PEXCreateLookupTable`
- `PEXCreateRenderer`

Colormap/Visual utilities

In this release, no changes have been made to either the HP-originated utilities (in `/usr/lib/PEX5/utilities`) or to the CGE utilities (in ... `/cge_utilities`) to make use of DBE. Applications that wish to use DBE can select a visual that can double-buffer using the utilities as provided, and then use `XdbeGetVisualInfo` to verify that DBE is supported on the visual. This is not different in nature from the current usage, where a visual can be selected using a double-buffering criterion, but the application must determine whether MBX is supported.

System Requirement/Release Dependencies

The HP PEX library (`libPEX5.sl`) generates references to some DBE client library entrypoints. The library that implements these is `libXext.sl`. It is absolutely necessary that the version of `libXext.sl` on a client system be new enough to implement these entrypoints, otherwise a runtime error (unsatisfied reference) will occur when PEXlib attempts to validate a Drawable ID passed into any procedure. The system release that includes HP PEXlib 5.1v3 and later releases also include an appropriate version of `libXext.sl`.

Whether or not the X server actually supports DBE is another matter. PEXlib will operate correctly whether or not the X server supports DBE. It is recommended that the application use `XdbeQueryExtension` or a more general Xlib extension query to verify that DBE is actually supported before attempting to create a back buffer name.

Compatibility Issues

No special compatibility issues are created by adding the support. However, if an application is modified to use DBE, then it will only compile and link successfully on systems that have the header files and the client library entrypoints. This is a portability consideration for application writers.

The Multi-buffering Extension (MBX)

MBX works as described in *PEXlib Programming Manual* (section 14.2), rendering to specific and separate image buffers. Buffer IDs can be used in any PEXlib call that requires a drawable ID. It is recommended that you use the function `XmbufQueryExtension` first, in order to determine whether the MBX extension is supported. Additionally, `PEXMatchRenderingTargets` reports `PEXBufferDrawable` targets on visuals that support multi-buffering. The visuals reported to be multi-buffer-capable are listed in the table below.

You should be aware of other characteristics of MBX that affect your programs. On systems with hardware double-buffering, setting up two buffers will provide hardware double-buffering sup port. However, on low-end systems or visual types without hardware double-buffering, operation is software-buffered.

See the *Graphics Administration Guide* for information about supported visuals on particular devices. If MBX is not available on your device, then you can use the Evans & Sutherland double-buffering escapes on the same visuals to achieve motion.

Evans & Sutherland Escapes for Double-Buffering

The escapes `PEXEscape` and `PEXEscapeWithReply` allow for access to the Evans & Sutherland method of double-buffering, an HP implementation-dependent functionality, that is described here. The structures for these functions are defined in the file `PEXHPlib.h` which also includes both the Evans & Sutherland escape requests and the HP companion escape request, `HPESCAPE_DFRONT`.

```
void PEXEscape(
    Display             *display,
    unsigned long       escape_id,
    int                 length,
    char                *escape_data)

char* PEXEscapeWithReply(
    Display             *display,
    unsigned long       escape_id,
    int                 length,
    char                *escape_data,
    unsigned long       *reply_length_return)
```

Using either escape, data parameters are as shown in the table below:

Table 6-26.
Double-Buffering Escape and -Escape With Reply Parameters

Value	Description
`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`escape_id`	The escape identifier.
`length`	The length, in bytes, of data for the escape request.
`*escape_data`	A pointer to data for the escape request: • For `ES_ESCAPE_DBLBUFFER`, use `esEscapeDblBuffer` • For `ES_ESCAPE_SWAPBUFFER`, use `esEscapeSwapBuffer` • For `ES_ESCAPE_SWAPBUFFERCONTENT`, use `esEscapeSwapBufferContent` • For `HP_ESCAPE_DFRONT`, use `hpEscapeSetRenderingBuffer`
`*reply_length_return`	Returns the length, in bytes, of the reply data (if applicable).

Table 6-27. Encoding HP Supported Double-Buffering Extensions

Extension/Size	Type/value	Explanation
PEXEscape (HP_ESCAPE_DFRONT)		
4	0x80070001	HP opcode (decimal1)
4	CARD32	drawable
4	BOOLEAN	render_to_front_buffer (True if visible buffer is drawing destination)
PEXEscape (ES_ESCAPE_DBLBUFFER)		
4	0x80040001	ES opcode (decimal1)
4	CARD32	drawable
4	CARD32	BufferMode (True if double-buffering)
PEXEscape (ES_ESCAPE_SWAPBUFFER)		
4	0x80040002	ES opcode (decimal2)
4	CARD32	drawable
PEXEscapeWithReply (ES_ESCAPE_SWAPBUFFERCONTENT−⟨*request*⟩)		
4	0x80040003	ES opcode (decimal3)
4	CARD32	drawable
PEXEscapeWithReply (ES_ESCAPE_SWAPBUFFERCONTENT−⟨*reply*⟩)		
4	0x80040003	ES opcode (decimal3)
4	CARD32	content (ES_DB_SWAP_CONTENT_UNDEFINED)

Evans & Sutherland Escape Requests

By specifying the Evans & Sutherland escape requests the following defined functionality is accessed.

- `ES_ESCAPE_DBLBUFFER`—This request sets up the specified drawable to be double buffered. Drawing commands directed at this drawable are written into the undisplayed buffer when the `bufferMode` is `ES_RENDERER_DBLBUFFER`. A back (undisplayed) buffer is allocated when this escape is received with `bufferMode` set to `ES_RENDERER_DBLBUFFER`. The back buffer is deallocated when this escape is received with `bufferMode` set to `ES_RENDERER_SINGLEBUFFER`.

  Sending this escape with `bufferMode` set to `ES_RENDERER_SINGLEBUFFER` when the drawable is already single-buffered has no effect. Sending this escape with `bufferMode` set to `ES_RENDERER_DBLBUFFER` when the drawable is already double-buffered has no effect. Sending this escape when the `RendererState` is `Rendering` has an effect that is implementation-dependent.

  This escape is not intended to enable mixing X and PEX graphics. Attempts to do so when the renderer is in double-buffer mode produces implementation-dependent results. It is recommended that applications use either the X Multi-buffer Extension (MBX) or this escape for double buffering, but not both. MBX should always be used if available.
- `ES_ESCAPE_SWAPBUFFER`—This request swaps buffers on the specified drawable. The undisplayed buffer becomes the displayed buffer and the previously displayed buffer is in a state described by the value returned by `ES_ESCAPE_SWAPBUFFERCONTENT`. Sending this escape with a drawable that is not double-buffered has no effect. Sending this escape when the `RendererState` is `Rendering` has an effect that is implementation-dependent.

- `ES_ESCAPE_SWAPBUFFERCONTENT`—This escape (the only one that is a `PEXEscapeWithReply`) returns the same value for a given drawable at all times. It is unnecessary to issue this escape after every swap because the state of the previously-displayed buffer remains consistent for the drawable. The possible values of the content are:
  - `ES_DB_SWAP_CONTENT_UNDEFINED` means that the content of the previously-displayed buffer is undefined. This is the HP value.
  - `ES_DB_SWAP_CONTENT_CLEAR_TO_BACKGROUND` means that the previously-displayed buffer is cleared to background after the "swap buffer" request.
  - `ES_DB_SWAP_CONTENT_UNCHANGED` means that the previously-displayed buffer content is unchanged after the "swap buffer" request.
  - `ES_DB_SWAP_CONTENT_FRONTBUFFER` means that the previously-displayed buffer has the same content as the currently-displayed buffer after the "swap buffer" request.

If the specified escape identifier is not supported, a value error is generated.

HP Escape Request

The HP escape, `HP_ESCAPE_DFRONT`, is a companion escape to the Evans & Sutherland double-buffering escapes described above. It allows you to set the Renderer drawing destination to either the front (visible) buffer or the back (hidden) buffer. This feature is useful for supporting interactive echoes over a visible rendered image and it allows buffer swapping to be reserved for actual image regeneration alone, rather than requiring both echoes and image regeneration to jointly control buffer swapping.

By specifying the HP companion escape request, the following defined functionality is accessed:

- `HP_ESCAPE_DFRONT`—If buffers have been allocated for the drawable via the `ES_ESCAPE_DBLBUFFER` request:
  - If `render_to_front_buffer` is `True`, the front, visible buffer is the renderer's drawing destination.
  - If `render_to_front_buffer` is `False`, the back, hidden buffer is the destination. The effect is visible at the next request that causes the renderer to draw.

  If no buffers have been allocated for the drawable, this escape has no effect until double buffering is turned on. Just as for the Evans & Sutherland escapes, this HP escape is intended to affect only PEX Renderer drawing, not X drawing. Effects on X drawing are implementation-dependent.

Inquiring Supported Escapes

You can use the enumerated type descriptors to inquire which double-buffering escapes are supported. These are also defined in the implementation-dependent header file, `PEXHPlib.h`.

The enumerated-type mnemonic strings and values returned from `PEXGetEnumTypeInfo` when inquiring with `PEXEscape` are:

Table 6-28. Inquiring Supported Escapes

EnumType Value	Enumerated Type Descriptor	Returned Mnemonic String
0x8401	ES_ESCAPE_ET_DBLBUFFER	ES_ESCAPE_DBLBUFFER
0x8402	ES_ESCAPE_ET_SWAPBUFFER	ES_ESCAPE_SWAPBUFFER
0x8403	ES_ESCAPE_ET_SWAPBUFFERCONTENT	ES_ESCAPE_SWAPBUFFERCONTENT
0x8701	HP_ESCAPE_ET_DFRONT	HP_ESCAPE_DFRONT

Structures

Search context resources are not supported with HP PEX 5.1v3 and later releases; however, protocol requests can be issued to PEX servers that *do* support search contexts.

Floating Point Formats/Conversions

The O'Reilly *PEXlib Programming Manual*, Section 15.6.4, "Copying Output Commands Between Servers," describes the transmission of data and the need for conversion of floating-point formats between servers. At this release of HP PEXlib, floating-point conversion support is not implemented. This means that an application using HP PEXlib to generate PEX protocol to a non-HP server that does not support `PEXIEEE_754_32` floating-point format will fail when `PEXInitialize` is called. Any attempt to use other floating-point formats will silently fail when using the PEX Protocol Method.

Lighting, Shading, and Depth Cueing

Texture Mapping

Texture mapping simulates a wide variety of surface material properties and detail for relatively modest costs in computation. Specific control of interior colors and transparency of an area primitive results from a special "mapped" correspondence between a texture image and 3D surface during rendering.

PEX support for texture mapping is currently under development by the PEX consortium, as noted in the *PEXlib Programming Manual* (16.1.9). However, HP is supporting CGE-extension texture mapping in the HP PEX 5.1v3 and 5.1v4 product releases. As such, a tutorial for texture mapping and the HP implementation details of the extension are covered in the later chapters of this book with the other advanced features (see Chapter 10).

3D Wireframe Modelling

If PowerShade is not installed, certain lighting and shading functions are disabled, leaving 3D wireframe functionality still available. Specific lighting and shading functions that are disabled in the wireframe configuration are:

- Lighting
- Surface- and polyline shading
- Alpha- or screen-door transparency
- Z-buffering
- Antialiasing
- Model clipping
- Facet distinguishing

In order for code to run unmodified on workstations regardless of PowerShade, errors as a result of calls to these non-supported functions are not reported. Also, the following attributes retain their default values despite attempts to change them:

- Facet distinguishing flag defaults to `FALSE`
- Model clip flag defaults to `PEXNoClip`

The following table illustrates the enumerated types and implementation-dependent constants that are different in the 3D wireframe configuration than when PowerShade is present.

Table 6-29. Enumerated Types and Implementation-Dependent Constants

LUT	HP PEX 5.1v4
Enumerated Types	
`PEXETHLHSRMode`	`PEXHLHSROff`
`PEXETPolylineInterpMethod`	`PEXPolylineInterpNone`
`PEXETSurfaceInterpMethod`	`PEXSurfaceInterpNone`
`PEXETReflectionModel`	`PEXReflectionNone`
Antialiasing GSE	Unsupported
Gamma Escape	Supported, but has no effect
Implementation-Dependent Constants	
`PEXIDTransparencySupported`	`False`

Hidden Line and Hidden Surface Removal

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib. See "Renderers", next.

Renderers

HP PEXlib offers immediate dynamics for all attributes, including Lookup Tables and Nameset contents.

This list describes the specifics of immediate dynamics for `hlhsr_mode`, if changed while the renderer is active:

Table 6-30. HLHSR Mode Transition Behaviors

To→ From↓	Off	Z Buffer	Z Buffer Read-Only	Z BufferID	Z BufferID Read-Only
Off		1	1	1	1
Z Buffer	3		2	2	2
Z Buffer Read-Only	3	2		2	2
Z BufferID	3	2	2		2
Z BufferID Read-Only	3	2	2	2	

1. Transitions from `Off` to any other mode may cause allocation of a Z Buffer and can potentially generate an allocation error. If traversal-time control of Z-Buffer comparisons is needed, HP recommends that a "Z BufferID" mode be used at `PEXBegin*` and `PEXSetHLHSRIdentifier` be used during traversal.
2. Switching between HLHSR modes other than `Off` can be done while the renderer is active. Z-buffer contents are preserved.
3. Transitions from any mode to `Off` may deallocate the Z Buffer and the contents may not be preserved.

The NPC Subvolume and Viewport

While this is standard PEX behavior, it is important to note that the viewport attribute value does not track the window size when `use_drawable` is `True`; therefore, inquiring it is not a substitute for standard ways of acquiring the window size such as `XGetWindowAttributes`.

Pipeline Contexts

There have been no additions by HP to this section.

Lookup Tables

Color Approximation

The color approximation lookup table (LUT) does not support arbitrary definitions of type `PEXColorSpace`. The default entry for the LUT may not represent the supported definition of that type for drawables of a particular depth and visual class. However, the predefined entry in a LUT that has been created for that visual and class does represent the supported definition.

Servers from other vendors may support arbitrary values. It is the application's responsibility to use the LUT appropriately. The color approximation LUT on most devices respects the dither flag. The implementation-dependent constant `DitheringSupported` is false on devices that never dither. It is `True` on devices where dithering is possible and on those devices the dither hint in the color approximation LUT entry may or may not have an effect, depending on the device. If it has no effect, then dithering is always used.

Color

When *realized* values are inquired from the color LUT, the values will be in the same color type used when the entry was specified.

Visible LUT behavior is specified in the table "LUT Default Entries" below. These are the values that are used when there isn't a current LUT of the specified type bound to a renderer, or if the entry being indexed does not exist and there is not any entry at the default index of the LUT.

The values shown in the table are also the values of the predefined entries for the LUTs in all cases except `Color` and `ColorApprox` LUTS. In the case of a `Color` LUT, entries 0–7 are defined as:

Table 6-31. Visible LUT Behavior

0	(0.0,0.0,0.0)	Black
1	(1.0,1.0,1.0)	White
2	(1.0,0.0,0.0)	Red
3	(1.0,1.0,0.0)	Yellow
4	(0.0,1.0,0.0)	Green
5	(0.0,1.0,1.0)	Cyan
6	(0.0,0.0,1.0)	Blue
7	(1.0,0.0,1.0)	Magenta

In the case of `ColorApprox` LUTs, the predefined entries depend on the drawable (visual) characteristics associated with the LUT.

Table 6-32. LUT Default Entries

LUT	HP PEX 5.1v2
LineBundle	
line_type	PEXLineTypeSolid
polyline_interp	PEXPolylineInterpNone
curve_approx	PEXApproxImpDep, (PEXApproxDCRelative), 1.0
line_width	1.0
line_color	{PEXColorTypeIndexed, 1}
MarkerBundle	
marker_type	PEXMarkerAsterisk
marker_scale	1.0
marker_color	{PEXColorTypeIndexed, 1}
TextBundle	
text_font_index	1
text_precision	PEXStrokePrecision
char_expansion	1.0
char_spacing	0.0
text_color	{PEXColorTypeIndexed, 1}
TextFont	
font	Roman8
View	
clip_flags	PEXClippingAll
clip_limits	(0.0, 0.0, 0.0), (1.0, 1.0, 1.0)
orientation	(identity matrix)
mapping	(identity matrix)

Table 6-32. LUT Default Entries (continued)

LUT	HP PEX 5.1v2
InteriorBundle	
interior_style	PEXInteriorStyleHollow
interior_style_index	1
surface_color	{PEXColorTypeIndexed, 1}
reflection_attr	{1.0, 1.0, 1.0, 0.0, 0.0 (PEXColorTypeIndexed, 1)}
reflection_model	PEXReflectionNone
surface_interp	PEXSurfaceInterpNone
bf_interior_style	PEXInteriorStyleHollow
bf_interior_style_index	1
bf_surface_color	{PEXColorTypeIndexed, 1}
bf_reflection_attr	{1.0, 1.0, 1.0, 0.0, 0.0 (PEXColorTypeIndexed, 1)}
bf_reflection_model	PEXReflectionNone
bf_surface_interp	PEXSurfaceInterpNone
surface_approx	{PEXApproxImpDep, (PEXApproxDCRelative), 1.0 ,1.0}
EdgeBundle	
surface_edges	PEXOff
surface_edge_type	PEXSurfaceEdgeSolid
surface_edge_width	1.0
surface_edge_color	{PEXColorTypeIndexed, 1}
Pattern	
	not supported

Table 6-32. LUT Default Entries (continued)

LUT	HP PEX 5.1v2
Light	
light_type	PEXLightAmbient
direction	(0, 0, 0) (not used for ambient light)
point	0.0 (not used for ambient light)
concentration	0.0 (not used for ambient light)
spread_angle	0.0 (not used for ambient light)
attenuation	0.0 (not used for ambient light)
color	{PEXColorTypeRGB, (1.0,1.0,1.0)}
DepthCue	
mode	PEXOff
front_plane	1.0
back_plane	0.0
front_scaling	1.0
back_scaling	0.5
color	{PEXColorTypeIndexed, 0}
ColorApprox	
approxType	PEXColorSpace
approxModel	PEXColorApproxRGB
max1	5
max2	5
max3	5
dither	PEXOff
mult1	36
mult2	6
mult3	1
weight1	1.0
weight2	0.0
weight3	0.0
basePixel	40
color	{PEXColorTypeRGB, (1.0,1.0,1.0)}

Namesets, Filters, and Searching

Hewlett-Packard makes no implementation-dependent additions to these functions of PEXlib.

Picking

HP PEX supports the `PEXPickLast` method for `PickOne` traversals and the `PEXPickAllAll` method for `PEXPickAll` traversals. Neither of these methods uses the current setting of HLHSR.

Echo and Highlighting Filter

The actual visual appearance of echo and highlight modes is an implementation-dependent choice that is not prescribed by the PEX and PEXlib specifications. The following are the echo and highlight attributes for HP PEXlib:

Echo Mode Attributes

- All primitive color attributes are set to the current echo color. The default echo color is white, but can be changed either by setting the `HPPEX_DHA_ECHO_COLOR` environment variable (before starting the client) or by calling `PEXSetEchoColor()`
- Line and surface edge types are solid
- Interior style is set to outline mode
- Drawing mode is "exclusive or" as described in the next section
- Lighting is disabled
- Line and surface color interpolation is disabled
- Antialiasing is disabled
- Depth cueing is disabled
- HLHSR computations are disabled
- Front- and backface facet distinguishing is disabled
- Alpha blending is disabled
- Line width is disabled

Highlight Mode Attributes

- All primitive color attributes are set to the highlight color. The default highlight color is white, but can be changed before the client is started by setting the `HPPEX_DHA_HIGHLIGHT_COLOR` environment variable or by calling `PEXHPOCCSetHighlightColor()`.
- Line color interpolation is disabled

Implications of The Exclusive Or Drawing Mode

HP PEXlib implements echo mode using "exclusive or" drawing mode for rapid display and erasure of echo images. This implementation causes the following behaviors:

- Draw and erase (`PEXEcho` and `PEXUnecho`) of primitives must be exactly paired to achieve the desired results. For example, consider a primitive that is first rendered with `PEXEcho` mode. A subsequent rendering with `PEXUnecho` mode erases the primitive expected, but a second `PEXUnecho` rendering causes the primitive to be displayed again rather than being erased as `PEXUnecho` implies.
- The actual echo color rendered varies from the specified echo color in different image locations based upon the frame buffer contents prior to echoing.

Error Handling

HP PEX prints additional information for PEX errors.

A Final Word About Writing Efficient Programs . . .

The discussion in Chapter 3 described how selection of the proper protocol method for the circumstances could improve performance (whether DHA, PEX, or X). In this section, the application developer can also improve performance by considering these general rules:

- Performance of HP PEXlib is improved if, in a series of OC commands, multiple OC targets are not alternated, or intermixed. The best performance is achieved when you send a series of OCs to one target, switch to the next, send series #2, switch, and so on. Using target one, then two, then three, then one, then two, then three, etc., causes multiple context switches and promotes thrashing.
- Similarly, avoid mixing attribute changes and primitives together, and avoid redundant attribute changes if possible.
- Namesets that start names at 0 and increment by 1 offer best performance.
- Do not use names higher than 1024 because some other implementations of PEX do not support these.

Also see Chapter 5 for hints on tuning performance. A variety of factors that affect performance have been documented in order to help an application writer better understand the variety of factors and tools that need to be considered when addressing performance issues. These performance hints are also available in the HP PEXlib on-line documentation.

Fast Macros

For higher performance, some of the PEXlib OC entry points are defined as macros instead of procedures. If it is necessary for your application that the PEXlib OC entrypoints be true procedures, you'll need to use the compile line option `-DHPPEX_PROCEDURES`. If your application requires one particular OC entrypoint to be a true procedure, then the C preprocessor statement, `#undef`⟨*entrypoint_name*⟩ can be used to undefine the macro for the named procedure after the `#include <X11/PEX5/PEXlib.h>`. When linked, the symbol will bind with the true procedure.

Using fast macros for an incremental gain in performance has some programming drawbacks. These are:

- You cannot take the address of a PEXlib entrypoint that is implemented as a macro. If the application had its own list that stored the address of PEXlib procedures, it would not compile while using fast macros.
- You *cannot* do this:

```
#define RENDER          display, resource_id, PEXOCRender

   PEXFillArea(RENDER, ...);
```

6

7 HP PEX 5.1v3—Selected 5.2 PEXlib Functionality

Overview of HP PEX5.1v3

This chapter is intended to give an overall view of what is new or different about HP-PEX in the 5.1v3 release. There are both new features and entrypoints, and changes in support for existing features.

Note that a newer release of HP-PEX is now available. For information on HP-PEX 5.1v4, see Chapter 8.

Background Information

Previous releases of HP-PEX have been PEX 5.1 based. The initial release (HP-PEX 5.1v1) was a richly featured, almost complete implementation of the PEX 5.1 standard, and had only a few extensions beyond the standard (most of them were related to double-buffering support). 5.1v1 was released in conjunction with HP-UX 9.01.

The second release (5.1v1.1) added support for the HP 712 systems, and included some defect fixes. This release was made available as a patch update to 9.01, but was most closely associated with HP-UX 9.03.

The third release (5.1v2) added support for the CGE PEX 1.0 extension set, including texture mapping, drafting primitives, and other features agreed upon by the participants in COSE. In addition, there were new HP extensions for capping, deformation, and other HP customer-requested features. Support was also added for the HCRX family of displays. This release was associated with HP-UX 9.05; in fact, HP-PEX runtime support was bundled with the basic system at this release.

HP PEX 5.1v3 includes some functionality and interfaces from the future PEX/PEXlib 5.2 standards. These features are being implemented in advance of the final standards, because HP believes that they will have significant value in many PEXlib applications. In some cases, minor differences between the HP implementation and the final PEX 5.2 standard may occur, but none should require more than very minor adjustments to make your application 5.2 conformant. It is important to note that 5.1v3 is *not* a complete PEX 5.2 implementation; instead, as the release name implies, it is PEX 5.1, plus CGE PEX 1.0 extensions, plus certain selected items from the PEX 5.2 *draft* standard. Some of these 5.2 features may be available only from HP for some time to come, so use of them is a consideration for portability and interoperability. Nevertheless, you may find them very valuable in the interest of performance, functionality, and experimentation with some important features of PEX/PEXlib 5.2.

Features have been added that improve performance as well as usability. See Chapter 5 here or in the on-line documentation.

Global Description of the HP-PEX 5.1v3 Release

Functionality Affecting Performance Improvements:

- OCC interface from the 5.2 PEXlib Specification
- Stride and Unpacked Data Models from the 5.2 PEXlib Specification
- Structure Permissions Accessible with `PEXSetStructurePermission()` from the PEXlib 5.2 Specification
- Documentation on “Performance Hints”

Additional 5.2 PEX Functionality

- Z-Buffer Read and Write Operations
- Plane Mask
- Drawing Mode

New Device and System Support

- HP VISUALIZE-8/-24/-48: New 3D Graphics Accelerated Devices
- DBE: Simple X Double-Buffering Extension

Programming Interfaces for Generating Output Commands

PEXlib Explicit Interface

PEXlib offers two major argument interfaces for the output command functions: the explicit interface and the output command context, or "OC Context" (OCC) interface.

The explicit interface is the interface defined for PEXlib 5.1 and is included in the PEXlib 5.2 specification for backwards compatibility, so PEXlib 5.1 programs compile with PEXlib 5.2 libraries. Although you may still use the explicit interface, you are encouraged to use the OCC interface to take advantage of the performance and functional improvements available in PEXlib 5.2. All output commands using the explicit interface use the same first three arguments:

`display`	Specifies the display connection.
`resource_id`	Specifies the resource identifier for the targeted renderer (for displaying output commands immediately) or structure (for storing output commands).
`req_type`	Specifies whether the application renders the output commands immediately (in which case the `resource_id` argument identifies the renderer resource), or stores the output commands in a structure (in which case the `resource_id` argument identifies the structure resource)

The explicit interface requires that you specify all arguments on every output command function you invoke.

7

PEXlib Output Command Context (OCC) Interface

The output command context interface (introduced with PEXlib 5.2) requires that the first argument always be the OC context, replacing the three arguments listed above. The OC context is an opaque structure that contains many of the arguments that are commonly found in the output command functions. Your application uses a set of special OCC manipulation functions to modify the fields in the opaque OC context. Then your application uses the OC Context in subsequent invocations of output command functions. The OCC interface uses far fewer arguments than the explicit interface, making coding easier and improving performance in some implementations.

Note: the attribute and primitive output commands introduced after 5.1 do not offer the explicit argument interface format. This is to encourage the use of the OCC argument interface in newer applications over the older explicit interface.

Flexible Data Formats

The OCC style functions for primitives that have facet and/or vertex data parameters allow you to supply the graphical data (coordinates, vertex attributes, facet attributes, and floating-point data) in packed, stride, or unpacked form. The packed form is the only form supported in PEXlib 5.1 and requires you to format the data into packed data structures defined by PEXlib. The stride form is more flexible and allows you to supply data formatted in application-defined structured arrays without the need to copy the data into the PEXlib-defined structures before invoking the PEXlib function. The unpacked form allows you to supply the data in separate lists for each data type.

Data Alignment

On many machine architectures, data alignment can be very important for performance and correct execution. For example, some machines require that pointers point to word boundaries when accessing word-length data items. Because the OCC interface allows you to control the offset applied to pointers for data accesses, you should be extra careful when specifying OCC pointers and offsets to ensure that you are following your machine's data alignment policies.

Output Command Context (OC Context or OCC)

The OC Context maintains the state of some common PEXlib arguments across PEXlib function calls. The OC context enables you to set the desired values of these arguments and then reuse them by specifying just the OC Context in several subsequent function calls. This eliminates the need to re-specify these same arguments in every function call, reducing redundancy and improving performance in some environments.

The OC Context itself is an opaque data structure with members that are referred to as values. Programs cannot alter the values of the OC Context directly. Instead, programs must use additional function calls to manipulate the values of the context. This ensures that the PEXlib implementation is informed whenever an OC Context value is changed.

Data Structures

The primary data structure used with the OC Context is the `PEXOCCValues` data structure which you use to set the value fields via the bitmask/value mechanism. It is important to realize that you do not alter the OC Context itself. You instead specify your desired changes in the `PEXOCCValues` data structure and use OCC functions to alter the OC Context with this data structure as input. The default values assigned to the OC Context, when your application creates it, are also given below:

```
typedef struct {                                          /* DEFAULT Values */
     Display          *display;                             /* undefined */
     PEXRenderer      renderer;                             /* undefined */
     PEXStructure     structure;                            /* undefined */
     PEXOCRequestType req_type;                             /* PEXOCRender */
     int              shape_hint;                           /* PEXShapeUnknown */
     int              ignore_edges;                         /* False */
     int              contour_hint;                         /* PEXContourUnknown */
     int              contours_all_one;                     /* False */
     unsigned int     facet_attributes;                     /* PEXGANone */
     unsigned int     line_vertex_attributes;               /* PEXGANone */
     unsigned int     marker_vertex_attributes;             /* PEXGANone */
     unsigned int     surface_vertex_attributes;            /* PEXGANone */
     unsigned int     edge_attributes;                      /* PEXGANone */
     unsigned int     facet_fp_data_count;                  /* 0 */
     unsigned int     line_vertex_fp_data_count;            /* 0 */
     unsigned int     marker_vertex_fp_data_count;          /* 0 */
     unsigned int     surface_vertex_fp_data_count;         /* 0 */
     int              color_type;                           /* PEXColorTypeRGB */
     char             *encoding_state;                      /* NULL Pointer */
     int              data_model;                           /* PEXDataPacked */
     union {
           PEXOCCStrideData     stride;
           PEXOCCUnpackedData   unpacked;
     } data_model_specs;
} PEXOCCValues;
```

```
typedef struct {                                        /* all members 0 */
     int              facet_stride;
     int              vertex_stride;
     int              facet_color_offset;
     int              facet_normal_offset;
     int              facet_fp_data_offset;
     int              vertex_coord_offset;
     int              vertex_color_offset;
     int              vertex_normal_offset;
     int              vertex_edge_offset;
     int              vertex_radius_offset;
     int              vertex_axes_offset;
     int              vertex_angle_offset;
     int              line_vertex_fp_data_offset;
     int              marker_vertex_fp_data_offset;
     int              surface_vertex_fp_data_offset;
     } PEXOCCStrideData;

typedef struct {                                        /* all members 0 */
     int              facet_color_size;
     int              facet_normal_size;
     int              facet_fp_data_count;
     int              vertex_coord_size;
     int              vertex_color_size;
     int              vertex_normal_size;
     int              vertex_edge_size;
     int              vertex_radius_size;
     int              vertex_axes_size;
     int              vertex_angle_size;
     int              line_vertex_fp_data_count;
     int              marker_vertex_fp_data_count;
     int              surface_vertex_fp_data_count;
     } PEXOCCUnpackedData;
```

The semantics of each member in the `PEXOCCValues` structure is the same as it is in the explicit interface.

Specify all size and offset members in terms of bytes. The ⟨*type*⟩`_fp_data_count` members represent the number of floating point values.

The `encoding_state` member is reserved for use by future internationalized text functions to retain character encoding state for languages that require it.

The `data_model` member indicates the data model used to pass vertex and facet data to OCC primitive functions. You should set this member to `PEXDataPacked`, `PEXDataStride`, or `PEXDataUnpacked`.

The OCC itself is defined as:

```
typedef struct _PEXOCC *PEXOCC;
    {
        XExtData     *ext_data;          /* hook for extension */
        /* PEXlib private data */
    }
#endif
*PEXOCC;
```

You cannot access any members within the OCC directly. You must simply pass the OCC to functions that modify or use the contents of the context structure.

Sample Usage of the OCC

To invoke the OC functions that use the OC Context, your application must first create an OC Context with the `PEXCreateOCC` function. This function returns an OC Context initialized with default values for future reference. The application can change the default context values when it creates the OC Context and/or can change values later by invoking the `PEXChangeOCC` function. These two functions set the value fields via a bitmask/value mechanism and are suitable when you are setting several values at once. There are also OC Context convenience functions that, through a simpler interface, enable you to set only one value per function in the OCC.

Here is a coded example of how to use the OC Context. The last two statements illustrate the difference between an OCC function and a non-OCC function:

```
PEXOCCValues        ocvalues;
unsigned int        mask;
PEXOCC              myocc;

ocvalues.display = my_display;
PEXSetOCCValueMask(&mask, PEXOCCMDisplay);
ocvalues.renderer = my_renderer;
PEXSetOCCValueMask(&mask, PEXOCCMRenderer);
ocvalues.color_type = PEXColorTypeRGB;
PEXSetOCCValueMask(&mask, PEXOCCMColorType);
ocvalues.surface_vertex_attributes = PEXGAColor;
PEXSetOCCValueMask(&mask, PEXOCCMSurfaceVertexAttributes);

myocc = PEXCreateOCC(&mask), &ocvalues);

PEXOCCTriangleStrip(myocc, NULL, count, vertices);

/* This is the old, explicit interface */
PEXTriangleStrip(display, resource_id, req_type, PEXGANone,
      PEXGAColor, 0, 0, PEXRGBFloat, NULL, count, vertices);
```

Invoke the PEXFreeOCC function to deallocate the memory associated with the OC Context.

Facet/Vertex Data Formats

There are three ways to provide facet and vertex data to PEXlib:

Packed Data Format (PEXlib 5.1 method)

Unlike the stride and unpacked formats, you can use the packed data format with either OCC primitive functions or with non-OCC primitive functions. The use of the packed data format with non-OCC functions, as with PEXlib 5.1 functions, is as follows (this example has no facet data and supplies colors and normals in the vertex data):

```
PEXVertexRGBNormal          my_vertices[20];
PEXArrayOfVertex            verts;

/* code to init my_vertices */

verts.rgb_normal = my_vertices;

PEXTriangleStrip(display, rend, PEXOCRender, 0,
                 PEXGAColor | PEXGANormal, PEXColorTypeRGB,
                 NULL, 20, verts);
```

You can also use packed data format with OCC style functions, which you may find useful when converting older code to use the OCC style functions:

```
PEXVertexRGBNormal        my_vertices[20];

/* code to init my_vertices */

PEXOCCTriangleStrip(context, NULL, 20, my_vertices);
```

This usage assumes that you have already initialized the OC Context fields with the correct values. In particular, you need to set the `display`, `renderer` (or `structure`), `req_type`, `surface_vertex_attributes`, `color_type`, and `data_model` (`PEXDataPacked`, of course) fields. Unless these fields change, you only need to set these once. You do not need to set any fields in the `data_model_specs` union.

Also, you do not need the `PEXArrayOfVertex` union to pass the vertex data in the function. The type of the facet and vertex data parameters is `PEXPointer`, which allows you to pass any type in this parameter. This is important for the stride interface described in the next section. However, for both facet and vertex data, you are responsible for making sure that the data you are passing accurately reflects the attributes that you set in the facet or vertex attribute fields.

PEXlib does not supply a set of structure type definitions for facet or vertex data that include floating-point data because there is no way to determine how many floating point numbers you want to supply with a facet or vertex. Therefore, you may find it convenient to design your own data structure to pass in the facet or vertex parameters of some OC functions when using the `PEXDataPacked` data model. You need to design the data structure with the following ordering rules in mind.

For facet data, the required order is:

`PEXColor*`	One of the PEXlib color types, if provided.
`PEXVector`	Normal, if provided.
`float`[n]	Floating point data, if provided.

For vertex data, the required order is:

`PEXCoord`	(or `PEXCoord2D`) Center.
`PEXColor*`	One of the PEXlib color types, if provided.
`PEXVector`	Normal, if provided.
`unsigned int`	Edges, if provided.
`float`[n]	Floating point data, if provided.

As an example, if you wish to supply vertex data for FillArea using 3D coordinates, RGB floating-point colors and three floating-point numbers (for texture data), you would create a structure like this:

```
typedef struct {
    PEXCoord        center;
    PEXColorRGB     rgb;
    float           texture_data[3];
} MY_PEXRGBTextureVertexData;
```

Remember to set the correct surface vertex attributes in the OCC. In this case you would make sure that the `PEXGA2D` flag is off, the `PEXGAColor` is on and the `PEXGAFloatData` flag is on. You set the `color_type` field in the OCC to `PEXColorTypeRGB` and the number of floats in the OCC (`surface_vertex_fp_data_count`) to three.

You may then pass the address of an array of these structures directly in the OCC form of a function:

```
MY_PEXRGBTextureVertexData      my_vertices[20];
/* code to init my_vertices */
PEXOCCTriangleStrip(context, NULL, 20, my_vertices);
```

Stride Data Format

The stride format allows you to access the facet and vertex data directly from application data structures if the data is arranged in a "structured array" format, where the application stores facet or vertex data in an array of structures; each structure in the structured array corresponds to data for a single facet or vertex along with other application specific data. You specify the size, or "stride" of each array element and the offsets of each of the facet or vertex data items within the array element. Using this format, you eliminate the need to copy the facet or vertex data from the application data structure to an array of PEXlib packed data structures.

The stride format was initially designed because applications often have data stored in different structures than the PEXlib OC entry points require. Sometimes application data is stored in a structure that represents a single vertex, or sometimes it is in a structure that represents all the colors that will be used. Often this data format does not change or changes infrequently. For an implementation of PEXlib this means that the specific data values will not need to be checked to see if they have changed which can improve performance in DHA, immediate mode.

Here is an illustration of the stride model, using vertex data:

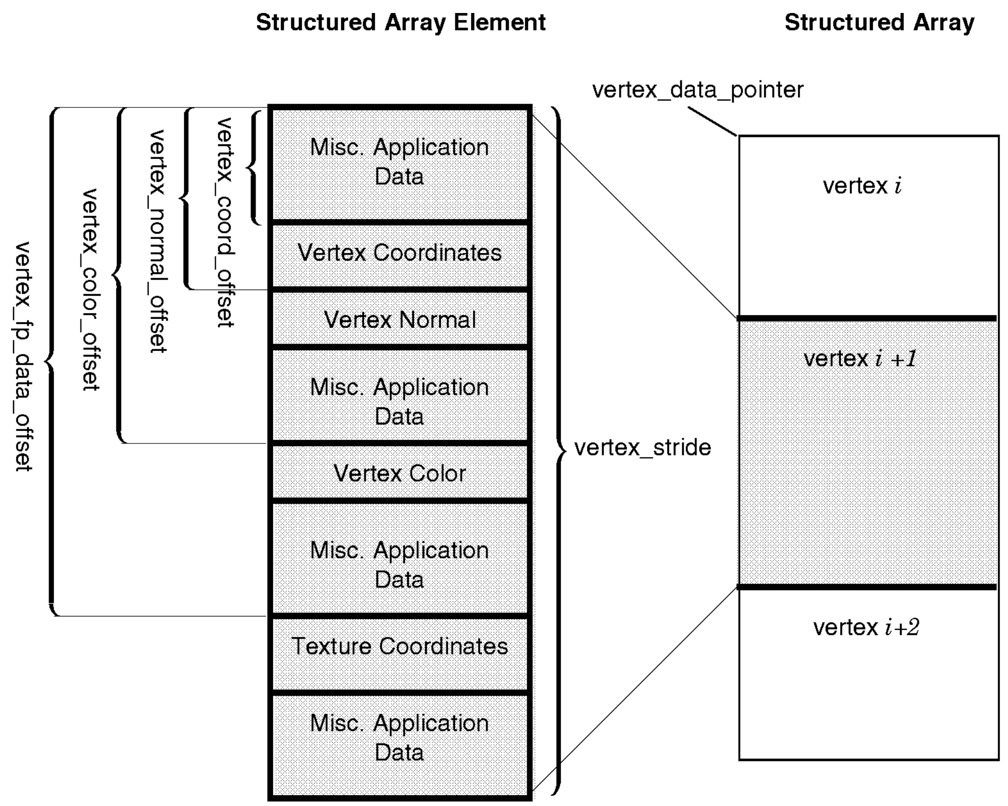


Figure 7-1. Stride model, with vertex data

In this example, you supply a pointer to the beginning of the structured array in the function call.

```
PEXOCCTriangleStrip(context, NULL, 20, vertex_data_pointer);
```

To use the `PEXDataStride` format, you need to set some additional members in the OC Context to inform PEXlib of the details of your data's format. The following steps use the "bitmask/value" method of initializing the OC Context. You may instead create the OC Context first and then use the convenience functions, or use a combination of the two approaches.

- Fill in the appropriate members of a `PEXOCCValues` data structure:
  - Assign all the members of the `PEXOCCStrideData` data structure that are applicable to your data. You only need to set the members for the vertex or facet data that you are actually using. The `PEXOCCStrideData` structure is a part of the `data_model_specs` union, so you should use the `stride` union member to access this structure.
  - Assign the value `PEXDataStride` to the `data_model` member of the `PEXOCCValues` structure.
  - Assign other members of the `PEXOCCValues` structure such as `display`, `renderer`, `color_type`, etc., as discussed in previous sections.
- Use the `PEXSetOCCValueMask` function to set a bit in a bitmask that corresponds to every member you initialize in the `PEXOCCValues` data structure.
- Use the `PEXOCCValues` data structure and the bitmask you have initialized to create an OC Context with the `PEXCreateOCC` function.

Once the OCC is created and initialized, you may use it multiple times to draw primitives using graphic data whose form is described by the `PEXOCCStrideData` structure you have defined. The OCC-style primitive functions accept data parameters of type `void*`, so you may pass a pointer to your data that is of any type, including application-specific types.

Some primitives, such as Fill Area Set and Polylines (Polyline Set) are defined by a nested list of vertices. Since you cannot work directly with a nested list of vertices, you use the "`PEXListOfVertexData` data structure", which looks like:

```
typedef struct {
    unsigned long  count;
    PEXPointer     vertices;
} PEXListOfVertexData;
```

For each fill-area-set contour or polyline in the set, you allocate one of these structures and fill it in with the number of points in that contour or line and a pointer to the first vertex in the list. Note that the type of the vertices pointer is `PEXPointer`, so you set this pointer to point to a structured array, just as in the non-nested case. The list of these `PEXListOfVertexData` structures must be contiguous and you pass the address of the first one in the vertex data argument of the primitive function call.

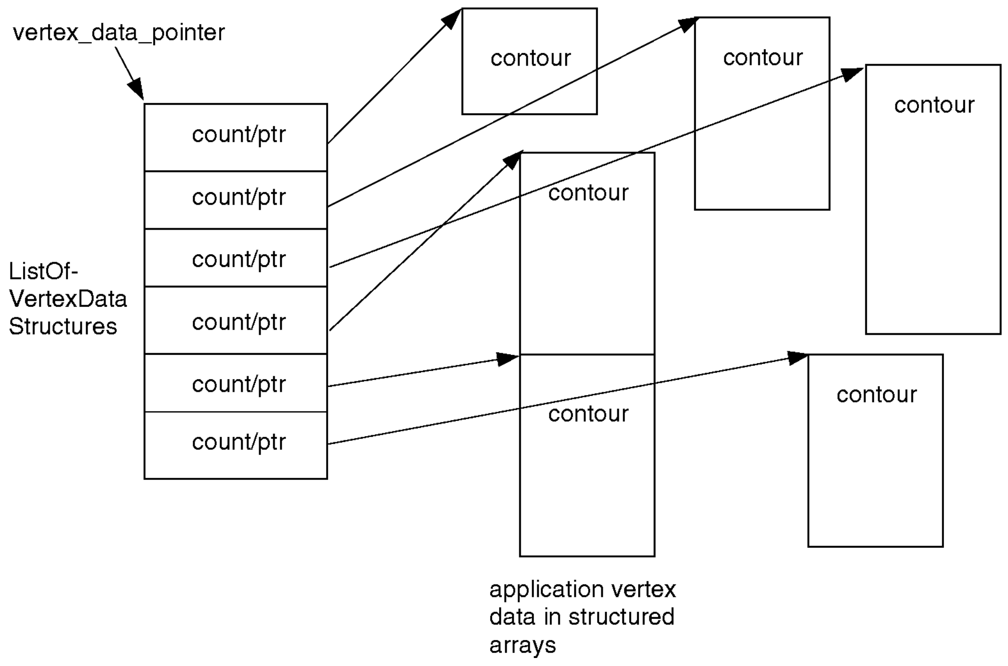


Figure 7-2. Application Vertex Data

Unpacked Data Format

You use this data format when your data is arranged in lists or arrays of points, colors, normals, etc. Each of these lists can reside anywhere in memory, since you supply a pointer to the start of each list. You also must specify the size of each element in the list by using the PEXOCCUnpackedData member of the data_model_specs union.

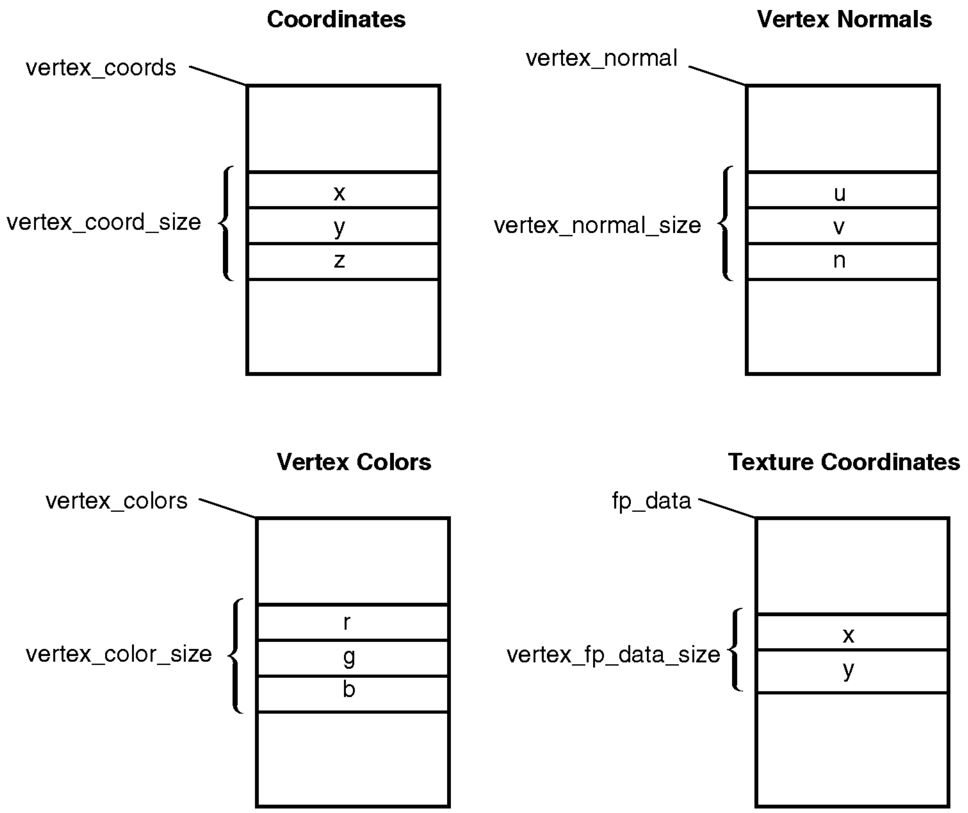


Figure 7-3. Unpacked Data Model

Using the PEXDataUnpacked format is similar to using the PEXDataStride format, except:

- Set the data_model member to the value of PEXDataUnpacked.
- Set the appropriate members of the PEXOCCUnpackedData member of the data_model_specs union with the values that reflect your data's organization. You only need to set the members for the vertex or facet data components that you are actually using.

For unpacked data, you need to pass pointers to the beginning of each list of data by placing them in the following structure and passing the address of this structure in the PEXlib function.

```
typedef struct {
    PEXPointer    coords;
    PEXPointer    colors;
    PEXPointer    normals;
    PEXPointer    edges;
    PEXPointer    radii;
    PEXPointer    axes;
    PEXPointer    angles;
    PEXPointer    fp_data;
} PEXUnpackedVertexData;
```

An example of using the unpacked data pointers:

```
PEXUnpackedVertexData          verts;

verts.coords = my_coords;
verts.colors = my_colors;
verts.normals = my_normals;
verts.fp_data = my_fp_data;
PEXOCCTriangleStrip(context, NULL, count, &verts);
```

When working with facet data, use the PEXUnpackedFacetData data structure.

Some primitives, such as fill area set and polylines (polyline set) are defined by a nested list of vertices. Since you cannot work directly with a nested list of vertices, you use the PEX `ListOfVertexData` data structure, which looks like:

```
typedef struct {
    unsigned long       count;
    PEXPointer          vertices;
} PEXListOfVertexData;
```

For each Fill Area Set Contour or Polyline in the set, you allocate one of these structures and fill it in with the number of points in that contour or line and a pointer to the first vertex in the list. Note that the type of the vertices pointer is `PEXPointer`, so you set this pointer to point to a `PEXUnpackedVertex` data structure, just as in the non-nested case. The list of these `PEXListOfVertexData` structures must be contiguous and you pass the address of the first one in the vertex data argument of the primitive function call.

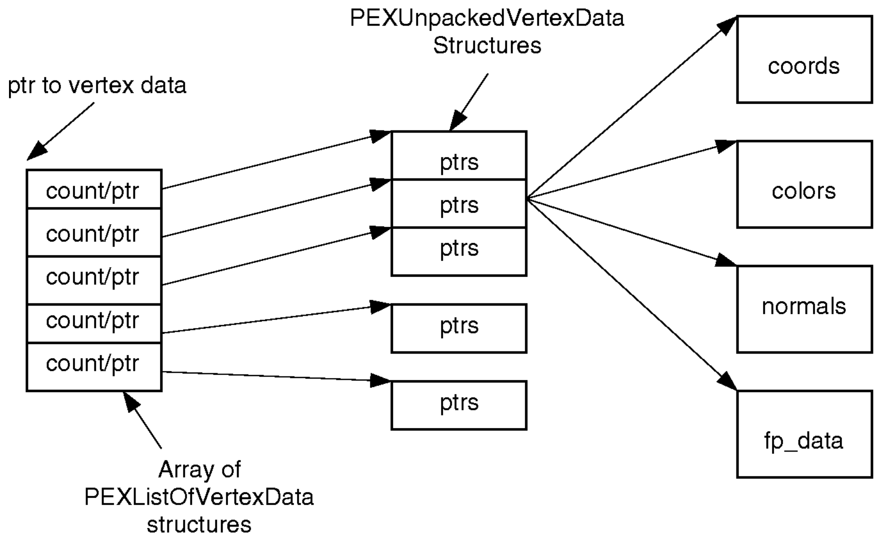


Figure 7-4. PEX Unpacked Vertex Data Structures

Errors and Output Command Errors

The PEX output commands are designed so that the PEX implementation can often continue to process an OC with an error in it by using defaults or fall-back values. However, in some cases, it is not practical to define a reasonable fall-back value and so the PEX implementation stops processing OCs and generates an error, usually a `BadPEXOutputCommand` error.

Many `BadPEXOutputCommand` errors are very specific to the output command and are listed with the function. There are another set of errors that are common to a number of output commands and are not generally listed with each OC function.

`BadPEXOutputCommand`: Most cases are listed with each OC. Some of the causes common to many OCs are:

- An argument expected to contain a value from a fixed enumeration contains an undefined value. An example of this is the `text_path` text attribute. If you stick with the listed possibilities, or possibilities allowed by an extension, then you should not get errors.
- The color data is not in a format supported by the PEX implementation. Since a large number of output commands use color data, this error is not listed with each OC function.
- Setting a bit in a bitmask to one when that bit is defined as unused. To avoid errors, set only bits that have been defined by PEXlib, though some PEX 5.1 implementations may tolerate the setting of undefined bits.

Run-Time Errors

The `PEXOCCSet*` functions ignore undefined bits in the mask argument. Any error in setting the OC context will not become apparent until the OCC is actually used in an OC function that uses the OC context.

`PEXOCCSet` Functions will return a bad status if the OCC functions do return a status, then the application can check for an error.

Simplified OCC functions for Primitive OCs

Beginning with PEXlib release 5.1v3, the output command context (OCC) makes it possible to generate primitives that use a wide variety of attributes with a smaller number of functions. The following table describes what OCC functions to use to generate the corresponding primitive in terms of the non-OCC function form. Primitives added after PEXlib release 5.1v3 will, in general, be accessible only via the OCC function format. In the 5.1v3 release, HP PEXlib does not support the `PEXGA2D` flag in fill area OCC primitives.

Table 7-1.
Relationship Between OCC and Non-OCC Primitive Functions

OCC Form	Non-OCC Form
PEXOCCAnnotationText	PEXAnnotationText
PEXOCCAnnotationText2D	PEXAnnotationText2D
PEXOCCCellArray	PEXCellArray
PEXOCCCellArray2D	PEXCellArray2D
PEXOCCEncodedAnnoText	PEXEncodedAnnoText
PEXOCCEncodedAnnoText2D	PEXEncodedAnnoText2D
PEXOCCEncodedText	PEXEncodedText
PEXOCCEncodedText2D	PEXEncodedText2D
PEXOCCExtendedCellArray	PEXExtendedCellArray
PEXOCCFillArea	PEXFillArea
PEXOCCFillArea	PEXFillArea2D
PEXOCCFillArea	PEXFillAreaWithData
PEXOCCFillAreaSet	PEXFillAreaSet
PEXOCCFillAreaSet	PEXFillAreaSet2D
PEXOCCFillAreaSet	PEXFillAreaSetWithData
PEXOCCGDP	PEXGDP
PEXOCCGDP	PEXGDP2D
PEXOCCIndexedFillAreaSets	PEXSetOfFillAreaSets
PEXOCCMarkers	PEXMarkers
PEXOCCMarkers	PEXMarkers2D

Table 7-1.
Relationship Between OCC and Non-OCC Primitive Functions (continued)

OCC Form	Non-OCC Form
PEXOCCPolyline	PEXPolyline
PEXOCCPolyline	PEXPolyline2D
PEXOCCPolylines	PEXPolylineSetWithData
PEXOCCQuadrilateralMesh	PEXQuadrilateralMesh
PEXOCCText	PEXText
PEXOCCText2D	PEXText2D
PEXOCCTriangleStrip	PEXTriangleStrip

Generating HP PEX 5.1 Output Commands

PEX 5.2 introduces five primitive output commands in the protocol definition (PEXOCFillAreaSetWithDataFP, PEXOCPolylineSetWithDataFP, PEXOCQuadrilateralMeshFP, PEXOCSetOfFillAreaSetsFP, PEXOCTriangleStripFP) that are similar to their 5.1 counterparts, except that they support optional floating-point values in the facet and vertex data. By default, the PEXlib 5.2 OCC-style functions generate these "FP" forms when connected to a PEX 5.2 server, or to an HP server support release 5.1v3. However, if connected to another vendor's PEX 5.1 server, HP-PEXlib generates the 5.1 version of these output commands. If HP-PEXlib is generating a 5.1 version of the opcode and you supply data via the OCC-style function that is in conflict with the capabilities of the PEX 5.1 output command, you may cause an output command error.

Note that you may also always generate PEX 5.1 output commands by using the non-OCC forms of the functions.

Examples

Below is an example illustrating two ways to set up for rendering a `PEXOCCTriangleStrip` using the `PEXDataStride` data model with an array of application defined vertex data structures. Note that the OCC setup is required only one time provided that nothing in the data format changes. The OCC can then be re-used on all subsequent triangle strip primitives.

```
/* Following is a sample definition of an application's data structure
   containing vertices, colors, normals, and other application data. */
typedef struct {
    PEXVector      normal;
    PEXColorRGB    color;
    PEXCoord       point;
        ⋮ (Other application data)
} app_vertex_def;

/* Define an array of app_vertex_def structures */
app_vertex_def       *app_vertices;
{
    Display          *dpy;
    PEXRenderer      rdr;
    PEXOCC           tri_context;
    PEXOCCValues     occ_values;
    unsigned long    occ_mask = 0;
```

7

```
/*   Method #1:
 *   Set the OCCValues structure for OCC context creation.
 *   Note that for convenience in this example, we utilize
 *   the default values for:
 *   facet_attributes      (PEXGANone),
 *   req_type              (PEXOCRender), and
 *   color_type            (PEXColorTypeRGB).  */
occ_values.display = dpy;
PEXSetOCCValueMask(&occ_mask, PEXOCCMDisplay);
occ_values.renderer = rdr;
PEXSetOCCValueMask(&occ_mask, PEXOCCMRenderer);
occ_values.surface_vertex_attributes = PEXGAColor | PEXGANormal;
PEXSetOCCValueMask(&occ_mask, PEXOCCMSurfaceVertexAttributes);
occ_values.data_model = PEXDataStride;
PEXSetOCCValueMask(&occ_mask, PEXOCCMDataModel);
occ_values.data_model_specs.stride.vertex_stride      = sizeof(app_vertex_def);
occ_values.data_model_specs.stride.vertex_coord_offset  = sizeof(PEXVector) +
                                                         sizeof(PEXColorRGB);
occ_values.data_model_specs.stride.vertex_color_offset  = sizeof(PEXVector);
occ_values.data_model_specs.stride.vertex_normal_offset = 0;
PEXSetOCCValueMask(&occ_mask, PEXOCCMDataModelSpecs);
/* Create the OC context */
tri_context = PEXCreateOCC(&occ_mask, &occ_values);

/*   Method #2
 *   Note that the OCC Convenience functions could also have been used to set
 *   up the OCC Context as below.  In this method, the OCC context is created
 *   with all default values, and then the convenience functions are used to
 *   change selected fields. */
PEXOCCStrideData stride_data;

tri_context = PEXCreateOCC(&occ_mask, &occ_values);
PEXSetOCCDisplay(tri_context, dpy);
PEXSetOCCRenderer(tri_context, rdr);
PEXSetOCCSurfaceVertexAttributes(tri_context, PEXGAColor|PEXGANormal);
PEXSetOCCDataModel(tri_context, PEXDataStride);

stride_data.vertex_stride         =  sizeof(app_vertex_def);
stride_data.vertex_coord_offset   =  sizeof (PEXVector) + sizeof (PEXColorRGB);
stride_data.vertex_color_offset   =  sizeof (PEXVector);
stride_data.vertex_normal_offset  =  0;

PEXSetOCCDataModelSpecs(tri_context, &stride_data);
```

```
/*  Malloc space for NUM_TRI_POINTS application vertices */
app_vertices=(app_vertex_def*)malloc((NUM_TRI_POINTS)*sizeof(app_vertex_def));

/*  Fill in vertex structures */
for(i = 0; i << NUM_TRI_POINTS; i++)
    {
    app_vertices[i].point   =  vertex coordinate data;
    app_vertices[i].color   =  vertex color data;
    app_vertices[i].normal  =  vertex normal data;
        :  (Fill in any other application vertex data)
}

/*   Render the triangle strip
 *   Note the PEXPointer cast on the app_vertices argument */
PEXOCCTriangleStrip(tri_context,                /* Triangle OCC */
                    (PEXPointer) NULL,          /* Facet Data   */
                    NUM_TRI_POINTS,             /* Num Points   */
                    (PEXPointer) app_vertices); /* Vertices     */

/*   If we want to render the triangle strip again but without colors and
 *   normals, then we only need to set the surface vertex attributes in the
 *   OC Context and call PEXOCCTriangleStrip() again. */

PEXSetOCCSurfaceVertexAttributes(tri_context,PEXGANone);
PEXOCCTriangleStrip(tri_context,                 /* Triangle OCC */
                    (PEXPointer)NULL,            /* Facet Data   */
                    NUM_TRI_POINTS,              /* Num Points   */
                    (PEXPointer)app_vertices);   /* Vertices     */
```

Below is the same example, this time illustrating the `PEXDataUnpacked` formatting method to be used in rendering.

```
/*  Following are the application vertex, color, and normal arrays. */
PEXVector     *normals;
PEXColorRGB   *colors;
PEXCoord      *vertices;
{
    Display         *dpy;
    PEXRenderer     rdr;
    PEXOCC          tri_context;
    PEXOCCValues    occ_values;
    unsigned long   occ_mask = 0;

/*  Note that the OCC Convenience functions are used to set up the OCC Context
 *  as below.  In this method, the OCC context is created with all default
 *  values, and then the convenience functions are used to change selected
 *  fields. */
PEXOCCUnpackedVertexData    unpacked_vertices;
PEXOCCUnPackedData          unpacked_data;

tri_context = PEXCreateOCC(&occ_mask, &occ_values);
PEXSetOCCDisplay(tri_context, dpy);
PEXSetOCCRenderer(tri_context, rdr);
PEXSetOCCSurfaceVertexAttributes(tri_context, PEXGAColor|PEXGANormal);
PEXSetOCCDataModel(tri_context, PEXDataUnpacked);
unpacked_data.vertex_coord_size = sizeof(PEXCoord);
unpacked_data.vertex_color_size = sizeof(PEXColorRGB);
unpacked_data.vertex_normal_size = sizeof(PEXVector);
PEXSetOCCDataModelSpecs(tri_context, &unpacked_data);

/* Malloc space for NUM_TRI_POINTS application vertices, colors, and normals */
vertices  =  (PEXCoord*)malloc((NUM_TRI_POINTS)*sizeof(PEXCoord));
colors    =  (PEXColorRGB*)malloc((NUM_TRI_POINTS)*sizeof(PEXColorRGB));
normals   =  (PEXVector*)malloc((NUM_TRI_POINTS)*sizeof(PEXVector));

/* Fill in vertex, color, and normal data */
for (i=0; i<< NUM_TRI_POINTS; i++) {
    vertices[i] = vertex coordinate data;
    colors[i]   = vertex color data;
    normals[i]  = vertex normal data;
}
/* Fill in the pointers to vertices, colors, and normals in the unpacked
 * vertices structure. */
unpacked_vertices.coords   =  vertices;
unpacked_vertices.colors   =  colors;
unpacked_vertices.normals  =  normals;

/*  Render the triangle strip
 *  Note the PEXPointer cast on the app_vertices argument */
PEXOCCTriangleStrip(tri_context,                       /* Triangle OCC  */
                    (PEXPointer) NULL,                 /* Facet Data    */
                    NUM_TRI_POINTS,                    /* Num Points    */
                    (PEXPointer) unpacked_vertices);   /* Vertices      */
}

/*   If we want to render the triangle strip again but without colors and
 *   normals, then we only need to set the surface vertex attributes in the OC
 *   Context and call PEXOCCTriangleStrip() again. */
PEXSetOCCSurfaceVertexAttributes(tri_context, PEXGANone);
PEXOCCTriangleStrip(tri_context,                      /* Triangle OCC  */
                    (PEXPointer)NULL,                 /* Facet Data    */
                    NUM_TRI_POINTS,                   /* Num Points    */
                    (PEXPointer)unpacked_vertices);   /* Vertices      */
```

Structure Permissions

Introduction

Structure permissions control access to a structure so performance optimization can take place. The application developer is signaling PEXlib that the given structure or element is static and should now be altered to improve performance. The two levels of permission are write-only and locked. The structure can be made write-only to allow only additions and replacements or it can be locked so no editing is allowed.

Background Information

The motivation for this feature is to improve performance without burdening the developer with implementation-dependent knowledge. After a structure is locked or made write-only an implementation is free to operate on the structure to improve performance. Permissions are easy to use but require some planning by the application developer to deal with related issues.

This creates another reason for developers to consider using structures because they can achieve performance gains and still have interoperability. Because there are two levels of permission in PEX structures there are some additional benefits for using PEX.

Performance optimizations cannot be automatically applied in immediate mode, so implementing them in structures increases the performance gap between immediate mode and structure mode (traversals already run somewhat faster than the same rendering via immediate mode). Most applications choose between these modes based on one or more of the following considerations: rendering performance, data stability, and data size.

Using Permission Features

Here is an overall summary of the edit operations that are allowed on structures with the various permissions:

Table 7-2. Allowed Structure Editing Operations

Operation	ReadWrite	WriteOnly	Locked
Move element pointer	allowed	allowed	error
copy elements into structure	allowed	allowed from another WriteOnly structure	error
delete elements	allowed	allowed	error
insert elements	allowed	allowed	error
overwrite elements	allowed	allowed	error
change structure references	allowed	allowed	allowed

To get the maximum benefits from this feature, the design of the application must accommodate the constraints of permissions. After a structure is locked it may not be edited. But many applications mix rendering and editing of structures.

The main reason to lock a structure is to increase its rendering speed. If the data in the structure is static for a long period of time (e.g. while a model is being interactively viewed but not modified), then locking is probably a worthwhile performance improvement. Perhaps even an entire structure hierarchy or sub-hierarchy may be locked. Some examples where static data might be viewed include:

- Walkthrough of an architectural model;
- Animation of a non-articulated object, e.g., for a video production;
- Interactive manipulation of a part or an assembly of parts, e.g., to show another designer what has been created in an MCAD package.

Tradeoffs and careful choices need to be made in cases where the data in structures may be modified, especially at interactive or animation speeds. Once a structure is locked, it cannot be unlocked, so the application must have a way to regenerate the data in a ReadWrite or WriteOnly structure.

An application that supports an alternating edit/view cycle of user interaction might operate on an editable hierarchy during the edit session, and then create a separate locked structure hierarchy while viewing or animating. However, the amount of extra data space required for the extra copy must be traded off against the performance improvement. One strategy is to keep a ReadWrite copy of the structure and edit it, then lock it after editing. Another strategy is to make the structure WriteOnly, retaining the ability to add and delete elements; this would allow only some performance optimizations to be performed, but no duplication of storage (such as is implied by keeping a separate Locked structure) would be required. Whatever strategy is used, applications must be designed to preserve or re-create structures or elements, changing of permissions must be optimized and the partitioning of structures must be well designed.

Here is another example: In an animated, articulated model (such as a robot arm), it would be impractical to lock the structures containing the modeling matrices that control the positions of dependent parts of the arm. It would be better to keep the actual attributes and primitives that make up each part of the arm in separate structures that can be locked, and keep the modeling matrices in editable structures.

An extra consideration arises in the case of an application that is going to perform picking on its structure hierarchy. WriteOnly permission preserves the element offsets of all elements in the structure, so a path returned by picking on a WriteOnly structure can be used for editing. Locked structures preserve offsets to execute structure commands however, primitive element offsets are not guaranteed to be preserved in a Locked structure, and several primitive elements may be merged and so become indistinguishable. This means that the path returned when a hit occurs on a primitive in a Locked structure is valid for that (locked) structure hierarchy, but may not be useful to guide editing on the ReadWrite version of the same structure. Therefore, it is recommended that picking traversals be done on ReadWrite or WriteOnly structures, even if the image with which a user might interact during picking was created with a Locked structure.

All these strategies come together when a complex model is built and modified by picking pieces to modify. The following outline shows how this might be done by keeping a ReadWrite copy of the structure.

```
PEXCreateStructure A
PEXCreateStructure ALocked
    :                                  (add elements or edit existing elements of structure
                                       A)
while( event )

VIEW:
    PEXCopyStructure A->ALocked
    PEXSetStructurePermission ALocked
    PEXStructureLocked
    PEXRenderNetwork ALocked           (rendering performance is optimum at this point
                                       because optimization was done)
PICK:
set pick aperture
path = PEXPickOne( A )
PEXBeginRendering
PEXAccumulateState( path )             (the path returned by PEXPickOne can be used to
                                       change line color or to redraw the picked element in
                                       some other color)
PEXRenderElements( path )
PEXEndRendering                        (the old structure is destroyed because it cannot
                                       be changed to ReadWrite and copying to it is not
                                       allowed)
PEXDestroyStructure ALocked
PEXFreePath(path)                      (exit PICK with an editable structure "A" that can be
                                       further edited and locked again as shown at the start
                                       of this example)
```

Changes to Existing Functionality

Pick Path

Picking and `PEXAccumulateState` continue to function as before even though they may now be traversing locked structures. The pick path that the picking operations return is usable by `PEXAccumulateState` applied to the same structures. It is not necessarily transferable to ReadWrite versions of the structures, because elements offsets of hit primitives may change.

It is possible to pick "through" a locked structure because `PEXExecuteStructure` elements will always be preserved in the locked structure.

PEXGetStructureInfo

The structure that is returned has an added field for the permission values. This is the new structure:

```
typedef struct {
    unsigned long    element_count;
    unsigned long    size;
    Bool             has_refs;
    unsigned short   edit_mode;
    unsigned long    element_pointer;
    unsigned short   structure_permission;
} PEXStructureInfo;
```

New error condition for attributes and primitives

Any of the OCs can now return a `BadPEXStructurePermission` error when the req_type is `PEXOCStore` or `PEXOCSingleStore`, if the referenced structure is locked. This affects both the old form and the new OCC form of OCs.

This also affects the CGE and HP OC extensions.

Z-Buffer Block Operations

Introduction

HP PEXlib 5.1v3 offers various ways to read blocks from the Z-buffer or write blocks into the Z-buffer.

There are two main reasons why you might want to read/write the Z-buffer. One is so you can implement your own picking that considers the Z-buffer state. The second reason is to save/restore either the entire window or only a section of the window. When saving/restoring 3D graphics, the Z-buffer needs to be saved/restored along with the frame buffer. An example implementation of a Z picking method is provided in the on-line examples.

HP PEXlib 5.2 provides four entry points for Z-buffer block operations. These entry points are: `PEXPutZBuffer`, `PEXGetZBuffer`, `PEXCopyPixmapToZBuffer` and `PEXCopyZBuffer ToPixmap`. HP PEXlib also supplies three new `PEXEscapeWithReply` opcodes for Z-buffer support: `PEXHPEscapeOpcodePutZBuffer`, `PEXHPEscapeOpcodeGetZBuffer` and `PEXEscapeOpcodeEVEInformation`. The standard 5.2 entrypoints are documented on-line. This chapter describes the HP escapes in more detail.

Background Information.

These Z-buffer block operations are new to PEXlib 5.2. They are classified as non-OC Rendering requests. Previously, PEXlib users had no way of directly accessing the Z-buffer.

To use the Z-buffer values you need such information as the depth of the Z-buffer, plus the minimum and maximum values in the Z-buffer. To give access to this functionality, HP PEXlib implements a new `PEXEscapeWithReply` opcode. It is called `PEXEscapeOpcodeEVEInformation`.

PEXEscapeWithReply: PEXHPEscapeOpcodeGetZBuffer

This section describes the `PEXHPEscapeOpcodeGetZBuffer` opcode.

PEXHPEscapeOpcodeGetZBuffer Syntax

The syntax for this `PEXEscapeWithReply` is:

```
char *PEXEscapeWithReply(
    Display          *display,
    unsigned long    escape_id,
    int              length,
    char             *escape_data,
    unsigned long    *reply_length_return)
```

PEXHPEscapeOpcodeGetZBuffer Parameters

`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`escape_id`	This is the opcode: `PEXHPEscapeOpcodeGetZBuffer`
`length`	The length, in bytes, of the data for the escape request.
`escape_data`	This is an array of ints with the following fields:
	`escape_data[0]` Renderer (the Renderer ID)
	`escape_data[1]` X (x coordinate of the block to read)
	`escape_data[2]` Y (y coordinate of the block to read)
	`escape_data[3]` Width (width of block to read)
	`escape_data[4]` Height (height of block to read)
`reply_length_return`	Length of the reply data in bytes.

PEXHPEscapeOpcodeGetZBuffer Description

PEXHPEscapeOpcodeGetZBuffer allows the user to read a block from the Z-buffer. This call is identical to PEXGetZBuffer except that the Z-buffer values are the raw hardware values, not normalized. The parameters that go into the escape_data structure are the same ones and in the same order as the parameters that are passed into PEXGetZBuffer.

Even though PEXEscapeWithReply returns a char pointer, you should cast this into a pointer to a structure of type PEXHPEscapeGetZBuffer which is defined in PEXHPlib.h. To access the Z-buffer values in this structure, start looking at &(GetZbufferStructure[1]).

PEXHPEscapeOpcodeGetZBuffer Example

Here is one way to use `PEXHPEscapeOpcodeGetZBuffer`:

```
int escape_data[20];
int  *zbuffer_data;
PEXHPEscapeGetZBuffer  *reply_data;

/* Set up to read a 2x3 Z-buffer block at (10,20) */
escape_data[0] = renderer;
escape_data[1] = 10;
escape_data[2] = 20;
escape_data[3] = 2;
escape_data[4] = 3;

/* Read raw Z-buffer values from the Z-buffer.  */
reply_data = (PEXHPEscapeGetZBuffer *)
    PEXEscapeWithReply(display,
        PEXHPEscapeOpcodeGetZBuffer,20,
        (char *)escape_data, &reply_length);

/* Point to the Z-buffer values */
zbuffer_data = (int *) &(reply_data[1]);

/* Grab the first Z-buffer value */
zbuf_value_1 = zbuffer_data[0];
/* Grab the second Z-buffer value */
zbuf_value_2 = zbuffer_data[1];
```

PEXEscape: PEXHPEscapeOpcodePutZBuffer

This section describes the `PEXHPEscapeOpcodePutZBuffer` opcode.

PEXHPEscapeOpcodePutZBuffer Syntax

The syntax for this `PEXEscape` is:

```
void PEXEscape(
    Display                 *display,
    unsigned long           escape_id,
    int                     length,
    char                    *escape_data )
```

PEXHPEscapeOpcodePutZBuffer Parameters

`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`escape_id`	This is the opcode: `PEXHPEscapeOpcodePutZBuffer`
`length`	The length, in bytes, of the data for the escape request.
`escape_data`	This is an array of `int`s with the following fields:

`escape_data[0]`	Renderer (the Renderer ID)
`escape_data[1]`	X (x coordinate of the block to write)
`escape_data[2]`	Y (y coordinate of the block to write)
`escape_data[3]`	Width (width of block to write)
`escape_data[4]`	Height (height of block to write)
`es-cape_data[5..N]`	Z-buffer values to write

PEXHPEscapeOpcodePutZBuffer Description

`PEXHPEscapeOpcodePutZBuffer` allows you to write a block into the Z-buffer. This call is identical to `PEXPutZBuffer` except that the Z-buffer values are the raw hardware values, not normalized. The parameters that go into the `escape_data` structure are the same ones and in the same order as the parameters that are passed into `PEXPutZBuffer`.

PEXHPEscapeOpcodePutZBuffer Example

Here is one way to use PEXHPEscapeOpcodePutZBuffer:

```
int escape_data[36];

/* Set up to write a 2x2 Z-buffer block at (10,20).
 * We will write the value 0x7FFFF000 into the Z-buffer */
escape_data[0] = renderer;
escape_data[1] = 10;
escape_data[2] = 20;
escape_data[3] = 2;
escape_data[4] = 2;
escape_data[5] = 0x7FFFF000;
escape_data[6] = 0x7FFFF000;
escape_data[7] = 0x7FFFF000;
escape_data[8] = 0x7FFFF000;

/*Write raw Z-buffer values into the Z-buffer.  */
PEXEscape(display,PEXHPEscapeOpcodePutZBuffer,36,
     (char *)escape_data);
```

PEXEscapeWithReply : PEXEscapeOpcodeEVEInformation

This section describes the PEXEscapeOpcodeEVEInformation entry point.

PEXHPEscapeOpcodeEVEInformation Syntax

The syntax for this PEXEscapeWithReply is:

```
void PEXEscapeWithReply(
    Display             *display,
    unsigned long       escape_id,
    int                 length,
    char                *escape_data
    unsigned long       *reply_length_return)
```

PEXHPEscapeOpcodeEVEInformation Parameters

`display`	A pointer to a display structure returned by a successful `XOpenDisplay` call.
`escape_id`	This is the opcode: `PEXHPEscapeOpcodeEVEInformation`
`length`	The length, in bytes, of the data for the escape request.
`escape_data`	This is an array of ints with the following fields: `escape_data[0]` Renderer (the Renderer ID)
`reply_length_return`	Length of the reply data in bytes.

PEXHPEscapeOpcodeEVEInformation Description

`PEXHPEscapeOpcodeEVEInformation` allows the user to inquire the details of the Z-buffer. Even though `PEXEscapeWithReply` returns a char pointer, the user should cast this into a pointer to a structure of type `PEXHPEscapeEVEInformation` which is defined in `PEXHPlib`.h. This structure defines what values are returned.

`EVEInfo[0]`	`unsigned int depth` (depth of the Z-buffer)
`EVEInfo[1]`	`unsigned int min_z` (minimum Z-buffer value)
`EVEInfo[2]`	`unsigned int max_z` (maximum Z-buffer value)

PEXHPEscapeOpcodeEVEInformation Example

Here is one way to use `PEXHPEscapeOpcodeEVEInformation`:

```
int escape_data[1];
PEXHPEscapeEVEInformation *EVEInfo;

/* Grab Z-buffer statistics.*/
escape_data[0] = renderer;

EVEInfo = (PEXHPEscapeEVEInformation *)
   PEXEscapeWithReply(display,
      PEXHPEscapeOpcodeEVEInformation,4,
      (char *) escape_data, &reply_length_return);

/* Grab the Z-buffer depth */
zbuffer_depth = EVEInfo->depth;
/* Grab the minimum Z-buffer value */
zbuffer_min_z = EVEInfo->min_z;
/* Grab the maximum Z-buffer value */
zbuffer_max_z = EVEInfo->max_z;
```

Plane Mask and Drawing Function

HP-PEXlib 5.1v3 supports the PEXlib 5.2 attribute output commands `PEXOCC-SetPlaneMask` and `PEXOCCSetDrawingFunction`. The online reference pages provide details on the interfaces to these entrypoints.

Both of these attributes are applied in the very last step in PEX rendering, when a source pixel value (derived from the rendering pipeline RGB value using the current Color Approximation table entry) is combined with a destination pixel value (already in the frame buffer). The plane mask determines which planes (bits) in the source pixel are to be combined with the corresponding bits of the destination pixel. The drawing function determines what logical function will be applied in the combining.

It is important to understand that the source, destination, and resulting new pixel values are related to the contents of the X Colormap. If you wish to use these functions to cause certain colors to appear on the screen, you must predict what source pixel value will be produced by PEX, and what resultant pixel value you need to access the desired X Colormap entry.

These pixel/color relationships are relatively easy to compute for a PseudoColor visual that uses a typical `PEXColorSpace` Colormap containing an orderly color "ramp". However, you should be aware that for other classes of Visuals, and on certain devices that do not use orderly color ramps, the mapping of pixels to colors may not be so simple to compute. Make sure your application allows for such variations in color environment if you want maximum portability.

On the other hand, if you only wish to use the functions to either completely enable or disable drawing (using the plane mask), or to draw in a way that can be later "undrawn" (for example, with an exclusive-OR drawing function value), then you do not need to be concerned with the Colormap contents.

8

HP PEX 5.1v4—More Selected 5.2 PEXlib Functionality and HP Extensions

Overview of HP PEX5.1v4

This chapter is intended to give an overall view of what is new or different about HP-PEX in this release. There are both new features and entrypoints, and changes in support for existing features.

Background Information

HP PEX 5.1v4 is a superset of HP PEX 5.1v3, which is described in the chapter called "HP PEX 5.1v3—Selected 5.2 PEXlib Functionality".

HP PEX 5.1v4 includes more functionality and interfaces from the future PEX/PEXlib 5.2 standards and HP extensions. These features are being implemented in advance of the final standards, because HP believes that they will have significant value in many PEXlib applications. In some cases, minor differences between the HP implementation and the final PEX 5.2 standard may occur, but none should require more than very minor adjustments to make your application 5.2 conformant. It is important to note that 5.1v4 is *not* a complete PEX 5.2 implementation; instead, as the release name implies, it is PEX 5.1, plus certain selected items from the PEX 5.2 *draft* standard, plus other extensions. Some of these 5.2 features may be available only from HP for some time to come, so use of them is a consideration for portability and interoperability. Nevertheless, you may find them very valuable in the interest of performance, functionality, and experimentation with some important features of PEX/PEXlib 5.2.

Global Description of the HP-PEX 5.1v4 Release

Additional Functionality

HP-PEX Release 5.1v4 includes support for the following new functionality and entry points. Further information is contained later in this chapter.

- Triangle Primitives
  - `PEXOCCTriangleFan`
  - `PEXOCCTriangles`
- Indexed Primitives
  - `PEXHPOCCIndexedMarkers`
  - `PEXHPOCCIndexedPolylines`
  - `PEXHPOCCIndexedTriangleFan`
  - `PEXHPOCCIndexedTriangleStrip`
  - `PEXOCCIndexedTriangles`
- User-Defined Line Types and Marker Glyphs
  - `PEXHPOCCSetUserLineType`
  - `PEXHPOCCSetUserMarkerGlyph`
- User-Defined Highlight Color
  - `PEXHPOCCSetHighlightColor`
- Face Lighting Control
  - `PEXHPOCCSetFaceLightingMode`
- Stereo Viewing
  - `PEXHPSetStereoMode`
- Wide Line Rendering Control
  - `PEXHPChangeRenderer`
- Polygon Offset Rendering Control
  - `PEXHPChangeRenderer`

New Device Support

HP-PEX Release 5.1v4 includes support for all devices supported with HP-PEX Release 5.1v3, plus support for two new devices:

- HP VISUALIZE-EG
- HP VISUALIZE-48XP

New Functionality Descriptions

Wideline Control

A new wideline control renderer attribute (`PEXHPRAWideLineControl`) is settable via `PEXHPChangeRenderer`.

This HP extended renderer attribute controls the method used to render wide lines. The attribute value `PEXHPWideLineControlStroked` instructs the renderer to draw wide lines as a series of multiple strokes. The default attribute value `PEXHPWideLineControlImpDep` allows the renderer to choose any method to render the wide lines.

Stereo Viewing

The HP-defined opcode for `PEXEscape`, `PEXHPEscapeOpcodeStereoMode`, places a specified window (and in some cases, the entire graphics display) in stereo mode, if the display device supports stereo display. When enabled, the application is expected to manage rendering of left and right views; this function only enables or disables the hardware state require to drive stereographic viewing equipment. The basic actions required of the application logic are explained below.

Information on HP graphics displays and monitors that support stereo viewing is not included here. For information about stereo viewing equipment that is compatible with HP displays, please contact your HP sales representative.

The escape data block can be set up using the supplied data structure type, `hpEscapeStereoMode`, defined in `PEXHPlib.h`:

```
typedef struct
{
   Window              window;
   unsigned int        enabled;
} hpEscapeStereoMode;
```

The *window* field should be set to the resource identifier of a window intended for PEX rendering. On all HP displays supported by HP-PEX to date, this must be a full-screen, borderless window. This is because the graphics display hardware does not have the capability of displaying some parts of the screen in stereo mode and some in non-stereo mode. Since most other X clients do not support this method of using the hardware, it is necessary to restrict the window configuration in this way.

The *enabled* field should be set to `True` to enable stereo display mode for the window, and `False` to disable it.

The direct-call interface, `PEXHPSetStereoMode`, is available for applications that do not have portability issues in using such platform-dependent entrypoints. Please see the reference page for `PEXHPSetStereoMode` for further details.

Here are the basic application actions and other important information required in order to perform stereo rendering:

1. The HP displays that support only full-screen stereo mode do so as follows: the frame buffer (to which the window is mapped) is split in half vertically, with the upper half being treated as the left-eye buffer, and the bottom half as the right-eye buffer. For example, a 1280×1024 window would be logically split into two 1280×512 buffers. Be aware that the vertical resolution of the window is effectively halved when in stereo mode, even though the graphics display hardware automatically alters its output video signal so that each buffer is alternately displayed using the entire screen. The stereo viewing hardware is connected to an output of the graphics display that synchronizes the shuttering mechanism being used (LCD, polarization, etc.) with the alternating left- and right-eye images.
2. The basic action of the application is to render two views per logical frame (a left-eye view and a right-eye view) into the corresponding halves of the window. It is important to always render the pair of images together, e.g. before a double-buffer swap, in order to avoid mismatched left and right images.
3. The application must manage two views, one for the left eye and one for the right eye. Normally, when modeling real-world geometric objects, two perspective views are set up to represent eyepoints that are spaced approximately as far apart as a pair of human eyes. However, sometimes the distance is made larger to exaggerate the visual parallax. Caution must be used to avoid causing headaches and nausea (literally) for the end user.
4. Note that the aspect ratio of each half-window buffer is different than in the normal non-stereo mode. The application must manage the rendering such that it occurs in the correct half-windows for the two views. This is best done in PEX by computing the view mapping matrix to "distort" the aspect ratio of the view window into either the upper or lower half of the NPC (Normalized

Projection Coodinate) space, while leaving the Renderer's NPC-to-viewport mapping unchanged.

If the PEX utility `PEXViewMappingMatrix` is used to compute the view mapping matrix, specify the view "window" (in view reference coordinates, or VRCs) normally, but specify the "viewport" (in NPCs) to be either the upper or lower half of the NPC space.

5. Most stereo viewing uses animation as a key feature to help the user get visual cues from the image. Typically, the application would use some type of double-buffering as part of the animation. All the supported mechanisms (Evans and Sutherland escapes, MBX, and DBE) can be used normally in stereo mode. The only difference in usage is to render both a left and a right image before swapping.
6. Code that relies on inquiries of the window's vertical resolution must be conditioned to divide the vertical resolution by two when in stereo mode. Any X rendering (including menus and other widgets) must be constrained to half of the window's height, and repeated in both halves. This is not automatically done by the X server or by most toolkit libraries, so the application has to take responsibility for this as well.

   Note that while in stereo display mode, every pixel is "stretched" vertically, so X raster fonts and other bitmaps will appear elongated.
7. When stereo mode is disabled, usage of the frame buffer goes back to normal, with the display's vertical resolution mapped to the window's height. In real use, an application would also go back to using a single view rendering per frame. While debugging stereo operation, it is sometimes useful to continue to render the half-window images with the hardware mode disabled, allowing the two images to be examined without stereo viewing equipment.

Triangle Primitives

In addition to the triangle strip primitive, HP PEXlib now supports the triangle fan and independent triangle primitives. All three primitives behave in the same manner, except in the way the vertices specify the geometry. See the following figure and descriptions for more detail.

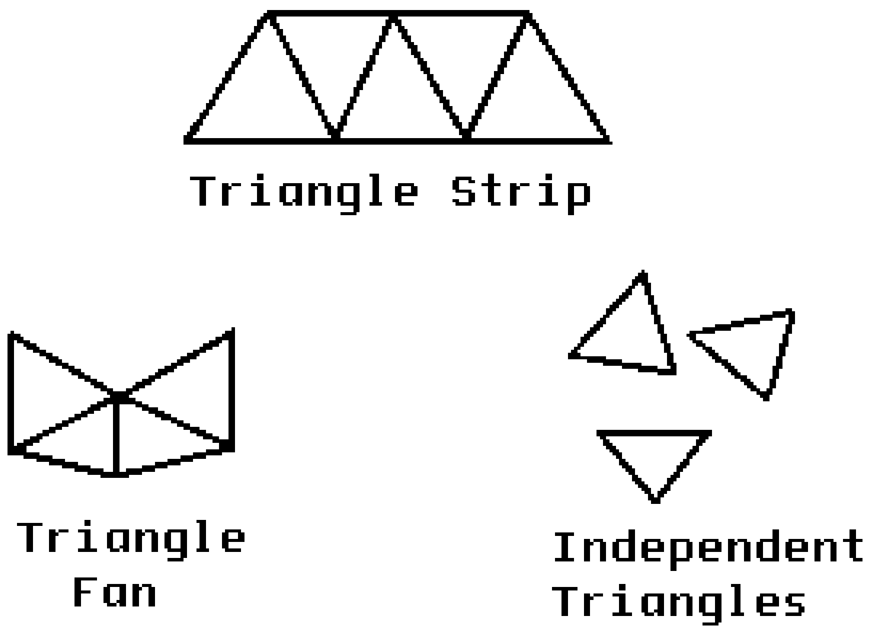


Figure 8-1. Triangle Primitive Examples

For the triangle strip, each triangle is formed by a vertex and the two vertices that precede it in the vertex list.

For the triangle fan, each triangle is formed by a vertex, the vertex that precedes it in the vertex list, and the first vertex in the list.

For the independent triangles, each three consecutive vertices in the vertex list define an independent triangle.

For more information on triangle primitives, see the following on-line reference pages:

- `PEXOCCTriangleFan`
- `PEXOCCTriangles`
- `PEXOCCTriangleStrip`

Indexed Primitives

In addition to the SOFAS primitive, HP PEXlib now supports additional primitives that use a connectivity list to index into a list of vertices. Markers, polylines, triangle fans, triangle strips, and independent triangles now support this method of specifying vectors.

With the addition of these new indexed primitives, most primitives using vertex lists have both indexed and non-indexed forms. The indexed form is useful for applications that have a long vertex list, but use only subsets of that list for each primitive. The indexed primitives let the application "pick and choose" which vertices from the list are actually used to draw each primitive, without needing to copy the chosen vertices into a temporary list.

In all cases, the indexed primitives behave like their non-indexed forms, except for the added level of indirection implied by the list of indicies.

For more information on indexed primitives, see the following on-line reference pages:

- `PEXHPOCCIndexedMarkers`
- `PEXHPOCCIndexedPolylines`
- `PEXHPOCCIndexedTriangleFan`
- `PEXHPOCCIndexedTriangleStrip`
- `PEXOCCIndexedTriangles`

User-Defined Linetypes and Marker-Glyphs

Before this release, the PEXlib programmer could only access PEX-defined linetypes and marker-glyphs. This release provides new functions to allow you to define custom linetypes and marker-glyphs.

You may specify a linetype with a 16-bit "on-off" bitmask and a repeat factor that "stretches" the pattern.

Define markers with a list of polylines, like a polyline set, to specify multiple "strokes". Each "stroke" may be a number of connected line segments. Markers cannot be defined with a "bitmap".

For more information on user-defined linetypes and marker-glyphs, see the following on-line reference pages:

- `PEXHPOCCSetUserLineType`
- `PEXHPOCCSetUserMarkerGlyph`

Highlight Color

Prior to this release, you had to select the highlight color prior to running the PEXlib application by setting an environment variable, or letting it remain the default color of white. PEXlib now supplies a function to allow you to change the highlight color at any point during program execution.

For more information on highlight color, see the following on-line reference page:

- `PEXHPOCCSetHighlightColor`

Face Lighting Control

Allows the HP PEX renderer to assume "bidirectional" implicit geometric normals for use in lighting calculations.

This function is useful when facet normals are not provided by the application and the ordering of vertices for surface area primitives, like a Fill Area, is inconsistent. If this function is used to configure PEXlib to assume bidirectional implicit geometric normals, facets that are implicitly back-facing due to their vertex order are illuminated as if they were front-facing.

For more information on face lighting control, see the following on-line reference page:

- `PEXHPOCCSetFaceLightingMode`

Polygon Offset

A new Polygon Offset attribute (`PEXHPRAPolygonOffset`) is settable via `PEXH-PChangeRenderer`.

The HP extended renderer attribute `PEXHPRAPolygonOffset` causes the interior pixels of front- and back-facing area primitives (polygons, triangle strips, quadrilateral meshes, polyhedra, and other such primitives), when in interior style `PEXInteriorStyleSolid` or `PEXExtInteriorStyleTexture`. This attribute is used to generate Z-buffer values that are offset from what the default Z-buffer values would be. This behavior allows an application to use an algorithm that may yield significantly better performance in rendering filled areas with edging (on those graphics devices that support the attribute) over the default PEX method for rendering edged areas. Details of the algorithm are explained below. Other uses to reduce rendering artifacts are also possible. This attribute is only useful when hidden surface rendering (`HLHSR`) is enabled (see the `PEXChangeRenderer` on-line help page).

An application can call `PEXGetImpDepConstants` with implementation-dependent constant name `PEXHPIDDoesPolygonOffset` to determine if this Renderer attribute is implemented on a particular target.

Note that the offset is applied in the device coordinate (DC) Z-axis only, not in any geometric space such as modeling coordinates or world coordinates. Thus, it displaces the rendered pixels after all modeling, viewing, and viewport transformations have been applied.

The offset value is computed from two parts:

1. A fixed offset (or bias) value that is always applied. The bias is specified as a device-independent floating point value. HP-PEX multiplies this value by the device-specific Z-buffer increment value. Thus, a bias value of 1.0 is typical.
2. A factor value that HP-PEX multiplies by each planar facet's maximum Z-gradient with respect to the DC X or Y axes. Areas that are orthogonal to the viewing direction have a Z-gradient of zero, so the factor has no effect. Areas that slope sharply away from the viewpoint have large Z-gradients so the factor value adds a significant additional offset. The factor is specified as a floating point value without units. A starting value of 1.0 is suggested, but depending on the nature of the geometry and the viewing transformation, an adjustment to achieve the desired rendering effect may be required.

The results of the bias computation and the factor computation are summed to create the DC Z offset that is applied to the area primitive. Positive bias and factor values result in Z-buffer values that are "farther away" from the viewer; this is the normal usage. The results are undefined for non-planar facets, as a single Z-gradient cannot be computed for them.

This attribute also requires an integer flag value indicating whether polygon offset is to be enabled or disabled. The bias and factor values are only used when the enable flag is set; however, they should always be given valid floating point values.

The data values that are part of the attribute are set in the `polygon_offset` substructure of the `PEXHPRendererAttributes` structure. This substructure is of type `PEXHPPolygonOffsetValues` and contains three fields that are used as follows:

`enabled`	Enable flag: set to `True` to enable polygon offset; set to `False` to disable application of the offset.
`offset`	Bias value.
`slope_factor`	Factor value.

Thus, a typical usage of `PEXHPChangeRenderer` to set this attribute might be:

```
PEXHPRendererAttributes         hp_attrs;
unsigned long                   hp_ra_mask;

hp_attrs.polygon_offset.enabled = True;
hp_attrs.polygon_offset.offset = 1.0;
hp_attrs.polygon_offset.slope_factor = 1.0;

hp_ra_mask = 0;
PEXHPSetRendererAttributeMask(hp_ra_mask, PEXHPRAPolygonOffset)

PEXHPChangeRenderer (display, renderer_id, &hp_ra_mask, &hp_attrs);
```

Improving Rendering of Edged Polygons

This attribute can be used to improve rendering of edged polygons, as follows:

The normal way to render edged areas in HP-PEX (and the way that must still be used in cases where `PEXHPIDDoesPolygonOffset` indicates that the functionality is not supported) is to set the interior style to `PEXInteriorStyleSolid`, and enable surface edging via `PEXSetSurfaceEdgeFlag`. HP-PEX renders the interior pixels in the fill color and the edge pixels in the edge color. Special rasterization is done to avoid "stitching" of edges.

"Stitching" occurs when scattered pixels of a primitive (in this case an edge vector) are not drawn because the Z-buffer values at those pixels already indicate that the primitive is "obscured" (in this case by the interior fill pixels).

Better performance can be achieved in rendering edged areas by filling many areas, and then rendering all the edges as polyline primitives in a second pass. Among other reasons behind the performance improvement, this distributes the cost of modifying the graphics pipeline state from "fill" mode to "vector" mode over many primitives, rather than switching modes during each area primitive.

Such a “grouping” of operations must be done at the application level. Obviously, the edging is done via line primitives in this algorithm, so line attributes must be set to the desired edge values.

The typical problem with this better-performing method of rendering is that when the edge vectors are rendered, a lot of stitching is visible because of the values already stored in the Z-buffer by the fill rendering. Offsetting the fill rendering in the Z-buffer can eliminate this stitching. Thus, this extended Renderer attribute allows for better-looking images using a faster rendering method.

Note that the per-area polygon offset computation does slightly slow down the rendering of filled areas, but for applications that can render significant numbers of area primitives followed by a few polyline primitives with many vertices (hundreds, perhaps), the “grouping” of area primitives and of polylines more than makes up for the computation overhead.

It should be noted that use of polygon offset can introduce artifacts in hidden-surface rendering. For example, if a solid object such as a cube is being rendered, then depending on the angle of the view, one side might have a higher Z-gradient than an adjoining side. Because the more sharply-angled side could be offset more (depending on the factor value), all the pixels of the adjoining side might not be obscured. This could result in ragged joints, especially with back-facing parts of the solid.

One way to avoid this particular artifact, if the geometry to be rendered is appropriate, is to enable back-face culling, which is a recommended practice in any case for performance reasons. Other artifacts in the conjunction or intersection of filled areas with each other or with other types of primitives can also occur.

Here is a partial code skeleton as an example of how an application can make use of polygon offset for edging of filled areas. Note that this sample code is only intended to show the basic logic; it may not be the most efficient code design in terms of geometry management or avoiding unnecessary attribute changes.

```
Display                 *display;
Window                  window;

PEXRenderer             renderer_id;
short                   id_name;
PEXImpDepConstant       *id_const_info;

PEXHPRendererAttributes hp_attrs;
unsigned long           hp_ra_mask;

:
display = XOpenDisplay(...);  /* Initialize an X connection. */
PEXInitialize (display, ...); /* Initialize PEX on the connection. */

:
/* Create a window or buffer drawable in a particular target
   Visual, to be used for PEX rendering. */

window = ...
:
/* Create a Renderer for this target. */
renderer_id = PEXCreateRenderer (display, window, ...);


/* Inquire whether polygon offset is supported by Renderers
   created for this target. */

id_name = PEXHPIDDoesPolygonOffset;
PEXGetImpDepConstants (display, window, 1, &id_name, &id_const_info);

:
/* Enable HLHSR (i.e., use of the Z-buffer) */
PEXChangeRenderer (display, renderer_id, ..., PEXRAHLHSRMode);

/* Enable back-face culling for performance, and to eliminate
   the most common polygon-offset artifact. */
PEXSetFacetCullingMode (display, renderer_id, ..., PEXBackFaces);

:
/* Set up fill attributes (other than interior style). */

:
if (id_const_info[0].integer) {

   /* Since the attribute is supported, use faster algorithm. */

   /* Set up line attributes with desired edge attribute values. */
```

```
    /* Set the interior style to solid, without edging. */
    PEXSetInteriorStyle (display, renderer_id, PEXOCRender,
                         PEXInteriorStyleSolid);
    PEXSetSurfaceEdgeFlag (display, renderer_id, PEXOCRender, PEXOff);

    /* Enable polygon offset. */
    hp_attrs.polygon_offset.enabled = True;
    hp_attrs.polygon_offset.offset = 1.0;
    hp_attrs.polygon_offset.slope_factor = 1.0;
    hp_ra_mask = 0;
    PEXHPSetRendererAttributeMask(hp_ra_mask, PEXHPRAPolygonOffset)

    PEXHPChangeRenderer (display, renderer_id, &hp_ra_mask, &hp_attrs);


    /* Collect area primitive geometry for edges to be rendered into
       polyline geometry format.  (In some cases, the same geometry arrays
       can be used for both filling and edging passes.) */
    .
    .
    .
    /* Fill:  Render primitives with or without edge flags (they will be
       ignored).  */

    ... /* Area primitive calls */

    /* Edge:  Render the edge geometry with or without move/draw flags. */

    ... /* Polyline calls */
}
else {
    /* Since the attribute is not supported, let HP-PEX draw the edges. */

    /* Set up edge attributes. */

    /* Set the interior style to solid, with edging. */
    PEXSetInteriorStyle (display, renderer_id, PEXOCRender,
                         PEXInteriorStyleSolid);
    PEXSetSurfaceEdgeFlag (display, renderer_id, PEXOCRender, PEXOn);

    /* Fill and Edge:  Render primitives with or without edge flags. */

    ... /* Area primitive calls */
}
```

9

Overview of CGE PEX Texture Mapping

This overview shows the parameters and data structures for PEXlib texture mapping calls. Use this section as a resource while developing texture-mapping programs; it explains texture-mapping parameters in detail. Note that in the "Parameters" sections, only those parameters *directly* related to texture mapping are described (for example, the Display argument is not explained in any detail). Please refer to the *PEXlib Programming Reference* and the *PEXlib Programming Manual* for descriptions of Display, and the other parameters that are not directly related to texture mapping; also see the HP PEX On-Line Reference for a hypertext version of this chapter, as well as the PEXlib Reference.

In the HP PEX 5.1v3 and later releases , the Output Command Context (OCC) interface for `PEXOCCTriangleStrip`, `PEXOCCQuadrilateralMesh`, `PEXOCCFillArea`, `PEXOCCFillAreaSet`, and `PEXOCCIndexedFillAreaSets` supports texture mapping and deformation data. The OCC interface is part of the 5.2 PEXlib specification; it is recommended over the Common Graphics Environment (CGE) extended output commands. However, in HP PEX 5.1v3 and later releases, the CGE interface to texture mapping must still be used; for example, you must use `PEXExtCreateTM` and the four TM LUTs from the CGE implementation, rather than the 5.2 equivalent definitions.

OCC versions can be used instead of `PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, `PEXExtQuadrilateralMesh`, if the appropriate OC Context has been established. (See the appropriate reference pages in the Alphabetical List of PEX Functions for more details.)

Also, in HP PEX releases 5.1v3 and later releases;, `PEXSetInteriorStyle` is defined to be `PEXExtInteriorStyleTexture` and the parameterization method specified is not `PEXExtTMParamExplicit`, then 5.1 Output Commands will be texturable for the Output Commands that correspond to the 5.2 Output Commands.

Table 9-1. Output Commands Texturable

5.1 Output Commands	5.2/HP Output Commands	CGE Output Commands
PEXFillAreaWithData	PEXOCCFillArea	
PEXFillAreaSetWithData	PEXOCCFillAreaSet	PEXExtFillAreaSetWithData
PEXSetOfFillAreaSets	PEXOCCIndexedFillAreaSets	PEXExtSetOfFillAreaSets
PEXQuadrilateralMesh	PEXOCCQuadrilateralMesh	PEXExtQuadrilateralMesh
PEXTriangleStrip	PEXOCCTriangleStrip	PEXExtTriangleStrip
	PEXOCCTriangleFan	
	PEXOCCTriangles	
	PEXOCCIndexedTriangles	
	PEXHPOCCIndexedTriangleStrip	
	PEXHPOCCIndexedTriangleFan	

Texture mapping can be accomplished in six main steps, each of which is described below:

1. Set up,
2. Texture preparation,
3. Geometry preparation,
4. Set up the LUTs (Binding, Coordinate Source, Sampling, and Composition),
5. Render, and
6. Clean up.

Optional functions are noted as such; all other function calls are required.

Step 1: Setup

Setting up for texture mapping involves ensuring that texture mapping is supported on the current implementation of PEXlib, and inquiring implementation-dependent constants that affect texture mapping.

- **Function:** `PEXGetEnumTypeInfo` (optional). Get enumerated type information to ensure that texture mapping is supported by this implementation of PEXlib. Returns non-zero if successful and zero if an error occurred.
- **Function:** `PEXFreeEnumInfo` (optional). Free memory allocated by `PEXGetEnumTypeInfo`.
- **Function:** `PEXGetImpDepConstants` (optional). Determine texture mapping implementation dependent constants. Returns non-zero if successful and zero if an error occurred.

`PEXGetEnumTypeInfo`: Parameters

```
int      *enum_types;
```

Set `enum_types` = `PEXETEscape`. If one of the entries returned in `enum_info_return` has the value `PEXExtEscapeChangePipelineContext`, texture mapping, along with other CGE PEX extensions, is supported by this implementation of PEXlib. The application may want to query additional enumerated types to get more detailed information about which texture mapping enumerated types are supported by this implementation.

These are the extended PEX enumerated types.

- `PEXExtETEnumType`
- `PEXExtETOC`
- `PEXExtETPC`
- `PEXExtETRA`
- `PEXExtETLUT`
- `PEXExtETID`
- `PEXExtETTMRenderingOrder`
- `PEXExtETTMCoordSource`
- `PEXExtETTMCompositeMethod`
- `PEXExtETTMTexelSampleMethod`
- `PEXExtETTMBoundaryCondition`
- `PEXExtETTMClampColorSource`
- `PEXExtETTMDomain`
- `PEXExtETTexelType`
- `PEXExtETTMResourceHint`
- `PEXExtETTMType`
- `PEXExtETTMParameterizationMethod`
- `PEXExtETTMPerspectiveCorrection`
- `PEXExtETTMSampleFrequency`
- `PEXExtETPrimitiveAAMode`
- `PEXExtETPrimitiveAABlendOp`
- `PEXExtETLineCapStyle`
- `PEXExtETLineJoinStyle`

The following are the extended enumerated types that may be returned.

- `PEXExtETOC`
  - `PEXExtOCTMPerspectiveCorrection`
  - `PEXExtOCTMSampleFrequency`
  - `PEXExtOCTMResourceHints`
  - `PEXExtOCActiveTextures`
  - `PEXExtOCBFActiveTextures`
  - `PEXExtOCFillAreaSetWithData`
  - `PEXExtOCSetOfFillAreaSets`
  - `PEXExtOCTriangleStrip`
  - `PEXExtOCQuadrilateralMesh`
  - `PEXExtOCPrimitiveAA`
  - `PEXExtOCLineCapStyle`
  - `PEXExtOCLineJoinStyle`
  - `PEXExtOCEllipse`
  - `PEXExtOCEllipse2D`
  - `PEXExtOCCircle2D`
  - `PEXExtOCEllipticalArc`
  - `PEXExtOCEllipticalArc2D`
  - `PEXExtOCCircularArc2D`
- `PEXExtETPC`
  - `PEXExtPCMinShift`
  - `PEXExtPCTMPerspectiveCorrection`
  - `PEXExtPCTMResourceHints`
  - `PEXExtPCTMSampleFrequency`
  - `PEXExtPCActiveTextures`
  - `PEXExtPCBFActiveTextures`
  - `PEXExtPCPrimitiveAA`
  - `PEXExtPCLineCapStyle`
  - `PEXExtPCLineJoinStyle`
  - `PEXExtPCMaxShift`
- `PEXExtETRA`
  - `PEXExtRAMinShift`
  - `PEXExtRATMBindingTable`
  - `PEXExtRATMCoordSourceTable`
  - `PEXExtRATMCompositionTable`
  - `PEXExtRATMSamplingTable`
  - `PEXExtRAMaxShift`

- PEXExtETLUT
  - PEXExtLUTTMBinding
  - PEXExtLUTTMCoordSource
  - PEXExtLUTTMComposition
  - PEXExtLUTTMSampling
- PEXExtETID
  - PEXExtIDMaxTextureMaps
  - PEXExtIDMaxFastTMSize
  - PEXExtIDPowerOfTwoTMSizesRequired
  - PEXExtIDSquareTMRequired
- PEXExtETTMRenderingOrder
  - PEXExtTMRenderingOrderPreSpecular
  - PEXExtTMRenderingOrderPostSpecular
- PEXExtETTMCoordSource
  - PEXExtTMCoordSourceVertexCoord
  - PEXExtTMCoordSourceVertexNormal
  - PEXExtTMCoordSourceFloatData
- PEXExtETTMCompositeMethod
  - PEXExtTMCompositeReplace
  - PEXExtTMCompositeModulate
  - PEXExtTMCompositeBlendEnvColor
  - PEXExtTMCompositeDecal
  - PEXExtTMCompositeDecalBackground
  - PEXExtTMCompositeReplaceBlendedColors
- PEXExtETTMTexelSampleMethod
  - PEXExtTMTexelSampleSingleBase
  - PEXExtTMTexelSampleLinearBase
  - PEXExtTMTexelSampleSingleInMipmap
  - PEXExtTMTexelSampleLinearInMipmap
  - PEXExtTMTexelSampleSingleBetweenMipmaps
  - PEXExtTMTexelSampleLinearBetweenMipmaps
- PEXExtETTMBoundaryCondition
  - PEXExtTMBoundaryCondClampColor
  - PEXExtTMBoundaryCondBoundary
  - PEXExtTMBoundaryCondWrap
  - PEXExtTMBoundaryCondMirror

- PEXExtETTMClampColorSource
  - PEXExtTMClampColorSourceAbsolute
  - PEXExtTMClampColorSourceExplicit
- PEXExtETTMDomain
  - PEXExtTMDomainColor1D
  - PEXExtTMDomainColor2D
  - PEXExtTMDomainColor3D
- PEXExtETTexelType
  - PEXExtTexelLuminanceInt8
  - PEXExtTexelLuminanceInt16
  - PEXExtTexelLuminanceAlphaFloat
  - PEXExtTexelLuminanceAlphaInt8
  - PEXExtTexelLuminanceAlphaInt16
  - PEXExtTexelRGBFloat
  - PEXExtTexelRGBInt8
  - PEXExtTexelRGBInt16
  - PEXExtTexelRGBAlphaFloat
  - PEXExtTexelRGBAlphaInt8
  - PEXExtTexelRGBAlphaInt16
  - PEXExtTexelLuminanceFloat
- PEXExtETTMType
  - PEXExtTMTypeMipMap
- PEXExtETTMParameterizationMethod
  - PEXExtTMParamExplicit
  - PEXExtTMParamReflectSphereVRC
  - PEXExtTMParamReflectSphereWC
  - PEXExtTMParamLinearVRC
- PEXExtETTMPerspectiveCorrection
  - PEXExtTMPerspCorrectNone
  - PEXExtTMPerspCorrectVertex
  - PEXExtTMPerspCorrectPixel
- PEXExtETTMSampleFrequency
  - PEXExtTMSampleFrequencyPixel
  - PEXExtTMSampleFrequencyInterpDep
- PEXExtETTMResourceHint
  - PEXExtTMResourceHintNone
  - PEXExtTMResourceHintSpeed
  - PEXExtTMResourceHintSpace

- PEXExtETPrimitiveAAMode
  - PEXExtPrimAANone
  - PEXExtPrimAAPoint
  - PEXExtPrimAAVector
  - PEXExtPrimAAPointVector
  - PEXExtPrimAAPolygon
  - PEXExtPrimAAPointPolygon
  - PEXExtPrimAAVectorPolygon
  - PEXExtPrimAAPointVectorPolygon
- PEXExtETPrimitiveAABlendOp
  - PEXExtPrimAABlendOpImpDep
  - PEXExtPrimAABlendOpSimpleAlpha
- PEXExtETLineCapStyle
  - PEXExtLineCapStyleButt
  - PEXExtLineCapStyleRound
  - PEXExtLineCapStyleProjecting
- PEXExtETLineJoinStyle
  - PEXExtLineJoinStyleImpDep
  - PEXExtLineJoinStyleRound
  - PEXExtLineJoinStyleMiter
  - PEXExtLineJoinStyleBevel

Additional Types of **PEXETEscape**

- PEXExtEscapeChangePipelineContext
- PEXExtEscapeGetPipelineContext
- PEXExtEscapeChangeRenderer
- PEXExtEscapeGetRendererAttributes
- PEXExtEscapeSetTableEntries
- PEXExtEscapeGetTableEntries
- PEXExtEscapeGetTableEntry
- PEXExtEscapeCreateTM
- PEXExtEscapeCreateTMDescription
- PEXExtEscapeFreeTM
- PEXExtEscapeFreeTMDescription
- PEXExtEscapeFetchElements
- PEXExtEscapeQueryColorApprox
- PEXExtEscapeCreateTMExtraData

Additional Types of **PEXETLineType**

- `PEXExtLineTypeCenter`
- `PEXExtLineTypePhantom`

Additional Types of **PEXETHatchStyle**

- `PEXExtHatchStyle45Degrees`
- `PEXExtHatchStyle135Degrees`

Additional Types of **PEXETInteriorStyle**

- `PEXExtInteriorStyleTexture`

`PEXGetImpDepConstants`: Parameters

```
   names[0] = PEXExtIDMaxTextureMaps;
```

Maximum number of texture maps that can be applied to a single primitive. For HP PEX, `PEXExtIDMaxTextureMaps` is device-dependent and should be inquired. For many HP systems, `PEXExtIDMaxTextureMaps` = 8; that is, a maximum of eight maps can be applied to a single primitive. Note that an unlimited number of textures can be loaded at any one time, but a maximum of eight can be applied to a single primitive.

```
   names[1] = PEXExtIDMaxFastTMSize;
```

Maximum size of any dimension of the base level of a texture map which allows an optimized implementation. Larger maps may not be optimized. A value of zero indicates that any size of texture map is equally optimized. For HP PEXlib, `PEXExtIDMaxFastTMSize` is device-dependent and should be inquired for the current device.

```
   names[2] = PEXExtIDPowerOfTwoTMSizesRequired;
```

True if the size of all dimensions of all texel arrays defining a texture map must be powers of two. For HP PEX, `PEXExtIDPowerOfTwoTMSizesRequired` = `True`. Note, however, that the `PEXExtCreateFilteredTM` and `PEXExtCreateFilteredTMFromWindow` utilities can be used to upsample texel arrays to a power of two.

```
   names[3] = PEXExtIDSquareTMRequired;
```

True if each level of a texture map must have equally sized dimensions. For HP PEX, `PEXExtIDSquareTMRequired` is `False`.

Step 2: Texture Preparation

Preparing a texture for use by PEXlib involves pre-filtering the texture to create a MIP map, importing the map into PEXlib and combining the map with parameterization and rendering information to create a texture map description.

- **Function:** `PEXExtCreateFilteredTM` (optional). Creates a filtered texture map from `base_map` and stores the results in `texel_array`. After this call, `texel_array` should be passed to `PEXExtCreateTM` to import it into PEXlib.
- **Function:** `PEXExtCreateFilteredTMFromWindow` (optional). Create a filtered texture map, `texel_array` from the X resources, `base_color_map` and `base_alpha_map`. After this call, the `texel_array` should be passed to `PEXExtCreateTM` to import it into PEXlib. Returns zero if successful.
- **Function:** `PEXExtCreateTM`. Converts the data described by the `domain`, `domain_data`, and `texel_arrays` into an internal texture map resource. Returns the X resource ID for the map.
- **Function:** `PEXExtFreeFilteredTM` (optional). Free texel data created by `PEXExtCreateFilteredTM` or `PEXExtCreateFilteredTMFromWindow` utilities. This routine may be called after the `texel_array` is used with `PEXExtCreateTM`.
- **Function:** `PEXExtCreateTMDescription`. Create a texture map description by combining texture resource(s) with parameterization and rendering information.

PEXExtCreateFilteredTM: Parameters

```
int domain;
```

Specifies the dimension of the texture map and how the texture map will affect a primitive. Only the primitive color can be affected. Supported values are:

PEXExtTMDomainColor1D: Texture mapping affects the color and alpha values using a 1D texture map.

PEXExtTMDomainColor2D: Texture mapping affects the color and alpha values using a 2D texture map.

```
struct PEXExtTMDomainData {
  union {
    struct {
      PEXEnumTypeIndex  tm_type;        /* Specifies the kind of filtered map
                                           to create.  Only PEXExtTMTypeMipMap
                                           (MIP map) is supported.  */
      PEXExtImpDepData  tm_type_data;   /* Not used by hppex */
      PEXEnumTypeIndex  texel_type;     /* Specifies the format of the data in
                                           the base_map.  Supported values and
                                           their associated types:  */
      PEXExtTexelRGBFloat       struct PEXExtTexelRGB {
                                  float red;           /* \            */
                                  float green;         /*  > 0.0 to 1.0 */
                                  float blue;          /* /            */
                                }
      PEXExtTexelRGBInt8        struct PEXExtTexelRGB8 {
                                  unsigned char red;    /* \          */
                                  unsigned char green;  /*  > 0 to 255 */
                                  unsigned char blue;   /* /          */
                                }
      PEXExtTexelRGBAlphaFloat  struct PEXExtTexelRGBAlpha {
                                  float red;           /* \            */
                                  float green;         /*  \ 0.0 to 1.0 */
                                  float blue;          /*  /           */
                                  float alpha;         /* /            */
                                }
      PEXExtTexelRGBAlphaInt8   struct PEXExtTexelRGBAlpha8 {
                                  unsigned char red;    /* \          */
                                  unsigned char green;  /*  \ 0 to 255 */
                                  unsigned char blue;   /*  /         */
                                  unsigned char alpha;  /* /          */
                                }
      unsigned short int num_levels;    /* Number of MIP map levels to create.
                                           If num_levels is set to zero, the
                                           optimum number of levels will be
                                           generated to create a full (MIP)
                                           map and num_levels will be updated
                                           to reflect the number of levels
                                           created.  */
    } color;
  } data;
} domain_data;
/*-------------------------------------------------------------------------*/
unsigned int power_of_two_tm_required;  /* Indicates whether the
                                        dimensions of texture maps must be a
                                        power of two.  Value returned by
                                        PEXGetImpDepConstants.  If true, this
                                        utility will apply image sizing to
                                        meet the power of two requirement.  If
                                        the image must be decreased in size, a
                                        box filter will be applied.  If the
                                        image must be enlarged, linear
                                        interpolation will be applied.  */
/*-------------------------------------------------------------------------*/
unsigned int square_tm_required;        /* Indicates whether or not the
                                        texture maps must be square.  Value
                                        returned by PEXGetImpDepConstants.  If
                                        true, this utility will apply image
                                        sizing to meet the square texture map
                                        requirement.  */
/*-------------------------------------------------------------------------*/
struct PEXExtTexelArray {
  PEXExtTexelDimension          dimension;      /* width, height, depth */
  union {
    PEXExtTexelLuminance        *luminance;         /* Not used by HP PEX */
    PEXExtTexelLuminance8       *luminance8;        /* Not used by HP PEX */
    PEXExtTexelLuminance16      *luminance16;       /* Not used by HP PEX */
    PEXExtTexelLuminanceAlpha   *luminance_alpha;   /* Not used by HP PEX */
    PEXExtTexelLuminanceAlpha8  *luminance_alpha8   /* Not used by HP PEX */
    PEXExtTexelLuminanceAlpha16 *luminance_alpha16; /* Not used by HP PEX */
    PEXExtTexelRGB              *rgb;           /* float R, G, B */
    PEXExtTexelRGB8             *rgb8;          /* unsigned char R, G, B */
    PEXExtTexelRGB16            *rgb16;         /* Not used by HP PEX */
    PEXExtTexelRGBAlpha         *rgb_alpha;     /* float R, G, B, A */
    PEXExtTexelRGBAlpha8        *rgb_alpha8;    /* unsigned char R, G, B, A*/
    PEXExtTexelRGBAlpha16       *rgb_alpha16;   /* Not used by HP PEX */;
  } array;
} *base_map;                                    /* Source texture map */
```

```
/*-------------------------------------------------------------------------*/
struct PEXExtTexelDimension {
  unsigned short int    t0;            /* Texture map width */
  unsigned short int    t1;            /* Texture map height */
  unsigned short int    t2;            /* Texture map depth: should be 0 */
};
/*-------------------------------------------------------------------------*/
PEXExtTexelArray **texel_array         /* Texture map array allocated and
                                       filled by this utility.  Pass this
                                       array to PEXExtCreateTM and then free
                                       it using PEXExtFreeFilteredTM.  */
```

PEXExtCreateFilteredTMFromWindow: Parameters

See also descriptions of domain, domain_data, power_of_two_tm_required, square_tm_required, and texel_array under "PEXExtCreateFilteredTM: Parameters".

```
unsigned int    luminance_channel_selector;     /* Luminance channel source
                                                   selector.  Unused by HP PEX */
/*--------------------------------------------------------------------------*/
XID             base_color_map;                 /* X window identifier of an
                                                   unobscured window to use as
                                                   the source texture map.  Use
                                                   for the texture map's color
                                                   data.  */
/*--------------------------------------------------------------------------*/
unsigned int    alpha_channel_selector; /* Alpha channel source
                                        selector.  Used only if alpha is to be
                                        included in the texture map and
                                        specified in the texel_type of
                                        domain_data.  The channels available as
                                        source of the alpha data are dependent
                                        upon the depth or visual class of the
                                        base_alpha_map window.  If the resource
                                        is a 24-bit resource, red, green and
                                        blue channels are available as a source
                                        for alpha data.  If it is an 8-bit
                                        resource, the whole 8-bit channel is
                                        used as the source.  Supported values
                                        are:
                                        PEXExtChannelNTSCLuminance:
                                          Alpha is a combination of the
                                          resource (red * 0.299 +
                                          green * 0.587 + blue * 0.114)
                                        PEXExtChannelRed:
                                          Alpha is in the red channel.
                                        PEXExtChannelGreen:
                                          Alpha is in the green channel.
                                        PEXExtChannelBlue:
                                          Alpha is in blue channel.  */
/*--------------------------------------------------------------------------*/
XID             base_alpha_map;         /* X window identifier of an
                                        unobscured window to use as the source
                                        texture map.  Used for the texture map's
                                        alpha data.  (May be NULL if alpha is
                                        not specified by the texel_type in
                                        domain_data.) */
```

PEXExtCreateTM: Parameters

Also see descriptions of domain, domain_data, and texel_arrays under PEX-ExtCreate FilteredTMParms.

```
PEXExtTexelArray        *texel_arrays;  /* Texture map data.  Number of arrays
                                        depends on the number of levels and
                                        the texture map type (as defined by
                                        domain and domain_data).  Texel arrays
                                        are ordered sequentially by logical
                                        levels, the base level being first in
                                        the list.  Every subsequent map is
                                        ordered by its level from the largest
                                        dimension down to the smallest
                                        dimension.  The texels are assumed to
                                        be stored in the order t0, t1, t2.
                                        texel_arrays may be created using
                                        PEXExtCreateFilteredTM or
                                        PEXExtCreateFilteredTMFromWindow.
                                        Because an internal copy of the data
                                        is kept, the memory used by
                                        texels_arrays may be freed immediately
                                        after calling PEXExtCreateTM using the
                                        function PEXExtFreeFilteredTM.  */
```

PEXExtFreeFilteredTM: Parameters

See also descriptions of domain, domain_data, and texel_array under "PEX-ExtCreateFilteredTM: Parameters".

PEXExtCreateTMDescription: Parameters

Note that for best results when using PEXExtTMParamReflectSphereWC, the boundary conditions should be set to PEXExtTMBoundaryCondWrap to achieve the most natural results. The boundary conditions, t0_boundary and t1_boundary are set in the Sampling LUT. The wrap boundary condition leads to better hiding of the texture seams in the case of World Coordinate reflection mapping.

```
int     parameterization;       /* The texture map parameterization type
                                defines how texture map coordinates are
                                derived.  They may be explicitly defined by
                                the primitive's vertex data or calculated via
                                a projection mapping by HP PEX.  Supported
                                values are:
                                  PEXExtTMParamExplicit: This method specifies
                                    that the texture coordinates are included
                                    with the primitive's vertex data.  The
                                    texture's entry in the Texture Coordinate
                                    Source Lookup Table defines how these
                                    coordinates are accessed.  The application
                                    can directly provide the coordinates with
                                    the primitive's vertex data or they can be
                                    derived using PEXExtTMCoord* utilities.
                                  PEXExtTMParamReflectSphereVRC: PEXlib
                                    derives the texture coordinates using an
                                    infinite sphere with (0,0,0) as its origin
                                    and the +Y axis as its axis of revolution.
                                    The texture seams lie on the positive and
                                    negative X axes.  A reflection vector in
                                    VRCs is computed to determine a point on
                                    the interior of the sphere.  Reflection, or
                                    environment, mapping results.
                                  PEXExtTMParamReflectSphereWC: PEXlib
                                    derives the texture coordinates.  The 3D
                                    source texture coordinates are normalized
                                    and conceptually projected onto an
                                    infinite sphere surrounding the object to
                                    calculate the 2D texture coordinates.  The
                                    axis of revolution of the sphere is the +Y
                                    axis (WCs).  The texture seam sweeps from
                                    the +X axis (WCs) in a counterclockwise
                                    direction [0..2pi].  A reflection vector
                                    in WCs is computed to determine a point on
                                    the interior of the sphere.  Reflection
                                    mapping results.
                                  PEXExtTMParamLinearVRC: Texture coordinates
                                    are determined by the PEXlib server with
                                    respect to a projection reference plane
                                    defined in view reference coordinates
                                    (VRC).  At the time of activation
                                    (PEXExtSetActiveTextures), equations p0
                                    and p1 are inversely transformed from VRC
                                    space back into Model Coordinate (MC)
                                    space.  Once there, they define a
                                    projection function such that objects
                                    appear to "swim" through a solid field of
                                    texture coordinates.  The result is texture
                                    mapping.  */
/*------------------------------------------------------------------------*/
struct PEXExtTMParameterizationData {
  union {
    struct {
      PEXMatrix         matrix; /* Reflection matrix used to transform the
                                texture coordinates relative to the projection
                                object (sphere).  Used only when
                                parameterization is set to
                                PEXExtTMParamReflectSphereVRC or
                                PEXExtTMParamReflectSphereWC.  */
    } reflection;
    struct {                    /* Linear equations used  */
      float             p0[4];  /* when parameterization  */
      float             p1[4];  /* is equal to            */
    } linear;                   /* PEXExtTMParamLinearVRC */
  } data;
} *param_data;
/*------------------------------------------------------------------------*/
int             tm_rendering_order;     /* Indicates whether the
                                        texture is applied before or after the
                                        specular component is calculated.
                                        Supported values are:
                                          PEXExtTMRenderingOrderPreSpecular:
                                            Specular component is computed
                                            after texturing.
                                          PEXExtTMRenderingOrderPostSpecular:
                                            Texture mapping is applied after
                                            the specular component has been
                                            computed.  */
/*------------------------------------------------------------------------*/
unsigned int    count;                  /* Number of texture map resource
                                        identifiers in *tm_ids.  */
/*------------------------------------------------------------------------*/
PEXExtTextureMap        *tm_ids;        /*A pointer to a list of texture map
                                        resource identifiers.  */
```

Step 3: Geometry Preparation

Geometry preparation is concerned with generating the texture coordinates for primitives that are to be texture-mapped. Note that an application has three choices for computing vertex coordinates:

- Set ⟨*parameterization*⟩ equal to `PEXExtTMParamReflectSphereVRC` or `PEXExtTMParamReflectSphereWC` and pass to `PEXExtCreateTMDescription` and allow PEXlib to calculate the texture coordinates. This produces view-dependent environment mapping.
- Set ⟨*parameterization*⟩ equal to `PEXExtTMParamExplicit` and pass it to `PEXExtCreateTMDescription` and compute the texture coordinates within the application and store them with the primitive's vertex data.
- Set ⟨*parameterization*⟩ equal to `PEXExtTMParamExplicit` and pass it to `PEXExtCreateTMDescription` and compute the texture coordinates using one of the `PEXExtTMCoord*` utilities described below.

The only PEXlib primitives that can be texture-mapped are either `PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, and `PEXExtQuadrilateralMesh`; or, alternatively, `PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh`.

- **Function:** `PEXExtTMCoordFillAreaSetWithData` (optional). Computes the texture coordinates for a fill area set with data and stores them in the vertex lists. Returns zero if successful.
- **Function:** `PEXExtTMCoordSetOfFillAreaSets` (optional). Computes the texture coordinates for a set of fill area sets and stores them in the specified vertex data fields. Returns zero if successful.
- **Function:** `PEXExtTMCoordTriangleStrip` (optional). Computes the texture coordinates for a triangle strip and stores them in the specified vertex data fields. Returns zero if successful.
- **Function:** `PEXExtTMCoordQuadrilateralMesh` (optional). Computes the texture coordinates for a quadrilateral mesh and stores them in the specified vertex data fields. Returns zero if successful.

PEXExtTMCoordFillAreaSetWithData: Parameters

```
struct PEXExtTMCoordData {      /* Parameterization data for this primitive */
  PEXEnumTypeIndex projection;  /* Projection method (or projection object).
                                Supported values:
                                  PEXExtTMProjectionSphereWC: The texture
                                    coordinates are derived using an infinite
                                    sphere as a projection object with (0,0,0)
                                    as its origin and the +Y axis as the axis
                                    of revolution.  The texture seams lie on
                                    the positive and negative X axes.  A
                                    direction vector in WCs is computed using
                                    either the vertex coordinate or vertex
                                    normal depending on the coord_source
                                    parameter to determine a point on the
                                    interior of the sphere.
                                  PEXExtMProjectionCylinderWC: The texture
                                    coordinates are derived using an infinite
                                    cylinder as a projection object with
                                    (0,0,0) as its origin and the +Y axis as
                                    the axis of revolution.  The texture seam
                                    sweeps from the +X axis in a
                                    counterclockwise direction.  If the
                                    coord_source is
                                    PEXExtTMCoordSourceVertexCoord, a ray
                                    perpendicular to the +Y axis in WCs
                                    through the vertex is computed to
                                    determine a point and its height on the
                                    interior of the cylinder.  If the
                                    coord_source is
                                    PEXExtTMCoordSourceVertexNormal, a
                                    direction vector in WCs is computed to
                                    determine a point on the interior of the
                                    cylinder.
                                  PEXExtTMProjectionLinearWC: The texture
                                    coordinates are derived from a linear
                                    projection using the equations defined by
                                    p0 and p1 and specified in the
                                    param_data.  Using coord_source equal to
                                    PEXExtTMCoordVertexCoord is recommended
                                    for the linear projection.  */
  PEXExtTMParameterizationData  param_data;
  PEXExtEnumTypeIndex          coord_source;
                               /* Specifies source coordinates for computing
                               the projections.  Supported values are:
                                 PEXExtTMCoordSourceVertexCoord: Use the
                                   vertex coordinate to compute the direction
                                   vector for the projection.
                                 PEXExtTMCoordSourceVertexNormal: Use the
                                   vertex normal to compute the direction
                                   vector for the projection.  The vertex
                                   normal will be computed if it does not
                                   already exist.  Using either the vertex
                                   coordinate or vertex normal will produce
                                   different visual results, one of which may
                                   be more pleasing to the end-user depending
                                   on the given primitive, texture, and
                                   desired results.  */
  unsigned short int           fp_data_index;
                               /* Index within the vertex floating point data
                               list in which to store the calculated texture
                               coordinates.  Space for two coordinates must
                               already exist and this index must point to a
                               valid location.  */
  PEXMatrix     mc_transform;  /* Applied to vertices and vertex normals in
                               model coordinates prior to computing the
                               specified projection.  */
} *tm_coord_data;
/*-----------------------------------------------------------------------------*/
struct PEXExtTMParameterizationData {
  union {
    struct {
      PEXMatrix         matrix;        /* If projection is either
                                       PEXExtTMProjectionSphere or
                                       PEXExtTMProjectionCylinderWC, this
                                       transform is applied to vertices and
                                       vertex normals after mc_transform has
                                       been applied.  It is used to orient
                                       the data relative to the projection
                                       object.  */
    } reflection
    struct {                           /* Linear projection          */
      float             p0[4];         /* equations used with        */
      float             p1[4];         /* PEXExtTMProjectionLinearWC */
    } linear;
    PEXExtImpDepData    imp_dep;       /* Not used by HP PEX */
  } data;
} *param_data;
```

```
/*------------------------------------------------------------------------*/
unsigned int            vertex_fp_data_size
                                        /* Number of floating point data
                                        values defined with the vertex.  To
                                        accommodate texture coordinates, this
                                        number should be at least two and may
                                        be higher if additional floating point
                                        data is included with the primitive's
                                        vertex data.  */
/*------------------------------------------------------------------------*/
unsigned int            vertex_attributes;
                                        /* The flag PEXExtGAData must be
                                        included in this mask if the texture
                                        coordinates are included in the
                                        vertex_lists.  */
/*------------------------------------------------------------------------*/
struct PEXExtListOfVertex{              /* Primitive vertices */
  unsigned long         count;          /* Number of vertices */
  PEXExtArrayOfVertex   vertices;       /* pointer to vertices */
} *vertex_lists;                        /* Space must be allocated within
                                        vertex lists to hold the vertices
                                        computed by this utility.  See the
                                        description under
                                        ``PEXExtFillAreaSetWithData'' for
                                        information on packing the texture
                                        coordinates into the vertex_list.  */
/*------------------------------------------------------------------------*/
union PEXExtArrayOfVertex {
  PEXCoord                      *no_data;
  PEXVertexIndexed              *index;
  PEXVertexRGB                  *rgb;
  PEXVertexHSV                  *hsv;
  PEXVertexHLS                  *hls
  PEXVertexCIE                  *cie;
  PEXVertexRGB8                 *rgb8;
  PEXVertexRGB16                *rgb16;
  PEXVertexNormal               *normal;
  PEXVertexEdge                 *edge;
  PEXVertexIndexedNormal        *index_normal;
  PEXVertexRGBNormal            *rgb_normal;
  PEXVertexHSVNormal            *hsv_normal
  PEXVertexHLSNormal            *hls_normal;
  PEXVertexCIENormal            *cie_normal;
  PEXVertexRGB8Normal           *rgb8_normal;
  PEXVertexRGB16Normal          *rgb16_normal;
  PEXVertexIndexedEdge          *index_edge;
  PEXVertexRGBEdge              *rgb_edge;
  PEXVertexHSVEdge              *hsv_edge;
  PEXVertexHLSEdge              *hls_edge;
```

```
      PEXVertexCIEEdge               *cie_edge;
      PEXVertexRGB8Edge              *rgb8_edge;
      PEXVertexRGB16Edge             *rgb16_edge;
      PEXVertexNormalEdge            *normal_edge;
      PEXVertexIndexedNormalEdge     *index_normal_edge;
      PEXVertexRGBNormalEdge         *rgb_normal_edge;
      PEXVertexHSVNormalEdge         *hsv_normal_edge;
      PEXVertexHLSNormalEdge         *hls_normal_edge;
      PEXVertexCIENormalEdge         *cie_normal_edge;
      PEXVertexRGB8NormalEdge        *rgb8_normal_edge;
      PEXVertexRGB16NormalEdge       *rgb16_normal_edge;
      PEXPointer                     with_fp_data;
    }
```

PEXExtTMCoordSetOfFillAreaSets: Parameters

See also "PEXExtTMCoordFillAreaSetWithData: Parameters" for a description of tm_coord_data, vertex_fp_data_size, vertex_attributes, and vertices.

PEXExtTMCoordTriangleStrip: Parameters

See "PEXExtTMCoordFillAreaSetWithData: Parameters" for a description of tm_coord_data, vertex_fp_data_size, and vertices.

PEXExtTMCoordQuadrilateralMesh: Parameters

See "PEXExtTMCoordFillAreaSetWithData: Parameters" for a description of tm_coord_data, vertex_fp_data_size, and vertices.

Step 4: Set Up the Look-Up Tables (LUTs)

Four LUTs—Binding, Coordinate Source, Composition, and Sampling—are used to control the mapping of textures onto primitives. These LUTs are created, manipulated, and inquired using the standard calls `PEXCreateLookupTable`, `PEXGetTableInfo`, `PEXGetDefinedIndices`, and the extended calls, `PEXExtSetTableEntries`, `PEXExtGetTableEntry`, `PEXExtGetTableEntries`, and `PEXExtFreeTableEntries`.

The function `PEXExtChangeRenderer` must be used to register the Texture Mapping Lookup Tables with the PEX renderer. `PEXExtSetRendererAttributeMask` or `PEXExtSetRendererAttributeMaskAll` may be used to set the mask needed by the `PEXExtChangeRenderer` call. The extended calls `PEXExtGetRendererAttributes` and `PEXExtFreeRendererAttributes` can be used to inquire and free the extended attributes of a PEX renderer.

- "Binding LUT": Associates a texture map with its orientation on a primitive, its texture composition and the texture-map sampling method by referencing the texture map description and indices into the Coordinate Source LUT, Composition LUT, and Sampling LUT. In general, there will be one Binding LUT entry for each primitive in the database that is to be texture-mapped.
- "Coordinate-Source LUT": Defines how a texture map is oriented on a primitive.
- "Composition LUT": Defines how the values in a texture map are blended with a primitive's color and alpha values.
- "Sampling LUT": Defines how a texture map is sampled as it is mapped onto a primitive.

Binding LUT

$\langle table\_type \rangle$ = `PEXExtLUTTMBinding`

```
struct PEXExtTMBindingEntry {
  PEXExtTMDescription   tm_description_id;      /* Texture ID.  Returned by
                                                PEXExtCreateTMDescription */
  PEXTableIndex         coord_source_index;     /* Index into Coordinate
                                                Source LUT */
  PEXTableIndex         composition_index;      /* Index into Composition
                                                LUT */
  PEXTableIndex         sampling_index;         /* Index into Sampling LUT */
}
```

Coordinate-Source LUT

$\langle table\_type \rangle$ = PEXExtLUTTMCoordSource

```
struct PEXExtTMCoordSourceEntry {
  PEXEnumTypeIndex      tm_source;      /* tm_source and fp_data_index are
                                        used only with texture maps that are
                                        defined for explicit parameterization;
                                        that is, parameterization is set to
                                        PEXExtTMParamExplicit when
                                        PEXExtCreateTMDescription is
                                        called.  For other parameterization
                                        methods, these values are ignored.
                                        The only supported value is:
                                          PEXExtTMCoordSourceFloatData: Source
                                            is included in the vertex's
                                            floating-point data list
                                            (default).  */
  unsigned short int    fp_data_index;  /* Specifies the location of the
                                        floating-point data at the end of
                                        vertex if tm_source is set to
                                        PEXExtTMCoordSourceFloatData.  */
  PEXMatrix             orientation;    /* The texture coordinates are
                                        transformed by the orientation matrix
                                        before they are interpolated.  */
}
```

Composition LUT

$\langle table\_type \rangle$ = PEXExtLUTTMComposition

```
struct PEXExtTMCompositionEntry {
  PEXEnumTypeIndex      method;         /* Specifies how the texture map is
                                        blended with a primitive's existing
                                        color and alpha data.  Supported
                                        values are:
                                          PEXExtTMCompositeReplace: The
                                            texture map data replaces the
                                            primitive's existing data
                                            (default operation).
                                          PEXExtTMCompositeModulate: The
                                            texture map and primitive data are
                                            blended.
                                          PEXExtTMCompositeDecal: The
                                            texture-map color and the
                                            primitive color are blended by the
                                            texture map alpha.  If the texture
                                            map alpha is not defined, the
                                            texture map color replaces the
                                            primitive color, and the
                                            primitive's alpha is replaced
                                            with 1.0.  */
  unsigned short        reserved;       /* Ignored */
  union {
    PEXColorSpecifier   decal_bkgd_color;/* Not used by HP PEX */
    struct {
      unsigned long     channel_number; /* R=0, G=1, B=2 */
      PEXColorSpecifier color;
    } blend_env;                        /* Not used by HP PEX */
    struct {
      unsigned long     channel_number; /* R=0, G=1, B=2 */
      PEXColorSpecifier color1;
      PEXColorSpecifier color2;
    } blend_repl;                       /* Not used by HP PEX */
    PEXExtImpDepData    imp_dep;        /* Not used by HP PEX */
  } data;
}
```

Sampling LUT

$\langle table\_type \rangle$ = PEXExtLUTTMSampling

```
struct PEXExtTMSamplingEntry {
  PEXEnumTypeIndex      minification_method;
                                /* Used when multiple texture-map texels map
                                to a single primitive pixel.  Supported values
                                are:
                                  PEXExtTMTexelSampleSingleBase: The closest
                                    single texel selected by the texture
                                    coordinate(s) is sampled from the base
                                    texture map level (default).
                                  PEXExtTMTexelSampleLinearBase: The 2^n
                                    closest texels selected by the texture
                                    coordinate(s) are sampled from the base
                                    texture map level.  Note that n is the
                                    dimension (1D, 2D, or 3D) of the texture
                                    map.  The weighted average of the selected
                                    texels is used.
                                  PEXExtTMTexelSampleSingleInMipmap: The
                                    closest single texel selected by the
                                    texture coordinate(s) is sampled from the
                                    closest texture map level to the sample
                                    depth.
                                  PEXExtTMTexelSampleLinearInMipmap: The
                                    2^n closest texels selected by the
                                    texture coordinate(s) are sampled from the
                                    closest texture map level to the sample
                                    depth.  Note that n is the dimension
                                    (1D, 2D, or 3D) of the texture map.  The
                                    weighted average of the selected texels is
                                    used.
                                  PEXExtTMTexelSampleBetweenMipmaps: The
                                    closest texel selected by the texture
                                    coordinate(s) is sampled from the two
                                    closest texture map levels to the sample
                                    depth.  The texel found at the exact
                                    sample depth by linear interpolation
                                    between these two sampled texels is used.
                                    If the sample depth is beyond the base or
                                    pinnacle texture-map levels, that level is
                                    used and this method behaves like
                                    PEXExtTMTexelSampleSampleInMipmap.
                                  PEXExtTMTexelSampleLinearBetweenMipmaps: The
                                    2^n closest texels selected by the
                                    texture coordinate (s) are sampled from
                                    the two closest texture map levels to the
                                    sample depth.  Note that n is the
                                    dimension (1D, 2D, or 3D) of the texture
                                    map.  A weighted average is taken of these
                                    2^n texels on each of the texture map
                                    levels.  The texel found at the exact
                                    sample depth by linear interpolation
                                    between these two calculated texels is
                                    used.  If the sample depth is beyond the
                                    base or pinnacle texture map levels, that
                                    level is used and this method behaves like
                                    PEXExtTMTexelSampleLinearInMipmap.  */
PEXEnumTypeIndex      magnification_method;
                                /* Used when a single texture-map texel is too
                                large to map to a single pixel of a primitive.
                                Supported values are:
                                  PEXExtTMTexelSampleSingleBase: The closest
                                    single texel selected by the texture
                                    coordinate(s) is sampled from the base
                                    texture map level (default).
                                  PEXExtTMTexelSampleLinearBase: The 2^n
                                    closest texels selected by the texture
                                    coordinate(s) are sampled from the base
                                    texture map level.  Note that n is the
                                    dimension (1D, 2D, or 3D) of the texture
                                    map.  The weighted average of the selected
                                    texels is used.  */
PEXEnumTypeIndex      t0_boundary_condition;  /* X boundary condition */
PEXEnumTypeIndex      t1_boundary_condition;  /* Y boundary condition */
PEXEnumTypeIndex      t2_boundary_condition;  /* Z boundary condition;
                                                 not used by HP PEX */
                                /* Specifies the texturing to be applied when
                                the texture coordinates select a point outside
                                the texture map.  Supported values are:
                                  PEXExtTMBoundaryCondClampColor: The
                                    color specified by the clamp color source
                                    is applied.
                                  PEXExtTMBoundaryCondBoundary:  The closest
                                    boundary texture map texel is used.
                                  PEXExtTMBoundaryCondWrap: Texel
                                    sampling wraps back to the opposite
                                    texture border creating a "rubber stamp"
                                    effect.
                                  PEXExtTMBoundaryCondMirror: Texel
                                    sampling is reversed across the texture
                                  map.  This produces the effect of
                                  alternating the texture map with a version
                                  of the map that is "backwards", "upside
                                  down", or both (depending on the values of
                                  t0_boundary_condition and
                                  t1_boundary_condition).  */
  PEXEnumTypeIndex      boundary_clamp_color_source;
                                /* Determines source of color if the
                                boundary_condition is
                                PEXExtTMBoundaryCondClampColor.  Supported
                                values are:
                                  PEXExtTMClampColorSourceAbsolute:
                                    Texturing is discontinued.  The
                                    primitive's color beyond the texture map
                                    boundary remains unchanged (default).
                                  PEXExtTMClampColorSourceExplicit: The color
                                    specified in clamp_color is used.  */
  PEXColorSpecifier     clamp_color;
                                /* Used when the boundary condition is
                                PEXExtTMBoundaryCondClampColor and the
                                clamp color source is
                                PEXExtTMClampColorSourceExplicit (default is
                                R = G = B = 0.0).  */
  float                 depth_sampling_bias_hint;
                                /* Used to adjust the sampling depth.  A
                                factor of -1.0 moves the sampling depth one
                                level toward the Mip map's base level,
                                effectively making the texturing more
                                detailed or jaggy.  A positive factor moves
                                the sampling depth away from the base level,
                                effectively blurring the texture (default is
                                0.0.).  */
  float                 t0_frequency_hint;     /* \ Frequency hints    */
  float                 t1_frequency_hint;     /* / for each dimension */
  float                 t2_frequency_hint;     /* t2 not used by HP PEX */
                                /* If a particular texture map has low spatial
                                frequency that would lead to unacceptable
                                blurring when sampling occurs, this hint maybe
                                set to a value between 0.0 and 1.0.  A value
                                of 1.0 indicates unknown spatial frequency or
                                high spatial frequency (that is, there are
                                abrupt changes of color in the texture map).
                                The default is 1.0.  */
}
```

Step 5: Render

The rendering step actually applies textures to primitives and controls texture-mapping rendering options. The individual calls included in the rendering step may be executed repeatedly in response to input from the user.

- **Function:** `PEXSetInteriorStyle` `PEXSetBFInteriorStyle`. Set front-face and back-face interior style.
- **Function:** `PEXExtSetTMPerspectiveCorrection` (optional). Sets the method of perspective correction for texture mapping. This determines how texture coordinate values in surface interiors are computed by PEXlib.
- **Function:** `PEXExtSetTMSampleFrequency` (optional). The texture mapping sample frequency specifies the frequency to use when sampling texels in a texture map.
- **Function:** `PEXExtSetTMResourceHints` (optional). Set texture mapping resource hints. Define preferences for resource usage and texture priorities.
- **Function:** `PEXExtSetActiveTextures`. Set currently active front-face/back-face textures.
- **Function:** `PEXExtChangePipelineContext`. Modify the extended pipeline context. Use `PEXExtGetPipelineContext` for inquiries.
- **Function:** `PEXExtFillAreaSetWithData` or `PEXOCCFillAreaSet` (optional). 3D fill-area primitives with additional data. To texture-map a fill area set, this routine must be called after setting interior style to `PEXExtInteriorStyleTexture`.
- **Function:** `PEXExtSetOfFillAreaSets` or `PEXOCCIndexedFillAreaSets` (optional). 3D Set of Fill area primitives with additional data. To texture map a set of fill-area sets, this routine must be called after setting interior style to `PEXExtInteriorStyleTexture`.
- **Function:** `PEXExtTriangleStrip` or `PEXOCCTriangleStrip` (optional). 3D Triangle Strip primitive with additional data. To texture map a triangle strip, this routine must be called after setting interior style to `PEXExtInteriorStyleTexture`.
- **Function:** `PEXExtQuadrilateralMesh` or `PEXOCCQuadrilateralMesh` (optional). 3D Quadrilateral Mesh primitive with additional data. To texture map a quadrilateral mesh, this routine must be called after setting interior style to `PEXExtInteriorStyleTexture`.

If the Output Commands used in the Rendering phase of texture mapping are placed in a structure, and an application must access those Output Commands, the following extended calls must be used:

- PEXExtCountOCs
- PEXExtDecodeOCs
- PEXExtEncodeOCs
- PEXExtFetchElements
- PEXExtFetchElementsAndSend
- PEXExtFreeOCData
- PEXExtGetSizeOCs

PEXSetInteriorStyle: Parameters

```
int     style;              /* Interior style.  Set style to
                            PEXExtInteriorStyleTexture to enable texture
                            mapping for subsequent primitives defined by
                            PEXExtFillAreaSetWithData,
                            PEXExtSetOfFillAreaSets, PEXExtTriangleStrip,
                            or PEXExtQuadrilateralMesh.  If the style
                            PEXExtInteriorStyleTexture is applied to any
                            other primitives, the interior style will be
                            treated as PEXInteriorStyleSolid.  */
```

PEXExtSetTMPerspectiveCorrection: Parameters

```
int     method;             /* Type of texture mapping interpolation to
                            apply.  Supported values are:
                              PEXExtTMPerspCorrectNone: Texture
                                coordinates are linearly interpolated
                                without any effort to apply perspective
                                correction (default).
                              PEXExtTMPerspCorrectPixel: As the
                                texture-mapping coordinates are
                                interpolated, their values are manipulated
                                at each step to account for a perspective
                                projection.  */
```

PEXExtSetTMSampleFrequency: Parameters

```
int     frequency;          /* Sample frequency.  Supported value is:
                              PEXExtTMSampleFrequencyPixel: Texels are
                                sampled once for each pixel.*/
```

PEXExtSetTMResourceHints: Parameters

```
int            optimization_hint;     /* Resource optimization approach to
                                      consider for all subsequently
                                      activated textures.  Supported values
                                      are:
                                        PEXExtTMResourceHintNone: Use the
                                          default optimization.
                                        PEXExtTMResourceHintSpeed: Attempt
                                          to optimize performance of texture
                                          mapping.
                                        PEXExtTMResourceHintSpace: Attempt
                                          to optimize memory usage of
                                          texture mapping.  */
unsigned int   count;                 /* Number of indices in the priorities
                                      list.  */
PEXTableIndex  priorities;            /* Array of Binding LUT indices in
                                      priority order (first texture is
                                      expected to be used most, etc.).  */
```

PEXExtSetActiveTextures: Parameters

```
unsigned short int      count;        /* The number of textures listed in
                                      the textures list */
unsigned short int      *textures;    /* An ordered list of texture Binding
                                      LUT indices corresponding to the
                                      texture maps to activate and apply to
                                      subsequent extended primitives when
                                      their interior style is
                                      PEXExtInteriorStyleTexture.  If
                                      texturing is enabled but this list is
                                      empty, a default black-and-white
                                      checkerboard texture is applied to the
                                      primitives.  */
```

PEXExtChangePipelineContext: Parameters

```
unsigned long   *value_mask;    /* Indicates which attribute values are
                                   specified.  PEXExtSetPCAttributeMask
                                   can be called to set up non-extended and
                                   extended portions of the value_mask.
                                   Texture-mapping pipeline context attributes
                                   are:
                                     PEXExtPCTMPerspectiveCorrection (see
                                       PEXExtSetTMPerspectiveCorrection).
                                     PEXExtPCTMResourceHints (see
                                       PEXExtSetTMResourceHints).
                                     PEXExtPCTMSampleFrequency (see
                                       PEXExtSetTMSampleFrequency).
                                     PEXExtPCActiveTextures (see
                                       PEXExtSetActiveTextures).
                                     PEXExtPCBFActiveTextures (see
                                       PEXExtSetBFActiveTextures). */
```

PEXExtFillAreaSetWithData: Parameters

```
unsigned int    vertex_fp_data_size;    /* Number of floating point values
                                        defined with each vertex.  This number
                                        should be increased by two if texture
                                        coordinates are included with the data
                                        in vertex_lists and were calculated
                                        by the utility
                                        PEXExtTMCoordFillAreaSetWithData.  */
unsigned int    vertex_attributes;      /* The flag PEXExtGAData must be
                                        included in this mask if the texture
                                        coordinates are included in the
                                        vertex_lists.  */
/*-----------------------------------------------------------------------*/
PEXExtListOfVertex      *vertex_lists;  /* If this primitive is to be textured
                                        with a map that was described to
                                        PEXExtCreateTMDescription using
                                        parameterization method
                                        PEXExtTMParamExplicit, the vertex
                                        lists must contain texture coordinates
                                        for each vertex.  The texture
                                        coordinates can be derived by the
                                        application or using
                                        PEXExtTMCoordFillAreaSetWithData.
                                        The data must be packed in vertex_lists
                                        in the order:
                                          1.  coordinate data
                                          2.  color data (if present)
                                          3.  normal (if present)
                                          4.  edge data (if present)
                                          5.  additional floating point data,
                                              such as texture coordinates */
/* For example, if the application has the coordinates, normal, and two
texture coordinates for each vertex, the following data structure should be
defined: */

typedef struct {                /* The first entry should be an existing
                                PEXArrayOfVertex member.  */
  PEXVertexNormal               coords_and_normal;
  float                         texture_coords[2];
} MyPEXVertexNormalTexCoord;
MyPEXVertexNormalTexCoord       my_vertex_data;
PEXExtArrayOfVertex             vertex_data;

my_vertex_data.coords_and_normal.point.x = MY_VERTEX_X;
my_vertex_data.coords_and_normal.point.y = MY_VERTEX_Y;
my_vertex_data.coords_and_normal.point.z = MY_VERTEX_Z;
my_vertex_data.coords_and_normal.normal.x = MY_VERTEX_NORMAL_X;
my_vertex_data.coords_and_normal.normal.y = MY_VERTEX_NORMAL_Y;
my_vertex_data.coords_and_normal.normal.z = MY_VERTEX_NORMAL_Z;
my_vertex_data.texture_coords[1] = MY_TEXTURE_COORDINATE_X;
my_vertex_data.texture_coords[2] = MY_TEXTURE_COORDINATE_Y;
/* Note that PEXExtTMCoordFillAreaSetWithData can be used to fill in
my_vertex_data.  */
vertex_data.with_fp_data = (PEXPointer) &my_vertex_data;
```

`PEXExtSetOfFillAreaSets`: Parameters

See description of `PEXExtFillAreaSetWithData`, `PEXOCCFillArea` or `PEXOCCFillAreaSet` for more information about `vertex_fp_data_size`, `vertex_attributes`, and `vertices`.

`PEXExtTriangleStrip`: Parameters

See description of `PEXExtTriangleStrip` or `PEXOCCTriangleStrip` for more information about
`vertex_fp_data_size`, `vertex_attributes`, and `vertices`.

`PEXExtTMCoordQuadrilateralMesh`: Parameters

See description of `PEXExtTMCoordQuadrilateralMesh` or `PEXOCCTMCoordQuadrilateralMesh` for more information about `vertex_fp_data_size`, `vertex_attributes`, and `vertices`.

Step 6: Cleanup

Cleanup involves releasing the memory used by texture resources when those resources are no longer needed.

- **Function:** `PEXExtFreeTM` (optional). Free a texture map resource.
- **Function:** `PEXExtFreeTMDescription` (optional). Free a texture map description resource.

`PEXExtFreeTM`: Parameters

```
PEXExtTextureMap        texture_map;    /* Texture resource to free; created
                                        using PEXExtCreateTM.  */
```

`PEXExtFreeTMDescription`: Parameters

```
PEXExtTMDescription tm_description; /* Texture description resource to free;
                                    created using PEXExtCreateTMDescription. */
```

10

Texture Mapping Tutorial

This tutorial describes the “big picture” of texture mapping to help you, the application developer, determine how best to integrate this technology into your applications. Listed herein are many of the considerations that must be addressed when presenting texture mapping to an end-user.

Texture mapping has many different uses, including:

- Data Display. Texture maps can be used to display many different kinds of scientific data including, but not limited to satellite, seismic, medical imaging, and geographic data.

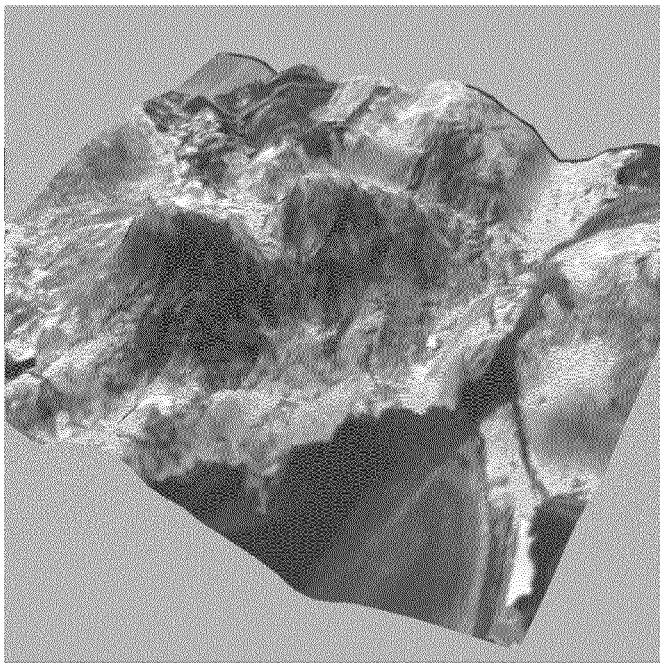


Figure 10-1. Texture Mapping to Display Data

- Realism. Texture mapping greatly increases the realism of a rendering and is especially valuable for presentation and design. Imagine how uninteresting and artificial-looking the image below would be with an unpatterned floor, unpatterned walls, a blank (or simple geometric) picture on the wall, blank windows, flat fireplace, etc.

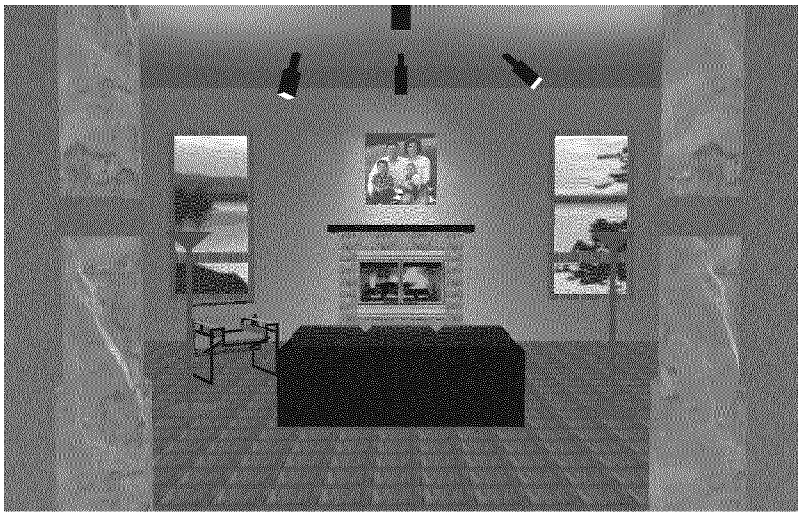


Figure 10-2. Texture Mapping to Add Realism

- Data Reduction. Data size can be drastically reduced by mapping details onto geometric objects instead of modeling those details. Imagine the size of the dataset required if, in the picture above, every brick in the fireplace, every tongue of flame in the fire, every tree branch and cloud outside the windows, every piece of wood in the parquet floor, every streak in the marble columns, and every bit of pattern in the wallpaper were individually geometrically modeled! It quickly becomes obvious that texture mapping can save enormous amounts of disk space and processing time.

Creating and Editing Textures

End-users will want to use texture maps from many different sources. For scientific data display applications, the texture maps will be created from real-world data and may be in almost any for mat. One of the application developer's jobs is to support the importation of this data into the application or to document the supported formats so the end user can translate their data into these formats.

- Some end-users will want to create their own texture maps using paint programs, scanners, video-in equipment, screen grabs, etc. Again, the application needs to provide a mechanism for the user to use files with these formats for texture maps.
- The binary version of a texture file reader, `read_texture`, is included in the directory ⟨*hp-examples*⟩`/TexMap`[1]. This utility will read files from several different formats and produce a PEXlib-compatible texture data structure suitable for passing to `PEXExtCreateFilteredTM`. A source code interface to the `read_texture` utility is provided in `read_tex.c` in the same directory. All of the example programs included in this tutorial use this interface to read texture files. The utility will read the files formats listed in the following table of file suffixes, formats, supported versions, and typical contents:

`.tif`	TIFF, version 5.0 (6.0 for TIFF JPEG). Contains PC, scanned, or FAX images. Note that TIFF images may be in uncompressed format, or any of the following compressed formats: JPEG, LZW, G3, G4, or Packbits.
`.jpg` or `.jpeg`	JFIF, version 8-R8. Contains JPEG-compressed images.
`.gif`	GIF, version 87a. Contains `xv` and `xgif` images.
`.xwd`	XWD, version X11. Contains pixmap images from `xwd` (Z format).
`.xbm` or `.bm`	XBM, version X11. Contains bitonal X bitmap images.
`.xpm` or `.pm`	XPM, version 3.0. Contains color X pixmap images.
`.bmf`	BMF, version 1. Contains Starbase bitmap images (Z format).

[1] The actual pathname of this directory depends on the file system structure. See the *Graphics Administration Guide* for details.

The file reader, `read_texture`, is provided for your convenience, particularly to use one of the example programs with a texture file of your own choosing. The file reader is not guaranteed to read all files in the supported formats; in particular, it does not support maps with an alpha channel included. For a more robust and complete solution, the HP product, HP B2157A "Image Developer's Toolkit for the S700" supplies source code for read/write, display, file format conversions, and compression/decompression for multiple file types. Please contact your local Hewlett-Packard Sales Office or the Customer Information Center at (800)752-0900 for more information on ordering this product.

- Libraries of predefined textures are also available. These provide myriad textures including interiors (carpets, tiles, wood grains), and natural textures (skies, rainbows, etc.).

Sources of Textures

The following products facilitate texture creation and editing. Please contact the companies directly for details and pricing. These products are listed for your information only, and except for the products from HP, do not represent an endorsement by HP.

- *Pixel!FX* Scanner support, image viewing and manipulation including resize, blend, rotate, etc. Supports TIFF, XWD, X bitmap and GIF formats. Contact: Mentalix, 1700 Alma Drive Suite 110, Plano, Texas 75075. Phone: (214)423-9377; Fax: (214)423-1145.
- *HP C1788A ScanJet IIc for Series 700 Workstations* High-performance, color and grayscale flatbed scanner. Scans to TIFF format. Contact your local Hewlett-Packard Sales Office or the Customer Information Center at (800)752-0900.
- *HP Z1100A RasterOps VideoLive Card* Digitizes and captures images from video to main memory on demand. Saves images to TIFF files. Contact your local Hewlett-Packard Sales Office or the Customer Information Center at (800)752-0900.
- *HP MPower* Software supports image scanning, view, and manipulation of images for scale, contrast, and brightness. Supports TIFF, JFIF, GIF, XWD, XBM, XBM, and BMF. Contact your local Hewlett-Packard Sales Office or the Customer Information Center at (800)752-0900.
- *HP B2157A Image Developer's Toolkit for S700* Developer's tool for bundled HP-UX Image Lib. ImageLib supports read/write, display, file format conversions and compression/decompression for multiple file types including TIFF, JFIF, GIF, XWD, XBM, and XPM. Includes source code. Contact your local Hewlett-Packard Sales Office or the Customer Information Center at (800)752-0900.

Predefined Textures

There are texture map libraries on CD-ROM available from the following sources. Please contact the companies listed for pricing and more information. These products are listed for your information only, and do not represent an endorsement by HP.

- *Pixar One Twenty Eight CD* Photographic textures include bricks, fabrics, environmental, landscaping, etc. Includes 128 512×512×24 bit images in TIFF format. Contact: Pixar, Attn: Renderman Retail, 1001 West Cutting Boulevard, Richmond, CA 94804. Phone: (510)236-4000, fax: (510)236-0388.
- *ImageCELs*® Includes building materials, environmental, "designer" patterns, landscaping, and industrial finishes. Supports file formats: 8-bit GIF, PCS, IFF, TIFF, TGA, and DIB; 16-bit TGA, I16, WIN; 24-bit TGA and CEL. To request a catalog, call (408)252-4706 (you must use fax machine handset). Contact: Imagetects™, P.O. Box 4, Saratoga, CA 95071-0004. Phone: (408)252-5487, fax: (408)252-7409.
- *PhotoDisc* Photographs of people, nature, places, etc. in PICT or JPEG-Compressed TIFF format. Contact: PhotoDisc, Inc. 2013 Fourth Avenue, Seattle, Washington 98121. Phone: (206)441-9355, fax: (206)441-9379.

User Interface Considerations

Determining how to present texture mapping to the end user of an application deserves careful consideration. When texture mapping is used to increase realism, "getting it right" relies on trial-and-error user interaction. Users will want to try different texture placements and rendering options before deciding which combination is best for the specific texture and geometry. The application developer's challenge is to give end users an appropriate amount of control over the options. Advanced texture-mapping users will want control over all possible parameters to achieve the desired effects. Novice users, however, would be confused by having too many choices. Throughout this document, user-interface considerations will be discussed as they pertain to the different features of PEXlib texture mapping.

At a minimum, a texture-mapping application will need to provide a list (or set of lists) of available textures. As discussed, these textures may come from a variety of sources. An application may want to provide searching mechanisms based on a texture name, picture icon of the texture, or key words (such as "woods") to facilitate texture retrieval. Hyperlink technologies are another possibility for searching and retrieving specific textures. Many applications will also need to provide a way for users to pick an object to receive the chosen texture(s).

Object partitioning must also be considered by users of texture mapping. If a user wants to realistically texture-map an office chair, for example, the cushions and legs must be partitioned into separate geometric objects so that upholstery can be applied to the cushions, and chrome or other material can be applied to the legs. For advanced users, the cushions may need to be partitioned further to accomplish the desired effect on the top and each side of each cushion. This partitioning must be accomplished when the model is first created or the application must provide a mechanism for object partitioning at any time during the design process.

Using PEXlib for Texture Mapping

There are six main steps required to use PEXlib texture mapping; these steps are described in the next sections of the tutorial:

- "Step 1: Setup"
  A. Ensure texture mapping support.
  B. Inquire implementation dependent constants.
- "Step 2: Texture Preparation"
  A. Create a filtered map resource from the source texture map and import it into PEXlib.
  B. Create a texture map description resource.
- "Step 3: Geometry Preparation"
  A. Compute texture coordinates with PEXlib utilities.
- "Step 4: Set up Texture Mapping Lookup Tables (LUTs)"
  A. Create one or more Coordinate Source LUT entries to specify how a texture is mapped onto a primitive.
  B. Create one or more Composition LUT entries to describe how texture map data is combined with the existing color and alpha data of a primitive.
  C. Create one or more Sampling LUT entries to specify how a filtered texture map is sampled or accessed.
  D. Create one or more Binding LUT entries to associate a texture description with entries in the Coordinate Source, Composition, and Sampling LUTs.
- "Step 5: Render"
  A. Set up rendering options.
  B. Enable texture mapping for subsequent primitives.
  C. Activate texture(s).
  D. Render primitives.
- "Step 6: Clean Up"
  A. Free resources used by texture mapping.

The graphics pipeline—that sequence of steps your graphical data goes through in the process of getting from your model to the finished image on the display—is as follows:

1. Prespecular texturing;
2. Lighting and shading;
3. Postspecular texturing;
4. Depth cueing;
5. Screen door transparency applied;
6. Replacement rules are applied;
7. Final alpha blending with the frame buffer (if supported in hardware).

Of course, your program may or may not use all of these features; the above merely shows the order in which the processes may occur.

Step 1: Setup

A. *Ensure texture mapping support.* Setup for texture mapping ensures that texture mapping is supported by the current PEXlib implementation by calling `PEXGetEnumTypeInfo`. (Don't forget to call `PEXFreeEnumInfo`, when finished with the memory allocated by `PEXGetEnumTypeInfo`.) See also "Step 1: Setup" in Chapter 9 in the Texture Mapping Overview.

B. *Inquire implementation dependent constants.* In general, all applications should inquire texture mapping implementation dependent constants via `PEXGetImpDepConstants`. Different implementations of PEXlib may return different values for these constants that must be considered. For example, the constant `PEXExtIDPowerOfTwoTMSizesRequired` must be passed by the application to either `PEXExtCreateFilteredTM` or `PEXExtCreateFilteredTMFromWindow`.

Step 2: Texture Preparation

A. *Create a filtered map resource from the source texture map and import it into PEXlib.*

Texture preparation involves creating a filtered map (a "MIP map") from your source map using `PEXExtCreateFilteredTM` or `PEXExtCreateFilteredTMFromWindow` and importing that map into PEXlib via `PEXExtCreateTM`.

Note that after `PEXExtCreateTM` has been called, `PEXExtFreeFilteredTM` can be called to reclaim memory used for the filtered map.

(See also "User Interface Considerations for Creating Filtered Texture Maps" and "Discussion: MIP Map".)

B. *Create a texture map description resource.*

Once the map has been imported into PEXlib, `PEXExtCreateTMDescription` must be called to combine texture identifier(s), parameterization and rendering information to form a texture map description. In addition to providing `PEXExtCreateTMDescription` with the texture resource identifier(s) returned by `PEXExtCreateTM`, three key parameters must be specified:

- `parameterization`
- `param_data`
- `rendering_order`

The parameterization parameter specifies how texture coordinates will be derived. Texture coordinates determine how a texture is mapped onto a primitive. The possible values for parameterization are `PEXExtTMParamExplicit`, `PEXExtTMParamReflectSphereVRC`, `PEXExtTMParamReflectSphereWC`, and `PEXExtTMParamLinearVRC`. The following list describes the different effects produced by using these different values.

- `PEXExtParamExplicit` (view-independent, standard mapping)

  The application calculates texture coordinates *or* uses the client-side utilities `PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, and/or `PEXExtTMCoordQuadrilateralMesh`.

  Visual result: The texture map is fixed to the primitive, regardless of the position of the camera or animation of the primitive.
- `PEXExtTMParamReflectSphereVRC` (view-dependent, reflection mapping)

  Texture coordinates are determined by the PEXlib server with respect to the current camera position.

  Visual result: The texture map is reflected onto the primitive. If the camera moves, the texture map will appear to move along with it and the object will reflect essentially the same portion of the texture map regardless of the point of view.
- `PEXExtTMParamReflectSphereWC` (view-dependent, reflection mapping)

  Texture coordinates are determined by the PEXlib server with respect to the current camera position.

  The texture map is reflected onto the primitive and produces "true" environment mapping. If the camera moves, the object will reflect a different portion of the texture map in much the same way as if you were to move around a chrome ball that reflects your environment.
- `PEXExtTMParamLinearVRC` (view-dependent, standard mapping)

  Texture coordinates are determined by the PEXlib server with respect to the current camera position.

  Texture coordinates are determined by the PEXlib server with respect to a projection reference plane defined in view reference coordinates (VRC).

  At the time of activation (`PEXExtSetActiveTextures`), equations *p0* and *p1* are inversely transformed from VRC space back into Model Coordinate (MC) space. Once there, they define a projection function such that objects appear to "swim" through a solid field of texture coordinates. This technique can sometimes be very effective in revealing the surface contours of an object in motion if the right type of texture map grid is employed.

> It should be noted that the visual result of this technique is subject to the currently active view orientation, local and global transforms at the time of activation. If scaling and rotation are incorporated in any of these matrices, repetition and distortion of the texture field may result.

The distinction between view-dependent and view-independent texture coordinates is an important one. View-independent texture mapping results in a texture fixed on a primitive that does not change when the point of view changes. View-dependent mapping means that the apparent texture does change depending on the point of view of the camera.

When view-independent mapping (`PEXExtParamExplicit`) is desired, the texture coordinates can be calculated once for each primitive and need not be recomputed if the position of the camera changes. For this reason, the client-side utilities, `PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, and `PEXExtTMCoordQuadrilateralMesh` are provided to pre-calculate the texture coordinates. These utilities are described in the next section.

View-dependent mapping, on the other hand, demands that the texture coordinates be calculated each time the view point is moved. In this case, it is logical for the server to calculate the coordinates each time a texture mapped primitive is rendered. View-dependent, server-side texture coordinates are derived for parameterization values `PEXExtTMParamSphereVRC`, `PEXExtTMParamSphereWC`, and `PEXExtTMParamLinearVRC`. The former two values cause a projection onto an infinite sphere to be used in calculating the texture coordinates, while `PEXExtTMParamLinearVRC` causes a linear projection to be used to compute the coordinates. See "Parameterization", below, for a more thorough explanation of the process of calculating texture coordinates.

Note that two of the four possible parameterization values, `PEXExtTMParamReflectSphereVRC` and `PEXExtTMParamReflectSphereWC`, result in "reflection mapping." The other two methods produce "standard" texture mapping. A reflection mapping differs from a "standard" texture mapping in that the texture coordinates for reflection mapping are based on the calculation of reflection vectors for each vertex in a primitive. The result is an object that reflects, or mirrors, the texture map. Reflection mapping can be likened to viewing the reflections of a room by looking at a shiny Christmas tree ornament. "Standard" texture mapping does not rely on reflection vectors at all. A texture is placed on an object much the same way as a piece of wrapping paper is applied to a gift.

Environment mapping is a form of reflection mapping where the texture map is a picture of the environment from the viewpoint of the object at the center of the environment. PEXlib reflection mapping can also be used to simulate chrome or other shiny materials. Because some shiny materials like chrome are not perfect reflectors, reflection mapping provides a relatively inexpensive way to simulate a shiny object.

The `param_data` parameter contains the linear equations *p0* and *p1* and a `reflection_matrix`. The values specified in *p0* and *p1* define the linear equations used for parameterization equal to `PEXExtTMParamLinearVRC`. The reflection_matrix can be used to orient a spherical or cylindrical projection object so that texture seams or distortions will appear where they are less noticeable once the texture is mapped onto a primitive. It is often desirable to align the axis of revolution of the projection object with the natural axis of symmetry for the geometrical object (if it has one). It is recommended that the matrix contain only 3D rotations and it should be noted that the matrix is never applied to the linear projection.

In addition to parameterization information, the `rendering_order` must be passed to `PEXExtCreateTMDescription`. The two alternative values are `PreSpecular`, meaning that any specular highlight is added after the texture component, and `PostSpecular`, which specifies that texture mapping affects the color after specular has been applied.

Parameterization

Surface parameterization is the name given to the process of generating texture coordinates. Texture coordinates (*t0*,*t1*) “tie” a 2D texture map to a 3D geometric model—a sphere, a cylinder or a plane:

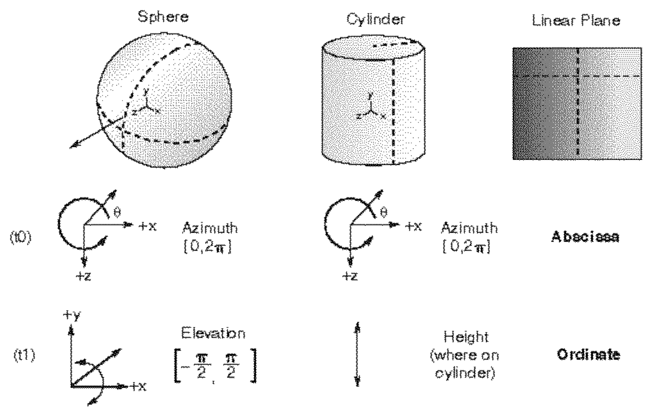


Figure 10-3. Coordinate Systems of the Three Types of Projection Objects

Finding the correspondence between a 2D map and 3D object is not as trivial as it might appear. In the case of PEXlib, mathematical projections are used to derive the correspondence, in a two-step process.

First, the geometric model is projected onto a standard volume, or projection object, and second, the projection object is "unfolded" to a flat, 2D surface that corresponds to a 2D texture map.

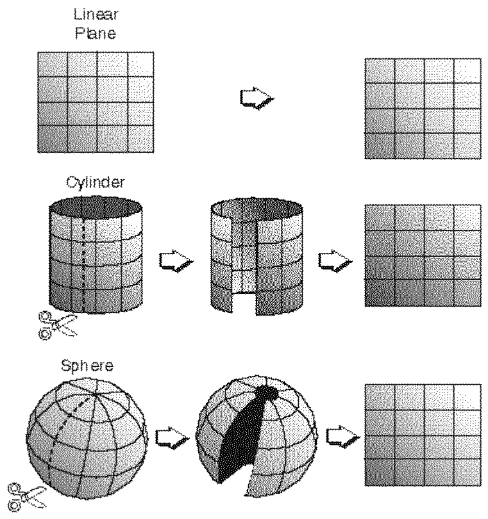


Figure 10-4. "Unfolding" a Projection Object

These two steps are described in more detail below.

A. The geometric model is conceptually placed in the center of a projection object. (PEXlib supports projection objects sphere, cylinder, and plane). For each vertex in the model, a vector is calculated that intersects the projection object.

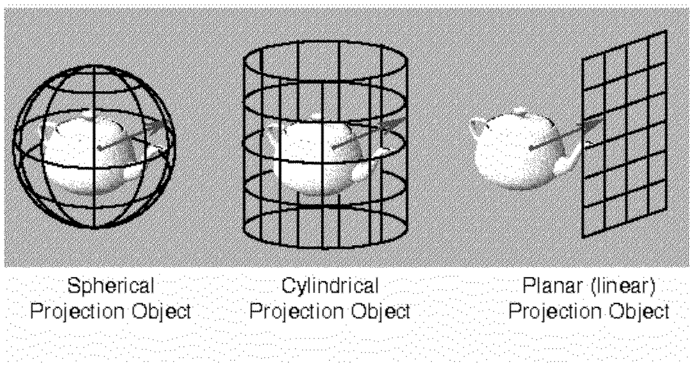

Spherical Projection Object
Cylindrical Projection Object
Planar (linear) Projection Object

Figure 10-5. Geometric Model with Various Projection Objects

Projection objects of may be rotated using the matrix passed to `PEXExtCreateTMDescription` or the `PEXExtTMCoord*` utilities.

Several different methods of projection, or methods to find the intersection, are supported including using a reflection vector, vertex coordinate, or vertex normal.

For spherical and cylindrical projections, *t0* corresponds to the intersection with the projection object in terms of an azimuth and will have a value between 0 and 2π. The *t1* coordinate for a spherical projection corresponds to an elevation between $-\pi/2$ and $\pi/2$. The *t1* coordinate for a cylindrical projection corresponds to a height on the cylinder. For a planar projection, *t0* and *t1* correspond to the abscissa and ordinate on the plane, respectively.

B. The second step in the process involves unfolding the projection object into a flat plane which trivially maps to a 2D texture map. For a spherical projection, this means mapping from the range $[0,2\pi]$ to [0,1] in X, and from the range $[-\pi/2,\pi/2]$ to [0,1] in Y. For a cylindrical projection, $[0,2\pi]$ maps to [0,1] in X and the $[0,h]$ maps to [0,1] in Y, where h is the height of the cylinder. Finally, for planar projections, [0,w] maps to [0,1] and [0,h] maps to [0,1] where w and h are the width and height of the projection plane, respectively.

Before the texture coordinates are used to access the texture map, they are first transformed by the orientation matrix in the Coordinate Source LUT. This transformation effectively allows the texture to be translated, rotated and scaled before it is applied to the geometry.

User Interface Considerations for Parameterization

An application may allow an advanced user to choose between the many different parameterization methods supported by PEXlib. The user interface for the methods may be presented in terms of view independence, reflection mapping vs. "standard" texture mapping, and projection objects. To further control the generation of texture coordinates, a user may also need to be able to manipulate the projection object matrix for each projection. For explicit projections created by `PEXExtTMCoord*`, a user may be given a choice of using the vertex coordinates or vertex normals when creating the projection vector for a projection. This allows the user to further "tweak" the results of the texture coordinates calculations to produce the desired effects.

Note that although PEXlib supports powerful texture coordinate generation techniques, some advanced texture mapping applications may want to extend the capabilities further and allow users to modify individual texture coordinates interactively, thus achieving complete control over texture placement. Such an application would want to display the actual texture coordinates and the texture mapped object and allow the user to pick one or more coordinates and move them using a mouse or other input device. Another scheme would display the 2D texture map and overlay the 2D texture coordinates. This would allow the user to manipulate the coordinates in 2D, a much simpler operation than manipulation in 3D.

Step 3: Geometry Preparation

A. *Compute texture coordinates with PEXlib utilities.*

Recall from the "Step 2: Texture Preparation" discussion, that when parameterization is set to `PEXExtParamExplicit` and passed to `PEXExtCreateTMDescription`, the texture coordinates are expected to be computed by the client-side PEXlib utilities (or by the application), not by the PEXlib server as determined by the other values of parameterization. PEXlib will generate texture coordinates for Extended or OC Context Fill Area Sets, Set of Fill Area Sets, Triangle Strips, or Quadrilateral Meshes via the utility routines `PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, or `PEXExtTMCoordQuadrilateralMesh`.

The only PEXlib primitives that can be texture-mapped are either `PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, and `PEXExtQuadrilateralMesh`; or, alternatively, `PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh`.

Texture coordinates created by the `PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, and `PEXExtTMCoordQuadrilateralMesh` routines result in "standard" texture mapping, as opposed to reflection mapping. Standard mapping is independent of the camera, and as such, may be calculated once for each primitive. These utilities calculate the texture coordinates and store them with the primitive's vertex data. The application must reserve space within the coordinate data for the texture coordinates before these utilities are called.

The PEXExtTMCoord* utilities require that the following information be specified:

- projection: PEXExtTMProjectionSphereWC, PEXExtTMProjectionCylinderWC, and PEXExtTMProjectionLinearWC are supported projection objects. The primitive described in a call to one of these utilities will be conceptually projected onto the projection object (sphere, cylinder, or plane) to derive the texture coordinates.

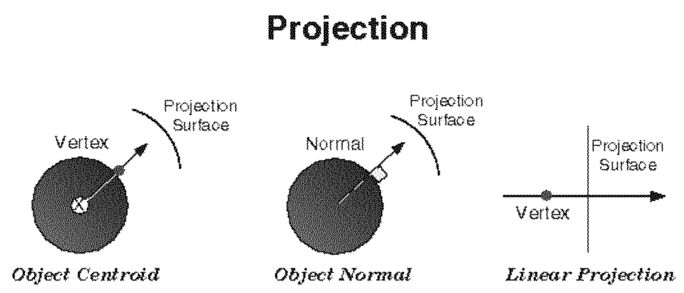


Figure 10-6. Projection Methods Used to Calculate Texture Coordinates

- matrix: For greater control over the positioning of texture coordinates relative to a primitive, the projection object can be transformed by the matrix in param_data for either the spherical or cylindrical projections. This can serve to change the orientation of the projection. The user may want to change the orientation of the projection to move texture seams or distortions where they will be less noticeable once the texture is mapped onto a primitive. It is often desirable to align the axis of revolution of the projection object with the natural axis of symmetry for the geometrical object (if it has one). The projection matrix is passed to the PEXExtTMCoord* utilities in the tm_coord_data parameter for explicit projections and to PEXExtCreateTMDescription for all other projections. It is recommended that the matrix contain only 3D rotations and should be noted that it is never applied to linear projections.
- coord_source: The coordinate source to be used to compute the texture coordinates must be supplied. For cylindrical and spherical projections, a direction vector is computed using either the vertex coordinate or the vertex normal. Using one or the other may produce results that are more pleasing to the end user depending on the primitive and texture map.

If `PEXExtTMCoordSourceVertexNormal` is selected, but normals are not supplied with the primitive's vertex data, the normals will be derived by the utility.

- `model_transform`: A model coordinate transform is provided to convert from model to world coordinates, if desired. One advanced use of the `model_transform`, for example, is a tire modeled once but instantiated four times. The model transform for each instantiation could be specified and the texture coordinates for each model transform stored in a different location of the tire primitives' vertex lists. This would result in four sets of texture coordinates being stored with each vertex. By using the `mc_transform`, each tire would be properly texture mapped according to its orientation in the scene.
- `vertex_attributes`: Note that an application must set the flag `PEXExtGAData` in the `vertex_attributes` parameter passed to the `PEXExtTMCoord*` utilities to notify PEXlib that there will be texture coordinates stored with the vertex data.

(See also "User Interface Considerations for Parameterization".)

Step 4: Set up Texture Mapping Lookup Tables (LUTs)

The texture mapping Lookup Tables control how a texture is positioned on an object and how the texture mapped object will appear once it is rendered. These LUTs are created, manipulated, and inquired using the standard calls, `PEXCreateLookupTable`, `PEXGetTableInfo`, `PEXGetDefinedIndices`, and the extended calls `PEXExtSetTableEntries`, `PEXExtGetTableEntry`, `PEXExtGetTableEntries`, and `PEXExtFreeTableEntries`.

Note that to change renderer attributes, including which Lookup Tables are associated with a renderer, the extended routine `PEXExtChangeRenderer` must be used.

A. Create one or more "Coordinate-Source LUT" in Chapter 9 entries to specify how a texture is mapped onto a primitive.

   The Coordinate Source LUT specifies how a texture is placed on the geometry. Of prime importance is the orientation matrix. Each texture coordinate is transformed by this matrix before accessing the texture map. Many applications will need to provide end-user access to the orientation matrix (via a mouse or other input device) so that textures can be positioned precisely on objects. For these applications, there is no exact way to know how a texture should be oriented and user input is indispensable. For example, only the end-user knows how a texture mapped label should be oriented on a package. For other applications, particularly when mapping real-world data, the position of the texture map is intrinsic to the texture data and the user will not need to position the textures on the object.

   The orientation matrix differs from the reflection matrix used by `PEXExtCreateTMDescription` and `PEXExtTMCoord*` in that it is applied to the texture coordinates before accessing the texture map while the reflection matrix is applied to the projection object as one of the steps taken to determine the texture coordinates.

B. Create one or more "Composition LUT" in Chapter 9 entries to describe how texture-map data is combined with the existing color and alpha data of a primitive.

   The Composition LUT specifies how a texture map is combined with a primitive's existing color and alpha values. Supported composition techniques are Replace, Modulate, and Decal:

C. The *Replace* operation overwrites the primitive's existing color with that of the texture map. Likewise, if a texture alpha is specified, the primitive's alpha is replaced. If alpha is not specified, the primitive's alpha remains unchanged.

D. The *Modulate* operation multiplies each component (R, G, B) of the primitive's existing color with each component of the texture map color. If texture alpha is specified, the primitive's alpha is multiplied by the texture alpha to determine the final alpha.

E. The *Decal* operation functions much like Replace when alpha is not included in the texture map, in that the texture map color replaces the primitive's color. However, alpha is set to 1.0. If, on the other hand, alpha is specified in the texture map, the following equation is used to determine the final blended color:

$$C_{out} = C_{in} \times (1 - t_a) + t_c \times t_a$$

where C_{in} is the primitive's existing color, t_a is the texture map alpha, and t_c is the texture map color.

F. Create one or more "Sampling LUT" in Chapter 9 entries to specify how a filtered texture map is sampled or accessed.

The entries in the Sampling LUT define how a texture map is sampled as it is mapped onto a primitive. Several different parameters determine the sampling method:

- When the texture coordinates for a primitive map multiple texels to a single pixel, the minification method is used to determine exactly how the texels should be used to determine the color for that pixel.
- When the texture coordinates map one texel to multiple pixels, the magnification method determines what values should be assigned to the multiple pixels.
- Boundary conditions `t0_boundary_condition`, `t1_boundary_condition`, and `t2_boundary_condition` specify how texturing should be applied when the texture coordinates select a point outside the texture map. The `t0_boundary_condition` is for the *t0* (horizontal) coordinate, `t1_boundary_condition` is used for the *t1* (vertical) coordinate, and `t2_boundary_condition` is not used.
- The `depth_sampling_bias_hint` can be used to adjust which level of a texture MIP map is sampled. This results in sharpening or blurring the texture detail depending upon the new level selected in the map.

- The `t0_frequency_hint`, `t1_frequency_hint`, and `t2_frequency_hint`, can be applied by advanced texture mapping users based on the actual data in a given texture map. If blurring in any one direction would be unacceptably high due to low spatial frequency, this hint may be set to a value between 0.0 and 1.0 for that direction. The affect will be to bias the texture map sampling to reduce the blurring.

G. Create one or more "Binding LUT" in Chapter 9 entries to associate a texture description with entries in the Coordinate Source, Composition, and Sampling LUTs. Binding LUT entries are passed to `PEXExtSetActiveTextures` to activate one or more textures for subsequent primitives.

Step 5: Render

A. Set up rendering options.

- Optionally use `PEXExtSetTMSampleFrequency`. The texture mapping sample frequency specifies the frequency to use when sampling texels in a texture map. The only supported value for frequency is `PEXExtTMSampleFrequencyPixel`, meaning that texture map texels are sampled once for each pixel.
- Optionally use `PEXExtSetTMPerspectiveCorrection` to control whether to apply perspective correction. Because texture coordinates are calculated for vertices only, these coordinates must be interpolated to find the appropriate values for the points between the vertices and on the interior of the primitives. This call determines whether perspective correction is applied during interpolation.
- Optionally use `PEXExtSetTMResourceHints`. This call allows the user to ask PEXlib to optimize texture mapping performance to the possible detriment of memory usage, or vice versa. It is also possible to specify a list of textures that are believed to be the ones most often used by the user. Note that as hints, the requests made by this call may or may not be followed. Although the way in which system resources are used may be affected by these resource hints, the actual image displayed will not change.
- Enable texture mapping for subsequent primitives.

  To texture-map a primitive, the interior style must be set to `PEXExtInteriorStyleTexture` via `PEXSetInteriorStyle`. Alternatively, bundle tables may be used and `PEXSetInteriorBundleIndex` may be called.

B. Activate texture(s).

   One or more textures must be activated using the call `PEXExtSetActiveTextures`. Backface textures can be activated by `PEXExtSetBFActiveTextures`. Note that the number of active textures allowed may be inquired using `PEXGetImpDepConstants`.

   Note that if the application needs to redefine the pipeline context, `PEXExtChangePipelineContext` can be called to set the texture mapping attributes perspective correction, resource hints, sample frequency, and front-face and back-face active textures.

C. Render primitives.

   Primitives that are to be texture mapped must be rendered using either one of the routines, `PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, or `PEXExtQuadrilateralMesh`; or, alternatively, `PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh`.

Step 6: Clean Up

A. Free resources used by texture mapping. The routines `PEXExtFreeTM` and `PEXExtFreeTMDescription` remove the association between a resource ID and the texture map and a resource ID and texture map description, respectively. The storage held by a resource will be freed when no other resource references it.

References

- Bier, Eric A. and Sloan, Kenneth R., Jr., "Two-Part Texture Mappings," *IEEE Computer Graphics and Applications*, Sept. 1986, The Computer Society, Los Alamitos, CA, pp. 40—53.
- Blinn, James F. and Newell, Martin E., "Texture and Reflection in Computer Generated Images," *Communications of the ACM*, Vol. 19, No. 10 (Oct 1976); Association for Computing Machinery, Inc., New York, 1976, pp. 542—547.
- Blinn, James F., "Simulation of Wrinkled Surfaces," *ACM Computer Graphics*, Vol. 12, No 3., (SIGgraph Proceedings 1978), Association for Computing Machinery, Inc., New York, 1978, pp. 286—292.
- Catmull, Edwin, "A Subdivision Algorithm for Computer Display of Curved Surfaces," doctoral dissertation, University of Utah, Salt Lake City, Utah, December 1974.
- Crow, Franklin C., "Summed-Area Tables for Texture Mapping," *ACM Computer Graphics*, Vol. 18, No. 3, (SIGGRAPH Proceedings 1984), Association for Computing Machinery, Inc., New York, 1984, pp. 207—212.
- Glassner, Andrew S., *3D Computer Graphics—A User's Guide for Artists and Designers*, Design Press, New York, 1989.
- Heckbert, Paul S., "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, Nov. 1986, The Computer Society, Los Alamitos, CA, 1986, pp. 56—67.
- Peachey, Darwyn R., "Solid Texture of Complex Surfaces," *ACM Computer Graphics*, Vol. 19, No. 3, (SIGgraph Proceedings 1985), Association for Computing Machinery, Inc., New York, 1985, pp. 279—286.
- Watt, Alan and Watt, Mark, *Advanced Animation and Rendering Techniques—Theory and Practice*, Addison-Wesley Publishing Company, ACM Press, New York, 1992.
- Williams, Lance, "Pyramidal Parametrics," *ACM Computer Graphics*, Vol. 17, No. 3, (SIGgraph Proceedings 1983), Association for Computing Machinery, Inc., New York, pp. 1—11.
- Wolberg, George, *Digital Image Warping*' IEEE Computer Society Press, Los Alamitos, 1990.

Detailed Discussions

This section contains more detailed discussions of some selected texture mapping-related topics below:

- "Discussion: MIP Map"
- "User Interface Considerations for Creating Filtered Texture Maps"
- "Parameterization"

Discussion: MIP Map

"MIP" stands for *multum in parvo*; literally, "many things in a small place."

- *Why use MIP maps?*

  MIP maps are generally used to reduce the aliasing effects that naturally take place with texture mapping. Because the cost and repetition of anti-aliasing calculations are high, pre-computing their values in the form of a MIP map can dramatically reduce the overall performance cost of anti-aliasing.

  When a texture map is accessed for a single screen pixel, one of three things can happen:
  - The pixel maps to less than one texel (this means that several screen pixels map to a single texel of the texture map.) The texel information must be "magnified" (by interpolating information from neighboring texels) to cover the pixel.
  - One pixel maps to a single texel (the ideal case)
  - One pixel maps to several texels. The texel information must be "minified" to represent the average color of all the covered texels.

  In cases 1 and 3, a MIP map allows a color to be easily approximated for the screen pixels in near-constant time. This averaging mechanism minimizes the effects of aliasing as much as possible in a cost-effective manner.

- *How is a MIP map created?*

  A MIP map is a pyramid of images with the base of the pyramid equal to the original source texture map. Each successive level of the texture map is created using a box filter for each texel in the next level. In other words, four texels in one level of the map (in a 2×2 square) are averaged to create one texel in the next, smaller level. Thus, each successive level in a MIP map is a quarter of area of the previous level. The number of levels in a MIP map is also referred to as the depth of the map. The highest, and smallest, level of the MIP map is referred to as the "pinnacle."

  PEXlib provides the utilities `PEXExtCreateFilteredTM` and `PEXExtCreateFilteredTMFromWindow` to create a MIP map from a source map provided by the application. One of the parameters passed to the `PEXExtCreateFilteredTM*` utilities is the number of levels to create. If zero is specified, the optimum number of levels will be created to produce a full MIP map.

- *How is a MIP map used?*

  The values specified in the Sampling LUT determine how the MIP map is actually sampled to find the appropriate color for a single pixel. The minification method specifies how the map should be sampled when one screen pixel maps to several texels and the magnification method determines what to do in the case of multiple pixels mapping to a single texel. The possible methods are detailed in the table below and vary depending on how many pixels per level are sampled and how many levels are sampled to determine the final color. Note that for magnification, only the first two, `PEXExtTMTexelSampleSingleBase` and `PEXExtTMTexelSampleLinearBase` are recommended. In the table, where "2^n" is specified, n refers to the texture map dimension: 1, 2, or 3.

Table 10-1. MIP Map Usage

Method (minification or magnification)	**Texels sampled per level**	**Level(s) sampled**
`PEXExtTMTexelSampleSingleBase`	1	Base level only
`PEXExtTMTexelSampleLinearBase`	2^n	Base level only
`PEXExtTMTexelSampleSingleInMipmap`	1	One level closest to sample depth
`PEXExtTMTexelSampleLinearInMipmap`	2^n	One level closest to sample depth
`PEXExtTMTexelSampleBetweenMipmaps`	1	Two levels closest to sample depth
`PEXExtTMTexelSampleLinearBetweenMipmaps`	2^n	Two levels closest to sample depth

Given these methods, the MIP map is sampled according to the following formula:

- The number of texels in the base map that are covered by a single pixel determines if the minification method or magnification method should be used. The number of texels also determines which level should be sampled if the minification method specifies a level other than the base level. The `depth_sampling_bias_hint` value in the Sampling LUT can shift this sampled level up or down to make the texture appear more detailed or more blurred.
- The minification method or magnification method, as appropriate, is used to determine how many pixels should be sampled.
- If more than one texel is sampled as specified by the minification method or magnification method, the texels are averaged to derive the final color value.

User Interface Considerations for Creating Filtered Texture Maps

Some PEXlib implementations require texture map dimensions to be a power of two and/or require square maps (your implementation's requirements can be determined by `PEXGetImpDepConstants`). The utilities `PEXExtCreateFilteredTM` and `PEXExtCreateFilteredTMFromWindow` will resize the maps if demanded by the implementation. In some rare instances, the user may want total control over how the images are resized to meet the implementation requirements. The user may want to specify whether the map be cropped to meet the requirements or shrunk or enlarged using a filter. In these cases, the application will need to perform the shrinking or enlargement itself, before calling `PEXExtCreateFilteredTM` (or `PEXExtCreateFilteredTMFromWindow`). The utilities `PEXExtCreateFilteredTM` and `PEXExtCreateFilteredTMFromWindow` filter according to these rules:

- If a power of two is required, each dimension is resized to the closest power of two. If the closest power of two is smaller than the original dimension of the map, the map is down-sampled using a box filter. If the closest power of two is greater than the original dimension, the map is up-sampled using linear interpolation.
- If texture maps are required to be square, the texture map will be enlarged using linear interpolation to have dimensions equal to the largest of the two original dimensions.
- If both power-of-two and square texture maps are required, the dimensions will first be resized to a power of two and then the largest dimension will be used as the dimension of the final map.

Troubleshooting

This troubleshooting section is separated into several sections; choose the desired texture-mapping subject matter.

Frequently-Asked Questions

Q. What areas should be checked if textures fail to render?

A. Texture mapping may fail when some of the following conditions occur:

- Neither the extended primitives (`PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, `PEXExtQuadrilateralMesh`) were called; nor were the OCC alternatives `PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh` called. These are the only primitives for which texture mapping is supported.
- The interior style `PEXExtInteriorStyleTexture` was not specified for either front and/or back faces.
- The PEX server does not support the 5.1 texture mapping extended renderer, determined by inquiring its support with `PEXGetEnumTypeInfo`.
- No texture maps or coordinates were supplied. In this case, the default map (a black-and-white checkerboard) is used, but since no texture coordinates existed, the default specifies a coordinate of (0,0) per vertex, which translates into a single point of a white square (or black square on some implementations non-HP implementations).
- Texture coordinates were out of range when the active texture coordinate clamp condition was `PEXExtTMBoundaryCondBoundary`. Scaling coordinates to be between [0,1) or attempting the clamp condition `PEXExtTMBoundaryCondWrap` (if it is supported) may resolve this.
- Texture coordinates were out of range when the active texture coordinate clamp condition was `PEXExtTMBoundaryCondClampColor`. If the clamp color specified was black (the default), then the surface should appear black as well. Try specifying a color of red (1,0,0) in the Sampling LUT entry to determine if the clamp condition is being used.
- The texture map input may be invalid. If so, try another map type and be certain it abides by any implementation-dependent constraints such as

PEXExtIDPowerOfTwoTMSizesRequired and PEXExtIDSquareTMRequired. Note that the utilities PEXExtCreateFilteredTM and PEXExtCreateFilteredTMFromWindow can be used to ensure the implementation constants are met.

- Data from the utilities for texture coordinates or filtered texture maps was invalid. Be certain to check the return status of these functions to determine the correctness of their data.

Q. Under what conditions might one see the default texture (a checkerboard) appear?

A. The default texture appears when:

- When the active [backface] texture list is empty (zero in length).
- When the active [backface] texture list specifies a nonexistent Binding LUT entry.
- When a user Binding LUT table is unspecified.
- When an invalid TMDescription or texture map ID is detected.

Q. What types of image file formats does PEXlib support?

A. None. The library supports utilities to perform xwd-like window grabs (PEXExtCreateFilteredTMFromWindow) once an image is displayed, however it cannot read or use a file format directly.

Q. Does PEXlib limit the number or size of textures which can be used?

A. PEXlib does not limit the number of textures which can be predefined by PEXExtCreateTM or PEXExtCreateTMDescription, however, implementations may limit the actual number of textures which can be applied to any one extended surface primitive at a time. This limit is specified by the constant PEXExtIDMaxTextureMaps on a per-primitive basis. The size of a texture may also be limited by the value specified in PEXExtIDMaxFastTMSize if texture mapping acceleration is available to a user. Both values can be inquired by PEXGetEnumTypeInfo.

Q. Why do the scaling matrices in the `TMCoordSrc` LUT behave inversely to their expected effect on a texture; for example, scaling values greater than 1 create multiple copies of a map on my surface?

A. Texture coordinate space is ideally defined between zero and one [0.0,1.0). When a texture coordinate is scaled by a value greater than one in the orientation matrix of a `TMCoordSrc` LUT entry, its value spans a texture coordinate space distance greater than or equal to one. In this case, the texture coordinates will, if the correct wrap or mirror boundary conditions are set, sample more than one texture "space" in a periodic fashion.

For example, the orientation matrix:

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

transforms the texture coordinates (0,0) and (1,1) to be the new forms (0,0) and (3,2). As the texture coordinates are interpolated across texture space, the boundary conditions for zero and one are examined and the appropriate wrap or clamp actions taken. (To wrap, compute the value $t'_0 = t_0 - \lfloor t_0 \rfloor$ and $t'_1 = t_1 - \lfloor t_1 \rfloor$.) In the above example, boundary conditions of `PEXExtTMBoundaryCondWrap` for *t0* and *t1* would result in three copies of the texture in the t_0 direction and two copies in the t_1 direction. This stems from the fact that the texture space range of zero to one [0,1) fits into the above ranges three and two times, respectively.

A matrix of the type:

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

would map the texture coordinates (0,0) and (1,1) to the new forms (0,0) and (0.25, 0.25) in texture space, thus sampling only a fraction of the total map.

Texture Maps

Q. How does one avoid or fix the unpleasant texture seams which may occur on an object?

A. Texture seams can become visible when:

- The boundary condition `PEXExtTMBoundaryCondWrap` is active in a given direction and opposing edges of a texture map do not match in color. To correct this, ensure the opposing edges of a map (top=bottom, left=right) share the same color. This can be performed in many interactive image processing and paint packages. Alternatively, the boundary condition `PEXExtTMBoundaryCondMirror`, if supported, may help in certain situations.
- Texture coordinates on a primitive do not quite stretch between [0,1), thus the underlying color of a primitive (such as white) reveals the edge of the map. In this situation, explicit texture coordinates can either be scaled and restored in the vertex to account for this condition (scale operation occurs once) or the orientation matrix in the `TMCoordSrc` LUT can hold a scale matrix (less efficient due to per image cost to rescale texture coordinates).
- Two textures in proximity to each other (by nature of their texture coordinates) do not overlap at their edges cleanly due to precision problems or because of contrasting colors. Under these situations, corrections may be possible by either editing textures to share edge colors explicitly or forcing one texture to overlap the other via its orientation matrix.
- The texture coordinate utility functions (`PEXExtTMCoord*`) are susceptible to producing parameterizations with obvious "seams." Seams created here occur when a projection method wraps over a boundary condition (both cylindrical or spherical projections have seams at their positive (+X) axes). Using the orientation matrix contained in the `param_data` argument of these utility functions can reorient this seam condition to other parts of an object's geometry (in many desirable cases, the side opposite the observer).

Q. Why does my input texture map appear to be inverted (upside down) on the textured surface?

A. The problem may be caused by the ordering of your texture map scanlines. PEXlib texture maps assume to have their origins (0,0) at the lower left edge of the quadrant. Many formats (such as X) assume the upper left. Maps should be inverted in scanline order if their origins are in conflict with PEXlib. Alternatively, an appropriate orientation matrix can flip these coordinates as well, although this method may cause more calculation and precision costs later in the pipeline. (The utility `PEXExtCreateFilteredTMFromWindow` already performs this inversion.)

Surface Parameterization

Q. When parameterizing certain objects (such as a sphere composed of polygons), why do texture maps appear to sometimes "swirl" around the natural poles of the object?

A. The poles of a sphere correspond to the "top" and "bottom" edges of a 2D texture as they collapse down to one point. Since the texture coordinate utilities `PEXExtTMCoord*` compute projections on a per-vertex basis, it is not possible to represent all points along these edges as one point. Instead, the projection utility will usually choose one or more points (depending upon the coordinate source selected), which will then be interpolated during rendering. When this occurs, a swirling effect appears which is often emphasized by having large primitives at the top (or bottom) of the sphere. This visual effect can be minimized by reducing the sizes of any "polar" primitives of the sphere, but it cannot be totally eliminated from this parameterization approach. (Ray tracers escape this problem by evaluating the projection equation at each pixel, not each vertex, thus minimizing the effect of the polar singularity.)

Q. Why do certain primitives (the extended primitives `PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, and `PEXExtQuadrilateralMesh`; or the OCC alternatives `PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh`) sometimes appear to have discontinuous textures applied across their surfaces, even though the texture coordinate utilities worked on the primitive all at once?

A. The primitives described above are shared-vertex types of surface primitives. Although facets of these primitives may share geometric vertices, they may not actually share the shading normals associated at these points. Thus, if the texture coordinate source is `PEXExtTMCoordSourceVertexNormal` and a facet normal must be substituted, the coordinates stored in the vertex will be those of the last shared facet to that vertex point (which may be incorrect for all other primitives). The only solutions to this problem are to ensure that facets sharing vertices also share vertex shading normals when using `PEXExtTMCoordSourceVertexNormal` or to use `PEXExtTMCoordSourceVertexCoord` when possible.

Q. What causes textures to become "squeezed" at unnatural locations on my object geometry?

A. Spherical projections computed by the texture coordinate utilities `PEXExtTMCoord*` create known artifacts around the poles of their projection object. These poles lie along the +Y axis in world space (WC). If the natural axis of symmetry for an object (such as a vase) uses a different axis (such as the Z axis), then the orientation matrix in the parameterization data record should incorporate a rotation of 90 degrees to bring this axis in conjunction with the +Y axis of the projection object. This will reduce the apparent distortion by keeping the lines of symmetry aligned.

Standard Mapping

Q. What operations affect the apparent position of a texture on the surface of a primitive?

A. The apparent position of a texture on a surface occurs due to the location of texture coordinates within the 2D space of a standard texture map. Texture coordinates select regions of the texture to "pin" to the vertices of a facet. To alter the selected regions, several measures can be taken:

- Texture coordinates can be "moved" during their creation by controlling the method of a surface parameterization. Both `PEXExtTMProjectionSphereWC` and `PEXExtTMProjectionCylinderWC` use orientation matrices in their parameterization, effectively transforming a texture coordinate before it is stored.
- A `TMCoordSrc` LUT entry contains a unique transformation matrix which can scale, translate, or rotate texture coordinates within 2D texture space. Use of this feature can aid in exactly positioning texture coordinates over the interesting areas of a texture map.
- Editing the contents of the target texture map to relocate, rescale, or otherwise alter the locations of interest with respect to the object's texture coordinates.

Environment Mapping

Q. Under certain conditions, why do some of the facets of a surface fail to texture map when using `PEXExtTMParamReflectSphereWC`?

A. The implementation of `PEXExtTMParamReflectSphereWC` under HP PEXlib detects a condition where a parameterized primitive may cross the natural seam of the texture (the seam of the WC sphere is at the +X axis). To correct this condition, try using the boundary clamp conditions `PEXExtTMBoundaryCondWrap` for both *t0* and *t1* to ensure correct continuity of the map across a seam.

Q. What advantage does `PEXExtTMParamReflectSphereWC` offer over `PEXExtTMParamReflectSphereVRC`?

A. If `PEXExtTMParamReflectSphereWC` is supported in your implementation, then reflections on a surface will change with respect to location of the view in world coordinates (WC). As an object or viewpoint changes position, so does the apparent reflection along its surface. This differs from `PEXExtTMParamReflectSphereVRC`, which computes reflections in eye coordinates (VRCs), a coordinate system which always keeps the reflected environment fixed behind the viewer. From a natural perspective, `PEXExtTMParamReflectSphereWC` behaves more like an enclosing environment as the eye position moves relative to the reflective surface. (HP PEXlib supports the functionality of `PEXExtTMParamReflectSphereWC`.)

Q. How are chrome or metallic effects best simulated?

A. Chrome and other highly reflective surfaces are only as interesting as the environments they are found in. In real-world situations, chrome objects are often photographed in special enclosures which include both light and dark regions. Texture maps to simulate this effect should include both light and dark regions of a map such as those found in sparsely lit rooms. Photographs or other image data can be converted specifically for this purpose. The quality of rendered reflection is also dependent upon the clarity of detail in a texture map. The sharper the detail contained in a map, the greater the impression of fine chrome there will be on a rendered model.

Performance

Q. Under what conditions will my texture mapping performance be the best?

A. Several steps can be taken to improve texture mapping throughput in the system:

- Attempt to render all extended surface primitives using the same set of textures at the same time. Activating different textures on a per-primitive basis can be costly.
- Minimize the mixture of surface primitives going to PEX. When the supported extended primitives for texture mapping (`PEXExtFillAreaSetWithData`, `PEXExtSetOfFillAreaSets`, `PEXExtTriangleStrip`, `PEXExtQuadrilateralMesh`), or the OCC versions of these primitives (`PEXOCCFillArea`, `PEXOCCFillAreaSet`, `PEXOCCIndexedFillAreaSets`, `PEXOCCTriangleStrip`, and `PEXOCCQuadrilateralMesh`) are mixed with other types of primitives, the overall cost to switch modes—texturing versus non-texturing—will increase.
- A single active texture map will generally perform better than multiple active texture maps. The performance cost, however, varies with general size of the map, type of map, number of maps, and the complexity of the texture operations associated with each map.
- Texture maps which use the "`Int8`" (byte) level organization save considerable memory resource on a given system. Larger maps, because of their dimension or the fact that they are comprised of "`Float`" data, will impact the operating system by increasing the application's use of swap space.
- Pre-transforming texture coordinate data stored at a vertex and using identity orientation matrices for all `TMCoordSrc` LUT entries will reduce the amount of computation spent during texture mapping. Although these matrices help position a texture on the surface of an object, it is often possible to fix these texture coordinate values by pre-transforming them one time since they are not likely to change after their location can be tied down.

Visual Quality

Q. My texture map has many fine details in it, some only one texel wide (such as lines). When I rotate the surface it is applied to, these lines tend to break up or flash. What can I do to correct this?

A. Texture mapping is all about sampling theory. Essentially, the more discrete samples taken, the better the reconstruction of the original, continuous signal. The artifact you are experiencing is sampling "aliasing," the loss of reconstruction information. To reduce or eliminate visual artifacts, try the following:

- Increase the size of your texture map. Since aliasing is based on a factor of screen pixel size to its coverage on the texture map, a greater size map with more information can often improve color sampling.
- Use a MIP map with more than one level. MIP maps are precomputed sampling filters which try to account for aliasing during sampling. Use of a MIP map can often improve image quality during animation due to problems with sampling.
- Use better sampling methods for your MIP map. `PEXExtTMTexelSampleSingleBase` uses only one sample and is prone to aliasing. `PEXExtTMTexelSampleLinearBetweenMipmaps` uses many samples and an neighbor interpolation filter to approximate colors, thus reducing the inaccuracy of single point sampling. (The only drawback to using more interpolation is a softening of sharp detail due to the blending used in the computation.)

Q. Portions of my textured surface appear to be blotched with regions of blurred detail. What are they and how can I correct this?

A. The artifact witnessed here occurs due to the design of the Mip Map anti-aliasing filter. As a renderer attempts to approximate an area of texels to sample in texture space, it selects different map levels to sample. The regions of blurred detail are visible transitions which are occurring due to the current sampling method. To reduce the visual impacts of these artifacts, several steps should be attempted:

- Create your own MIP maps using better filters than those offered by `PEXExtCreateFilteredTM` and `PEXExtCreateFilteredTMFromWindow`. These utilities employ box- and linear filters to minify and scale incoming image data. Other image processing filters such as Gaussian can lead to improved images.
- Increase the level of filtering.
  `PEXExtTMTexelSampleLinearBetweenMipmaps` provides the highest quality of visual control, however, it also requires the greatest amount of computation; this can reduce pipeline throughput.
- Increase the size of the `depth_sampling_bias_hint` in the `TMSampling` LUT to greater than one to force sampling to occur in deeper maps away from the base map. The overall blurring of detail will increase, however it will appear more uniform. Decrease the size to improve sharpness.
- Decrease the `t*_frequency_hint` in any of the *t0* or *t1* directions. Depending upon the level of detail in a map and its frequency, this bias will reduce the contribution of *t0* or *t1* gradient calculations such that the levels of Mip Map will change more rapidly based upon the contributing factor of the other's (*t1* or *t0*) gradient. This hint, however, may not be implemented under all implementations or devices.
- Reduce the level of high frequency detail in the original texture map. Areas of high change (gradient) reveal greater levels of visual artifacts when viewed on a textured surface.
- Decrease the size of the primitives to which the texture will be applied. Sometimes the interaction between light shade interpolation and texture surface causes mach-bands or other artifacts to become more pronounced. Increasing the level of detail can sometimes improve the final visual outcome.

A Sample Output from `xdpyinfo` and `pexdpyinfo`

Introduction

In "Determining A Server's Features" in Chapter 6, you were introduced to two helpful utilities, **`xdpyinfo`** and **`pexdpyinfo`**. Sample output from these are illustrated here for an HP 9000 Model 725/100 with HP VISUALIZE-24 graphics:

`xdpyinfo`

```
name of display:    hpsys00:0.0
version number:    11.0
vendor string:    Hewlett-Packard Company
vendor release number:    600000
maximum request size:  4194300 bytes
motion buffer size:  100
bitmap unit, bit order, padding:    32, MSBFirst, 32
image byte order:    MSBFirst
number of supported pixmap formats:    4
supported pixmap formats:
    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
keycode range:    minimum 10, maximum 135
focus:  PointerRoot
number of extensions:    13
    BIG-REQUESTS
    DOUBLE-BUFFER
    HP-SMT
    HPExtension
    MIT-SHM
    MIT-SUNDRY-NONSTANDARD
    Multi-Buffering
    SHAPE
    X3D-PEX
    XIE
    XInputExtension
    XTEST
    XTestExtension1
default screen number:    0
number of screens:    1

screen #0:
  dimensions:    1280x1024 pixels (342x273 millimeters)
  resolution:    95x95 dots per inch
  depths (4):    1, 8, 12, 24
  root window id:    0x2a
  depth of root window:    8 planes
  number of colormaps:    minimum 1, maximum 4
  default colormap:    0x28
  default number of colormap cells:    256
  preallocated pixels:    black 0, white 1
  options:    backing-store YES, save-unders YES
  largest cursor:    64x64
  current input event mask:    0x0
  number of visuals:    8
  default visual id:  0x21
  visual:
    visual id:    0x20
    class:    PseudoColor
    depth:    8 planes
    available colormap entries:    256
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x21
    class:    PseudoColor
    depth:    8 planes
    available colormap entries:    256
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x22
    class:    PseudoColor
    depth:    8 planes
    available colormap entries:    255
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x23
    class:    TrueColor
    depth:    8 planes
    available colormap entries:    8 per subfield
    red, green, blue masks:    0xe0, 0x1c, 0x3
  significant bits in color specification:    8 bits
visual:
  visual id:    0x24
  class:    DirectColor
  depth:    12 planes
  available colormap entries:    16 per subfield
  red, green, blue masks:    0xf00, 0xf0, 0xf
  significant bits in color specification:    8 bits
visual:
  visual id:    0x25
  class:    TrueColor
  depth:    12 planes
  available colormap entries:    16 per subfield
  red, green, blue masks:    0xf00, 0xf0, 0xf
  significant bits in color specification:    8 bits
visual:
  visual id:    0x26
  class:    DirectColor
  depth:    24 planes
  available colormap entries:    256 per subfield
  red, green, blue masks:    0xff0000, 0xff00, 0xff
  significant bits in color specification:    8 bits
visual:
  visual id:    0x27
  class:    TrueColor
  depth:    24 planes
  available colormap entries:    256 per subfield
  red, green, blue masks:    0xff0000, 0xff00, 0xff
  significant bits in color specification:    8 bits
```

pexdpyinfo

```
PEX information for hpsys00:0.0

PEX EXTENSION INFORMATION

   major version number:               5 (0x5)
   minor version number:               1 (0x1)
   release number:                     10400 (0x28a0)
   vendor:                             Hewlett-Packard Company
   subset information:                 Immediate
   subset information:                 Structure Rendering

PEX ENUMERATED TYPES

   PEXETATextStyle
      Type 0:                          NotConnected (1) (0x1)
      Type 1:                          Connected (2) (0x2)
   PEXETColorApproxModel
      Type 0:                          RGB (1) (0x1)
   PEXETColorApproxType
      Type 0:                          ColorSpace (1) (0x1)
      Type 1:                          ColorRange (2) (0x2)
      Type 2:                          HP_ColorApproxTypeIndexed (34560) (0x8700)
   PEXETColorType
      Type 0:                          Indexed (0) (0x0)
      Type 1:                          RGBFloat (1) (0x1)
      Type 2:                          HP_ColorTypeRGBA (34560) (0x8700)
   PEXETCurveApproxMethod
      Type 0:                          HP_AdaptiveDC (1) (0x1)
      Type 1:                          WCS_Relative (9) (0x9)
      Type 2:                          NPC_Relative (10) (0xa)
      Type 3:                          DC_Relative (11) (0xb)
   PEXETDisplayUpdateMode
   PEXETFloatFormat
      Type 0:                          IEEE_754_32 (1) (0x1)
   PEXETGDP2D
   PEXETGDP
   PEXETGSE
      Type 0:                          HP_GSE_SET_ANTIALIAS_MODE (34561) (0x8701)
   PEXETHatchStyle
      Type 0:                          45Degrees (34561) (0x8701)
      Type 1:                          135Degrees (34562) (0x8702)
      Type 2:                          ExtHatchStyle45Degrees (36864) (0x9000)
      Type 3:                          ExtHatchStyle135Degrees (36865) (0x9001)
   PEXETHLHSRMode
      Type 0:                          Off (1) (0x1)
      Type 1:                          ZBuffer (2) (0x2)
   Type 2:                       ZBufferId (6) (0x6)
   Type 3:                       HP_HLHSRZBufferReadOnly (34560) (0x8700)
   Type 4:                       HP_HLHSRZBufferIDReadOnly (34561) (0x8701)
PEXETInteriorStyle
   Type 0:                       Hollow (1) (0x1)
   Type 1:                       Solid (2) (0x2)
   Type 2:                       Hatch (4) (0x4)
   Type 3:                       Empty (5) (0x5)
   Type 4:                       ExtInteriorStyleTexture (36864) (0x9000)
PEXETLightType
   Type 0:                       Ambient (1) (0x1)
   Type 1:                       WCS_Vector (2) (0x2)
   Type 2:                       WCS_Point (3) (0x3)
   Type 3:                       WCS_Spot (4) (0x4)
PEXETLineType
   Type 0:                       Solid (1) (0x1)
   Type 1:                       Dashed (2) (0x2)
   Type 2:                       Dotted (3) (0x3)
   Type 3:                       DashDot (4) (0x4)
   Type 4:                       HP_Centerline (34561) (0x8701)
   Type 5:                       HP_Phantom (34562) (0x8702)
   Type 6:                       ExtLineTypeCenter (36864) (0x9000)
   Type 7:                       ExtLineTypePhantom (36865) (0x9001)
PEXETMarkerType
   Type 0:                       Dot (1) (0x1)
   Type 1:                       Cross (2) (0x2)
   Type 2:                       Asterisk (3) (0x3)
   Type 3:                       Circle (4) (0x4)
   Type 4:                       X (5) (0x5)
   Type 5:                       HP_Triangle (34560) (0x8700)
   Type 6:                       HP_Square (34561) (0x8701)
   Type 7:                       HP_Diamond (34562) (0x8702)
   Type 8:                       HP_CrossSquare (34563) (0x8703)
PEXETModelClipOperator
   Type 0:                       Replace (1) (0x1)
   Type 1:                       Intersection (2) (0x2)
PEXETParaSurfCharacteristics
   Type 0:                       None (1) (0x1)
   Type 1:                       HP_InteriorEdging (2) (0x2)
PEXETPickDeviceType
   Type 0:                       DC_HitBox (1) (0x1)
   Type 1:                       NPC_HitVolume (2) (0x2)
PEXETPolylineInterpMethod
   Type 0:                       None (1) (0x1)
   Type 1:                       Color (2) (0x2)
PEXETPromptEchoType
PEXETReflectionModel
   Type 0:                       NoShading (1) (0x1)
   Type 1:                       Ambient (2) (0x2)
   Type 2:                     Diffuse (3) (0x3)
   Type 3:                     Specular (4) (0x4)
PEXETRenderingColorModel
   Type 0:                     RGB (1) (0x1)
PEXETSurfaceApproxMethod
   Type 0:                     HP_AutoMesh (1) (0x1)
   Type 1:                     WCS_Relative (9) (0x9)
   Type 2:                     NPC_Relative (10) (0xa)
   Type 3:                     DC_Relative (11) (0xb)
PEXETSurfaceEdgeType
   Type 0:                     Solid (1) (0x1)
   Type 1:                     Dashed (2) (0x2)
   Type 2:                     Dotted (3) (0x3)
   Type 3:                     DashDot (4) (0x4)
PEXETSurfaceInterpMethod
   Type 0:                     None (1) (0x1)
   Type 1:                     Color (2) (0x2)
PEXETTrimCurveApproxMethod
   Type 0:                     HP_AdaptToSurfaceCriteria (1) (0x1)
PEXETEscape
   Type 0:                     SetEchoColor (1) (0x1)
   Type 1:                     QueryColorApprox (33025) (0x8101)
   Type 2:                     ES_ESCAPE_DBLBUFFER (33793) (0x8401)
   Type 3:                     ES_ESCAPE_SWAPBUFFER (33794) (0x8402)
   Type 4:                     ES_ESCAPE_SWAPBUFFERCONTENT (33795) (0x8403)
   Type 5:                     HP_ESCAPE_DFRONT (34561) (0x8701)
   Type 6:                     HP_ESCAPE_SET_GAMMA_CORRECTION (34562) (0x8702)
   Type 7:                     ExtEscapeChangePipelineContext (36864) (0x9000)
   Type 8:                     ExtEscapeGetPipelineContext (36865) (0x9001)
   Type 9:                     ExtEscapeChangeRenderer (36866) (0x9002)
   Type 10:                    ExtEscapeGetRendererAttributes (36867) (0x9003)
   Type 11:                    ExtEscapeSetTableEntries (36868) (0x9004)
   Type 12:                    ExtEscapeGetTableEntries (36869) (0x9005)
   Type 13:                    ExtEscapeGetTableEntry (36870) (0x9006)
   Type 14:                    ExtEscapeCreateTM (36871) (0x9007)
   Type 15:                    ExtEscapeFreeTM (36873) (0x9009)
   Type 16:                    ExtEscapeFetchElements (36875) (0x900b)
   Type 17:                    ExtEscapeQueryColorApprox (36876) (0x900c)
   Type 18:                    ExtEscapeCreateTMExtraData (36877) (0x900d)
   Type 19:                    HP_EscapeChangePipelineContext (34563) (0x8703)
   Type 20:                    HP_EscapeGetPipelineContext (34564) (0x8704)
   Type 21:                    HP_EscapeChangeRenderer (34565) (0x8705)
   Type 22:                    HP_EscapeGetRendererAttributes (34566) (0x8706)
   Type 23:                    HP_EscapeEVEInformation (34689) (0x8781)
   Type 24:                    HP_EscapeGetZBuffer (34690) (0x8782)
   Type 25:                    HP_EscapePutZBuffer (34691) (0x8783)
   Type 26:                    HP_EscapeStereoMode (34568) (0x8708)
   Type 27:                    HP_EscapeLockStructure (34569) (0x8709)
PEXETPickAllMethod
```

```
   Type 0:                         All (1) (0x1)
PEXETPickOneMethod
   Type 0:                         Last (1) (0x1)
ExtEnumType
   Type 0:                         ExtEnumType (36864) (0x9000)
   Type 1:                         ExtOC (36865) (0x9001)
   Type 2:                         ExtPC (36866) (0x9002)
   Type 3:                         ExtRA (36867) (0x9003)
   Type 4:                         ExtLUT (36868) (0x9004)
   Type 5:                         ExtID (36869) (0x9005)
   Type 6:                         ExtTMRenderingOrder (36870) (0x9006)
   Type 7:                         ExtTMCoordSource (36871) (0x9007)
   Type 8:                         ExtTMCompositeMethod (36872) (0x9008)
   Type 9:                         ExtTMTexelSampleMethod (36873) (0x9009)
   Type 10:                        ExtTMBoundaryCondition (36874) (0x900a)
   Type 11:                        ExtTMClampColorSource (36875) (0x900b)
   Type 12:                        ExtTMDomain (36876) (0x900c)
   Type 13:                        ExtTexelType (36877) (0x900d)
   Type 14:                        ExtTMParameterizationMethod (36880) (0x9010)
   Type 15:                        ExtTMType (36879) (0x900f)
   Type 16:                        ExtTMPerspectiveCorrection (36881) (0x9011)
   Type 17:                        ExtTMSampleFrequency (36882) (0x9012)
   Type 18:                        ExtTMResourceHint (36878) (0x900e)
   Type 19:                        ExtPrimitiveAAMode (36883) (0x9013)
   Type 20:                        ExtPrimitiveAABlendOp (36884) (0x9014)
   Type 21:                        ExtLineCapStyle (36885) (0x9015)
   Type 22:                        ExtLineJoinStyle (36886) (0x9016)
   Type 23:                        HP_TransparencyMethod (34560) (0x8700)
   Type 24:                        HP_AlphaBlendFunction (34561) (0x8701)
ExtOC
   Type 0:                         ExtOCTMPerspectiveCorrection (36864) (0x9000)
   Type 1:                         ExtOCTMSampleFrequency (36865) (0x9001)
   Type 2:                         ExtOCTMResourceHints (36866) (0x9002)
   Type 3:                         ExtOCActiveTextures (36867) (0x9003)
   Type 4:                         ExtOCBFActiveTextures (36868) (0x9004)
   Type 5:                         ExtOCFillAreaSetWithData (36869) (0x9005)
   Type 6:                         ExtOCSetOfFillAreaSets (36870) (0x9006)
   Type 7:                         ExtOCTriangleStrip (36871) (0x9007)
   Type 8:                         ExtOCQuadrilateralMesh (36872) (0x9008)
   Type 9:                         ExtOCPrimitiveAA (36873) (0x9009)
   Type 10:                        ExtOCLineCapStyle (36874) (0x900a)
   Type 11:                        ExtOCLineJoinStyle (36875) (0x900b)
   Type 12:                        ExtOCEllipse (36876) (0x900c)
   Type 13:                        ExtOCEllipse2D (36877) (0x900d)
   Type 14:                        ExtOCCircle2D (36878) (0x900e)
   Type 15:                        ExtOCEllipticalArc (36879) (0x900f)
   Type 16:                        ExtOCEllipticalArc2D (36880) (0x9010)
   Type 17:                        ExtOCCircularArc2D (36881) (0x9011)
   Type 18:                        HP_OCSetAlphaBlendFunction (34560) (0x8700)
   Type 19:                          HP_OCSetDeformationMode (34561) (0x8701)
   Type 20:                          HP_OCSetDeformationValueLocation (34562) (0x8702)
   Type 21:                          HP_OCSetCappingPlanes (34563) (0x8703)
   Type 22:                          HP_OCPolylineSetWithData (34564) (0x8704)
   Type 23:                          HP_OCMarkersWithData (34565) (0x8705)
   Type 24:                          HP_OCIndexedTriangleStrip (34567) (0x8707)
   Type 25:                          HP_OCIndexedTriangleFan (34568) (0x8708)
   Type 26:                          HP_OCIndexedMarkers (34569) (0x8709)
   Type 27:                          HP_OCIndexedPolylines (34570) (0x870a)
   Type 28:                          HP_OCFaceLightingMode (34571) (0x870b)
   Type 29:                          HP_OCUserLineType (34572) (0x870c)
   Type 30:                          HP_OCUserMarkerGlyph (34573) (0x870d)
   Type 31:                          HP_OCHighlightColor (34574) (0x870e)
ExtPC
   Type 0:                           ExtPCTMPerspectiveCorrection (36864) (0x9000)
   Type 1:                           ExtPCTMResourceHints (36865) (0x9001)
   Type 2:                           ExtPCActiveTextures (36867) (0x9003)
   Type 3:                           ExtPCBFActiveTextures (36868) (0x9004)
   Type 4:                           ExtPCPrimitiveAA (36869) (0x9005)
   Type 5:                           ExtPCLineCapStyle (36870) (0x9006)
   Type 6:                           ExtPCLineJoinStyle (36871) (0x9007)
   Type 7:                           ExtPCTMSampleFrequency (36866) (0x9002)
   Type 8:                           HP_PCAlphaBlendFunction (34560) (0x8700)
   Type 9:                           HP_PCDeformationMode (34561) (0x8701)
   Type 10:                          HP_PCDeformationValueLocation (34562) (0x8702)
   Type 11:                          HP_PCFaceLightingMode (34563) (0x8703)
   Type 12:                          HP_PCUserLineType (34564) (0x8704)
   Type 13:                          HP_PCUserMarkerGlyph (34565) (0x8705)
   Type 14:                          HP_PCHighlightColor (34566) (0x8706)
ExtRA
   Type 0:                           ExtRATMBindingTable (36864) (0x9000)
   Type 1:                           ExtRATMCoordSourceTable (36865) (0x9001)
   Type 2:                           ExtRATMCompositionTable (36866) (0x9002)
   Type 3:                           ExtRATMSamplingTable (36867) (0x9003)
   Type 4:                           HP_RATransparencyMethod (34560) (0x8700)
   Type 5:                           HP_RAWideLineControl (34561) (0x8701)
   Type 6:                           HP_RAPolygonOffset (34562) (0x8702)
ExtLUT
   Type 0:                           ExtLUTTMBinding (36864) (0x9000)
   Type 1:                           ExtLUTTMCoordSource (36865) (0x9001)
   Type 2:                           ExtLUTTMComposition (36866) (0x9002)
   Type 3:                           ExtLUTTMSampling (36867) (0x9003)
ExtID
   Type 0:                           ExtIDMaxTextureMaps (36864) (0x9000)
   Type 1:                           ExtIDMaxFastTMSize (36865) (0x9001)
   Type 2:                           ExtIDPowerOfTwoTMSizesRequired (36866) (0x9002)
   Type 3:                           ExtIDSquareTMRequired (36867) (0x9003)
   Type 4:                           HP_IDDeformationSupported (34560) (0x8700)
   Type 5:                           HP_IDCappingPlanesSupported (34561) (0x8701)
   Type 6:                        HP_IDInterferenceSupported (34562) (0x8702)
   Type 7:                        HP_IDPolygonOffsetSupported (34563) (0x8703)
ExtTMRenderingOrder
   Type 0:                        ExtTMRenderingOrderPreSpecular (36864) (0x9000)
   Type 1:                        ExtTMRenderingOrderPostSpecular (36865) (0x9001)
ExtTMCoordSource
   Type 0:                        ExtTMCoordSourceFloatData (36866) (0x9002)
ExtTMCompositeMethod
   Type 0:                        ExtTMCompositeReplace (36864) (0x9000)
   Type 1:                        ExtTMCompositeModulate (36865) (0x9001)
   Type 2:                        ExtTMCompositeDecal (36867) (0x9003)
ExtTMTexelSampleMethod
   Type 0:                        ExtTMTexelSampleSingleBase (36864) (0x9000)
   Type 1:                        ExtTMTexelSampleLinearBase (36865) (0x9001)
   Type 2:                        ExtTMTexelSampleSingleInMipmap (36866) (0x9002)
   Type 3:                        ExtTMTexelSampleLinearInMipmap (36867) (0x9003)
   Type 4:                        ExtTMTexelSampleSingleBetweenMipmaps (36868) (0x9004)
   Type 5:                        ExtTMTexelSampleLinearBetweenMipmaps (36869) (0x9005)
ExtTMBoundaryCondition
   Type 0:                        ExtTMBoundaryCondClampColor (36864) (0x9000)
   Type 1:                        ExtTMBoundaryCondBoundary (36865) (0x9001)
   Type 2:                        ExtTMBoundaryCondWrap (36866) (0x9002)
   Type 3:                        ExtTMBoundaryCondMirror (36867) (0x9003)
ExtTMClampColorSource
   Type 0:                        ExtTMClampColorSourceAbsolute (36864) (0x9000)
   Type 1:                        ExtTMClampColorSourceExplicit (36865) (0x9001)
ExtTMDomain
   Type 0:                        ExtTMDomainColor1D (36864) (0x9000)
   Type 1:                        ExtTMDomainColor2D (36865) (0x9001)
ExtTexelType
   Type 0:                        ExtTexelRGBFloat (36870) (0x9006)
   Type 1:                        ExtTexelRGBInt8 (36871) (0x9007)
   Type 2:                        ExtTexelRGBAlphaFloat (36873) (0x9009)
   Type 3:                        ExtTexelRGBAlphaInt8 (36874) (0x900a)
   Type 4:                        HPTexelAlphaRGBFloat (34560) (0x8700)
   Type 5:                        HPTexelAlphaRGBInt8 (34561) (0x8701)
ExtTMParameterizationMethod
   Type 0:                        ExtTMParamExplicit (36864) (0x9000)
   Type 1:                        ExtTMParamReflectSphereVRC (36865) (0x9001)
   Type 2:                        ExtTMParamReflectSphereWC (36866) (0x9002)
   Type 3:                        ExtTMParamLinearVRC (36867) (0x9003)
ExtTMType
   Type 0:                        ExtTMTypeMipMap (36864) (0x9000)
ExtTMPerspectiveCorrection
   Type 0:                        ExtTMPerspCorrectNone (36864) (0x9000)
   Type 1:                        ExtTMPerspCorrectPixel (36866) (0x9002)
ExtTMSampleFrequency
   Type 0:                        ExtTMSampleFrequencyPixel (36864) (0x9000)
ExtTMResourceHint
```

```
    Type 0:                        ExtTMResourceHintNone (36864) (0x9000)
    Type 1:                        ExtTMResourceHintSpeed (36865) (0x9001)
    Type 2:                        ExtTMResourceHintSpace (36866) (0x9002)
 ExtPrimitiveAAMode
    Type 0:                        ExtPrimAANone (36864) (0x9000)
 ExtPrimitiveAABlendOp
    Type 0:                        ExtPrimAABlendOpImpDep (36864) (0x9000)
    Type 1:                        ExtPrimAABlendOpSimpleAlpha (36865) (0x9001)
 ExtLineCapStyle
    Type 0:                        ExtLineCapStyleButt (36864) (0x9000)
 ExtLineJoinStyle
    Type 0:                        ExtLineJoinStyleImpDep (36864) (0x9000)
    Type 1:                        ExtLineJoinStyleMiter (36866) (0x9002)
 HP_TransparencyMethod
    Type 0:                        HP_TransparencyMethodScreenDoor (34560) (0x8700)
 HP_AlphaBlendFunction
    Type 0:                        HP_AlphaBlendFunctionSrcColor (34560) (0x8700)
    Type 1:                        HP_AlphaBlendFunctionSimpleAlpha (34561) (0x8701)

PEX IMPLEMENTATION-DEPENDENT CONSTANTS

 PEXIDDitheringSupported:          YES
 PEXIDDoubleBufferingSupported:    YES
 PEXIDTransparencySupported:       YES
 PEXIDBestColorApprox:             0
 PEXIDChromaticityRedU:            0.450
 PEXIDChromaticityRedV:            0.522
 PEXIDLuminanceRed:                1.000
 PEXIDChromaticityGreenU:          0.120
 PEXIDChromaticityGreenV:          0.561
 PEXIDLuminanceGreen:              1.000
 PEXIDChromaticityBlueU:           0.175
 PEXIDChromaticityBlueV:           0.157
 PEXIDLuminanceBlue:               1.000
 PEXIDChromaticityWhiteU:          0.188
 PEXIDChromaticityWhiteV:          0.466
 PEXIDLuminanceWhite:              1.000
 PEXIDNominalLineWidth:            1
 PEXIDNumSupportedLineWidths:      0
 PEXIDMinLineWidth:                1
 PEXIDMaxLineWidth:                16383
 PEXIDNominalEdgeWidth:            1
 PEXIDNumSupportedEdgeWidths:      1
 PEXIDMinEdgeWidth:                1
 PEXIDMaxEdgeWidth:                1
 PEXIDNominalMarkerSize:           3
 PEXIDNumSupportedMarkerSizes:     0
 PEXIDMinMarkerSize:               3
 PEXIDMaxMarkerSize:               2147483647
```

```
   PEXIDMaxNameSetNames:                2147483647
   PEXIDMaxModelClipPlanes:             6
   PEXIDMaxNonAmbientLights:            15
   PEXIDMaxNURBOrder:                   6
   PEXIDMaxTrimCurveOrder:              6
   PEXIDMaxHitsEventSupported:          YES
   PEXExtIDMaxTextureMaps:              8
   PEXExtIDMaxFastTMSize:               0
   PEXExtIDPowerOfTwoTMSizesRequired:   YES
   PEXExtIDSquareTMRequired:            NO
   HP_IDDeformationSupported:           YES
   HP_IDCappingPlanesSupported:         YES
   HP_IDInterferenceSupported:          YES
   HP_IDPolygonOffsetSupported:         NO

PREDEFINED LOOKUP TABLE ENTRIES

   PEXLUTColorApprox
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               1 (0x1)

      Entry 0
         type:                          1 (0x1)
         model:                         1 (0x1)
         max1:                          7 (0x7)
         max2:                          7 (0x7)
         max3:                          3 (0x3)
         dither:                        1 (0x1)
         mult1:                         32
         mult2:                         4
         mult3:                         1
         weight1:                       0.000
         weight2:                       0.000
         weight3:                       0.000
         base_pixel:                    0

   PEXLUTColor
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               8 (0x8)

      Entry 0
         type:                          PEXColorTypeRGB
         value
            red:                        0.000
            green:                      0.000
            blue:                       0.000

      Entry 1
         type:                          PEXColorTypeRGB
```

```
     value
        red:                        1.000
        green:                      1.000
        blue:                       1.000

  Entry 2
     type:                          PEXColorTypeRGB
     value
        red:                        1.000
        green:                      0.000
        blue:                       0.000

  Entry 3
     type:                          PEXColorTypeRGB
     value
        red:                        1.000
        green:                      1.000
        blue:                       0.000

  Entry 4
     type:                          PEXColorTypeRGB
     value
        red:                        0.000
        green:                      1.000
        blue:                       0.000

  Entry 5
     type:                          PEXColorTypeRGB
     value
        red:                        0.000
        green:                      1.000
        blue:                       1.000

  Entry 6
     type:                          PEXColorTypeRGB
     value
        red:                        0.000
        green:                      0.000
        blue:                       1.000

  Entry 7
     type:                          PEXColorTypeRGB
     value
        red:                        1.000
        green:                      0.000
        blue:                       1.000

PEXLUTDepthCue
  Maximum Entries:                  65535 (0xffff)
```

```
   Predefined Entries:              1 (0x1)

   Entry 0
      mode:                         0 (0x0)
      front_plane:                  1.000
      back_plane:                   0.000
      front_scaling:                1.000
      back_scaling:                 0.500
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                0 (0x0)

PEXLUTEdgeBundle
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
      edge_flag:                    0 (0x0)
      type:                         1 (0x1)
      width:                        1.000
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                1 (0x1)

PEXLUTInteriorBundle
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
      style:                        1 (0x1)
      style_index:                  0 (0x0)
      reflection_model:             1 (0x1)
      interp_method:                1 (0x1)
      bf_style:                     1 (0x1)
      bf_style_index:               0 (0x0)
      bf_reflection_model:          1 (0x1)
      bf_interp_method:             1 (0x1)
      surface_approx
         method:                    1 (0x1)
         u_tolerance:               1.000
         v_tolerance:               1.000
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                1 (0x1)
      reflection_attr
         ambient:                   1.000
         diffuse:                        1.000
         specular:                       1.000
         specular_conc:                  0.000
         transmission:                   0.000
         specular_color
            type:                        PEXColorTypeIndexed
            value
               indexed:                  1 (0x1)
      bf_color
         type:                           PEXColorTypeIndexed
         value
            indexed:                     1 (0x1)
      bf_reflection_attr
         ambient:                        1.000
         diffuse:                        1.000
         specular:                       1.000
         specular_conc:                  0.000
         transmission:                   0.000
         specular_color
            type:                        PEXColorTypeIndexed
            value
               indexed:                  1 (0x1)

PEXLUTLight
   Maximum Entries:                      65535 (0xffff)
   Predefined Entries:                   1 (0x1)

   Entry 1
      type:                              1 (0x1)
      direction
         x:                              0 (0x0)
         y:                              0 (0x0)
         z:                              0 (0x0)
      point
         x:                              0 (0x0)
         y:                              0 (0x0)
         z:                              0 (0x0)
      concentration:                     0.000
      spread_angle:                      0.000
      attenuation1:                      0.000
      attenuation2:                      0.000
      color
         type:                           PEXColorTypeRGB
         value
            red:                         1.000
            green:                       1.000
            blue:                        1.000

PEXLUTLineBundle
```

```
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
      type:                         1 (0x1)
      interp_method:                1 (0x1)
      curve_approx
         method:                    1 (0x1)
         tolerance:                 1.000
      width:                        1.000
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                1 (0x1)

PEXLUTMarkerBundle
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
      type:                         3 (0x3)
      scale:                        1.000
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                1 (0x1)

PEXLUTPattern
   Failed to get PEX lookup table info

PEXLUTTextBundle
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
      font_index:                   1 (0x1)
      precision:                    2 (0x2)
      char_expansion:               1.000
      char_spacing:                 0.000
      color
         type:                      PEXColorTypeIndexed
         value
            indexed:                1 (0x1)

PEXLUTTextFont
   Maximum Entries:                 65535 (0xffff)
   Predefined Entries:              1 (0x1)

   Entry 1
```

```
         font_index:                    65536 (0x10000)

   PEXLUTView
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               1 (0x1)

      Entry 0
         clip_flags:                    7 (0x7)
         clip_limits
            min
               x:                       0.000
               y:                       0.000
               z:                       0.000
            max
               x:                       1.000
               y:                       1.000
               z:                       1.000

   PEXExtLUTTMBinding
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               0 (0x0)

   PEXExtLUTTMCoordSource
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               0 (0x0)

   PEXExtLUTTMComposition
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               0 (0x0)

   PEXExtLUTTMSampling
      Maximum Entries:                  65535 (0xffff)
      Predefined Entries:               0 (0x0)


AVAILABLE PEX FONTS

      -hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-hp-roman8
      -hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-hp-roman8
      -hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-hp-roman8
      -hp-PEX polygonal sans serif-bold-r-normal-normal-0-0-0-0-p-0-hp-roman8
      -hp-PEX polygonal serif-bold-r-normal-normal-0-0-0-0-p-0-hp-roman8
      -hp-PEX polygonal serif-bold-r-normal-accel-0-0-0-0-p-0-hp-roman8
      -hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-hp-japaneseeuc
      -hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
      -hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
      -hp-PEX polygonal sans serif-bold-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
      -hp-PEX polygonal serif-bold-r-normal-normal-0-0-0-0-p-0-hp-japaneseeuc
      -hp-PEX stick-medium-r-normal-normal-0-0-0-0-m-0-iso8859-1
```

```
    -hp-PEX stick-medium-r-normal-normal-0-0-0-0-p-0-iso8859-1
    -hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-m-0-iso8859-1
    -hp-PEX simplex sans serif-medium-r-normal-normal-0-0-0-0-p-0-iso8859-1

SUPPORTED PEX VISUALS

  Target 0
    type:                              Window
    depth:                             12 (0xc)
    visual:                            TrueColor
  Target 1
    type:                              Window
    depth:                             12 (0xc)
    visual:                            DirectColor
  Target 2
    type:                              Window
    depth:                             24 (0x18)
    visual:                            TrueColor
  Target 3
    type:                              Window
    depth:                             24 (0x18)
    visual:                            DirectColor
  Target 4
    type:                              Window
    depth:                             8 (0x8)
    visual:                            TrueColor
  Target 5
    type:                              Window
    depth:                             8 (0x8)
    visual:                            PseudoColor
  Target 6
    type:                              Window
    depth:                             8 (0x8)
    visual:                            PseudoColor
  Target 7
    type:                              Window
    depth:                             8 (0x8)
    visual:                            PseudoColor
  Target 8
    type:                              Buffer
    depth:                             12 (0xc)
    visual:                            TrueColor
  Target 9
    type:                              Buffer
    depth:                             12 (0xc)
    visual:                            DirectColor
```

FINAL TRIM SIZE : 7.5 in x 9.0 in

Glossary

Abscissa

The value representing the distance of a point from the Y-axis in the Cartesian coordinate system, measured along a line parallel to the X-axis. (Compare "Ordinate".)

Active Texture List

A list of binding table entries which specifies the currently active textures within the rendering pipeline. All appropriate primitives will receive texture mapping effects sequentially evaluated from this list. Two separate lists exist for front- and backface distinguishing. (See `PEXExtSetActiveTextures` and `PEXExtSetBFActiveTextures`).

Alpha Blending

The operation of blending a source pixel color with a destination (frame buffer) pixel color according to some rule on alpha values.

Alpha Transparency

An application of alpha blending to achieve the effect of transparent primitives; requires a multi-pass algorithm in order to generate realistic images.

Anti-aliasing

A method for producing high-quality images using pixel coverage and blending techniques, most noticeable as smooth lines and polygon edges.

Azimuth

The horizontal angular distance from a fixed reference direction to a point.

Binding Lookup Table

A lookup table whose entries represent entire texture maps to the rendering pipeline. Each entry within the binding lookup table contains reference information for an X texture resource, texture coordinates for each primitive, color composition rules, and texture map sampling and quality controls. (See "Binding LUT" in Chapter 9).

Boundary Condition: Clamp Absolute
When a texture coordinate accesses a texel outside of the texture map, texturing is discontinued and the primitive's existing color data is used. (See "Sampling LUT" in Chapter 9.)

Boundary Condition: Clamp Color
When a texture coordinate accesses a texel outside of the texture map, texturing is discontinued and the "clamp col or" is applied. (See "Sampling LUT" in Chapter 9.)

Boundary Condition: Mirror
When a texture coordinate accesses a texel out side of the texture map, sampling is reversed across the texture map. (See "Sampling LUT" in Chapter 9.)

Boundary Condition: Wrap
When a texture coordinate accesses a texel out side of the texture map, sampling wraps back to the opposite texture border creating a "rubber stamp" effect. (See "Sampling LUT" in Chapter 9.)

Capping
A visualization technique, used with model clipping that *re-closes* a volume that has been clipped, making the object appear as though it had been cut away.

Color Ramp
The colors in a colormap for `PEXColorSpace` which are expected to represent a "sampling" of the color space.

Composition Lookup Table
A lookup table with entries used to determine how texture map values will be applied to the current color of the rendering pipeline. Entries within this table control the blending and replacement rules for each texture-mapping operation. (See "Composition LUT" in Chapter 9.)

Composition Type: Decal

If alpha is not included in the texture map, the texture map color replaces the primitive's existing color. If, on the other hand, alpha is specified in the texture map, the following equation is used to determine the final blended color:

$$C_{out} = C_{in} \times (1 - t_a) + t_c \times t_a$$

where C_{in} is the primitive's existing color, t_a is the texture map alpha and t_c is the texture map color.

(See "Composition LUT" in Chapter 9.)

Composition Type: Modulate

The texture map color and alpha (if it exists) blend with the primitive's existing color and alpha. The texture color (alpha) is multiplied by the primitive's color (alpha) to determine the final result. (See "Composition LUT" in Chapter 9.)

Composition Type: Replace

The texture map color and alpha (if it exists) overwrite the primitive's existing color and alpha. (See "Composition LUT" in Chapter 9.)

Coordinate Source Lookup Table

A lookup table with entries used to determine how texture coordinates are to be derived for texture mapping. Texture coordinates are either explicitly stored with a vertex as floating point data or they are derived from other vertex data (point, color, normal). (See "Coordinate-Source LUT" in Chapter 9.)

Deformation

A technique for computing a displacement in model coordinates for each vertex of a primitive, based on data supplied with the vertex.

Interference Checking

A method for visualizing and detecting inter-penetrating solids by highlighting overlapping caps within a clip plane.

Magnification Method

The process used to determine the final color for a screen pixel when more than one screen pixel maps to one texture map texel. (See "Sampling LUT" in Chapter 9.)

Minification Method
The process used to determine the final color for a screen pixel when one screen pixel maps to more than one texture-map texel. (See "Sampling LUT" in Chapter 9.)

MIP Map

A pre-computed area-sampling mechanism which fixes the cost of approximating the average color over a large number of pixels in a texture image. A MIP map (and RIP map, which involves additional rectangular dimensions) is created as an image "pyramid" of down-sampled maps, or "levels." Traditionally, each pixel in a level n equals the average of four pixels beneath it at level $n+1$. Thus the resolution of each map becomes half of the preceding level until the top level is reached, which has one pixel representing the average color of the entire original base-level map.

See also "Discussion: MIP Map" in Chapter 10.

Ordinate
The value representing the distance of a point from the X-axis in the Cartesian coordinate system, measured along a line parallel to the Y-axis. (Compare "Abscissa".)

Overlay Planes
Display hardware often has two kinds of display planes, image and overlay. The image plane allows the hardware to help the graphics commands run faster and more efficiently. Overlay planes are graphics frame buffer planes that store pixel data that is independent of the image buffer. These planes can be used for alpha text, windows, cursors, or menus as well as graphics. They can be written to and turned on and off independently of the graphics, or image, planes.

Parameterization Lookup Table
A lookup table with entries used to determine how texture coordinates are to be derived for texture mapping. Texture coordinates are either explicitly stored with a vertex as floating-point data or they are derived from other vertex data (point, color, normal).

Preparation

The data-processing phase before rendering where texture data are loaded and area primitives with data are "surface parameterized" with texture coordinates per vertex.

Projection Object

A standard volume used in calculation of texture coordinates. A primitive is conceptually placed at the center of a projection object and each of the primitive's vertices are projected onto the projection object. The intersections of the vertices with the projection object determine the texture coordinates. (See `PEXExtCreateTMDescription` and `PEXExtTMCoord*`) Spherical, cylindrical, and planar projection objects are possible objects.

Reflection Mapping

A type of texture mapping that uses reflection vectors to calculate texture coordinates. The result is an object that reflects or mirrors the texture map much like a shiny Christmas tree ornament reflects its environment. See also **Standard Mapping**.

Sampling Lookup Table

A lookup table with entries used to control texture map sampling (derivation of a color sample from an image) and rendering quality hints for each texture. (See "Sampling LUT" in Chapter 9.)

Standard Mapping

Standard texture mapping refers to the mapping of a texture onto an object much like a piece of wrapping paper is applied to a package. See also **Reflection Mapping**.

Surface Parameterization

A mathematical projection of a 3D surface onto a 2D surface which, in effect, "ties" facet data to corresponding regions of a texture image map, thus orienting an image on the primitive. (See the texture parameterization utilities, such as `PEXExtTMCoordFillAreaSetWithData`.)

Texel

One **tex**ture map **el**ement. Texel is analogous to the term "pixel"—a texel is to a texture map as a pixel is to a bitmap. A texel may be a floating point value, an 8-bit integer, or in another for mat.

Texture Map

A 1D, 2D, or 3D data set consisting of “texels” (or texture elements). A 1D texture map is an array of values; a 2D texture map is a two-dimensional image, and a 3D texture map is a set of 2D texture maps. HP PEX supports 1D and 2D texture maps. (See **texel**.)

Texture Mapping

A rendering effect which enhances the surface detail of an area primitive for usually less cost than explicitly modeling the information. Texture mapping controls interior color and transparency through special “mapped” correspondences between texture images and area primitives during the rendering phase.

View-Dependent Mapping

Texture mapping that changes with the position of the camera. If the camera (or point of view) changes, the orientation of the texture map on the texture-mapped object changes.

View-Independent Mapping

Texture mapping that does not change with the position of the camera. A texture map is fixed on an object and does not change even if the position of the camera or object changes. Describes the characteristics of a virtual colormap that has been or can be created for use on a particular screen.

Visual Class

Distinguishes between color or monochrome, whether the color map is read/write or read-only, and whether a pixel value provides a single index to the colormap or is decomposed into separate indices for red, green, or blue values.

Index