CGE PEX 5.1 Portability Guide

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 801-6749-10
Revision A, August 1994

SunSoft
A Sun Microsystems, Inc. Business

Contents

Figures

Tables

Code Samples

Introduction to Portable Programming with CGE PEX 5.1 Extensions

1

Purpose and Scope of this Guide

The purpose of this guide is to assist you in creating highly portable 3D graphic applications on platforms supporting the Common Open Software Environment. The programming interface that you will use for this purpose is known as the Common Graphics Environment PEX 5.1 Extensions (CGE PEX 5.1). Using this interface, you can create applications that are highly portable and interoperable on platforms that support CGE PEX 5.1. By following this guide, you should have success creating applications that meet this goal. You will find that the graphic portions of your application will compile effortlessly on all CGE PEX 5.1 platforms, eliminating the need for vendor specific *drivers* or code paths. You will also find that the graphical results on each CGE PEX 5.1 platform will be more complete and consistent, again reducing the need for vendor specific code paths.

This guide is written specifically for application developers who want to create portable and interoperable applications for the CGE PEX 5.1 platforms. This guide is not a PEX or a PEXlib tutorial. There are many other references available that serve as better and more general tutorial material. See Appendix A, “Other Useful References.” This guide is derived from the Common Graphics Environment PEX 5.1 Extensions specification which you may obtain from your workstation vendor or from the X Consortium.[1] Or perhaps your vendor has integrated a version of this information with their PEX product.

While you may find some useful techniques in this guide that apply to creating portable vanilla PEX 5.1 applications, this guide does not specifically address this more general goal. Also, this guide does not include a tutorial on the new CGE PEX 5.1 functionality, specifically texture-mapping. It explains how to add texture mapping to your application so that it will work on all CGE PEX 5.1 platforms. Please refer to product documentation and other literature listed in Appendix A, "Other Useful References" to learn the fundamentals of features such as texture mapping.

This guide will help you interpret the basic CGE PEX 5.1 specifications, especially the more subtle, yet useful, information that is crucial to your application's success. You will also find a large number of hints and techniques that are the result of a cumulative total of years of experience in creating portable graphics applications in multi-vendor environments. PEX application programming is still in its infancy stage, so many of the solutions to common problems and difficulties have not yet disseminated through the industry. This guide contains the solutions to many of these problems and will help you put your PEX application well ahead of your competition's.

Motivation to Use CGE PEX 5.1 Extensions

We hope that you have chosen to create your next application with CGE PEX 5.1 for one or many of the following reasons:

Increased Portability

The CGE PEX 5.1 vendors are committed to supplying you with a CGE PEX 5.1 programming library that is consistent across all CGE PEX 5.1 platforms. There are no *missing* or *conflicting* function calls on any platform and all existing functions are implemented identically. In fact, some of the same code has been

1. You can obtain the specification from the X Consortium's FTP (File Transfer Protocol) server. The server's address is ftp.x.org. Login as "anonymous", giving your Internet address as the password. The specification can be found in: /contrib/PEXlib in the file CGE-PEX-51-Spec-V1.txt.Z.

shared by the vendors to implement the CGE PEX 5.1 library, so you might be using products on different vendors' platforms that were derived from the same source code base!

This promise of portability reduces the need for you to worry about programming graphics, an area that has historically been riddled with vendor extensions and incompatibilities.

Increased Interoperability

Your CGE PEX 5.1 application interoperates with another vendor's CGE PEX 5.1 server with no difficulty. This allows your application to execute on one machine, while interacting with a user running a CGE PEX 5.1 server on another workstation, even if the workstations come from different vendors. Interoperability gives your customers with large heterogeneous installations the ability to host or port applications onto fewer types of host machines, while still making them available for use from a wide variety of workstations.

Early Access to Some PEX 5.2 functionality

One of the most exciting benefits about CGE PEX 5.1 is that it provides you with the earliest consistent multi-vendor implementation and includes some of the functionality slated for PEX 5.2. The early availability of these functions will help you put more powerful applications in the hands of your customers without you having to resort to differing vendor-specific PEX extensions or switching to another graphics support package.

- Texture Mapping - This is the biggest and most exciting new feature in CGE PEX 5.1. Texture Mapping allows you to project pixmap-like images onto various PEX primitives, giving them appearances not possible with standard PEX.
- Anti-Aliasing - This allows you to create higher quality pictures by instructing the PEX server to draw specified primitives with a reduced "jaggies" effect. Some PEX vendors have provided this as a vendor extension in PEX 5.1 products. You can now count on it in CGE PEX 5.1.
- Transparency - Although this feature is a part of the PEX 5.1 specification, it was not required. It is now a part of CGE PEX 5.1, and offers you the ability to create objects that transmit light to varying degrees, that objects on the other side may appear clear or sheer, gauzy or diaphanous.

- Drafting Primitives - These are the common and useful circle, ellipse, and arc primitives that many vendors have provided as PEX 5.1 vendor extensions. They are now a common part of CGE PEX 5.1.

Note – It is impossible to guarantee that any of these functions will appear in PEX 5.2 in exactly the same way they appear in CGE PEX 5.1, because PEX 5.2 was not in a final state at the time CGE PEX 5.1 developed. However, the creators of CGE PEX 5.1 have done everything possible to follow PEX 5.2 directions and keep CGE PEX 5.1 as close to those directions as possible.

Higher Level of PEX 5.1 Functionality

One of the criticisms of PEX 5.1 is that too many features were left optional and that too few vendors implemented these optional functions. CGE PEX 5.1 raises the level of required implementation, giving you a graphics support system with a higher level of guaranteed functionality. This means that your application will have to make fewer queries for optional function. Fewer optional features means fewer possible configurations your application will have to handle with separate code paths.

Relationship of CGE PEX 5.1 to PEX 5.1

So, enough of the sales pitch. You're convinced, right? Now, what does all this mean to you if you are currently using PEX 5.1?

CGE PEX 5.1 Extensions As a Superset of PEX 5.1

Well, the name sure says it, but what do we really mean?

CGE PEX 5.1 is a set of extensions built on top of PEX 5.1 using the X Consortium standard rules and interoperability conventions for extending PEX 5.1. (See Chapter 8, "Interoperability Conventions"). These rules and guidelines are engineered to allow PEX 5.1 extensions to exist without affecting the operation of any core PEX 5.1 functions. This means that if you were to suddenly upgrade your graphics support software to CGE PEX 5.1, your PEX 5.1 applications would still work without modification.

If you want to take advantage of new CGE PEX 5.1 function (for example, texture mapping), you'll need to modify your application to make use of this function. Note that some of the PEX 5.1 optional function that is now required in CGE PEX 5.1 (for example, transparency) might suddenly start to work in your application.

CGE PEX 5.1 Programs in 5.1 Environments

If you add a CGE PEX 5.1 function, like texture mapping, to your application, your application may not work with a vanilla PEX 5.1 server. This is because the vanilla 5.1 server will not understand the CGE PEX 5.1 extensions and will probably return errors.

You'll need to decide if you want your application to continue to work with vanilla PEX 5.1 servers. If so, you'll have to add logic to your application to avoid the CGE PEX 5.1 functions when connected to a PEX 5.1 server.

Relationship of CGE PEX 5.1 to PEX 5.2

While moving your application to CGE PEX 5.1 gives you early access to PEX 5.2 function, it does not guarantee that your CGE PEX 5.1 program will port without effort to PEXlib 5.2. Since PEXlib 5.2 will be backwards compatible with PEXlib 5.1, the non-CGE portions of your application should port with no effort. However, the CGE-specific portions may need some tweaking to work with PEXlib 5.2.

At this stage, PEXlib 5.2 is still not completely defined, so the CGE vendors cannot predict all the changes that you will have to make. The intention is for changes to be limited to minor syntactical changes, but CGE vendors cannot guarantee this will be the case. You may choose to make the necessary changes when you upgrade your application from CGE PEX 5.1 to PEXlib 5.2. You can then make any CGE->5.2 changes as well as the changes required for your application to take advantage of any PEX 5.2 function not found in CGE PEX 5.1. This situation is illustrated in Figure 1-1 on page 6.

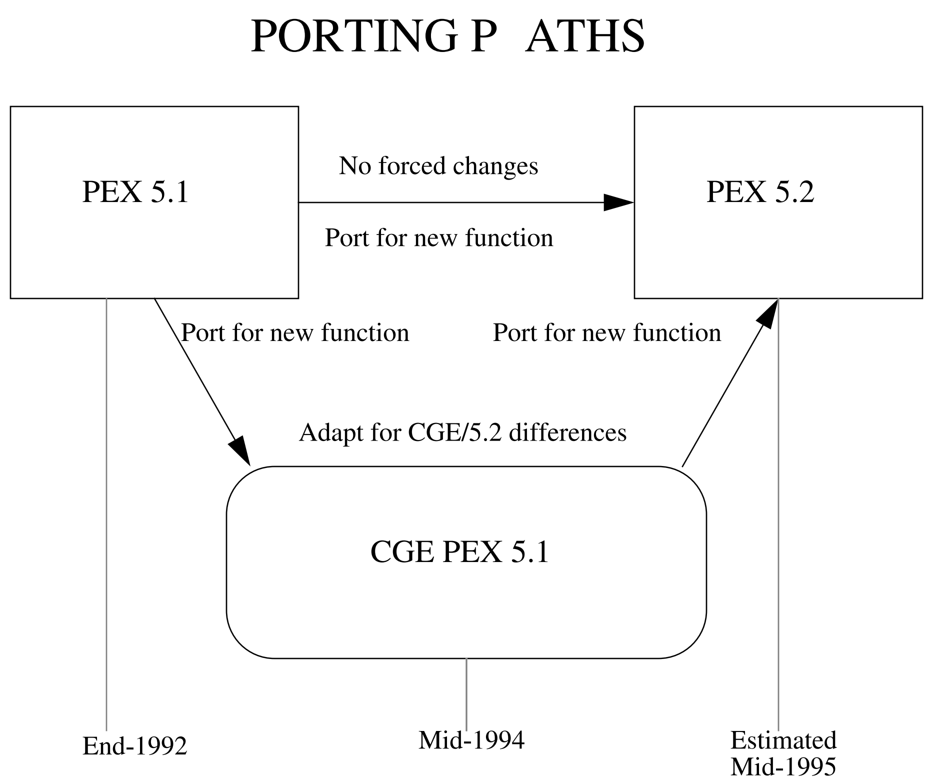

Figure 1-1 Porting Paths

CGE PEX 5.1 Extension Mechanics

This section explains how PEX 5.1 was extended to obtain CGE PEX 5.1. Skip this section if this sort of detail does not interest you.

CGE PEX 5.1 Follows PEX 5.1 Interoperability Conventions

See Chapter 8, "Interoperability Conventions" for more information. This information is also useful for general PEX 5.1 applications.

Vendor ID

To avoid conflicts between vendors adding extensions to PEX 5.1, the X Consortium assigns an 8-bit vendor ID to vendors wishing to implement extensions. The vendor ID is used in the protocol encoding of the extension to distinguish the extension protocol from another vendor's extensions.

The vendor ID 0x10 is IBM's vendor ID, but is being used by all vendors designing or implementing CGE PEX 5.1. CGE is using IBM's vendor ID because it has the least amount of conflict with existing IBM extensions. IBM ensures that other IBM extensions do not conflict with the CGE PEX 5.1 extensions.

According to the PEX 5.1 Interoperability Conventions, 16-bit quantities, such as Output Command types and Enumerated Type indices, are encoded with values starting at 0x9000, while 32-bit quantities like Escape opcodes start at 0x80100000.

New Output Commands Encoded As Vendor-Specific Output Commands

CGE PEX 5.1 specifies new Output Commands (OCs) as vendor-specific OCs, setting the high bit on and using the vendor ID for the next 7 bits. This is in contrast to the current 5.2 direction of using GDPs and GSEs for extension OCs. Besides inquiry, you should see little functional difference between the two methods. You can query the existence of the new OCs by getting the CGE PEX 5.1 enumerated type information for CGE PEX 5.1 output commands.

New Requests Encoded As PEXEscape and PEXEscapeWithReply

The new CGE PEX 5.1 requests are converted by PEXlib to PEXEscape or PEXEscapeWithReply requests, as this is the only way to add requests to PEX 5.1.

Extended Renderer Attributes and Extended Pipeline Context Attributes Accessed with New Requests

CGE PEX 5.1 adds several attributes to the renderer and the pipeline context. Due to protocol restrictions, you cannot access these new attributes with the PEX 5.1 requests. Therefore, CGE PEX 5.1 provides requests to change and get these extended attributes.

You cannot create either of these resources with the extended attributes; you must create them with PEX 5.1 requests and then later change them with the new CGE PEX 5.1 requests (escapes).

Preparing CGE PEX 5.1 Extensions Programs

2

CGE PEX 5.1 Packaging

Your workstation vendor provides the CGE PEX 5.1 libraries in one or both of two ways: the *integrated library* in which the CGE PEX 5.1 function is built directly into the PEXlib library and the *layered library* in which the source code of CGE PEX 5.1 is provided, enabling you to build your application on any system that has the PEX 5.1 libraries.

Integrated Libraries

The *integrated library* contains the CGE PEX 5.1 function built directly into the PEXlib library that you link to your application program. In this case, the library is provided only in object-code format and the source is generally not available. This is the best method to use when you know that the CGE PEX 5.1 library will always be available on the system, because the *integrated library* will likely be the most optimized library available.

Library files may vary across CGE vendors; but most likely each vendor will provide the following header files:

`PEXExt.h` and `PEXExtlib.h`

and supply the *integrated library* code itself in the same library file as the standard PEX 5.1 support:

`libPEX5.a` or `libPEX5.sl` (example of a shared lib)

Layered Libraries

The *layered library* allows you to build your application on any system where only PEXlib 5.1 libraries are available. This library consists of a set of macros and/or functions that convert your application's CGE PEX 5.1 function calls to the equivalent PEX 5.1 PEXEscape or PEXEscapeWithReply function calls. The source code for this library is provided so you can compile it on any system that has PEXlib 5.1. With this library, you can port your application to platforms that do not support CGE PEX 5.1 libraries in their integrated form, with the intent of displaying your application on PEX servers running on other platforms that do support the CGE PEX 5.1 extensions.

In addition to the files provided with the *integrated library*, the vendor also provides several other C source and header files. These files are required to build the layer portion of the *layered library*. In order to use the *layered library*, you need to compile this layer code first to obtain object files that you include when linking your application.

Source Code Requirements

There is only one source code change that you must make to access the CGE PEX 5.1 functions. You need to add a preprocessor include directive to include the CGE PEX 5.1 header file after the directive that includes the standard PEXlib 5.1 header file.

```
#include <X11/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h> /* Add this line */
```

The `PEXExtlib.h` file includes `PEXExt.h`, so you do not have to.

Compilation

You need to make sure that your compiler can locate the include file mentioned above. Many compilers find the `X11/PEX5` directory during the searching of system default header files. If your PEX header files are stored in a different area, use the `-I` compiler directive to help the compiler locate the files.

It does not matter if you are using the integrated or layered libraries when compiling your application source code modules because they compile in the same way.

Linking

When using the integrated library, you link the library in the same way as you would link a standard PEXlib 5.1 application. To do this, use the compiler/linker directive:

```
-lPEX5
```

If this library is not in your system's default library directives, use the `-L` directive to tell the linker where to look.

If you are using the layered library, include the additional object files that contain the CGE PEX 5.1 code, as mentioned above.

A Word About Makefiles and Imakefiles

Typically it is difficult to create a Makefile that works on a large number of different platforms. Though the various Unix[1] standardization efforts reduce this problem, vendors ship various compilers and tools that use a large variety of options and syntax variations. Therefore, portable Makefiles currently are not offered. However, modifying Makefiles is probably one of the smaller and easier tasks of porting a large application from one vendor's machine to another. It is certainly reasonable to create and maintain a separate Makefile for each platform.

The `imake` program is an X Consortium tool that generates Makefiles using preprogrammed platform configuration data. You may find that this tool is a possible alternative to handling multiple Makefiles.

Vendor Extensions

Yes, some vendors even provide PEX extensions other than this CGE PEX 5.1 extension. You should use these extensions carefully in order to keep your program portable and interoperable. This probably means coding a path for platforms that have the extension and a path for those that don't.

1. Unix is a registered trademark licensed exclusively by X/Open Company Ltd. X/Open is a trademark of X/Open Company Ltd.

The vendor extension may be wholly contained and described in header files that define constants and data structures that you will use in PEXEscape or PEXEscapeWithReply function calls. (Some header files may define macros that "hide" the PEXEscape function call from you.) To help keep source code using these extensions consistent, use a preprocessor directive naming an extension library to include the files:

```
#include <X11/PEX5/extensions/filename.h>
```

This will probably be the convention adopted for PEXlib 5.2, so you are ahead of that game once again. The actual filename is constructed out of an abbreviation of the owning vendor's name and the function that the extension provides.

Other types of vendor extensions may be provided via C language source code files. However, you may have to compile such code before using the extensions.

Finally, a vendor may choose to extend PEXlib by adding new entry points directly to the PEXlib implementation, without supplying source code. In these cases, you may have to code "stub" routines, so that you can compile and link your application on other platforms. Remember, in this case, you must code your application to avoid the vendor extension calls if the extensions are not available on your application's platform.

You probably will find vendor extension files only on individual vendor's machines. So, if you'd like your application code to compile on other vendors' platforms, be sure to include a copy of these files with your application.

Design Considerations

You will have to decide ahead of time how you want your application to work in situations where the CGE might not be present. Do you want your application to work *only* when there is a CGE PEX server doing the display work for the user? Or, do you want your application to go ahead and still run, although not as efficiently, on a vanilla PEX 5.1 server?[1]

1. An implementation of PEX 5.1 without any CGE extensions or any other extensions is referred to as a vanilla PEX 5.1 server or application.

The CGE-only PEX Server Application

In this case, your application *requires* that the PEX server support the CGE PEX 5.1 extensions before it does anything. This requires that your users have the necessary CGE support on the system they are running the PEX server. This makes your application more "exclusive", but perhaps simpler, internally. All you need to do in the application is to check for the existence of the CGE PEX 5.1 extensions. If they are there, then continue. Else, the application should terminate, printing a polite message that the CGE extensions are not present.

The CGE / PEX 5.1 Application

In this case, your application determines at an early stage whether or not the CGE PEX 5.1 extensions are present in the PEX server. The application continues executing, using the extensions if they are present in the server, ignoring them in the application if they are not. This means that when it is time for the application to use a CGE-specific feature, it must check a flag to see if the CGE support is there. Then, the application executes either a CGE path or non-CGE path. The non-CGE path can be as simple as not performing the specific function or can be as complex as an elaborate emulation of the feature. It all depends on how important the feature is to the application.

Critical Features

These are features of CGE PEX that your application cannot do without. For example, if the entire purpose of your application is to use texture mapping to illustrate the stress points on a metal frame, then your application would probably not want to continue if the CGE support were not present.

Visual-Only Features

These are features that improve the appearance of your application, but are not critical to its usability. For example, some applications would work OK if anti-aliasing were not available. Applications like these will likely continue in *degraded mode* if CGE support is not present.

Initializing an Application Using the CGE PEX 5.1 Extensions 3

Getting Started

In many ways, your application begins just like any other PEXlib application.

Initializing PEXlib

You initialize PEXlib the same way you initialize any PEXlib application.

Detecting a CGE PEX 5.1 Implementation

One of the first things you will want to do in your application is to find out if the PEX server has the CGE extensions. Then you can decide either to abort the application or configure the application to run without the extensions. You should query the server for the list of Escapes that the PEX extension supports using the PEXGetEnumTypeInfo request. If the server has the CGE extensions, then you will find the escape identifiers, starting at 0x9000 (PEXETEscapeChangePipelineContext), in the list. The following is a code sequence that performs the query:

```
{
    int status, i, found;
    int enum_types[1];
    unsigned long *enum_counts;
    PEXEnumTypeDesc *enum_values, *p_enum;

    enum_types[0] = PEXETEscape;

    status = PEXGetEnumTypeInfo(dpy,
                                drawable,
                                1,
                                enum_types,
                                PEXETCounts | PEXETIndex,
                                &enum_counts,
                                &enum_values);
    if (!status) {
        /* failure */
    }

    p_enum = enum_values;
    found = False;
    for (i=0; i<enum_counts[0]; i++) {
        if (((unsigned short) (p_enum++)->index) ==
        ((unsigned short) PEXETEscapeChangePipelineContext))
        found = True;
    }
}
```

Checking for Optional Features

One of the goals of CGE PEX 5.1 is to reduce the occurrence of *optional features* that application developers could not always count on being present in the PEX server implementation. While there is much more mandated functionality, there are still allowable differences in the degree that some of the features might be implemented across CGE PEX server implementations. For example, texture mapping is a required function, but only 2D texture maps are required to be implemented by all CGE PEX servers.

Once you have established that your application is connected to a CGE PEX server, you can then either count on the required functionality being there without further query, or you can issue additional PEXGetEnumTypeInfo requests to determine what additional functionality might be available. Continuing with the texture mapping case, you can now go ahead and use 2D textures without issuing further queries. But if you also wanted to use 1D textures, you would have to query the PEXExtTMDomain enumerated type to make sure that the server supported 1D textures.

Many people design applications that are flexible in a few ways, depending on the level of this sort of optional support. The application usually queries the features for which it can adjust once during initialization and keeps the results around for easy access. It is usually a good idea to perform these queries only once, because round-trips to the server can be expensive and the data does not change during the session anyway.

Multi-Buffering

In accordance with the preferred industry-standard way of implementing double-buffering, CGE PEX 5.1 requires that platforms supporting CGE PEX 5.1 must also provide the X Multi-Buffering Extension (MBX). This means if you are using MBX 3.2 or later, then your applications are assured of portability on any CGE PEX 5.1 platform. This eliminates the need for you to invoke XListExtension or XQueryExtension functions. Use MBX in your applications any time double-buffering is desired. To use double-buffering via MBX, invoke:

```
XmbufCreateBuffers()
```

to create the image buffers. If the device only supports one MBX image buffer, then you could either *live* with single buffering, or find out if the server supports rendering to pixmaps. Issue PEXMatchRenderingTargets to do so. If rendering to pixmaps is supported, copy the pixmap to the window to simulate double-buffering.

For every window or pixmap drawable that is supported by a PEX implementation, a corresponding MBX buffer drawable, whether single-buffered or double-buffered, will be supported on any server supporting CGE PEX 5.1.

Fonts

All vendors who support CGE PEX 5.1 use the X-Windows Logical Font Description (XLFD) font naming format. Loading and using a font with a `CHARACTER_REGISTRY` and `CHARACTER_ENCODING` property of ISO8859-1

(*.*.*.*.*.*.*.*.*.*.*.*.ISO8859-1)

ensures you of at least one PEX-usable font on any CGE PEX 5.1 platform.

PEX Subsets

Do not use the workstation subset in your CGE PEX application. It is not required for a CGE PEX implementation, so you are not guaranteed that it will always be available. Once you are certain that the server supports CGE PEX 5.1, you also do not need to check for the renderer or structure subsets. They are required in CGE PEX 5.1.

Opening a Window, Selecting a Visual, Creating a Colormap, and Establishing Color Approximation

The following are the four basic steps to PEX Color support:

- Select a Visual in which to create the Window.
- 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 selected Visual with the correct Colormap.

To help you complete these four basic steps, CGE PEX 5.1 offers utility extensions. These utilities are shipped as source code to give your application control over policy decisions regarding choice of visual and colormap organization. However, applications that support CGE PEX 5.1, should be able to use these functions without changes on any CGE PEX 5.1 supported platform.

As with other functions, call the PEXInitialize function prior to using these utilities.

Though there are a number of Visual and Colormap utilities, the main ones that will help you with the four steps listed above include:

- PEXUtSimpleWindowAndColormap which operates with "soft" criteria (discussed in "Visual Selection" on page 19) and creates a Window for PEX rendering.
- PEXUtMakeWindowAndColormap which is similar to the PEXUtSimpleWindowAndColormap function but offers you more explicit control over the criteria (see "Visual Selection" on page 19) and attributes used in selecting the Visual and creating the Window.
- PEXUtSelectVisual which is a lower-level utility that offers you control over the criteria used to select a Visual (see "Visual Selection" on page 19). You can then use other CGE-supplied utilities, or code of your own, to handle the Colormap and Color Approximation issues and the creation of a Drawable. Some of the CGE PEX utilities include: PEXUtMakeColormap, PEXUtGetStandardColormapInfo, and PEXUtCheckColorApproximation. See the CGE reference material for more details.

Visual Selection

If you have a simple application that does not have complicated requirements for the Visual, then you probably prefer to use the PEXUtSimpleWindowAndColormap utility. Otherwise, use the Visual selection criteria described in this section to provide input to the PEXUtMakeWindowAndColormap or PEXUtSelectVisual utility extensions.

You can supply one or more criterion for selection of a Visual. Criteria can be "hard" which indicates that if the results are not met by the Visual, then unacceptable results are produced. The utility returns a failure code or continues to the next criteria set if you have supplied more criteria. Criteria can also be "soft" which indicates that the characteristics are desirable but not required for a Visual to be acceptable.

There are two means of Visual Selection:

- Supply a single criteria set with some combination of "hard" and "soft" criteria. If the utility does not find a Visual that meets all of your "hard" criteria, then the utility returns an error code and your application can either quit or try another set of criteria. Failure to meet "soft" criteria is not considered a complete failure, though the utility informs you which ones were not met.

- Supply several sets of criteria (a "degradation" path) for Visual selection. Failure to meet the "hard" criteria in one set causes the utility to continue to the next set. If the utility reaches the last set that you supplied and cannot meet the requirements, then the utility returns an error code. Failure to meet "soft" criteria does not cause the utility to continue to the next set. Therefore, you probably want your degradation path to list "hard" criteria in the early sets and relax the requirements (for example, by listing "soft" criteria) in the later sets. When the utility finds a Visual that meets a set of criteria, the utility returns information on which set was successful.

Colormap Creation and Color Approximation

The Colormap and Color Approximation CGE utility extensions provide an interface to applications to ease the Colormap tasks. If you want to use these utilities in your application, then include the PEXUtCmap.h header file.

For PEX color support interoperability, use standard Colormaps. In general, with PEX 5.1, it is recommended that you use RGB_BEST_MAP and/or RGB_DEFAULT_MAP. You can use the PEXUtGetStandardColormapInfo utility to obtain a Colormap initialization and Color Approximation setup that PEX is likely to support in the specified Visual. If no interoperability property was used to obtain the Colormap description, the utility provides an *educated guess* about what color setup might be supported by PEX. The interoperable conventions include:

- As mentioned above, use standard Colormaps. In PEX it is recommended that you use RGB_BEST_MAP and/or RGB_DEFAULT_MAP. CGE PEX 5.1 servers are likely to define these properties and describe Colormaps that are supported by PEX via the PEXColorSpace approximation. With CGE PEX 5.1 utilities, this interoperable property is easy to meet by including the PEXUtStandardColormapProperty criterion as either *hard* or *soft* in each criteria set (see "Visual Selection" on page 19 for details on *hard* and *soft* criteria).
- Use the PEX 5.1 PEXEscapeQueryColorApprox escape or the CGE PEX 5.1 PEXExtQueryColorApprox escape to verify the setups available. An alternative setup is returned if the specified Colormap setup is not supported for the PEX ColorApproximation and your application then can choose either to adapt to the alternative configuration or to select another

Visual. The CGE PEX 5.1 utilities, PEXUtMakeWindowAndColormap and PEXUtSimpleWindowAndColormap, enable you to use the supplied alternative configuration.

To use the PEXUtMakeWindowAndColormap utility, for example, set one or two *hard* criteria to indicate the most essential characteristics required of a Visual or Window and any *soft* criteria. Ensure one of the *hard* or *soft* criteria is PEXUtStandardColormapProperty so that such properties are accessed. If two Visuals meet the *hard* criteria and an equal amount of *soft* criteria, the Visual with the most color resolution is selected. If neither Visual has more color resolution than the other, then the first Visual in the list provided by the server is selected.

If you create a PEX Color Approximation entry (for example, by calling the PEXUtSelectVisual or PEXUtGetStandardColormapInfo utility), you can verify that the PEX server supports the Color Approximation entry you want by issuing the CGE PEX 5.1 utility PEXUtCheckColorApproximation. Based on information from the server, the utility verifies that the Color Approximation setup is supported by PEX on the specified Visual. If the specified Color Approximation is not supported, then the utility attempts to supply an alternative Color Approximation.

Drawable Creation

Some of the utilities create the drawable for you and return it to you as a parameter. Others give you a Visual and Colormap that you would use on your own call to XCreateWindow.

3

Utilities

In addition to the CGE PEX 5.1 utility extensions, vendors who support CGE PEX 5.1 fully support the existing PEX 5.1 utility functions. PEX 5.1 utilities apply to PEX 5.1 functions only. You need to invoke other CGE utilities in order to obtain information applicable to CGE PEX 5.1 functions. For example, PEXGetSizeOCs returns values that apply to PEX 5.1 output commands but does not return information about CGE PEX 5.1 OCs, such as the output command PEXExtEllipse. Extended functions are provided to support the CGE PEX 5.1 extensions as described in the following table:

Table 3-1 Extended Functions for CGE PEX 5.1

PEX 5.1 Function	CGE PEX 5.1
PEXCountOCs	PEXExtCountOCs
PEXDecodeOCs	PEXExtDecodeOCs
PEXEncodeOCs	PEXExtEncodeOCs
PEXFreeOCData	PEXExtFreeOCData
PEXGetSizeOCs	PEXExtGetSizeOCs
PEXFetchElementsAndSend	PEXExtFetchElementsAndSend
PEXFreePCAttributes	PEXExtFreePCAttributes
PEXFreeRendererAttributes	PEXExtFreeRendererAttributes
PEXFreeTableEntries	PEXExtFreeTableEntries
None. Use bits to set value mask.	PEXExtRendererAttributeMask
None. Use bits to set value mask.	PEXExtRendererAttributeMaskAll
PEXSetPCAttributeMask	PEXExtSetPCAttributeMask
PEXSetPCAttributeMaskAll	PEXExtSetPCAttributeMaskAll

In addition to these utilities, CGE PEX 5.1 also provides Texture Mapping utilities:

Table 3-2 Texture Mapping Utilities for CGE PEX 5.1

PEX 5.1 Function	CGE PEX 5.1
None	PEXExtCreateFilteredTM
None	PEXExtCreateFilteredTMFromWindow
None	PEXExtFreeFilteredTM
None	PEXExtTMCoordFillAreaSetWithData
None	PEXExtTMCoordQuadrilateralMesh
None	PEXExtTMCoordSetOfFillAreaSets
None	PEXExtTMCoordTriangleStrip

Using Features of the CGE PEX 5.1 Extensions 4

Required CGE Features

Once you know you have a CGE server, you can use the features that are a required part of CGE without querying them. See "Minimum Conformance" on page 75.

Optional CGE Features

Some CGE features are still optional. You'll need to invoke the PEXGetEnumTypeInfo request to find out what is supported with your particular implementation.

Working with the Extended Renderer Attributes

CGE PEX 5.1 defines a number of new renderer attributes:

- `TMBindingTable`
- `TMCoordSourceTable`
- `TMCompositionTable`
- `TMSamplingTable`

You cannot access these new attributes with the standard PEXlib 5.1 functions because PEXlib 5.1 is not extensible enough to allow these functions to access vendor-extended attributes. Instead, CGE PEX 5.1 supplies you with two new functions to manipulate these attributes. You should use the PEXExtSetRendererAttributeMask or the PEXExtSetRendererAttributeMaskAll macros to set up bitmasks for these new functions.

PEXExtChangeRenderer()

Use this function to alter both the standard and CGE PEX 5.1 extended renderer attributes. The interface is slightly different from the standard 5.1 function in that you must supply an array of two long words that contain the bitmask indicating which attributes are supplied. The longer bitmask is required to accommodate the additional renderer attributes. In a sense, this interface is similar to the PC functions in PEX 5.1.

You cannot create a renderer with any of the new attributes specified, because CGE PEX 5.1 does not supply a new function to create a renderer. You must first create a renderer with the standard PEX 5.1 function, and then use PEXExtChangeRenderer to modify the extended attributes.

PEXExtGetRendererAttributes()

Use this function to get both the standard and CGE PEX 5.1 extended renderer attributes. This function also has a longer bitmask to accommodate the extra attributes as described for the PEXExtChangeRenderer function.

Working with the Extended Pipeline Context

CGE PEX 5.1 defines a number of new pipeline context attributes:

- `TMPerspectiveCorrection`
- `TMResourceHints`
- `TMSampleFrequency`
- `ActiveTextures`
- `BFActiveTextures`
- `PrimitiveAA`
- `LineCapStyle`
- `LineJoinStyle`

You cannot access these new attributes with the standard PEXlib 5.1 functions because PEXlib 5.1 is not extensible enough to allow these functions to access vendor-extended attributes. Instead, CGE PEX 5.1 supplies you with two new functions to manipulate these attributes. You should use the PEXExtSetPCAttributeMask or the PEXExtSetPCAttributeMaskAll macros to set up bitmasks for these new functions.

The PEXlib 5.1 PEXCopyPipelineContext function will only copy the standard PEX 5.1 PC attributes. There is no extended form of PEXCopyPipelineContext.

PEXExtChangePipelineContext()

Use this function to alter both the standard and CGE PEX 5.1 extended pipeline context Attributes. The interface is the same as in the PEXlib 5.1 function, except that you can specify the extra bits and attributes.

You cannot create a pipeline context with any of the new attributes specified, because CGE PEX 5.1 does not supply a new function to create a pipeline context. You must first create a pipeline context with the standard function, and then use PEXExtChangePipelineContext to modify the extended attributes.

PEXExtGetPipelineContext()

Use this function to get both the standard and CGE PEX 5.1 extended pipeline context Attributes. This function works the same way as the standard function, except that you can specify the extended bits in the bitmask.

Working with the New Lookup Tables

CGE PEX 5.1 defines a number of new lookup tables (LUTs):

- `TMBinding`
- `TMCoordSource`
- `TMComposition`
- `TMSampling`

You can use all standard PEXlib 5.1 functions to manipulate the new LUTs except for the functions that get and set table entries. Use the following functions instead:

- `PEXExtGetTableEntry`
- `PEXExtGetTableEntries`
- `PEXExtSetTableEntries`

These functions have the same syntax as the standard PEXlib 5.1 functions. However, you must still use the new functions to get or set entries in the new LUTs because only the new functions understand the format of the new table entries. You may also use these functions to get or set entries in the PEX 5.1 standard LUTs.

Also, there are no predefined entries for the new LUTs, therefore the function PEXGetPredefinedEntries is not useful for the new LUTs.

Renderer Dynamics

As with some PEX 5.1 servers, some CGE PEX 5.1 servers cannot apply LUT, nameset, or renderer attribute changes immediately after you change them with a PEXlib function if the renderer is in the rendering state. The changes do not actually take affect until the next BeginRendering call. Such renderers are called non-dynamic. Non-dynamic renderers may force the application to call BeginRendering and EndRendering more often than is needed for a dynamic renderer. This can lead to different program logic for each type of renderer.

For best performance, consider the issue of renderer dynamics carefully and consider using extra code paths, based on the renderer dynamics. Be careful not to program for a non-dynamic renderer only, because you could lose performance if the same code is run on a dynamic renderer.

Also, the new CGE PEX 5.1 extended renderer attributes are all considered to be dynamic. There are no new functions to query the renderer dynamics for these new attributes.

Programming Example 5

The Sample Program

This chapter contains the program listings of a simple CGE PEX 5.1 program that displays a rotating torus, with or without texture mapping. This short program shows you how to determine if CGE PEX 5.1 is present and how you can avoid queries, once you know CGE PEX 5.1 is there. It also shows you how to apply texture mapping to an object with a minimum amount of work. All CGE vendors ship the following files with their CGE PEX 5.1 products which provides you with the opportunity to experiment with the files:

- `ptorus.h` - header file for entire program. It contains definitions of common data structures and function prototypes. (See "ptorus.h - Header File for Main Program" on page 30.)
- `ptorus.c` - main program. Be sure to look at this file. (See "ptorus.c - Main Program" on page 33.)
- `view.c` - code for handling viewing calculations and setting the view table entries. Not extremely important for illustrating CGE. (See "view.c - View Computations Example" on page 45.)
- `model.c` - code for creating the model. It generates a torus and stores it into a structure for later display. A compile-time definition controls whether or not you want the PEX server to apply a texture to the torus. The code in this file also sets up the pipeline context and performs other model-specific initialization. (See "model.c - Model and Scene Generation Example" on page 50.)

ptorus.h - *Header File for Main Program*

Code Example 5-1 Header File for Main Program Example

```
*/
        CGE PEX 5.1 Simple Programming Example
        Header file
        Define common function prototypes and data structures
*/

/*
    This structure is used to describe a view using a
    camera field-of-view model.
*/

typedef struct
{
    float refx,refy,refz;
    float camx,camy,camz;
    float upx,upy,upz;
    float field_of_view;
    float front,back;
    int perspective;
} camera_arg;

/*
    This procedure sets the view table entry to
    the view described by the camera argument.
*/

extern void set_view_camera(
#if NeedFunctionPrototypes
        Display                 *dpy,
        Window                  window,
        camera_arg              *ca,
        PEXRenderer             renderer,
        PEXRendererAttributes   *rend_attr,
        int                     index
#endif
);

/*
    This procedure generates the geometric model with appropriate
    surface attributes.
```

Code Example 5-1 Header File for Main Program Example (Continued)

```
*/

extern void Torus(
#if NeedFunctionPrototypes
        Display                  *dpy,
        XID                      resource_id,
        PEXOCRequestType         req_type
#endif
);

/*
    This procedure creates a structure containing the geometric
    model.
*/

extern PEXStructure create_model(
#if NeedFunctionPrototypes
        Display                  *dpy
#endif
);

/*
    This procedure sets up viewing and lighting.
*/

extern void create_scene(
#if NeedFunctionPrototypes
        Display                  *dpy,
        camera_arg               *camera,
        int                      *num_lights,
        PEXLightEntry            **lights,
        PEXTableIndex            **lss
#endif
);

/*
    This procedure sets up initial attributes in the Pipeline
    Context.
*/

extern void create_initial_state(
#if NeedFunctionPrototypes
        Display                  *dpy,
        PEXPipelineContext       plc,
```

Code Example 5-1 Header File for Main Program Example (Continued)

```
        int                      num_lights,
        PEXTableIndex            *lss
#endif
);

/*
    This procedure sets up the Renderer and creates the useful
    associated resources (Lookup Tables and Pipeline Context).
*/

extern PEXRenderer create_renderer(
#if NeedFunctionPrototypes
        Display                  *dpy,
        Window                   window,
        PEXRendererAttributes    *rend_attrs
#endif
);
```

ptorus.c - *Main Program*

Code Example 5-2 Main Program Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        Main program file
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>

#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Xutil.h>
#include <X11/extensions/multibuf.h>

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

#include <PEXUtCmap.h>

#include "ptorus.h"

/*
    Forward declarations of utility procedures in this file.
*/

static void redraw(
#if NeedFunctionPrototypes
        Display *dpy,
        Multibuffer *drawable,
        PEXRenderer renderer,
        PEXStructure structure,
        Multibuffer *buffers
#endif
);

static void p_animate(
#if NeedFunctionPrototypes
```

Code Example 5-2 Main Program Example (Continued)

```
        Display *dpy,
        PEXRenderer rid,
        PEXStructure sid
#endif
);

/*
    This data structure describes the enumerated type values this
    program requires. They are checked in the main program below.
    Note that the only check we really need to make is to look for
    the CGE Escape Identifiers. If they are present, we can be
    sure that we have a CGE server and can assume that a large
    number of features are supported.
    This program requires no features beyond what CGE PEX 5.1
    requires, so we just need to check for CGE support.
*/

static struct enum_entry {
    unsigned int        enum_type;
    unsigned int        enum_value;
    char                *message;
} enums_to_check[]={

    { PEXETEscape,             PEXExtEscapeChangePipelineContext,
        "CGE PEX 5.1 Support" },

};
#define NUM_ENUMS_TO_CHECK (sizeof(enums_to_check)/sizeof(struct
                                     enum_entry))

/*
    The main program. Takes one possible command-line argument to
    name the X server, -display <display_name>.
*/

main (argc, argv)
        int argc;
        char **argv;
{
    char                        *prog_name;
    char                        *display_name;

    int                         screen;
```

Code Example 5-2 Main Program Example (Continued)

```
    Display                      *dpy;
    Window                       window;
    Multibuffer                  renderBuf;
    Multibuffer                  buffers[2];
    int                          mbx_buf_count;

    PEXExtensionInfo             *ext_info;
    char                        err_string[PEXErrorStringLength];

    XVisualInfo                  vis_info;
    XStandardColormap            cmap_info;
    PEXColorApproxEntry          capx_info;

    XSizeHints                   hints;
    PEXUtVisualCriteria          criteria;
    PEXUtWindowSpecification     window_info;
    int                          crit_index;
    unsigned int                 unmet;
    Atom                         prop_atom;
    int                          result;
    XColor                       returned_background;
    XEvent                       event;
    Colormap                     cmap_id;

    Atom                         xa_WM_DELETE_WINDOW;
    Atom                         xa_MOTIF_WM_MESSAGES;
    Atom                         xa_WM_PROTOCOLS;
    Atom                         protocols[2];
    char                         *title;

    PEXRendererAttributes        rend_attrs;
    PEXRenderer                  renderer;
    PEXStructure                 structure;

    int                          num_lights;
    PEXLightEntry                *lights;
    PEXTableIndex                *lss;

    int                          visible;
    camera_arg                   camera;

    /*
        Get the program name for error messages.
    */
```

Code Example 5-2 Main Program Example (Continued)

```
    if (prog_name = strrchr(argv[0], '/'))
        prog_name++;
    else
        prog_name = argv[0];

    /*
        Parse the command line arguments.
    */
    if ((argc > 2)
            && argv[1]
            && !strcmp( "-display", argv[1] ))
        display_name = XDisplayName( argv[2] );
    else
        display_name = XDisplayName( NULL );

    /*
        Open the X connection.
    */
    if ( !(dpy = XOpenDisplay( display_name )) )
    {
       fprintf(stderr, "%s: can't open display %s -- exiting\n",
                       prog_name, display_name);
        exit (1);
    }

    screen = DefaultScreen(dpy);

    /*
        Initialize PEXlib on the connection.
    */
    if (PEXInitialize (dpy, &ext_info, PEXErrorStringLength,
                        err_string))
    {
       fprintf (stderr, "%s: PEXInitialize() failed on %s: %s\n",
                prog_name, display_name, err_string);
        exit (1);
    }

    /*
        Select a Visual for PEX rendering and create a Window
        on the Visual.
        We're going to hint that we want Double Buffering for
        smooth animation and to use a Standard Colormap to
        promote the sharing of Colormap resources on the server.
```

Code Example 5-2 Main Program Example (Continued)

```
         If we don’t get either or both, we’ll stumble on.
     */

     hints.x = 128;
     hints.y = 128;
     hints.width = 256;
     hints.height = 256;
     hints.flags = (USSize|USPosition);

     criteria.hard_criteria_mask = 0;
     criteria.soft_criteria_mask = PEXUtDBCapability |
                                   PEXUtStandardColormapProperty;

     criteria.double_buffering_capability = PEXUtDBPEX;
     criteria.standard_colormap_property = True;

     window_info.attr_mask = CWEventMask;
     window_info.attrs.event_mask = ( ExposureMask        |
                                     VisibilityChangeMask |
                                     StructureNotifyMask);

     window_info.title = “PEX Torus”;
     window_info.size_hints = hints;
     window_info.parent = RootWindow (dpy, screen);
     window_info.border_width = 0;
     window_info.background_color_name = “black”;
     window_info.border_color_name = “white”;

     result = PEXUtMakeWindowAndColormap (dpy, screen,
                                         1, &criteria,
                                        &window_info, &window,
                                        &vis_info, &cmap_info,
                                         &capx_info,
                                         &crit_index, &unmet,
                                         &prop_atom,
                                         &returned_background,
                                         &cmap_id);

     if (result != PEXUtSuccess)
     {
         static char *PEXUtFailStrings[] =
         {
             “PEXUtBadAlloc”,              /* -4 maps to 0 */
             “PEXUtPEXFailure”,            /* -3 maps to 1 */
```

Code Example 5-2 Main Program Example (Continued)

```
            "PEXUtXFailure",                  /* -2 maps to 2 */
            "PEXUtCriteriaFailure",           /* -1 maps to 3 */
            "PEXUtSuccess",                   /*  0 maps to 4 */
            "PEXUtQualifiedSuccess",          /* +1 maps to 5 */
            "PEXUtAlternativeSuccess",        /* +2 maps to 6 */
        };

        fprintf(stderr,"%s: visual selection or window creation
                 failed: %s\n", prog_name, PEXUtFailStrings
                 [ result + 4 ]);
        exit(1);
    }

    /*
       Report soft criteria failures, but don't quit.
    */
    if (unmet & PEXUtDBCapability)
        fprintf(stderr,
              "%s: could not find Visual for double buffering\n",
                 prog_name);
    if (unmet & PEXUtStandardColormapProperty)
        fprintf(stderr,
               "%s: could not find Standard Colormap Property\n",
                prog_name);

    /*
        Set up some window manager properties (see the ICCCM).
    */

    xa_WM_DELETE_WINDOW   = XInternAtom(dpy, "WM_DELETE_WINDOW",
                                            False);
    xa_MOTIF_WM_MESSAGES = XInternAtom(dpy, "_MOTIF_WM_MESSAGES",
                                            False);
    xa_WM_PROTOCOLS      = XInternAtom(dpy, "WM_PROTOCOLS", False);
    protocols[0] = xa_WM_DELETE_WINDOW ;
    protocols[1] = xa_MOTIF_WM_MESSAGES;
    XSetWMProtocols(dpy, window, protocols, 2);

    title = "PEX Torus";
    XChangeProperty(dpy, window, XA_WM_ICON_NAME, XA_STRING,
                 8, PropModeReplace, (unsigned char *) title, 8);

    /*
        Configure the window for double-buffering via MBX.
```

Code Example 5-2 Main Program Example (Continued)

```
    */

    mbx_buf_count = XmbufCreateBuffers( dpy, window, 2,
                              MultibufferUpdateActionBackground,
                              MultibufferUpdateHintFrequent,
                              buffers );
    if ( mbx_buf_count == 0 ) {
        fprintf(stderr,
            "%s: can't create MBX buffers on display %s, visual
             %x\n",
                prog_name, display_name, vis_info.visualid);
        exit(1);
    }

    /*
        Handle case where only one MBX buffer was created.
        This will make the swap work, even if one buffer.
    */
    if ( mbx_buf_count == 1 )
      buffers[1] = buffers[0];

      renderBuf = buffers[1];


    /*
       Check the Enum Types to be sure that CGE PEX is supported
    */
    {
        int status, i, j, found;
        int enum_types[NUM_ENUMS_TO_CHECK];
        unsigned long *enum_counts;
        PEXEnumTypeDesc *enum_values, *p_enum;

        for (i=0; i<NUM_ENUMS_TO_CHECK; i++)
            enum_types[i] = enums_to_check[i].enum_type;

        status = PEXGetEnumTypeInfo (dpy, renderBuf,
                          NUM_ENUMS_TO_CHECK, enum_types,
                          PEXETCounts|PEXETIndex,
                          &enum_counts, &enum_values);
        if (!status) {
            fprintf(stderr,
            "%s: inquiry of enumerated types failed - exiting\n",
                prog_name);
```

Code Example 5-2 Main Program Example (Continued)

```
            exit(1);
        }

        p_enum = enum_values;
        for (i=0; i<NUM_ENUMS_TO_CHECK; i++) {

            found = False;

            for (j=0; j<enum_counts[i]; j++) {
                if (((unsigned short) (p_enum++)->index) ==
                ((unsigned short) enums_to_check[i].enum_value))
{
                    found = True;
                }
            }

            if (!found) {
                fprintf(stderr,
                   "%s: this program requires support for %s\n",
                    prog_name, enums_to_check[i].message);
                exit(1);
            }
        }

        PEXFreeEnumInfo (NUM_ENUMS_TO_CHECK, enum_counts,
                         enum_values);
    }

    /*
        Create the model and scene to be animated.
    */

    structure = create_model(dpy);
    create_scene(dpy, &camera, &num_lights, &lights, &lss);

    /*
        Create a Renderer and various lookup tables.
        Set up the view, lights, and Pipeline Context.
    */

    renderer = create_renderer(dpy, window, &rend_attrs);

    set_view_camera(dpy, window, &camera, renderer, &rend_attrs,
                      1);
```

Code Example 5-2 Main Program Example (Continued)

```
    PEXSetTableEntries(dpy, rend_attrs.light_table, lss[0],
                        num_lights, PEXLUTLight, lights);
    PEXSetTableEntries(dpy, rend_attrs.color_approx_table, 0, 1,
                        PEXLUTColorApprox, &capx_info);
    create_initial_state (dpy, rend_attrs.pipeline_context,
                           num_lights, lss);

    /*
        Now enter the main event-processing and animation loop.
    */

    visible = False;
    while (True)
    {

        /*
            First handle any pending events.
        */

        while (XPending(dpy))
        {
            XNextEvent( dpy, &event );

            switch ( event.type ) {
            case ConfigureNotify:
                /*
                    Adapt the view to the window configuration
                    and render the image.
                */
                set_view_camera(dpy, window, &camera, renderer,
                                  &rend_attrs, 1);
                redraw(dpy, &renderBuf, renderer, structure,
                         buffers);
                break;

            case Expose:
                /*
                    Flush remaining Expose events.
                */
                while ( XCheckTypedWindowEvent( dpy, window,
                        Expose, &event ));
                /* empty statement */
                visible = True;
```

Code Example 5-2 Main Program Example (Continued)

```
                /*
                    Adapt the view to the window configuration
                    and render the image.
                */
                set_view_camera(dpy, window, &camera, renderer,
                                  &rend_attrs, 1);
                redraw(dpy, &renderBuf, renderer, structure,
                         buffers);
                break;

            case ClientMessage:
              if (event.xclient.message_type == xa_WM_PROTOCOLS)
                {
                    if (event.xclient.data.l[0] ==
                         xa_WM_DELETE_WINDOW)
                    {
                        /*
                           Terminate the double-buffering on the
                            window.
                            Close the Window and the connection.
                        */

                        XmbufDestroyBuffers( dpy, window );
                        XDestroyWindow (dpy, window);
                        XCloseDisplay( dpy );
                        exit(0);
                    }
                }
                break;
            }
        } /* event handling */

        /*
            After handling events, render the next frame in the
            animation.
        */

        if (visible)
          redraw(dpy, &renderBuf, renderer, structure, buffers);

    } /* main loop */

} /* main */
```

Code Example 5-2 Main Program Example (Continued)

```
/*
    This procedure draws one frame and swaps the buffers.
*/
static void redraw(dpy, drawable, renderer, structure, buffers)
        Display *dpy;
        Multibuffer *drawable;
        PEXRenderer renderer;
        PEXStructure structure;
        Multibuffer *buffers;
{
    PEXBeginRendering(dpy, (Drawable) *drawable, renderer);
    p_animate(dpy, renderer, structure);
    PEXEndRendering(dpy, renderer, True);

    /* Swap the buffers. */
    XmbufDisplayBuffers( dpy, 1, drawable, 0, 0 );
    *drawable = ((*drawable == buffers[0]) ? buffers[1] :
buffers[0]);

    XSync(dpy, False);

} /* redraw */


/*
    This procedure generates the animation and does the rendering
    of a single frame.
*/

static float xangle = 0.01;
static float yangle = 0.1;

static void p_animate(dpy, rid, sid)
        Display *dpy;
        PEXRenderer rid;
        PEXStructure sid;
{
    PEXMatrix model_mtx, xrot_mtx, yrot_mtx;

    xangle += 0.01;
    if (xangle > 2*M_PI) xangle = 0.01;
    PEXRotate(PEXXAxis, xangle, xrot_mtx);

    yangle += 0.1;
```

Code Example 5-2 Main Program Example (Continued)

```
    if (yangle > 2*M_PI) yangle = 0.1;
    PEXRotate(PEXYAxis, yangle, yrot_mtx);

    PEXMatrixMult (xrot_mtx, yrot_mtx, model_mtx);

    PEXSetLocalTransform(dpy, rid, PEXOCRender, PEXReplace,
                              model_mtx);
    PEXExecuteStructure(dpy, rid, PEXOCRender, sid);

} /* p_animate */
```

view.c - *View Computations Example*

Code Example 5-3 View Computations Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        View Computations

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include <X11/PEX5/PEXlib.h>

#include "ptorus.h"

/*
Linear algebra utilities.
*/

void vector_subtract(c, a, b)
float *c;
float *a;
float *b;
{
    *c++ = *a++ - *b++;
    *c++ = *a++ - *b++;
    *c++ = *a++ - *b++;
}

double vector_length(v)
    register float *v;
{
    register int i;
    register double s;

    for (i = 0, s = 0.0; i < 3; i++, v++)
        s += *v * *v;
    s = sqrt(s);
    return(s);
}
```

Code Example 5-3 View Computations Example (Continued)

```
void vector_scale(v, s)
    register float *v, s;
{
    *v++ *= s;
    *v++ *= s;
    *v++ *= s;
}


/*
    This procedure creates a view table entry and installs it.
*/

void set_view_camera(dpy, window, ca, renderer, rend_attr, index)
    Display                     *dpy;
    Window                      window;
    camera_arg                  *ca;
    PEXRenderer                 renderer;
    PEXRendererAttributes       *rend_attr;
    int                         index;
{
    double viewDistance;

    PEXCoord            viewReferencePoint;
    PEXCoord            projectionReferencePoint;
    PEXVector           viewUpVector, viewPlaneNormal;
    PEXCoord2D          viewWindow[2];
    double              viewPlane, frontPlane, backPlane;
    PEXNPCSubVolume     viewPort;
    PEXViewEntry        view;
    int                 err, perspective;
    double              windowScale;
    double              winAspect;
    int                 winWidth, winHeight;

    XWindowAttributes   win_attrs;

    /*
     * Compute the ViewOrientationMatrix from viewReferencePoint,
    * viewPlaneNormal and viewUpVector.  This stuff is independent
     * of the resize method.
     */
```

Code Example 5-3 View Computations Example (Continued)

```
    viewReferencePoint.x = ca->refx;
    viewReferencePoint.y = ca->refy;
    viewReferencePoint.z = ca->refz;

    vector_subtract((float *) &viewPlaneNormal, &(ca->camx),
                    &(ca->refx));
    viewDistance = vector_length(&viewPlaneNormal);
    vector_scale(&viewPlaneNormal, 1.0 / viewDistance);

    viewPlane  = 0;
    backPlane  = -10000;
    frontPlane = viewDistance - 1;

    viewUpVector.x = ca->upx;
    viewUpVector.y = ca->upy;
    viewUpVector.z = ca->upz;

    err = PEXViewOrientationMatrix(&viewReferencePoint,
                                   &viewPlaneNormal,
                                   &viewUpVector,
                                   view.orientation);

    /*
       Compute the size (and shape) of the viewWindow based upon
       the camera's field of view and the aspect ratio of the
       window to which we're about to render.
    */

    XGetWindowAttributes (dpy, window, &win_attrs);

    windowScale = atan(ca->field_of_view * M_PI / 360.0)
                    * (viewDistance - viewPlane);
    winWidth    = (win_attrs.width  > 1) ? (win_attrs.width  - 1)
                : 1;
    winHeight   = (win_attrs.height > 1) ? (win_attrs.height - 1)
                : 1;
    winAspect   = (double) winWidth / (double) winHeight;

    viewPort.min.x = 0;    viewPort.max.x = 1;
    viewPort.min.y = 0;    viewPort.max.y = 1;
    viewPort.min.z = 0;    viewPort.max.z = 1;
    rend_attr->npc_subvolume = viewPort;

    viewWindow[0].y = -windowScale;
```

Code Example 5-3 View Computations Example (Continued)

```
    viewWindow[1].y =  windowScale;
    viewWindow[0].x = -windowScale;
    viewWindow[1].x =  windowScale;

    view.clip_flags = PEXClippingAll;
    perspective = True;
    projectionReferencePoint.x = 0;
    projectionReferencePoint.y = 0;
    projectionReferencePoint.z = viewDistance;

    /*
     * Compute the aspect ratio of the viewPort so we fill the
     * window. From that the ViewMappingMatrix is computed.
     */

    /*
     * This method varies only the renderer's npc_subvolume to
    * map the 0-1 range to the major dimension of the destination
     * drawable. The minor dimension is reduced by the aspect
     * ratio of the drawable and then centered.
     */

    if (winWidth > winHeight)
    {
        double aspect = (double) winHeight / (double) winWidth;

        rend_attr->npc_subvolume.min.y = (1 - aspect) / 2;
        rend_attr->npc_subvolume.max.y =
                        rend_attr->npc_subvolume.min.y + aspect;
    }
    else if (winWidth < winHeight)
    {
        rend_attr->npc_subvolume.min.x = (1 - winAspect) / 2;
        rend_attr->npc_subvolume.max.x =
                                rend_attr->npc_subvolume.min.x +
                                winAspect;
    }

    view.clip_limits = viewPort;
    PEXChangeRenderer(dpy, renderer, PEXRANPCSubVolume,
                        rend_attr);

    err = PEXViewMappingMatrix(viewWindow, &viewPort,
                                  perspective,
```

Code Example 5-3 View Computations Example (Continued)

```
                                    &projectionReferencePoint,
                                    viewPlane, backPlane, frontPlane,
                                    view.mapping);

    PEXSetTableEntries(dpy, rend_attr->view_table, index, 1,
                             PEXLUTView,
                             (PEXPointer) &view);

} /* set_view_camera */
```

model.c - *Model and Scene Generation Example*

Code Example 5-4 Model and Scene Generation Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        Model and scene generation

*/
#define TEXTURE                                /* Remove this line to
                                                  disable texture */
#include <stdio.h>
#include <math.h>

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

#include "ptorus.h"

/*
    This procedure normalizes a vector to a magnitude of 1.0.
*/

NormalizeVector( vector )
    PEXVector *vector;
{
    float r;

    /* Transform a vector to a unit vector. */
    r = vector->x * vector->x
        + vector->y * vector->y
        + vector->z * vector->z;

    if ( r > 0 ) {
      r = 1.0 / sqrt( r );
      vector->x *= r;
      vector->y *= r;
      vector->z *= r;
    }
} /* NormalizeVector */

/*
    This procedure generates the output commands that constitute
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
    the model and the closely-associated surface attributes.
*/

void Torus(dpy, resource_id, req_type)
Display *dpy;
XID resource_id;
PEXOCRequestType req_type;
{
  /*
   * Torus parameters
   */
    float xaxis                         = 10.0,
    yaxis                               = 10.0,
    zaxis                               = 10.0,
    radius                              = 4.0;
    PEXCoord center                     = {0,0,0};
    int depth                         = 4; /* quality control */

    /*
     * Misc variables
     */
    int i, j;                           /* i/phi    j/theta */
    float u, v;
    float phi, theta, step;
    float fphiSize, fthetaSize;
    int phiSize, thetaSize;
    int count;
#ifdef TEXTURE
    /* "Custom" vertex mapping with texture data */
    typedef struct _PEXExtVertexNormal {
      PEXCoord point;
      PEXVector normal;
      PEXCoord data;
    } PEXExtVertexNormal;
    PEXExtVertexNormal *torus;
    PEXExtArrayOfVertex quadmesh;
#else
    PEXVertexNormal *torus;
    PEXArrayOfVertex quadmesh;
#endif
    PEXReflectionAttributes ra;
    PEXColor color;
    PEXArrayOfFacetData facetdata;
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
    /*
     * Set up a green object.
     */
    color.rgb.red                          = 0.1;
    color.rgb.green                        = 0.9;
    color.rgb.blue                         = 0.1;
    ra.ambient                             = 0.2;
    ra.diffuse                             = 1;
    ra.specular                            = 1;
    ra.specular_conc                       = 15;
    ra.transmission                        = 0;
    ra.specular_color.type                 = PEXColorTypeRGB;
    ra.specular_color.value.rgb.red        = 0.0402;
    ra.specular_color.value.rgb.green      = 0.4012;
    ra.specular_color.value.rgb.blue       = 0.0402;
    PEXSetSurfaceColor(dpy, resource_id, req_type,
                        PEXColorTypeRGB, &color);
    PEXSetReflectionAttributes(dpy, resource_id, req_type, &ra);
#ifdef TEXTURE
    PEXSetInteriorStyle(dpy, resource_id, req_type,
PEXExtInteriorStyleTexture);
#endif

    /*
     * Compute step size and allocate space for verticies.
     */
    step = (float)(M_PI / pow( (double)2, (double)depth ));
    fphiSize = (2 * M_PI / step) + 1.;
    fthetaSize = (2 * M_PI / step) + 1.;
    phiSize = (int)fphiSize;
    thetaSize = (int)fthetaSize;
#ifdef TEXTURE
    torus = (PEXExtVertexNormal *) malloc (phiSize * thetaSize *
                                    sizeof(PEXExtVertexNormal));
#else
    torus = (PEXVertexNormal *) malloc (phiSize * thetaSize *
                                    sizeof(PEXVertexNormal));
#endif

    /*
     * Sweep from -pi to pi along both u and v to generate
       verticies.
     */
    count = 0;
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
    u = -M_PI - step;
    for( i = 0; i < phiSize; i++ ) {
      u += step;
      v = -M_PI - step;
      for( j = 0; j < thetaSize; j++, count++ ) {
        v +=  step;
        /*
         * Compute vertex coordinate and normal
         */
        torus[count].point.x = xaxis *
          (radius + cos(v)) * cos(u) + center.x;
        torus[count].point.y = yaxis *
          (radius + cos(v)) * sin(u) + center.y;
        torus[count].point.z = zaxis * sin(v) + center.z;
        torus[count].normal.x = (1 / xaxis) * cos(v) * cos(u);
        torus[count].normal.y = (1 / yaxis) * cos(v) * sin(u);
        torus[count].normal.z = (1 / zaxis) * sin(v);
        NormalizeVector (&(torus[count].normal));
#ifdef TEXTURE
        torus[count].data.x = (u + M_PI) / (2 * M_PI);
        torus[count].data.y = (v + M_PI) / (2 * M_PI);
        torus[count].data.z = 0.0;
#endif
        }
    }

#ifdef TEXTURE
    quadmesh.with_fp_data = (PEXPointer)torus;
    PEXExtQuadrilateralMesh (dpy,
                             resource_id,
                             req_type,
                             PEXShapeUnknown,
                             3,
                             PEXGANone,
                             PEXGANormal | PEXExtGAData,
                             PEXColorTypeIndexed,
                             facetdata,
                             thetaSize,
                             phiSize,
                             quadmesh);
#else
    quadmesh.normal = (PEXVertexNormal *)torus;
    PEXQuadrilateralMesh (dpy,
                          resource_id,
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
                                req_type,
                                PEXShapeUnknown,
                                PEXGANone,
                                PEXGANormal,
                                PEXColorTypeIndexed,
                                facetdata,
                                thetaSize,
                                phiSize,
                                quadmesh);
#endif
    free(torus);
} /* Torus */

/*
     This procedure creates a structure containing the geometric
     model.
*/

PEXStructure create_model(dpy)
        Display          *dpy;
{
        PEXStructure str_return;

        /*
            Create and load the structure containing the model.
        */
        str_return = PEXCreateStructure(dpy);
        Torus (dpy, str_return, PEXOCStore);

        return str_return;

} /* create_model */

/*
    This procedure creates the light table entries and activation
    list for the scene.
*/

#define NUM_LIGHTS 5
PEXLightEntry ltbl[NUM_LIGHTS];
PEXTableIndex lndx[NUM_LIGHTS];

pex_cube_lights(dpy, num_lights, lights, lss)
        Display *dpy;
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
         int *num_lights;
         PEXLightEntry **lights;
         PEXTableIndex **lss;
{
         ltbl[0].type = PEXLightAmbient;
         ltbl[0].color.type = PEXColorTypeRGB;
         ltbl[0].color.value.rgb.red   = 0.1;
         ltbl[0].color.value.rgb.green = 0.1;
         ltbl[0].color.value.rgb.blue  = 0.1;
         lndx[0] = 1;

         ltbl[1].type = PEXLightWCPoint;
         ltbl[1].color.type = PEXColorTypeRGB;
         ltbl[1].color.value.rgb.red   = 0.7653;
         ltbl[1].color.value.rgb.green = 0.763499;
         ltbl[1].color.value.rgb.blue  = 0.7651;
         ltbl[1].point.x = -19.0;
         ltbl[1].point.y = 0.0;
         ltbl[1].point.z = 200.0;
         ltbl[1].attenuation1 = 1;
         ltbl[1].attenuation2 = 0;
         lndx[1] = 2;

         ltbl[2] = ltbl[1];
         ltbl[2].point.x = 17.0;
         ltbl[2].point.y = 200.0;
         ltbl[2].point.z = -41.0;
         lndx[2] = 3;

         ltbl[3] = ltbl[1];
         ltbl[3].point.x = 175.0;
         ltbl[3].point.y = -83.0;
         ltbl[3].point.z = -51.0;
         lndx[3] = 4;

         ltbl[4] = ltbl[1];
         ltbl[4].point.x = -181.0;
         ltbl[4].point.y = -83.0;
         ltbl[4].point.z = 0.0;
         lndx[4] = 5;

         *num_lights = NUM_LIGHTS;
         *lights = ltbl;
         *lss = lndx;
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
} /* pex_cube_lights */

/*
    This procedure creates the viewing and lighting for the scene.
*/

void create_scene(dpy, camera, num_lights, lights, lss)
        Display *dpy;
        camera_arg *camera;
        int *num_lights;
        PEXLightEntry **lights;
        PEXTableIndex **lss;
{
        /*
            Load the camera describing the view.
        */
        camera->refx = 0.0;
        camera->refy = 0.0;
        camera->refz = 496.0;
        camera->camx = 0.0;
        camera->camy = 0.0;
        camera->camz = 497.0;
        camera->upx = 0.0;
        camera->upy = 1.0;
        camera->upz = 0.0;
        camera->field_of_view = 18.4;
        camera->front = 0.0;
        camera->back = 9999.0;
        camera->perspective = True;

        /*
            Load the light table entries.
        */
        pex_cube_lights(dpy, num_lights, lights, lss);

} /* create_scene */

/*
    This procedure sets up a Pipeline Context with the initial
    attributes required for the scene and model rendering.
*/

void create_initial_state (dpy, plc, num_lights, lss)
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
        Display *dpy;
        PEXPipelineContext plc;
        int num_lights;
        PEXTableIndex *lss;
{
    PEXPCAttributes pc_attrs;
    unsigned long attr_mask[3];

    attr_mask[0] = 0;
    attr_mask[1] = 0;
    attr_mask[2] = 0;

    PEXSetPCAttributeMask(attr_mask, PEXPCCullingMode);
    pc_attrs.culling_mode = PEXBackFaces;

    PEXSetPCAttributeMask(attr_mask, PEXPCDistinguishFlag);
    pc_attrs.distinguish = False;

    PEXSetPCAttributeMask(attr_mask, PEXPCSurfaceEdgeFlag);
    pc_attrs.surface_edges = PEXOff;

    PEXSetPCAttributeMask(attr_mask, PEXPCInteriorStyle);
    pc_attrs.interior_style = PEXInteriorStyleSolid;

    PEXSetPCAttributeMask(attr_mask, PEXPCReflectionModel);
    pc_attrs.reflection_model = PEXReflectionSpecular;

    PEXSetPCAttributeMask(attr_mask, PEXPCSurfaceInterp);
    pc_attrs.surface_interp = PEXSurfaceInterpColor;

    PEXSetPCAttributeMask(attr_mask, PEXPCViewIndex);
    pc_attrs.view_index = 1;

    PEXSetPCAttributeMask(attr_mask, PEXPCLightState);
    pc_attrs.light_state.count = num_lights;
    pc_attrs.light_state.indices = lss;

    PEXChangePipelineContext (dpy, plc, attr_mask, &pc_attrs);

} /* create_initial_state */

/*
    This procedure sets up the Renderer and creates the useful
    associated resources (Lookup Tables and Pipeline Context).
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
*/

PEXRenderer create_renderer(dpy, window, rend_attrs)
        Display                 *dpy;
        Window                  window;
        PEXRendererAttributes   *rend_attrs;
{
    unsigned long itemMask = 0;
    unsigned long attr_mask[3];
    PEXRenderer renderer;

    itemMask |= PEXRAPipelineContext;
    attr_mask[0] = 0;
    attr_mask[1] = 0;
    attr_mask[2] = 0;
    rend_attrs->pipeline_context = PEXCreatePipelineContext(dpy,
                attr_mask, NULL);

    itemMask |= PEXRAViewTable;
    rend_attrs->view_table = PEXCreateLookupTable(dpy, window,
                PEXLUTView);

    itemMask |= PEXRALightTable;
    rend_attrs->light_table = PEXCreateLookupTable(dpy, window,
                PEXLUTLight);

    itemMask |= PEXRAColorApproxTable;
    rend_attrs->color_approx_table = PEXCreateLookupTable(dpy,
                window, PEXLUTColorApprox);

    itemMask |= PEXRABackgroundColor;
    rend_attrs->background_color.type         = PEXColorTypeRGB;
    rend_attrs->background_color.value.rgb.red   = 0.5;
    rend_attrs->background_color.value.rgb.green = 0.5;
    rend_attrs->background_color.value.rgb.blue  = 0.5;

    itemMask |= PEXRAHLHSRMode;
    rend_attrs->hlhsr_mode = PEXHLHSRZBuffer;

    itemMask |= PEXRAClearImage;
    rend_attrs->clear_image = True;

    renderer = PEXCreateRenderer(dpy, window, itemMask,
                             rend_attrs);
```

Code Example 5-4 Model and Scene Generation Example (Continued)

```
    return renderer;

} /* create_renderer */
```

Texture Mapping in this Example

The preceding code example illustrates the technique and code for activating texture mapping with a minimum amount of effort and code. Here is an explanation of the changes made to `model.c` to enable texturing with the default texture:

```
#ifdef TEXTURE
    /* "Custom" vertex mapping with texture data */
    typedef struct _PEXExtVertexNormal {
      PEXCoord point;
      PEXVector normal;
      PEXCoord data;
    } PEXExtVertexNormal;
    PEXExtVertexNormal *torus;
    PEXExtArrayOfVertex quadmesh;
#else
    PEXVertexNormal *torus;
    PEXArrayOfVertex quadmesh;
#endif
```

This program creates a torus with a QuadMesh primitive. When not texturing, the only data needed for each vertex is the vertex normal to create a smooth shading effect and so the standard PEXVertexNormal data structure is sufficient. However, when texturing, the program also provides texture parameterization data. This data describes to the PEX server what part of the texture map is applied to the object at each vertex. The extra data in this case is a PEXCoord supplied with each vertex to specify the texture coordinate to be mapped to the object at each vertex.

Since there are a wide variety of texturing methods, there are also a variety of parameterization methods that require different amounts and types of extra data per vertex. Because of the large number of possibilities, PEXlib cannot supply Vertex data structures for all possibilities. Therefore, you must create your own to match the type of texturing you intend to use. Note in the code above the new PEXExtVertexNormal data structure that allows for one texture coordinate per vertex.

```
#ifdef TEXTURE
    PEXSetInteriorStyle(dpy, resource_id, req_type,
          PEXExtInteriorStyleTexture);
#endif
```

This function call is the key to turning on texturing. It tells the PEX server to apply a texture to the object instead of using a solid color (The interior style is set to PEXInteriorStyleSolid in the Pipeline Context). Note that instead of changing the interior style attribute with an OC, you could change it instead by initializing the Pipeline Context value to PEXExtInteriorStyleTexture.

```
#ifdef TEXTURE
    torus = (PEXExtVertexNormal *) malloc (phiSize * thetaSize *
          sizeof(PEXExtVertexNormal));
#else
    torus = (PEXVertexNormal *) malloc (phiSize * thetaSize *
          sizeof(PEXVertexNormal));
#endif
```

The code above simply allocates space for the torus data. Different code is needed to handle the differing data types since the data for each vertex is of different size. A PEXExtVertexNormal structure is bigger than a PEXVertexNormal by the size of the extra texture data.

```
#ifdef TEXTURE
        torus[count].data.x = (u + M_PI) / (2 * M_PI);
        torus[count].data.y = (v + M_PI) / (2 * M_PI);
        torus[count].data.z = 0.0;
#endif
```

The code above computes the texture coordinates and stores them in the extra data for each vertex. The texture coordinates are computed so that the entire default texture wraps around the torus exactly once. The texture coordinates range from 0.0 to 1.0.

Imagine cutting the torus once through one of its cylindrical cross sections, straightening the cut torus out into a cylinder, slicing the cylinder along one of its long sides, and then unrolling the result onto a flat surface. The parameterizations the program supplies above specify that the PEX server should apply the texture to this flat surface in one complete “patch”, since the texture coordinates at the edges of this flattened surface will be 0.0 along two adjacent edges and 1.0 along the opposite edges. This way, you will see the entire texture mapped exactly once around the torus, after it is conceptually rolled back up into a torus.

The code above will generate the required texture coordinate data, since u and v range from -PI to PI, resulting in texture values for x and y between 0.0 and 1.0.

```
#ifdef TEXTURE
    quadmesh.with_fp_data = (PEXPointer)torus;
    PEXExtQuadrilateralMesh (dpy,
                             resource_id,
                             req_type,
                             PEXShapeUnknown,
                             3,
                             PEXGANone,
                             PEXGANormal | PEXExtGAData,
                             PEXColorTypeIndexed,
                             facetdata,
                             thetaSize,
                             phiSize,
                             quadmesh);
#else
    quadmesh.normal = (PEXVertexNormal *)torus;
    PEXQuadrilateralMesh (dpy,
                          resource_id,
                          req_type,
                          PEXShapeUnknown,
                          PEXGANone,
                          PEXGANormal,
                          PEXColorTypeIndexed,
                          facetdata,
                          thetaSize,
```

```
                        phiSize,
                        quadmesh);
#endif
```

Finally, the program sends the QuadMesh primitive to the PEX server. The CGE version of the function call uses an extra parameter to specify how much texture data there is per vertex (the fifth parameter in the first function call above) and also turns on the flag that indicates the presence of texture data.

Texture Mapping: Applying Your Own Texture 6

The previous chapter illustrates a program that displays a simple object and shows how to texture the object using the default texture with just a few extra steps. The default texture, a checkerboard pattern, is not particularly interesting. You probably have your own textures that you would like to map onto an object.

The following code fragments illustrate how to create a texture from raw data that is compiled into a program. The data itself is too large to present here, but you should find this code useful for dealing with this sort of texture data.

This partial example defines three textures. The texture data is defined in Code Example 6-1 as follows:

Code Example 6-1 Texture Data Example

```
/*
       CGE PEX 5.1 Simple Programming Example with Texture Mapping

         Texture data

*/

PEXExtTexelRGBAlpha8
mandrill[128][128] = {
/* Row 0 */
0x96,0xAC,0xAC,0xFF,0xA0,0xAB,0xA0,0xFF,0x9F,0xA7,0x9D,0xFF,0x9
E,0xA9,0x9E,0xFF,
.... /* more data */
};
```

Code Example 6-1 Texture Data Example (Continued)

```
PEXExtTexelRGBAlpha8
landsat[128][128] = {
/* Row 0 */
0x11,0x22,0x74,0xFF,0x11,0x23,0x74,0xFF,0x3F,0x90,0xA7,0xFF,0x8
6,0xFF,0xF0,0xFF,
.... /* more data */
};

PEXExtTexelRGBAlpha8
stone[128][128] = {
/* Row 0 */
0xC9,0xDC,0xE7,0xFF,0xCF,0xE3,0xEF,0xFF,0xD4,0xE7,0xF3,0xFF,0xD
C,0xEF,0xFA,0xFF,
.... /* more data */
};
```

Most of the data has been omitted because of its size.

Code Example 6-2 illustrates how to set up the textures.

Code Example 6-2 Texture Preparation Example

```
/*
        CGE PEX 5.1 Simple Programming Example with Texture Mapping

          Texture preparation, LUT initialization, and clean up.

*/

#include <stdio.h>
#include <X11/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h>
#include "texture.h"                        /* texture data */

static PEXExtTextureMap         texture_IDs[3];
static PEXExtTMDescription      texture_desc_IDs[3];

/*
        This procedure initializes the textures and imports them
        into PEXlib.
*/
void init_textures(dpy,drawable)
    Display *dpy;
    Drawable drawable;
```

Code Example 6-2 Texture Preparation Example (Continued)

```
{
    static unsigned short names[] =     {
   PEXExtIDPowerOfTwoTMSizesRequired,PEXExtIDSquareTMRequired };
    PEXImpDepConstant                    *constants;
    PEXExtTexelArray                    base_array, *texel_array;
    PEXEnumTypeIndex                     domain;
    PEXExtTMDomainData                   domain_data;
    PEXEnumTypeIndex                     parameterization;
    PEXExtTMParameterizationData         param_data;
    PEXEnumTypeIndex                     tm_rendering_order;
    unsigned int                        power_of_two_tm_required;
    unsigned int                         square_tm_required;
    int                                  i, status;
    char                                 *map_name;

    /*
       Get texture mapping related implementation dependent
       constants
    */
    if (!PEXGetImpDepConstants(dpy, drawable, 2, names,
         &constants)) {
       fprintf(stderr,"Implementation dependent constants inquiry
                 failed.\n");
        fprintf(stderr,"Exiting\n");
        exit(1);
    }

    power_of_two_tm_required = (unsigned long int)
        constants[0].integer;
   square_tm_required = (unsigned long int) constants[1].integer;

    XFree(constants);

    /*
       Initialize domain data
    */

    domain = (PEXEnumTypeIndex) PEXExtTMDomainColor2D;

    domain_data.data.color.tm_type =
        (PEXEnumTypeIndex) PEXExtTMTypeMipMap;
    domain_data.data.color.texel_type =
        (PEXEnumTypeIndex) PEXExtTexelRGBAlphaInt8;
```

Code Example 6-2 Texture Preparation Example (Continued)

```
    /*
       Only need one level in the MIP Map.
    */

    domain_data.data.color.num_levels = (unsigned short int) 1;

    /*
       Initialize all 3 texture resources
    */

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

        switch ( i ) {
                case 0 :
                        base_array.dimension.t0 = (unsigned short
                                                        int) 128;
                        base_array.dimension.t1 = (unsigned short
                                                        int) 128;
                        base_array.dimension.t2 = (unsigned short
                                                        int) 0;
                         base_array.array.rgb_alpha8 =
                        (PEXExtTexelRGBAlpha8 *) &mandrill[0][0];
                         map_name = "MANDRILL";
                         break;
                case 1 :
                         base_array.dimension.t0 =
                        base_array.dimension.t1 = (unsigned short
                                                        int) 128;
                        base_array.dimension.t2 = (unsigned short
                                                        int) 0;
                         base_array.array.rgb_alpha8 =
                         (PEXExtTexelRGBAlpha8 *) &landsat[0][0];
                         map_name = "LANDSAT";
                         break;
                case 2 :
                         base_array.dimension.t0 =
                        base_array.dimension.t1 = (unsigned short
                                                        int) 128;
                        base_array.dimension.t2 = (unsigned short
                                                        int) 0;
                         base_array.array.rgb_alpha8 =
                           (PEXExtTexelRGBAlpha8 *) &stone[0][0];
                         map_name = "STONE";
                         break;
```

Code Example 6-2 Texture Preparation Example (Continued)

```
}

/*
 * Create filtered texture map
 */

status = PEXExtCreateFilteredTM(
              (int) domain,
              &domain_data,
              power_of_two_tm_required,
              square_tm_required,
              (PEXExtTexelArray *) &base_array,
              (PEXExtTexelArray **) &texel_array);

if ( status ) {
        fprintf(stderr,
        "Failed to filter map [%s]...\n", map_name);
};

/*
 * Give map to PEXlib now...
 */

texture_IDs[i] = PEXExtCreateTM(
                        dpy,
                        domain,
                        & domain_data,
                        texel_array);

if ( ! texture_IDs[i] ) {
        fprintf(stderr, "Failed to Create TM
        [%s]...Exiting\n",
                map_name);
        exit ( -1 );
}

/*
 * Dispose of temporary storage
 */

PEXExtFreeFilteredTM(
                domain,
                & domain_data,
                texel_array);
```

Code Example 6-2 Texture Preparation Example (Continued)

```
    }

    /*
     * Initialize parameterization data
     */

    parameterization = (PEXEnumTypeIndex) PEXExtTMParamExplicit;

    /*
     * Initialize TM rendering for maps...
     */

    tm_rendering_order =
                (PEXEnumTypeIndex)
                PEXExtTMRenderingOrderPostSpecular;

    /*
     * Initialize all 3 texture description resources
     */

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

        switch ( i ) {
                case 0 :
                        map_name = "MANDRILL";
                        break;
                case 1 :
                        map_name = "LANDSAT";
                        break;
                case 2 :
                        map_name = "STONE";
                        break;
        }

        /*
         * Create map descriptions within PEXlib now...
         */

        texture_desc_IDs[i] = PEXExtCreateTMDescription(
                                dpy,
                                parameterization,
                                & param_data,
                                tm_rendering_order,
                                1,
```

Code Example 6-2 Texture Preparation Example (Continued)

```
                                  & texture_IDs[i]);

         if ( ! texture_desc_IDs[i] ) {
                 fprintf(stderr,
                      "Failed to Create TM Description [%s]...\n",
                         map_name);
                 exit ( -1 );
         }
     }
} /* init_textures */

/*
         This procedure sets up the texture mapping LUTs.
*/
set_texture_luts(dpy, ext_rend_attrs)
    Display *dpy;
    PEXExtRendererAttributes *ext_rend_attrs;
{

    PEXExtTMBindingEntry         binding[3];
    PEXExtTMCoordSourceEntry     coord_source[1];
    PEXExtTMCompositionEntry     composition[1];
    PEXExtTMSamplingEntry        sampling[1];
    int                          i;

    /*
       Setup coord source LUT...
    */

    coord_source[0].tm_source =
         (PEXEnumTypeIndex) PEXExtTMCoordSourceFloatData;
    coord_source[0].fp_data_index = (unsigned short int) 0;
    PEXIdentityMatrix( coord_source[0].orientation );

    PEXExtSetTableEntries(dpy,
                             ext_rend_attrs->tm_coord_source_table,
                               1,
                               1,
                               PEXExtLUTTMCoordSource,
                               coord_source
                               );

    /*
       Setup composition LUT...
```

Code Example 6-2 Texture Preparation Example (Continued)

```
    */

    composition[0].method = (PEXEnumTypeIndex)
    PEXExtTMCompositeModulate;

    PEXExtSetTableEntries(dpy,
                          ext_rend_attrs->tm_composition_table,
                           1,
                           1,
                           PEXExtLUTTMComposition,
                           composition);

    /*
       Setup sampling LUT...
    */

    sampling[0].minification_method =
        (PEXEnumTypeIndex) PEXExtTMTexelSampleSingleBase;

    sampling[0].magnification_method =
        (PEXEnumTypeIndex) PEXExtTMTexelSampleSingleBase;

    sampling[0].t0_boundary_condition =
        (PEXEnumTypeIndex) PEXExtTMBoundaryCondClampAbsolute;
    sampling[0].t1_boundary_condition =
        (PEXEnumTypeIndex) PEXExtTMBoundaryCondClampAbsolute;

    sampling[0].boundary_clamp_color_source =
            (PEXEnumTypeIndex) PEXExtTMClampColorSourceExplicit;
    sampling[0].boundary_clamp_color_source =
            (PEXEnumTypeIndex) PEXExtTMClampColorSourceBoundary;
    sampling[0].depth_sampling_bias_hint =
                (float) 0.0;
    sampling[0].t0_frequency_hint =
                (float) 1.0;
    sampling[0].t1_frequency_hint =
                (float) 1.0;

    PEXExtSetTableEntries(dpy,
                           ext_rend_attrs->tm_sampling_table,
                           1,
                           1,
                           PEXExtLUTTMSampling,
                           sampling);
```

Code Example 6-2 Texture Preparation Example (Continued)

```
    /*
       Setup binding LUT...
    */

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

        binding[i].tm_description_id =
                (PEXExtTMDescription) texture_desc_IDs[i] ;
        binding[i].coord_source_index =
                (PEXTableIndex) 1;
        binding[i].composition_index =
                (PEXTableIndex) 1;
        binding[i].sampling_index =
                (PEXTableIndex) 1;
    }

    PEXExtSetTableEntries(dpy,
                        ext_rend_attrs->tm_binding_table,
                        1,
                        3,
                        PEXExtLUTTMBinding,
                        binding);
} /* set_texture_luts */

/*
        This procedure releases the texture map resources.
*/

void cleanup_textures(dpy)
    Display *dpy;
{
    int i;

    /*
       Free texture descriptions...
    */

    for (i = 0; i < 3; i++)
        PEXExtFreeTMDescription(
                dpy,
                texture_desc_IDs[i]
                );
```

Code Example 6-2 Texture Preparation Example (Continued)

```
    /*
       Free texture resources...
    */

    for (i = 0; i < 3; i++)
        PEXExtFreeTM(
                dpy,
                texture_IDs[i]);
} /* cleanup_textures */
```

With the textures set up by Code Example 6-2, the following code, Code Example 6-3, draws a cube with each of the three textures applied to two opposite faces of the cube.

Code Example 6-3 Draw a Cube Example

```
        /*
            Left and Right sides of the cube
        */
        binding_lut_entry = 1;
        PEXExtSetActiveTextures(dpy, resource_id, req_type, 1,
            &binding_lut_entry);

        vertex_data.vertices.with_fp_data = (PEXPointer) cube[0];
        PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
            PEXShapeUnknown, False, PEXContourDisjoint, 2,
            PEXGANone, (0 | PEXGANormal | PEXExtGAData),
            PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
            &vertex_data);

        vertex_data.vertices.with_fp_data = (PEXPointer) cube[1];
        PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
            PEXShapeUnknown, False, PEXContourDisjoint, 2,
            PEXGANone, (0 | PEXGANormal | PEXExtGAData),
            PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
            &vertex_data);

        /*
            Back and Front sides of the cube
        */
        binding_lut_entry = 2;
        PEXExtSetActiveTextures(dpy, resource_id, req_type, 1,
            &binding_lut_entry);
```

Code Example 6-3 Draw a Cube Example (Continued)

```
        vertex_data.vertices.with_fp_data = (PEXPointer) cube[2];
         PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
             PEXShapeUnknown, False, PEXContourDisjoint, 2,
             PEXGANone, (0 | PEXGANormal | PEXExtGAData),
             PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
             &vertex_data);

        vertex_data.vertices.with_fp_data = (PEXPointer) cube[3];
         PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
             PEXShapeUnknown, False, PEXContourDisjoint, 2,
             PEXGANone, (0 | PEXGANormal | PEXExtGAData),
             PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
             &vertex_data);

         /*
             Top and Bottom sides of the cube
         */
         binding_lut_entry = 3;
         PEXExtSetActiveTextures(dpy, resource_id, req_type, 1,
             &binding_lut_entry);

        vertex_data.vertices.with_fp_data = (PEXPointer) cube[4];
         PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
             PEXShapeUnknown, False, PEXContourDisjoint, 2,
             PEXGANone, (0 | PEXGANormal | PEXExtGAData),
             PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
             &vertex_data);

        vertex_data.vertices.with_fp_data = (PEXPointer) cube[5];
         PEXExtFillAreaSetWithData (dpy, resource_id, req_type,
             PEXShapeUnknown, False, PEXContourDisjoint, 2,
             PEXGANone, (0 | PEXGANormal | PEXExtGAData),
             PEXColorTypeRGB, 1, (PEXFacetData *) NULL,
             &vertex_data);
```

Conformance Summary 7

Minimum Conformance

Vendors who conform to the specifications outlined by the Common Graphics Environment conform to the following *minimum* requirements. These requirements have been provided here as a reference, so you can code to the maximum portability on CGE platforms.

Enumerated Types

Individual vendors may choose to support more enumerated types than the minimum required for CGE PEX 5.1 compliance. You should invoke the PEXGetEnumTypeInfo function to obtain the actual enumerated types supported. Use the PEXETEnumType to obtain the complete list of CGE PEX 5.1 enumerated types supported by your vendor platform:

Table 7-1 CGE PEX 5.1 Enumerated Types Supported

PEX 5.1 Class	CGE PEX 5.1 Types Supported
PEXETATextStyle	PEXATextNotConnected, PEXATextConnected
PEXETColorApproxModel	PEXColorApproxRGB
PEXETColorApproxType	PEXColorSpace
PEXETColorType	PEXColorTypeIndexed, PEXColorTypeRGB
PEXETCurveApproxMethod	PEXApproxImpDep
PEXETDisplayUpdateMode	Not required.

Table 7-1 CGE PEX 5.1 Enumerated Types Supported (Continued)

PEX 5.1 Class	CGE PEX 5.1 Types Supported
PEXETEscape	Not required. See Table 7-2 on page 77 for details on extensions.
PEXETFloatFormat	PEXIEEE_754_32
PEXETGDP2D	Not required.
PEXETGDP	Not required.
PEXETGSE	Not required.
PEXETHatchStyle	Not required. See Table 7-4 on page 78 for details on extensions.
PEXETHLHSRMode	PEXHLHSROff, PEXHLHSRZBuffer, PEXHLHSRZBufferID
PEXETInteriorStyle	PEXInteriorStyleHollow, PEXInteriorStyleSolid, PEXInteriorStyleHatch, PEXInteriorStyleEmpty. See Table 7-5 on page 78 for details on extensions.
PEXETLightType	PEXLightAmbient, PEXLightWCVector, PEXLightWCPoint, PEXLightWCSpot.
PEXETLineType	PEXLineTypeSolid, PEXLineTypeDashed, PEXLineTypeDotted, PEXLineTypeDashDot. See Table 7-3 on page 78 for details on extensions.
PEXETMarkerType	PEXMarkerDot, PEXMarkerCross, PEXMarkerAsterisk, PEXMarkerCircle, PEXMarkerX
PEXETModelClipOperator	PEXModelClipReplace, PEXModelClipIntersection
PEXETParaSurfCharacteristics	PEXPSCNone, PEXPSCImpDep
PEXETPickAllMethod	PEXPickAllAll
PEXETPickDeviceType	PEXPickDeviceDCHitBox
PEXETPickOneMethod	PEXPickLast
PEXETPolylineInterpMethod	PEXPolylineInterpNone, PEXPolylineInterpColor
PEXETPromptEchoType	Not required.

Table 7-1 CGE PEX 5.1 Enumerated Types Supported (Continued)

PEX 5.1 Class	CGE PEX 5.1 Types Supported
PEXETReflectionModel	PEXReflectionNone, PEXReflectionAmbient, PEXReflectionDiffuse, PEXReflectionSpecular
PEXETRenderingColorModel	PEXRenderingColorModelImpDep
PEXETSurfaceApproxMethod	PEXApproxImpDep
PEXETSurfaceEdgeType	PEXSurfaceEdgeSolid
PEXETSurfaceInterpMethod	PEXSurfaceInterpNone, PEXSurfaceInterpColor
PEXETTrimCurveApproxMethod	PEXApproxImpDep

Escapes

Some escapes are introduced with CGE PEX 5.1. The following escapes are required to be supported by all CGE PEX 5.1 compliant servers. The PEXETEscape enumerated type lists the escapes supported by your server:

Table 7-2 CGE PEX 5.1 Escape Extensions

CGE PEX 5.1 Escape Extensions
PEXExtEscapeChangePipelineContext, PEXExtETMEscapeChangePipelineContext
PEXExtEscapeChangeRenderer, PEXExtETMEscapeChangeRenderer
PEXExtEscapeGetRendererAttributes, PEXExtETMEscapeGetRendererAttributes
PEXExtEscapeSetTableEntries, PEXExtETMGetTableEntries
PEXExtEscapeGetTableEntry, PEXExtETMGetTableEntry
PEXExtEscapeCreateTM, PEXExtETMCreateTM
PEXExtEscapeCreateTMFromWindow, PEXExtETMCreateTMFromWindow
PEXExtEscapeCreateTMDescription, PEXExtETMCreateDescription
PEXExtEscapeFreeTM, PEXExtETMFreeTM
PEXExtEscapeFreeTMDescription, PEXExtETMEscapeFreeTMDescription
PEXExtEscapeFetchElements, PEXExtETMEscapeFetchElements
PEXExtEscapeQueryColorApprox, PEXExtETMEscapeQueryColorApprox

Line Types

Some extended line types are introduced with CGE PEX 5.1. The following line types are required to be supported by all CGE PEX 5.1 compliant servers, in addition to the line types defined by PEX 5.1. The PEXETLineType enumerated type lists the line types supported by your server:

Table 7-3 CGE PEX 5.1 Line Types

CGE PEX 5.1 Line Types
PEXExtLineTypeCenter, PEXExtETMLineTypeCenter
PEXExtLineTypePhantom, PEXExtETMLineTypePhantom

Hatch Styles

Some hatch styles are introduced with CGE PEX 5.1. The following hatch styles are required to be supported by all CGE PEX 5.1 compliant servers. The PEXETHatchStyle enumerated type lists the hatch styles supported by your server:

Table 7-4 CGE PEX 5.1 Hatch Styles

CGE PEX 5.1 Hatch Styles
PEXExtHatchStyle45Degrees, PEXExtETMHatchStyle45Degrees
PEXExtHatchStyle135Degrees, PEXExtETMHatchStyle135Degrees

Interior Styles

An extended interior style is introduced with CGE PEX 5.1. The following interior style is required to be supported by all CGE PEX 5.1 compliant servers. The PEXETInteriorStyle enumerated type lists the interior styles supported by your server:

Table 7-5 CGE PEX 5.1 Interior Styles

CGE PEX 5.1 Interior Styles
PEXExtInteriorStyleTexture, PEXExtETMInteriorStyleTexture

Extended Enumerated Types

In addition, the following extended enumerated types are required to be supported by all CGE PEX 5.1 servers. PEXExtETEnumType extension returns the list of enumerated types supported by your server. The list includes PEXExtETEnumType itself. You can use this extension to verify that CGE PEX is supported by your server. The extended enumerated types include:

Table 7-6 CGE PEX 5.1 Enumerated Type Extensions

CGE PEX 5.1 Enum Type Extension	Definition
PEXExtETEnumType	Returns the list of extended enumerated types supported. This table contains the possible values.
PEXExtETOC	Returns the list of extended output commands supported. See Table 7-7 on page 82 for details.
PEXExtETPC	Returns the list of extended pipeline context attributes supported. See Table 7-8 on page 83 for details.
PEXExtETRA	Returns the list of extended renderer attributes supported. See Table 7-8 on page 83 for details.
PEXExtETLUT	Returns the list of extended lookup tables supported. See Table 7-10 on page 84 for details.
PEXExtETID	Returns the list of extended implementation-dependent constants supported. See Table 7-13 on page 87 for details.
PEXExtETTMRenderingOrder	Specifies where in the rendering pipeline a texture map is to be applied. See Table 7-14 on page 87 for details.
PEXExtETTMCoordSource	Specifies the coordinate source within a primitive's vertex data of a texture coordinate. See Table 7-15 on page 88 for details.
PEXExtETTMCompositeMethod	Specifies how the texture map is blended with the primitive's existing data. See Table 7-16 on page 88 for details.
PEXExtETTMTexelSampleMethod	Specifies how texels are sampled from the texture map and mapped to primitive pixels when the area covered by a texel and pixel differ. See Table 7-17 on page 89 for details.

Table 7-6 CGE PEX 5.1 Enumerated Type Extensions (Continued)

CGE PEX 5.1 Enum Type Extension	Definition
PEXExtETTMBoundaryCondition	Specifies the texturing to be applied to the primitive when the texture coordinates select a point beyond a boundary of the texture map. See Table 7-18 on page 89 for details.
PEXExtETTMClampColorSource	Specifies what color is to be applied if the texture mapping boundary condition ClampColor is used. See Table 7-19 on page 90 for details.
PEXExtETTMDomain	Specifies the texture dimension and what aspect of a primitive is to be modified by texture mapping. See Table 7-20 on page 90 for details.
PEXExtETTexelType	Defines the format of the texture map texel values. See Table 7-21 on page 91 for details.
PEXExtETTMResourceHint	Defines what type of resource optimization hints are to be made. See Table 7-22 on page 92 for details.
PEXExtETTMType	Specifies the type of texture map. See Table 7-23 on page 92 for details.
PEXExtETTMParameterizationMethod	Defines how texture map coordinates are derived. See Table 7-24 on page 93 for details.
PEXExtETTMPerspectiveCorrection	Specifies the method used to compute texture coordinate values in surface interiors when rendering textured surface primitives. See Table 7-25 on page 93 for details.
PEXExtETTMSampleFrequency	Specifies the frequency to sample texels in a texture map when texturing surface primitives. See Table 7-26 on page 94 for details.
PEXExtETPrimitiveAAMode	Specifies what modifications should be made to the image to correct aliasing effects. See Table 7-27 on page 94 for details.

Table 7-6 CGE PEX 5.1 Enumerated Type Extensions (Continued)

CGE PEX 5.1 Enum Type Extension	Definition
PEXExtETPrimitiveAABlendOp	Specifies how blending is accomplished with primitive anti-aliasing. See Table 7-28 on page 95 for details.
PEXExtETLineCapStyle	Specifies how the ends of a wide line are rendered. See Table 7-29 on page 95 for details.
PEXExtETLineJoinStyle	Specifies how the intersection between adjacent segments of a wide line are rendered. See Table 7-30 on page 96 for details.

Extended Output Commands

Individual vendors may choose to support more output commands than the minimum required for CGE PEX 5.1 compliance. From the list of OCs defined by the non-extended PEX 5.1, CGE PEX 5.1 requires that all OCs be supported though some CGE vendors may support OCs with the following exceptions:

- *Cell Arrays* may be rendered as simple polyline boundaries, using the current polyline attributes. These OCs include: PEXCellArray, PEXCellArray2D, and PEXExtendedCellArray.
- *Pattern functions* must be supported but are not required to have any visual effect. These OCs include: PEXSetPatternAttributes, PEXSetPatternAttributes2D and PEXSetPatternSize.
- *Text precision* is fully supported for stroked fonts; however:
- PEXStringPrecision: height, width and expansion are evaluated as closely as possible, but in a workstation-dependent way. Up and base vectors, path, alignment, and spacing may not be applied. Clipping is also workstation-dependent.
- PEXCharPrecision: spacing is evaluated exactly. Height, width, expansion, and up and base vectors are evaluated as closely as possible, but in a workstation-dependent way. Clipping is performed at least on a character-by-character basis.
- PEXStrokePrecision: all attributes and clipping are applied exactly.

In addition, the following extended OCs are required to be supported by all CGE PEX 5.1 servers:

Table 7-7 CGE PEX 5.1 OC Extensions

CGE PEX 5.1 OC Extension	**CGE PEX 5.1 Extensions Supported**
PEXExtTMPerspectiveCorrection	Required.
PEXExtTMSampleFrequency	Required.
PEXExtSetTMResourceHints	Required.
PEXExtOCActiveTextures	Required.
PEXExtOCBFActiveTextures	Required.
PEXExtOCFillAreaSetWithData	Required.
PEXExtOCSetOfFillAreaSets	Required.
PEXExtOCTriangleStrip	Required.
PEXExtOCQuadrilateralMesh	Required.
PEXExtOCPrimitiveAA	Required.
PEXExtOCLineCapStyle	Required
PEXExtOCLineJoinStyle	Required.
PEXExtOCEllipse	Required.
PEXExtOCEllipse2D	Required.
PEXExtOCCircle2D	Required.
PEXExtOCEllipticalArc	Required.
PEXExtOCEllipticalArc2D	Required.
PEXExtOCCircularArc2D	Required.

Pipeline Context Attributes

Individual vendors may choose to support more pipeline context attributes than the minimum required for CGE PEX 5.1 compliance. From the list of pipeline context attributes defined by the non-extended PEX 5.1, CGE PEX 5.1 requires that all attributes be supported. In addition, the following extended pipeline context attributes apply to extended pipeline contexts and are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETPC enumerated type lists the extended pipeline context attributes supported by the server:

Table 7-8 CGE PEX 5.1 Pipeline Context Extensions

CGE PEX 5.1 Pipeline Context Extension	CGE PEX 5.1 Extensions Supported
PEXExtPCTMPerspectiveCorrection	Required.
PEXExtPCTMResourceHints	Required.
PEXExtPCTMSampleFrequency	Required.
PEXExtPCActiveTextures	Required.
PEXExtPCBFActiveTextures	Required.
PEXExtPCPrimitiveAA	Required.
PEXExtPCLineCapStyle	Required.
PEXExtPCLineJoinStyle	Required.

Renderer Attributes

Individual vendors may choose to support more renderer attributes than the minimum required for CGE PEX 5.1 compliance. From the list of renderer attributes defined by the non-extended PEX 5.1, CGE PEX 5.1 requires that all attributes be supported. The non-extended renderer attributes and requests only apply to PEX 5.1, the following extended renderer attributes apply to extended rendering and are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETRA enumerated type lists the extended renderer attributes supported by the server:

Table 7-9 CGE PEX 5.1 Renderer Extensions

CGE PEX 5.1 Renderer Extension	CGE PEX 5.1 Extensions Supported
PEXExtRATMBindingTable	Required.
PEXExtRATMCoordSourceTable	Required.
PEXExtRATMCompositionTable	Required.
PEXExtRAMTMSamplingTable	Required.

Lookup Tables

Individual vendors may choose to support more lookup tables than the minimum required for CGE PEX 5.1 compliance. From the list of lookup tables defined by the non-extended PEX 5.1, CGE PEX 5.1 requires the following:

- Sparse indexing is supported which enables you to define noncontiguous indices in the lookup table.
- The default table values are required (see Table 7-8 on page 83)
- When either set or realized values are inquired, implementations return a value. However, implementations may or may not distinguish between set and realized values.
- No table is required to have predefined entries.

Table 7-10 CGE PEX 5.1 Lookup Table Requirements

CGE PEX 5.1 Lookup Table Requirements	**Default Index**	**Index Value Range**	**Minimum Entries Required**
PEXLUTColor	1	0 .. 65534	256
PEXLUTColorApprox	0	0 .. 65534	1
PEXLUTDepthCue	0	0 .. 65534	2 (ON and OFF are required)
PEXLUTEdgeBundle	1	1 .. 65535	20
PEXLUTInteriorBundle	1	1 .. 65535	20
PEXLUTLight	1	1 .. 65535	8
PEXLUTLineBundle	1	1 .. 65535	20
PEXLUTMarkerBundle	1	1 .. 65535	20
PEXLUTPattern	1	1 .. 65535	0 (pattern is not required)
PEXLUTTextBundle	1	1 .. 65535	20
PEXLUTTextFont	1	1 .. 65535	1
PEXLUTView	0	0 .. 65534	64

In addition, the following extended lookup tables are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETLUT enumerated type lists the extended lookup tables supported by the server:

Table 7-11 CGE PEX 5.1 Lookup Table Extensions

CGE PEX 5.1 Lookup Table Extension	Default Index	Index Value Range	Minimum Entries Required
PEXExtLUTTMBinding	0	0 .. 65534	20
PEXExtLUTTMComposition	0	0 .. 65534	20
PEXExtLUTTMCoordSource	0	0 .. 65534	20
PEXExtLUTTMSampling	0	0 .. 65534	20

Implementation-Dependent Constants

Individual vendors may choose to support more implementation-dependent constants than the minimum required for CGE PEX 5.1 compliance. You should invoke the PEXGetImpDepConstants function to obtain the actual implementation-dependent constants supported. Use PEXGetImpDepConstants to obtain the complete list of PEX 5.1 implementation-dependent constants supported by your vendor platform:

Table 7-12 PEX 5.1 Constants Supported by CGE PEX 5.1

PEX 5.1 Constants	PEX 5.1 Constants Supported by CGE PEX 5.1
PEXIDBestColorApprox	implementation-dependent
PEXIDDitheringSupported	implementation-dependent
PEXIDDoubleBufferingSupported	implementation-dependent
PEXIDMaxEdgeWidth	implementation-dependent
PEXIDMaxHitsEventSupported	True
PEXIDMaxLineWidth	At least 4 pixels
PEXIDMaxMarkerSize	At least the size of the largest screen dimension
PEXIDMaxModelClipPlanes	At least 6
PEXIDMaxNameSetNames	At least 1024
PEXIDMaxNonAmbientLights	At least 7

Table 7-12 PEX 5.1 Constants Supported by CGE PEX 5.1 (Continued)

PEX 5.1 Constants	PEX 5.1 Constants Supported by CGE PEX 5.1
PEXIDMaxNURBOrder	At least 6
PEXIDMaxTrimCurveOrder	At least 6
PEXIDMinEdgeWidth	At least 1 pixel
PEXIDMinLineWidth	At least 1 pixel
PEXIDMinMarkerSize	implementation-dependent
PEXIDNominalEdgeWidth	1 pixel
PEXIDNominalLineWidth	1 pixel
PEXIDNominalMarkerSize	implementation-dependent
PEXIDNumSupportedEdgeWidths	At least 1
PEXIDNumSupportedLineWidths	0 (continuous)
PEXIDNumSupportedMarkerSizes	0 (continuous)
PEXIDTransparencySupported	True. The transmission coefficient varies from 0.0 (opaque) to 1.0 (fully transparent). If transparency is supported through the screen door algorithm, then a minimum of 4 screens are supported.
PEXIDChromaticityRedU	implementation-dependent
PEXIDChromaticityRedV	implementation-dependent
PEXIDLuminanceRed	implementation-dependent
PEXIDChromaticityGreenU	implementation-dependent
PEXIDChromaticityGreenV	implementation-dependent
PEXIDLuminanceGreen	implementation-dependent
PEXIDChromaticityBlueU	implementation-dependent
PEXIDChromaticityBlueV	implementation-dependent
PEXIDLuminanceBlue	implementation-dependent
PEXIDChromaticityWhiteU	implementation-dependent
PEXIDChromaticityWhiteV	implementation-dependent
PEXIDLumianceWhite	implementation-dependent

In addition, the following extended implementation-dependent constants are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETID enumeration type lists the extended implementation-dependent constants supported by your server:

Table 7-13 CGE PEX 5.1 Implementation-dependent Constants

CGE PEX 5.1 Imp. Dep. Extension	CGE PEX 5.1 Imp. Dep. Supported
PEXExtIDMaxTextureMaps	implementation-dependent
PEXExtIDMaxFastTMSize	implementation-dependent
PEXExtIDPowerOfTwoTMSizesRequired	implementation-dependent
PEXExtIDSquareTMRequired	implementation-dependent

Texture Mapping Rendering Order

Texture mapping rendering order is introduced with CGE PEX 5.1. The following texture mapping rendering orders are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMRenderingOrder enumerated type lists the texture mapping rendering orders supported by your server:

Table 7-14 CGE PEX 5.1 Texture Mapping Rendering Orders

CGE PEX 5.1 Rendering Order	CGE PEX 5.1 Types Supported
PEXExtTMRenderingOrderPreSpecular	Not required.
PEXExtTMRenderingOrderPostSpecular	Required.

Texture Mapping Coordinate Source

Texture mapping coordinate source is introduced with CGE PEX 5.1. The following texture mapping coordinate sources are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMCoordSource enumerated type lists the texture mapping coordinate sources supported by your server:

Table 7-15 CGE PEX 5.1 Texture Mapping Coordinate Sources

CGE PEX 5.1 Coordinate Source	CGE PEX 5.1 Type Supported
PEXExtTMCoordSourceVertexCoord	Not required.
PEXExtTMCoordSourceVertexNormal	Not required.
PEXExtTMCoordSourceFloatData	Required.

Texture Mapping Composite Method

Texture mapping composite method is introduced with CGE PEX 5.1. The following texture mapping composite methods are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMCompositeMethod enumerated type lists the texture mapping composite methods supported by your server:

Table 7-16 CGE PEX 5.1 Texture Mapping Composite Methods

CGE PEX 5.1 Composite Method	CGE PEX 5.1 Types Supported
PEXExtTMCompositeReplace	Required.
PEXExtTMCompositeModulate	Required.
PEXExtTMCompositeBlendEnvColor	Not required.
PEXExtTMCompositeDecal	Not required.
PEXExtTMCompositeDecalBackground	Not required.
PEXExtTMCompositeReplaceBlendedColors	Not required.

Texture Mapping Texel Sample Method

Texture mapping texel sample method is introduced with CGE PEX 5.1. The following texture mapping sample methods are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMTexelSampleMethod enumerated type lists the texture mapping sample methods supported by your server:

Table 7-17 CGE PEX 5.1 Texture Mapping Texel Sample Methods

CGE PEX 5.1 Texel Sample Method	CGE PEX 5.1 Methods Supported
PEXExtTMTexelSampleSingleBase	Required.
PEXExtTMTexelSampleLinearBase	Not required.
PEXExtTMTexelSampleSingleInMipMap	Not required.
PEXExtTMTexelSampleLinearInMipMap	Not required.
PEXExtTMTexelSampleSingleBetweenMipMaps	Not required.
PEXExtTMTexelSampleLinearBetweenMipMaps	Not required.

Texture Mapping Boundary Condition

Texture mapping boundary condition is introduced with CGE PEX 5.1. The following texture mapping boundary conditions are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMBoundaryCondition enumerated type lists the texture mapping boundary conditions supported by your server:

Table 7-18 CGE PEX 5.1 Texture Mapping Boundary Conditions

CGE PEX 5.1 Boundary Condition	CGE PEX 5.1 Conditions Supported
PEXExtTMBoundaryCondClampColor	Required.
PEXExtTMBoundaryCondClampAbsolute	Required.
PEXExtTMBoundaryCondWrap	Not required.
PEXExtTMBoundaryCondMirror	Not required.

Texture Mapping Clamp Color Source

Texture mapping clamp color source is introduced with CGE PEX 5.1. The following texture mapping clamp color sources are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETTMClampColorSource enumerated type lists the texture mapping clamp color sources supported by your server:

Table 7-19 CGE PEX 5.1 Texture Mapping Clamp Color Sources

CGE PEX 5.1 Clamp Color Source	CGE PEX 5.1 Type Supported
PEXExtTMClampColorSourceBoundary	Required.
PEXExtTMClampColorSourceExplicit	Required.

Texture Mapping Domain

Texture mapping domain is introduced with CGE PEX 5.1. The following texture mapping domains are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMDomain enumerated type lists the texture mapping domains supported by your server:

Table 7-20 CGE PEX 5.1 Texture Mapping Domains

CGE PEX 5.1 Domain	CGE PEX 5.1 Type Supported
PEXExtTMDomainColor1D	Not required.
PEXExtTMDomainColor2D	Required.
PEXExtTMDomainColor3D	Not required.

Texture Mapping Texel Type

Texture mapping texel type is introduced with CGE PEX 5.1. The following texture mapping texel types are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMTexelType enumerated type lists the texture mapping texel types supported by your server:

Table 7-21 CGE PEX 5.1 Texture Mapping Texel Types

CGE PEX 5.1 Texel Type	CGE PEX 5.1 Types Supported
PEXExtTexelLuminanceFloat	Not required.
PEXExtTexelLuminanceInt8	Not required.
PEXExtTexelLuminanceInt16	Not required.
PEXExtTexelLuminanceAlphaFloat	Not required.
PEXExtTexelLuminanceAlphaInt8	Not required.
PEXExtTexelLuminanceAlphaInt16	Not required.
PEXExtTexelRGBFloat	Required.
PEXExtTexelRGBInt8	Required.
PEXExtTexelRGBInt16	Not required.
PEXExtTexelRGBAlphaFloat	Required.
PEXExtTexelRGBAlphaInt8	Required.
PEXExtTexelRGBAlphaInt16	Not required.

Texture Mapping Resource Hints

Texture mapping resource hints is introduced with CGE PEX 5.1. The following texture mapping resource hints are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETTMResourceHints enumerated type lists the texture mapping resource hints supported by your server:

Table 7-22 CGE PEX 5.1 Texture Mapping Resource Hints

CGE PEX 5.1 Resource Hints	CGE PEX 5.1 Types Supported
PEXExtTMResourceHintNone	Required.
PEXExtTMResourceHintSpeed	Required.
PEXExtTMResourceHintSpace	Required.

Texture Mapping Type

Texture mapping type is introduced with CGE PEX 5.1. The following texture mapping types are required to be supported by all CGE PEX 5.1 compliant servers. The PEXExtETTMType enumerated type lists the texture mapping types supported by your server:

Table 7-23 CGE PEX 5.1 Texture Mapping Types

CGE PEX 5.1 Texture Mapping Type	CGE PEX 5.1 Types Supported
PEXExtTMTypeMipMap	Required.

Texture Mapping Parameterization Method

Texture mapping parameterization method is introduced with CGE PEX 5.1. The following texture parameterization methods are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below.The PEXExtETTMParameterizationMethod enumerated type lists the texture mapping parameterization methods supported by your server:

Table 7-24 CGE PEX 5.1 Texture Mapping Parameterization Methods

CGE PEX 5.1 Parameterization Method	CGE PEX 5.1 Methods Supported
PEXExtTMParamExplicit	Required.
PEXExtTMParamReflectSphereVRC	Required.
PEXExtTMParamReflectSphereWC	Not required.
PEXExtTMParamLinearVRC	Not required.

Texture Mapping Perspective Correction

Texture mapping perspective correction is introduced with CGE PEX 5.1. The following texture mapping perspective corrections are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMPerspectiveCorrection enumerated type lists the texture mapping perspective corrections supported by your server:

Table 7-25 CGE PEX 5.1 Texture Mapping Perspective Corrections

CGE PEX 5.1 Perspective Correction	CGE PEX 5.1 Types Supported
PEXExtTMPerspCorrectNone	Required.
PEXExtTMPerspCorrectVertex	Not required.
PEXExtTMPerspCorrectPixel	Required.

Texture Mapping Sample Frequency

Texture mapping sample frequency is introduced with CGE PEX 5.1. The following texture mapping sample frequencies are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETTMSampleFrequency enumerated type lists the texture mapping sample frequencies supported by your server:

Table 7-26 CGE PEX 5.1 Texture Mapping Sample Frequencies

CGE PEX 5.1 Sample Frequency	CGE PEX 5.1 Types Supported
PEXExtTMSampleFrequencyPixel	Required.
PEXExtTMSampleFrequencyInterpDep	Not required.

Primitive Anti-Aliasing Mode

Primitive anti-aliasing mode is introduced with CGE PEX 5.1. The following primitive anti-aliasing modes are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETPrimitiveAAMode enumerated type lists the primitive anti-aliasing modes supported by your server:

Table 7-27 CGE PEX 5.1 Primitive Anti-Aliasing Modes

CGE PEX 5.1 Primitive Anti-Aliasing Mode	CGE PEX 5.1 Modes Supported
PEXExtPrimAANone	Required.
PEXExtPrimAAPoint	Not required.
PEXExtPrimAAVector	Not required.
PEXExtPrimAAPointVector	Not required.
PEXExtPrimAAPolygon	Not required.
PEXExtPrimAAPointPolygon	Not required.
PEXExtPrimAAVectorPolygon	Not required.
PEXExtPrimAAPointVectorPolygon	Not required.

Primitive Anti-Aliasing Blend Operation

Primitive anti-aliasing blend operation is introduced with CGE PEX 5.1. The following primitive anti-aliasing blend operations are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETPrimitiveAABlendOp enumerated type lists the primitive anti-aliasing blend operations supported by your server:

Table 7-28 CGE PEX 5.1 Primitive Anti-Aliasing Blend Operations

CGE PEX 5.1 Blend Operations	CGE PEX 5.1 Operations Supported
PEXExtPrimAABlendOpImpDep	Required.
PEXExtPrimAABlendOpSimpleAlpha	Not required.

Line Cap Style

Line cap style is introduced with CGE PEX 5.1. The following line cap styles are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtLineCapStyle enumerated type lists the line cap styles supported by your server:

Table 7-29 CGE PEX 5.1 Line Cap Styles

CGE PEX 5.1 Line Cap Styles	CGE PEX 5.1 Styles Supported
PEXExtLineCapStyleButt	Required.
PEXExtLineCapStyleRound	Not required.
PEXExtLineCapStyleProjecting	Not required.

Line Join Style

Line join style is introduced with CGE PEX 5.1. The following line join styles are required to be supported by all CGE PEX 5.1 compliant servers, with the exceptions noted below. The PEXExtETLineJoinStyle enumerated type lists the line join styles supported by your server:

Table 7-30 CGE PEX 5.1 Line Join Styles

CGE PEX 5.1 Line Join Styles	CGE PEX 5.1 Styles Supported
PEXExtLineJoinStyleImpDep	Required.
PEXExtLineJoinStyleRound	Not required.
PEXExtLineJoinStyleMiter	Not required.
PEXExtLineJoinStyleBevel	Not required.

Interoperability Conventions 8

Interoperability Conventions

CGE PEX 5.1 utilizes many of the PEX interoperability conventions in its design to ensure interoperability across all servers. The conventions are provided here for those interested in the interoperability rules. As its background information, you may choose to ignore this information:

1. X Consortium vendors wishing to define their own individual extension (for example, enumerated types (ET), enumerated type values (ETV), escapes (ESC), output commands (OCs), table types (TTs), GDPs, or GSEs should obtain a unique vendor id (a number) by the MIT registry.

   Vendor id numbers range from 1 to 126. The numbers 0 and 127 are reserved and are not distributed.

   If all vendor id numbers are distributed, a new vendor wishing a number needs to negotiate with an existing registered vendor and share that number space; otherwise, the new vendor has to wait until a release of PEX that expands vendor space.

2. A vendor-specific ET, ETV, OC, or TT has the high bit set (bit 15) and the vendor id encoded in the high bits (bits 14 through 8) of the type or value.
   - Bit 15 a one
   - Bits 14..8 vendor id
   - Bits 7..0 the type or value (a number in the range of 0..255)

3. Vendor-specific GDP, GSE, and ESC ids have the high bit set (bit 31), and the vendor id encoded in the high bits of the id:
   - Bit 31 a one
   - Bits 30..23 zeros
   - Bits 22..16 vendor id
   - Bits 15..8 zeros
   - Bits 7..0 the id (a number in the range of 0..255)
4. All vendor-specific ETs, ETVs, GDPs, GSEs, and ESCs can be inquired via the PEXGetEnumeratedTypeInfo request (the PEXlib function is PEXGetEnumTypeInfo). Because the PEXGetEnumerateTypeInfo interface is only 16 bits, GDP, GSE, and ESC ids are packed into 16 bits only for purposes of returning the values through PEXGetEnumeratedTypeInfo. Therefore all ETs, ETVs and GDP, GSE and ESC ids, as returned by PEXGetEnumeratedTypeInfo, are encoded:
   - Bit 15 a one
   - Bits 14..8 vendor id
   - Bits 7..0 type, value, or id (a number in the range of 0..255)
   - Specifically, the GDP, GSE, and ESC ids are packed as follows:
   - Bit 31 packs into bit 15
   - Bits 22..16 packs into bits 14..8
   - Bits 7..0 packs into bits 7..0
5. The list of vendor-specific OCs can be inquired via the PEXGetEnumeratedTypeInfo request using the PEXExtETOC enumerated type.
6. The list of vendor-specific table types can be inquired via the PEXGetEnumeratedTypeInfo request using the PEXExtETLUT enumerated type. An alternative way to determine if a vendor-specific table type is supported is to make a PEXGetTableInfo request with the table type (PEXlib function is also named PEXGetTableInfo). If the request is successful, then the table type is supported. If an error (either LookupTable or Value) is generated, then the table type is not supported.
7. Defining a separate X extension is the only way to add events or errors to PEX. XQueryExtension can be used to determine the base offsets for any new errors or events.

8. New requests can be added via the PEXEscape request. See Items 1-4 regarding the encoding of escape identifiers.

9. Not all servers support Big Request Extension (BRE). If BRE is supported by the server, large requests (greater than the maximum request size) can be sent to the server; however, individual OCs will still be limited to 64K words.

10. OCNil is encoded as opcode 0xFFFF. (This is the current PEX-SI OC opcode, and would conform to the encoding specified, if PEX-SI had vendor id 127: 0xFFFF = [1,vendor 127,value 255]). OCNil is valid only for element searching.

11. If facet normals are not given, the server should compute them as described in the *PHIGS PLUS International Standard* (ISO/IEC 9592-4:1992, pp. 37-39, clauses 4.5.5.2 - 4.5.5.4, Facet Normal, Facet Orientation, and Reflection Normal). The server should not modify any geometric or reflectance facet normals that the client explicitly defines.

12. The direction vector associated with the WCS_Vector and WCS_Spot light types points in the direction the light would travel from these sources.

13. Specular exponents are used as specified in Annex C of the *PHIGS PLUS International Standard* (ISO/IEC 9592-4:1992, pp 163-165).

14. Not all servers support all drawables. Use the PEXMatchRendererTargets request to determine supported drawable types (PEXlib function is PEXMatchRenderingTargets). The ordering of supported rendering targets returned by servers should not be interpreted as a recommendation for usage because the list may not necessarily be ordered.

15. Double-buffering with the PHIGS workstation resource, if supported, requires that the resource be created with the double-buffering mode enabled. To determine if double-buffering can be enabled, use the PEXGetImpDepConstants request (PEXlib functions also call PEXGetImpDepConstants).

16. Servers return their native floating point format as the first entry in the list of supported floating point formats returned by PEXGetEnumeratedTypeInfo.

17. Implementations that do not support the pattern interior style do not guarantee support for the pattern lookup table (a LookupTable error is returned from PEXCreateLookupTable and PEXGetTableInfo). Regardless of support for the pattern interior style, servers which support the PHIGS workstation resource also support the pattern LUT.

18. Implementations return a Value error from PEXCreateLookupTable and PEXGetTableInfo for attempted use of vendor specific lookup tables that are not supported.

19. Double buffering via MBX might not be available across all devices. The application should always first inquire whether MBX can create more than one image buffer. If only one MBX image buffer can be created, then the application could also inquire support for rendering to pixmaps which, if supported, could be used to accomplish the same effect as double-buffering by rendering the pixmap and copying to the window.

20. Not all servers support arbitrary color approximation settings. To address this, these conventions should be followed.

    a. A PEX server should possess either or both of the X standard colormap properties: RGB_BEST_MAP and RGB_DEFAULT_MAP. These properties contain entries that describe IDs of Colormaps that are supported for PEX Color Approximation by that server. PEX clients can use these properties to find Colormaps to share, and/or to derive PEX Color Approximation setups that are supported. In all respects, the content and usage of these properties under PEX should conform to the ICCCM.

    b. A PEX server that does not support arbitrary valid Color Approximation configurations should provide the QueryColorApprox escape. PEX clients can use this escape to verify support for a specific Color Approximation configuration, and/or to obtain a list of one or more supported configurations.

Other Useful References

A

For additional information about PEX and PEXlib, see:

- PEXlib Specification and C Language Binding (X Consortium, 1992)
- PEX Protocol Specification (X Consortium, 1992)
- PEX Protocol Encoding (X Consortium, 1992)
- PEXlib Programming Manual by Tom Gaskins (O'Reilly and Associates, 1993)
- PEXlib Programming Reference by Tom Gaskins (O'Reilly and Associates, 1993)
- PEXlib: A Tutorial by Paula Womack (Prentice-Hall, 1993)
- PEXlib: A Reference Manual by Mark Graff (Prentice-Hall, 1993)
- Building Applications with PEXlib by Jan Hardenbergh (Prentice-Hall, 1994)