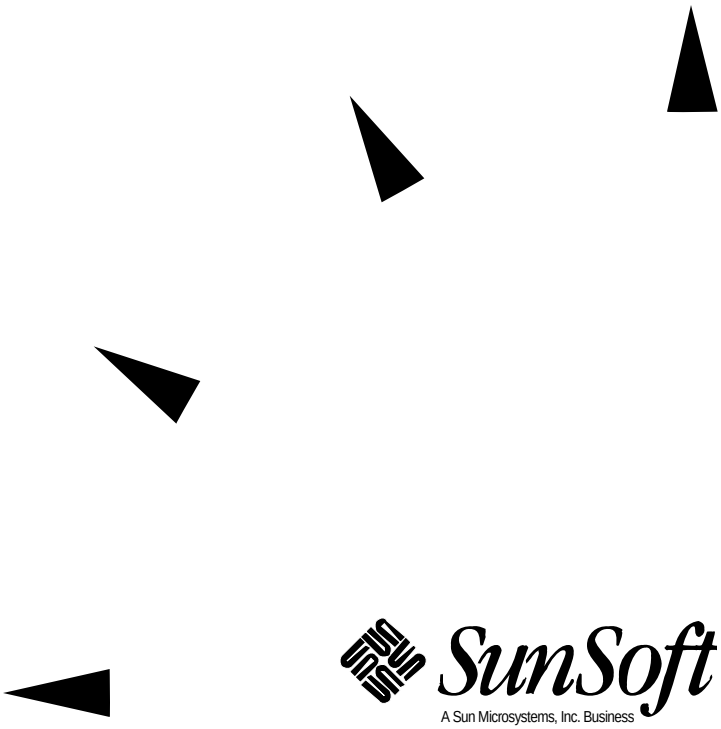
Solaris VISUAL Overview for Driver Developers



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

SunSoft
A Sun Microsystems, Inc. Business

Contents

Solaris VISUAL Overview for Driver Developers

This document provides an introduction to the Solaris™ Graphics Porting Interfaces (GPIs), the low-level interfaces for porting the Solaris graphics environment — Solaris VISUAL™ — to frame buffers, graphics accelerators, and other graphics devices. All the components and porting options are covered at a high level, and a guide to the detailed Solaris VISUAL Driver Developer Kit (DDK) documents is provided.

Solaris VISUAL

Solaris VISUAL is the Solaris windows and graphics environment, including the graphics libraries and the X11R5 server. Solaris VISUAL includes Application Programming Interfaces (APIs) for a wide variety of graphics functionality, including 2-D and 3-D geometric graphics, imaging and digital video, stencil/paint style graphics (PostScript™), and basic pixel graphics (X11). Figure 1 illustrates the components of Solaris VISUAL at a high level. Applications may be built using a set of application libraries, including libraries from SunSoft™ and from third parties, as well. These application libraries are built on a set of foundation libraries that are part of the Solaris system — one for each major area of graphics functionality. Some of these foundation libraries are also available directly to application developers. Each foundation library defines a Graphics Porting Interface (GPI) that is the interface for porting the library to hardware devices. A more extensive introduction to Solaris VISUAL may be found in the *Solaris VISUAL* White Paper.

Device developers may port their device to the Solaris VISUAL environment by porting one or more of the Solaris GPIs to the device. The Solaris DDK provides information enabling developers to do this. A device might be a frame buffer, graphics accelerator, input device, frame grabber, image compression device, etc. In some cases, a device might also be a software component, e.g., an optimized version of a compression algorithm or rendering pipeline. The Solaris GPI interfaces can be used to support such “software devices.”

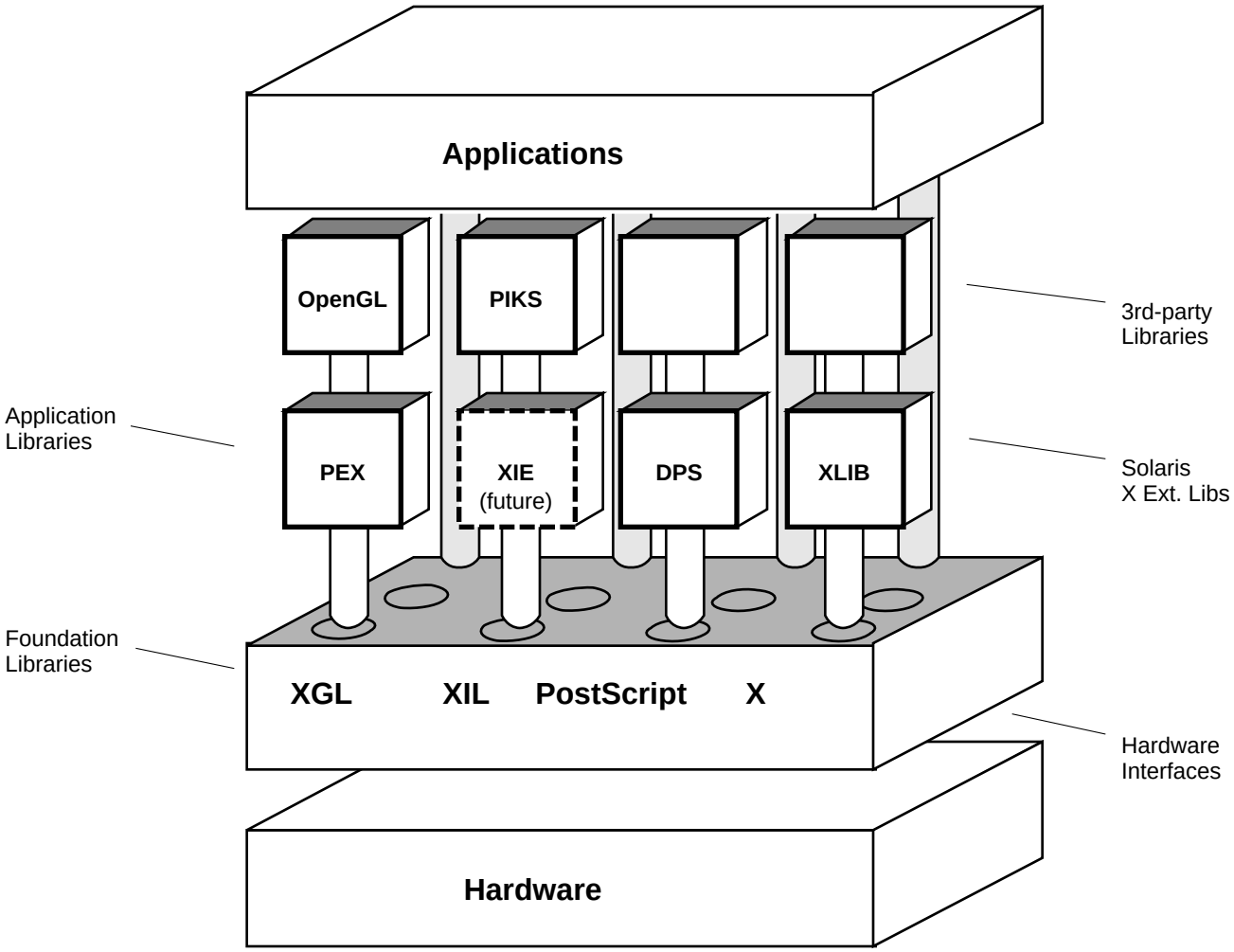


Figure 1 Solaris VISUAL Components

The Solaris VISUAL GPIs are designed to satisfy the following requirements of Independent Hardware Vendors (IHVs) and OEMs:

1. The ability to port the Solaris VISUAL environment using only interface documentation, test/verification suites, and sample code provided in the DDK (i.e., no source licensing required)
2. The ability to port at a functional level appropriate to the device (e.g., utilize Solaris-provided software implementations where devices lack certain functionality and make full use of device capabilities when available)
3. The ability to get a simple port going with minimal effort and to tune the port with additional effort
4. The ability to produce hardware and associated device software that can be installed in the field by end users

Subsequent sections of this overview provide further explanation of the Solaris VISUAL architecture, a guide to the components of the DDK, and a few hints to developers along the way.

Solaris VISUAL Architecture

Figure 2 shows a more detailed diagram of the Solaris VISUAL architecture. A set of standard APIs are provided for application developers — PEXlib, XIElib (future), DPSlib, and Xlib. These APIs are all based on the X Window System in that they are bindings to the X11 protocol (Xlib) or extensions to it (PEXlib, XIElib, and DPSlib). The Solaris environment provides a window system server based on the X11R5 server from the X Consortium. The server provides resource management for the graphics resources of a machine and enables network distributed graphics by accepting connections for communication of graphics requests using the X11 protocol or an extension (see below).

The implementation of the standard APIs relies on a set of four foundation libraries — XGL™, XIL™, the PostScript foundation, and the X foundation. Of these, only XGL and XIL are accessible to application developers.

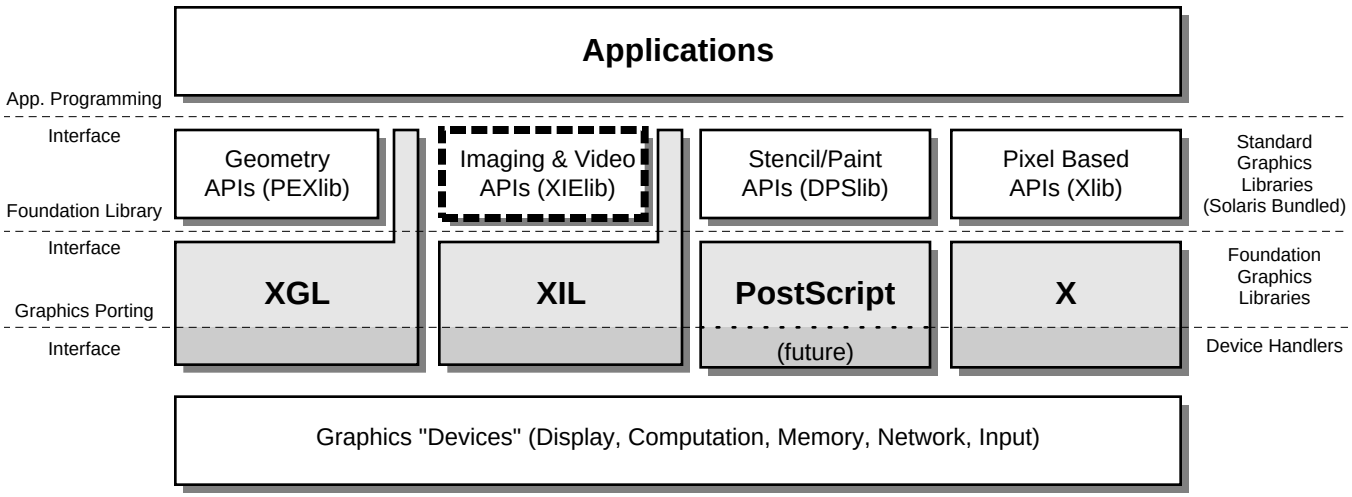


Figure 2 Solaris VISUAL Architecture

Each foundation library has a GPI appropriate to the functionality covered by the library. An implementation of a GPI for a particular device is called a device handler (this is different from a kernel device driver — see below). Device handlers are dynamically-loadable shared object modules. They are located and loaded as needed as applications or the window system execute. IHVs can distribute device handlers for their devices and users can install them on their machines for operation with the Solaris device-independent libraries and window system. It is not necessary for an IHV to port to each GPI — only the X GPI (DDX) is required (see “Porting Strategies”). An IHV can optionally choose to implement additional GPIs to enhance performance in particular areas of functionality (e.g., the XGL GPI for 3D rendering performance).

Table 1 shows for each graphics area (geometry, imaging, stencil / paint, and basic pixel graphics) the corresponding standard API (X extension library), foundation library, and wire protocol. (Note that these APIs may be mixed within an application by drawing to the same or different windows using several APIs.)

Table 1 Solaris VISUAL Components

Graphics Area	Foundation Library	Standard API (X Ext. Lib)	Wire Protocol
Basic Pixel Operations	X	Xlib	X11
Stencil/Paint Graphics	PostScript	DPSlib	DPS
Imaging & Digital Video	XIL	XIElib (future)	XIE (future)
2-D/3-D Geometry	XGL	PEXlib	PEX

As indicated in Figure 2, Solaris 2.4 does not fully implement Solaris VISUAL Architecture. XIE and XIElib are still under development by the X Consortium, but XIL is available for writing imaging applications. In addition, no PostScript GPI is available in Solaris 2.4. The PostScript foundation utilizes the X GPI (DDX) in this release. (This is true both for device handlers provided in Solaris and for those developed by IHVs. See the section on "Graphics Porting Interfaces.") A PostScript GPI may be provided in a future release.

The types of devices that are supported by a GPI vary. All three GPIs (XGL, XIL, and X) support rendering to system memory, dumb frame buffers, and graphics accelerators. In addition, the X GPI provides an interface for input devices (tablets, button and dial devices, etc.). The XIL GPI supports image input devices, compute devices, storage devices, and compression devices, in addition to displays.

Distributed Graphics

The X Window System implements distributed graphics using a traditional client/server model as shown in Figure 3. In a strict implementation of this model, the window server and the client application are separate UNIX processes. The client may be on the same machine as the server or a different machine. Client and server communicate only via IPC mechanisms, and the communication involves network traffic when they run on different machines. The client calls the X graphics libraries which emit protocol requests to the server. The server accepts X11 protocol requests (or protocol extension requests) and invokes device-

dependent code (e.g., DDX) to render the specific primitives. In the Solaris implementation, rendering is accomplished by one of the foundation libraries and a device handler associated with the attached display hardware. Solaris VISUAL supports distributed graphics across a network in this fashion.

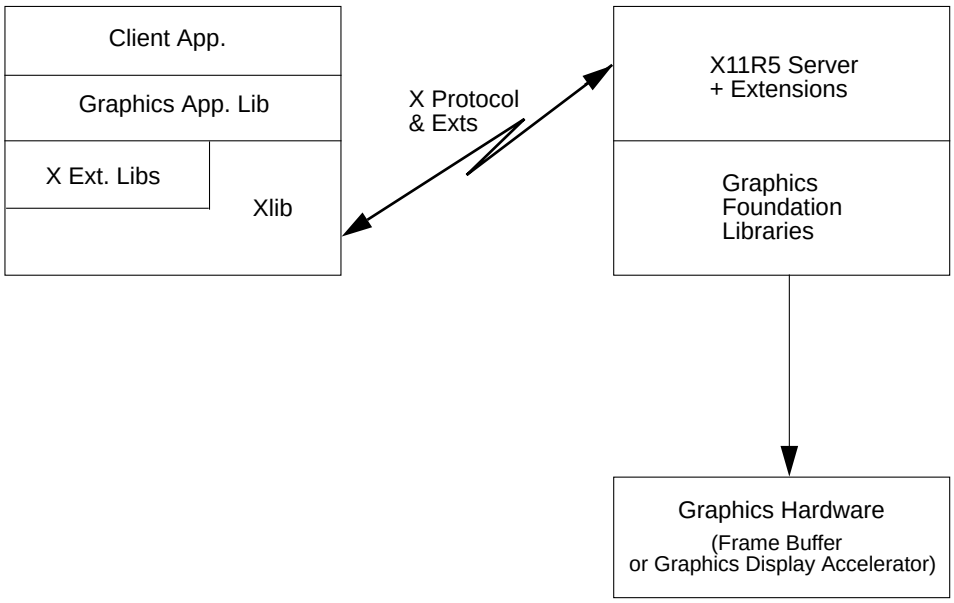


Figure 3 Distributed Graphics Client/Server Model

Direct Graphics Access

When the client and server processes are on the same machine, it is possible to achieve much higher performance on frame buffers and graphics display accelerators in some cases by avoiding IPC communication to the server and directly accessing the graphics hardware from the client process. Solaris VISUAL implements Direct Graphics Access (DGA) to take maximum advantage of high performance graphics devices (see Figure 4). DGA is a set of shared memory communication and synchronization primitives that allow foundation libraries and device handlers to bypass the server but still maintain screen integrity as though all graphics were still rendered by the server.

DGA interfaces include a client interface for writers of foundation libraries and device handlers and a device porting interface. The client interface is chiefly of interest to programmers writing device handlers for XGL or XIL, since these are the foundation libraries which utilize DGA in Solaris 2.4. The device porting interface must be implemented by IHVs who implement DGA for their devices, and is part of the X GPI. Both of these interfaces are documented in the *OpenWindows Server Device Developer's Guide.*

An IHV is not required to implement DGA for a device. Some devices may not see great performance increases from it. On some devices, it may be difficult to efficiently implement, since it requires device context switching. For more information, see the section on "Porting Strategies."

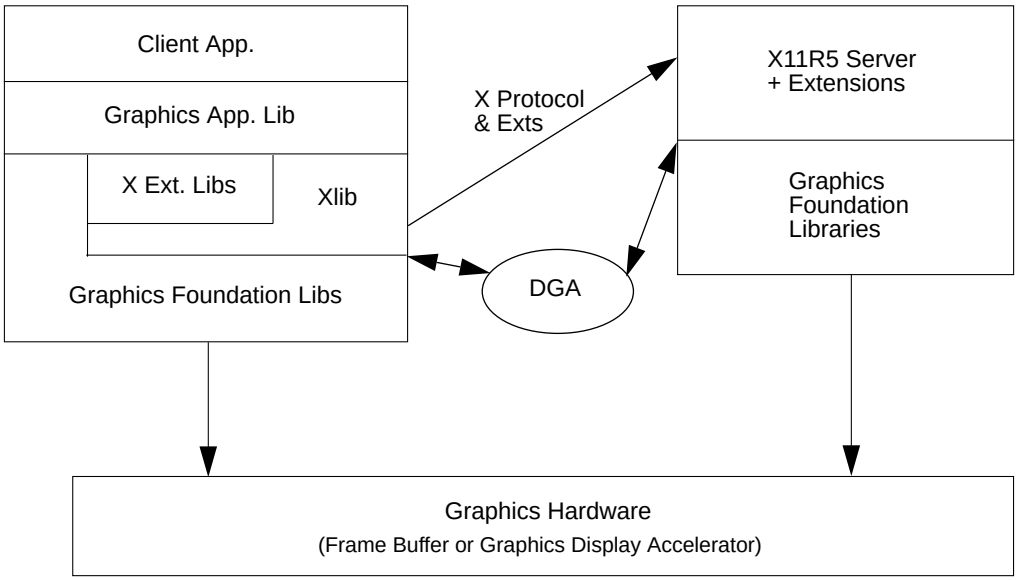


Figure 4 Direct Graphics Access Acceleration for Local Case

Graphics Porting Interfaces

Figure 5 illustrates the GPI architecture for Solaris VISUAL. Note that Solaris 2.4 does not contain a PostScript GPI. Instead, PostScript uses the X GPI. The figure shows how this is possible. The XGL, XIL, and PostScript foundation libraries all have the ability to render via the X foundation. This capability establishes the minimal port feature of Solaris VISUAL — it is only necessary to port DDX in order to get all of Solaris VISUAL running.

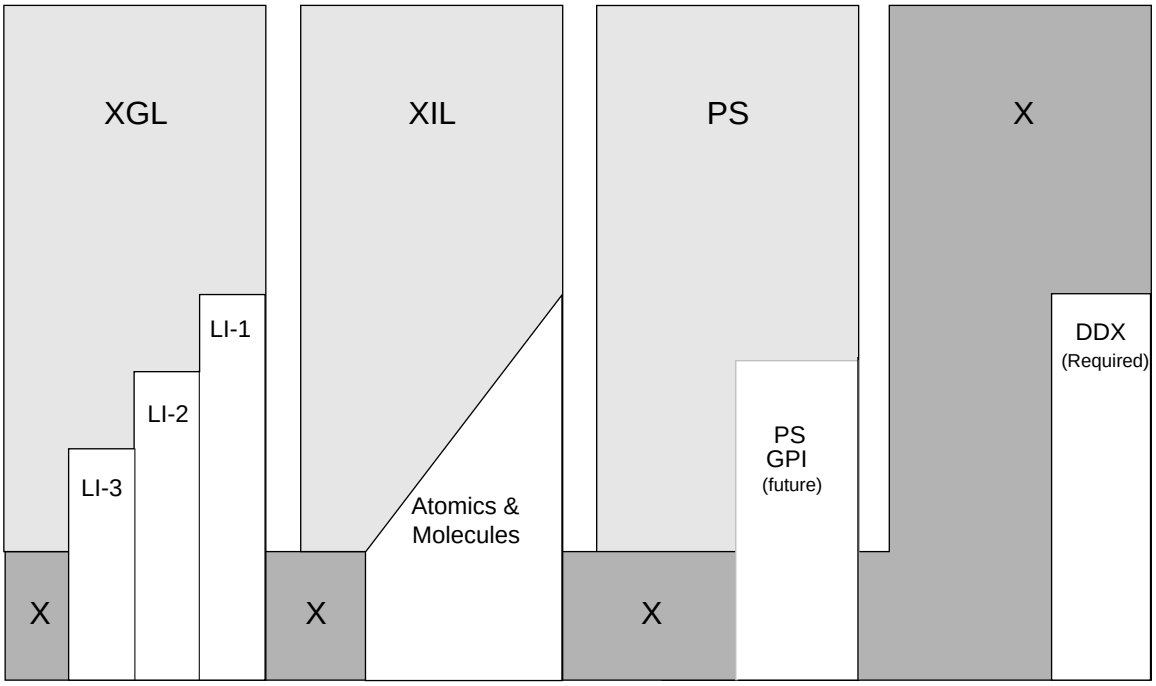


Figure 5 Graphics Porting Interfaces

DDX is the “device dependent X” interface defined in the X Consortium sample server. DDX (the X GPI) is described in the *OpenWindows Server Device Developer’s Guide*, which documents all the interfaces SunSoft provides to IHVs for porting the X11R5-based window server. These interfaces include DDX and Solaris value-added enhancements to DDX for support of advanced graphics hardware features. Input device interfaces are covered in that guide as well.

The XIL GPI is documented in the *XIL Device Porting and Extensibility Guide.* XIL defines a set of low-level atomic imaging functions. An IHV can replace atomic memory-based functions with device-specific functions. In addition, sequences of atomic functions, called *molecules*, can be provided in optimized form for use by XIL. These molecules can greatly improve the performance of an XIL device port.

The XGL GPI consists of three levels of device pipeline interface, called LI-1, LI-2, and LI-3. These three levels correspond to different cut points in the typical geometric graphics pipeline. IHVs may implement different GPI functions at different levels to tailor a port for a particular device. XGL provides a software pipeline implementation of LI-1 and LI-2 for use by device porters. The XGL GPI is described in two documents in the DDK. The *XGL Architecture Guide* describes the internal XGL architecture as it relates to device pipelines. The *XGL Device Pipeline Porting Guide* provides the interface definitions and detailed explanations for LI-1, LI-2, and LI-3.

Note – All of the GPIs documented in the Solaris 2.4 DDK are ‘‘unstable’’ interfaces. This means that they are not yet mature enough to guarantee binary or source compatibility from release to release. No gratuitous changes will be made to the interfaces, but experience and feedback from IHVs may lead to changes in the interfaces. If you are using these interfaces, you can forward your comments, feedback, and queries on interface revisions to:

```
visual-ddk-feedback@sun.com
```

This allows us to give you timely information on changes that may be made in future releases of Solaris. SunSoft’s goal is to solidify the interfaces as soon as feasible, so that both source and binary compatibilty can be guaranteed across releases. Once this happens, the interfaces will evolve in compatible ways as new features are added.

The GPI interfaces are versioned with major and minor numbers. The implementation of the versioning has implications for writers of device handlers. Please consult the documents for each GPI for versioning information. In Solaris 2.4, the GPI version numbers are as follows:

DDX — *1.1*
XGL GPI — *4.0*
XIL GPI — *1.2*

Porting Tasks

Porting the Solaris VISUAL environment to a new device generally involves the following tasks:

1. **Write a loadable kernel device driver.**
2. **Write a loadable DDX device handler.**
3. **Optionally, write loadable XGL and XIL handlers as well for higher performance on geometric graphics and imaging applications.**
4. **Provide configuration information for inclusion in configuration files used by the foundation libraries to locate and load device handlers.**
5. **Satisfy platform-specific hardware requirements.**

The kernel device driver generally handles device functions that cannot be safely or conveniently done in device handlers. Since device handlers are loaded into executing processes, they run in *user* mode, not *kernel* mode. Actual graphics rendering is typically not done by the device driver — it is the responsibility of the handlers. Examples of functions usually performed by a kernel driver include initializing the device, performing simple (policy- free) allocation of device resources, mapping the device into user process address spaces, updating control registers that are unsafe to map into user address spaces, performing time-critical device update functions (e.g., those which must synchronize with vertical retrace), tracking hardware cursors, and context switching stateful graphics devices.

The DDK document *Writing Device Drivers* provides the information needed to develop a graphics device driver. The services provided by a device driver are usually used only by foundation library device handlers for the device. There

are a small number of `ioctl` calls used by device-independent code in Solaris VISUAL. These are discussed in the first document above. Otherwise, the driver interface is private to the device.

Porting strategies for writing device handlers are covered in the following section.

Configuration information needed by each foundation library is covered in the GPI documents referred to earlier. A device porter only needs to provide configuration information for a foundation library if he provides a device handler for the library. In Solaris 2.4, only X and XIL define configuration files.

Platform-specific hardware requirements include issues such as bus interfaces and boot requirements. For SBus-based SPARC® systems, this information is covered in the *OpenBoot Command Reference Manual* and *Writing FCode Programs*. Note that on *x86* platforms, the system is initially booted with the display adapter in text mode. The X Windows server (DDX device handler) is responsible for initializing the hardware into the appropriate graphics mode at server startup. Many *x86* display adapters cannot be probed reliably to determine their physical characteristics and configuration; therefore, this information is generally stored in the server configuration file, `OWconfig`. (For more information on `OWconfig`, see the *OpenWindows Server Device Developer's Guide*.) Also note that the calls to the PC's BIOS ROM from a Unix process such as the server are not supported in this release. Some device initialization may be performed in the graphics kernel device driver (for example, through the server invoking `ioctl()` calls). The manner of partitioning the initialization roles between the server and the kernel driver is an implementation choice left to the developer. In all cases, however, the DDX device handler drives the initialization of the graphics display adapter into graphics mode, and restores text mode as the server is exited.

Porting Strategies

The Solaris VISUAL libraries and window server are structured such that an implementation of DDX enables all libraries, the server, and server extensions to run.

The reason to implement the XGL or XIL GPIs is *performance*. A simple port of Solaris VISUAL using only the DDX interface will not perform as well as a tuned port involving the XGL and XIL GPIs for devices that are able to

accelerate higher-level geometric or imaging functions. The decision about whether to implement one or both of these GPIs is a tradeoff between porting effort and performance.

Even within a GPI, several choices are available to an IHV. For example, the SunSoft DDX interface has several capabilities beyond those of the X Consortium sample server. These are mostly for support of advanced graphics hardware with features such as window ids, multiple plane groups, multiple hardware colormaps, etc. A simple port not utilizing these additional interfaces is possible even on devices that have one or more of these features. Addtional effort is required to take advantage of these interfaces.

The XIL GPI offers several opportunities for optimizing performance on particular devices. Any of the atomic imaging operations can be implemented in a device handler. Furthermore, sequences of operations, called *molecules*, can be selected for device-specific implementation. The best choices for performance are highly device-dependent, and each atomic function or molecule represents additional porting effort. The *XIL Device Porting and Extensibility Guide* provides advice on making these choices.

The XGL GPI offers interfaces appropriate for devices ranging from dumb frame buffers to high-end accelerators. The selection of the LI-1, LI-2, or LI-3 interfaces is not a global decision — it can be made for each primitive. Thus, an IHV can do a tuned port for a device in an efficient manner, and can start with a simple port and optimize later.

The Solaris DDK includes sample code for several device ports of each GPI. These are intended to get device porters started on their ports.

One implication of porting either the XGL or XIL GPIs to a frame buffer or graphics display accelerator is that an IHV must also implement the DGA functionality in the DDX GPI, because both XGL and XIL GPIs require the client DGA interface referred to above. A further choice for IHVs utilizing DGA is whether to implement graphics device context switching. Generally, devices for which XGL or XIL ports are worthwhile will have device state. Since DGA allows multiple processes to access the device, some form of context management must be implemented (see the next paragraph for exceptions to this). The *Writing Device Drivers* document describes system facilities that may be used to implement graphics context switching in a kernel device driver (although an IHV could implement context switching in some other fashion than through these facilities).

One circumstance for which device context switching is not required is devices that are purely dumb frame buffers, i.e., a device handler does not need to set up state for graphics operations. Another example is a device for which an IHV cannot easily provide context switching between multiple processes because of device limitations (e.g., write-only device registers), but which has functionality that can greatly accelerate XGL or XIL. In a future release of Solaris, IHVs will be able to choose to support DGA, but to disable it for any process except the window server. This means, for example, that both DDX and XGL device handlers could be active in the server process, but no other process would load device handlers for the device (client processes would use X11 or extension protocols to communicate graphics requests to the server, even when on the local machine). In this case, the device handlers within a single process would cooperate to maintain the device state, but only a single process would touch the device. This could make sense for an XGL handler as a way to accelerate the PEX extension in the server. This choice is not available to IHVs in Solaris 2.4.

Porting Environment

The Solaris DDK provides sample `Makefiles` along with the sample code.

For DDX porting, an ANSI C compiler is required. Test programs can be written using the `include` files and documentation for Xlib and DPSlib provided in the Solaris runtime product.

For XGL or XIL porting using the Solaris 2.4 DDK, the SPARCompiler™ 2.0.1 C++ compiler or ProCompiler™ 2.0.1 C++ compiler is required, since these GPIs are C++ interfaces. In addition, the Solaris Software Developer Kit (SDK) is needed for writing test programs, since it contains the necessary API `include` files and documentation.

Testing a Device Handler

Test suites are also provided in the DDK for XGL and XIL device handlers. These are described in the *XGL Test Suite User's Guide* and the *XIL Test Suite User's Guide*, respectively.

Xlib test suites (which exercise DDX device handlers) are publicly available. See the *OpenWindows Server Device Developer's Guide*.

Software Distribution

Device drivers and handlers developed by IHVs should be distributed to customers as a set of Solaris 2.*x* packages. For general information on creating packages, see the *Application Packaging Developer's Guide.*

Specific information on naming conventions and installation directories for kernel drivers, DDX handlers, XGL handlers, and XIL handlers is provided in *Writing Device Drivers, OpenWindows Server Device Developer's Guide*, *XGL Device Pipeline Porting Guide*, and the *XIL Device Porting and Extensibility Guide*, respectively.

References

The following references are all included as part of the Solaris 2.4 documentation:

- *Solaris VISUAL* White Paper
- *OpenWindows Server Device Developer's Guide*
- *XIL Device Porting and Extensibility Guide*
- *XGL Architecture Guide*
- *XGL Device Pipeline Porting Guide*
- *Writing Device Drivers*
- *OpenBoot Command Reference Manual*
- *Writing FCode Programs*
- *XGL Test Suite User's Guide*
- *XIL Test Suite User's Guide*
- *Application Packaging Developer's Guide*