SGI® OpenGL Multipipe™ User's Guide

Version 2.5.2

007-4318-017

CONTRIBUTORS

Written by Ken Jones and Jenn Byrnes

Illustrated by Chrystie Danzer

Production by Karen Jacobson and Ken Jones

Engineering contributions by Ye Cong, Craig Dunwoody, Bill Feth, Alpana Kaulgud, Claude Knaus, Ravid Na'ali, Jeffrey Ungar, Christophe Winkler, Guy Zadicario, and Hansong Zhang

New Features in This Release

OpenGL Multipipe 2.5.2 contains the following enhancements:

- Support for compositor pixel averaging to provide transparent antialiasing
- Support for multiple compositors
- **glDrawPixels()** clipping now enabled by default (not tied to geometry culling)
- Improved application stability and bugfixes

Record of Revision

Version	Description
001	August 2000 Beta release.
002	November 2000 Updated for release 1.0 of the OpenGL Multipipe product.
003	February 2001 Updated for release 1.1 of the OpenGL Multipipe product. New features: - Increased overall performance - Support for overlapping screens, as in SGI Reality Center facilities
004	May 2001 Updated for release 1.2 of the OpenGL Multipipe product. New features: - Transparent OpenGL Pipe Management - Subset of multipipe applications made aware of Xinerama
005	August 2001 Updated for release 1.3 of the OpenGL Multipipe product. New features: - Enhanced Support for Multithreaded Applications - Enhanced tgl Script
006	November 2001 Updated for release 1.4 of the OpenGL Multipipe product.

Bugfixes:

- Enhanced GLX conformance for context manipulation

- Support for pixmaps, pbuffers, and GLXWindows

Beta features:

- Curved Screen Support

This allows you to run applications on a non-planar Reality Center in immersive mode by adapting the 3D projections to the display layout.

- Window Manager Support for Aware Windows

All applications started in aware mode can now be under window manager control by using the customized window manager included with this release.

007 February 2002

Updated for release 1.4.1 of the OpenGL Multipipe product.

Broader application support

Bugfixes:

- Enhanced OpenGL conformance for applications using **glCallList()** within another display list

- Stability improvements to (beta) aware window manager

008 April 2002

Updated for release 1.4.2 of the OpenGL Multipipe product.

- Broader application support

- Stability improvements to (beta) aware window manager

009 October 2002

Updated for release 2.1 of the OpenGL Multipipe product.

Features:

Replacement of the Transparent OpenGL (TGL) layer with a proxy render library and render servers

Support for additional servers. The list of supported servers now includes the following:

- Silicon Graphics Onyx

- Silicon Graphics Onyx2

- SGI Onyx 3000, InfiniteReality

- SGI Onyx 3000, InfinitePerformance
- Silicon Graphics Octane2

010 May 2003

Updated for release 2.1.2 of the OpenGL Multipipe product.

Features:

- Performance enhancements over the OpenGL Multipipe 2.1.1 and 2.1 releases
- Increased application compatibility for vertex array applications
- Option of running in master render mode or slave-only mode
- Support for compositor-based systems
- Seamless cursor movement across overlapped or composited screen regions (IRIX 6.5.20 or later required)
- Support for SGI-SCREEN-CAPTURE and ReadDisplay X extensions in SGI Xinerama mode (IRIX 6.5.20 or later required)
- An API introduced for integration of multipipe applications with SGI Xinerama
- Hardware swap-synchronization option (Swap Ready, Genlock)
- Support for additional platforms. The list of supported visualization systems now includes the following:
  - o SGI Onyx 3000 series with InfinitePerformance graphics
  - o SGI Onyx 3000 series with InfiniteReality graphics
  - o SGI Onyx 350
  - o Silicon Graphics Octane2
  - o Silicon Graphics Onyx
  - o Silicon Graphics Onyx2

011 July 2003

Updated for release 2.2 of the OpenGL Multipipe product.

Features:

- Performance enhancements over the 2.1.2 release including the following:
  - o Pixel drawing enhancements
  - o Geometry culling, which can improve application performance for some applications beyond what can be achieved on a single pipe
- Support for the DMX (Distributed Multihead X) meta display server

- Support for DMX and SGI Xinerama meta display servers by the MPC API for multipipe-aware applications
- Support for additional platforms. The list of supported visualization systems now includes the following:
  - o SGI Onyx 3000 series with InfinitePerformance graphics
  - o SGI Onyx 3000 series with InfiniteReality graphics
  - o SGI Onyx 350
  - o Silicon Graphics Octane2
  - o Silicon Graphics Onyx
  - o Silicon Graphics Onyx2
  - o Silicon Graphics Onyx4 UltimateVision

012 December 2003

Updated for release 2.3 of the OpenGL Multipipe product.

Features:

- Performance enhancement options including (disabled by default):
  - o Geometry culling to optional, additional OpenGL clip planes
  - o OpenGL viewport clipping and geometry culling to improve fill and geometry performance when using a compositor
  - o Spatial partitioning of large display lists for efficient geometry culling
- Baseline performance enhancements over OpenGL Multipipe 2.2
  - o Swap synchronization and frame latency control improvements between slave rendering processes
  - o Improved performance when using geometry culling
- Support for Onyx4 OpenGL extensions
- Support for additional platforms. The list of supported visualization systems now includes the following:
  - o SGI Onyx 3000 series with InfinitePerformance graphics
  - o SGI Onyx 3000 series with InfiniteReality graphics
  - o SGI Onyx 350
  - o Silicon Graphics Octane2
  - o Silicon Graphics Onyx
  - o Silicon Graphics Onyx2
  - o Silicon Graphics Onyx4 UltimateVision
  - o Silicon Graphics Tezro

013 April 2004

Updated for release 2.3.1 of the OpenGL Multipipe product.

Enhancements over OpenGL Multipipe 2.3:

- Capability to expand the OpenGL viewport size beyond the single-pipe limit
- Support for overlay visuals in the DMX proxy X server if supported by underlying X servers
- Support for applications that use GLX pbuffers or GLX pixmaps
- Improved application compatibility

014 October 2004

Updated for release 2.4 of the OpenGL Multipipe product.

Enhancements over OpenGL Multipipe 2.3.1:

- Support for most OpenGL Multipipe features on Silicon Graphics Vizualization System for Linux 64-bit platforms
- Small object culling (small relative to screen space)
- Drawpixels culling
- Improved application compatibility

015 November 2004

Updated for release 2.4.1 of the OpenGL Multipipe product.

Enhancements over OpenGL Multipipe 2.4:

- Improved daemon processing
- Improved support for static tiling modes of hardware compositors
- Enhanced application compatibility

016 January 2005

Updated for release 2.5 of the OpenGL Multipipe product.

Enhancements over OpenGL Multipipe 2.4.1:

- Support for vertex buffer objects
- The optional use of the omprun command on selected platforms
- The use of X11 resources to control performance features as an alternative to command-line options on omprun
- Improved application stability

017	June 2005 Updated for release 2.5.2 of the OpenGL Multipipe product. Enhancements over OpenGL Multipipe 2.5.1: - Support for compositor pixel averaging to provide transparent antialiasing - Support for multiple compositors **- glDrawPixels()** clipping now enabled by default (not tied to geometry culling) - Improved application stability and bugfixes

Contents

About This Guide

This guide describes the OpenGL Multipipe product, which allows you to run single-pipe applications in a multipipe environment without modification. You can seamlessly move single-pipe application windows across the single logical display that OpenGL Multipipe creates from multiple pipes. Both multipipe applications and single-pipe applications run concurrently.

Related Publications

The following SGI documents contain additional information that may be helpful:

- *InfiniteReality Video Format Combiner User's Guide*
- *POWER Onyx and Onyx Rackmount Owner's Guide*
- *SGI Scalable Graphics Compositor User's Guide*
- *IRIX Admin: Software Installation and Licensing*
- *SGI ProPack for Linux Start Here*

These books might also be helpful:

- Dave Shreiner, OpenGL Architecture Review Board, Mason Woo, Jackie Neider and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4*. Reading, MA: Addison Wesley Longman Inc., 2003. ISBN 0-321-17348-1.
- Nye, Adrian, *Volume One: Xlib Programming Manual*. Sebastopol, California: O'Reilly & Associates, Inc., 1992.

Obtaining Publications

You can obtain SGI documentation in the following way:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, select **Help** from the Toolchest, and then select **InfoSearch**. Or you can type `infosearch` on a command line.
- On IRIX, you can also view release notes by typing either `grelnotes` or `relnotes` on a command line.
- You can also view man pages by typing man< *title*> on a command line.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages and programming language structures.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
interface	This font denotes the names of graphical user interface (GUI) elements such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, fields, and lists. Functions are also denoted in bold with following parentheses.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names.

Right angle brackets (>)	These brackets indicate a path through menus to a menu option. For example, "**File > Open**" means "Under the **File** menu, choose the **Open** option."

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, contact SGI. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Use the Feedback option on the Technical Publications Library Web page:

  http://docs.sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  Technical Publications
  SGI
  1500 Crittenden Lane, M/S 535
  Mountain View, CA 94043-1351

SGI values your comments and will respond to them promptly.

OpenGL Multipipe Overview

This overview of OpenGL Multipipe consists of the following sections:

- "What OpenGL Multipipe Provides"
- "Architecture of OpenGL Multipipe"
- "Components of OpenGL Multipipe"
- "Supported Platforms"

What OpenGL Multipipe Provides

SGI has always been focused on high-end graphics solutions. The Onyx family of scalable visualization supercomputers allows you to have multiple graphics pipes on one single-system-image machine in order to reach new visualization performances. These multipipe systems are commonly used to drive expanded visualization systems such as SGI Reality Center facilities. OpenGL Multipipe extends the use of these powerful supercomputers to a broad spectrum of graphics applications without the requirement of modifying the applications.

Many existing graphics applications—such as Netscape or applications based on Open Inventor, for example—are constrained to run on a single pipe. On these single-pipe applications, you can choose the pipe on which to open the application's windows, but the windows cannot be dragged from one pipe to another. The main reason is that the graphics pipes are separate logical units and are handled by an X server as different, unconnected screens. This means that the X server does not provide any functionality to group multiple screens into a single logical display. A second reason is that OpenGL applications connect directly to a specified graphics pipe and bypass the X protocol layer.

In the past, displaying an application on multiple screens required you to explicitly write the application for that purpose. You had to use tools like the OpenGL Performer or OpenGL Multipipe SDK libraries to help you create these multipipe applications. These tools allow you to explicitly open windows on different screens and to draw into them

using OpenGL. However, this solution lacks consistency. In fact, all of the windows on the different pipes are independent; hence, moving or iconifying one window on one screen will not handle the other windows accordingly.

OpenGL Multipipe has been designed to overcome these difficulties. The goal is to group pipes managed by the X server in order to create a consistent, single virtual screen as shown in Figure 1-1. This means that the applications are unaware of the underlying hardware configuration. Rather, they only know about a single display and behave accordingly.

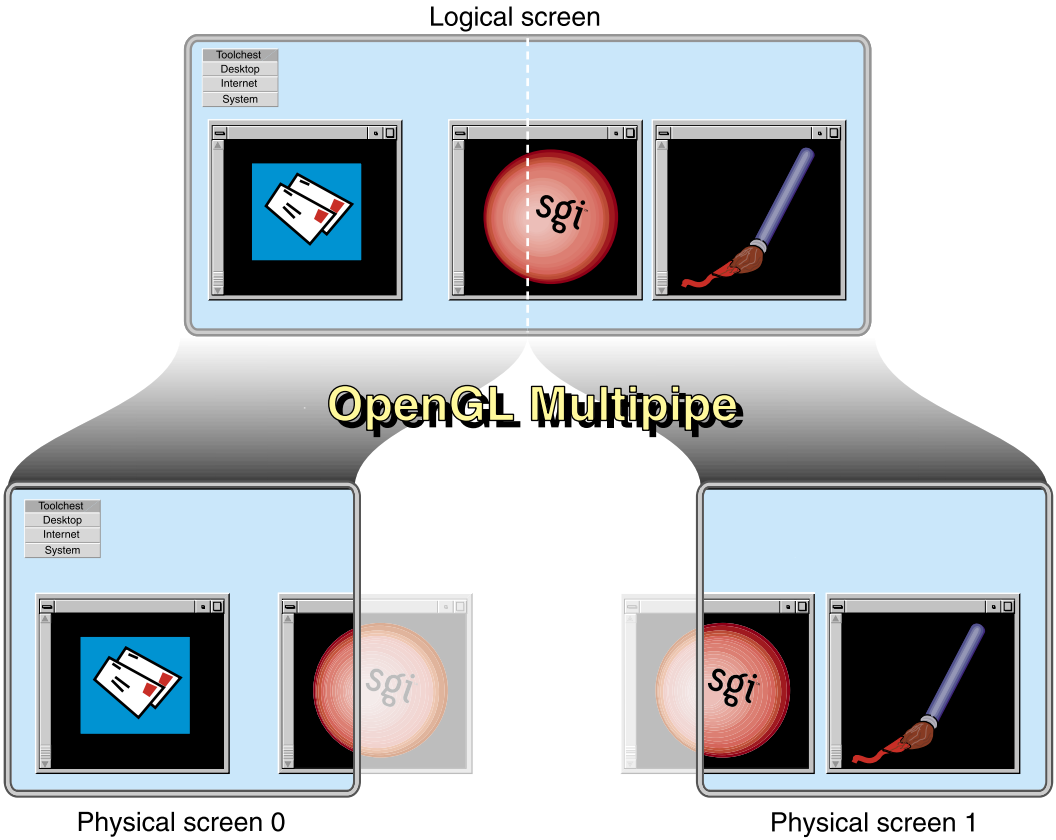

Figure 1-1 OpenGL Multipipe with Non-Overlapping Screens

In contrast to Figure 1-1, if you have screens that overlap each other (such as in an SGI Reality Center wall display with edge blending), OpenGL Multipipe allows you to

specify the amount of this overlap. Figure 1-2 shows the image blended on overlapping screens.

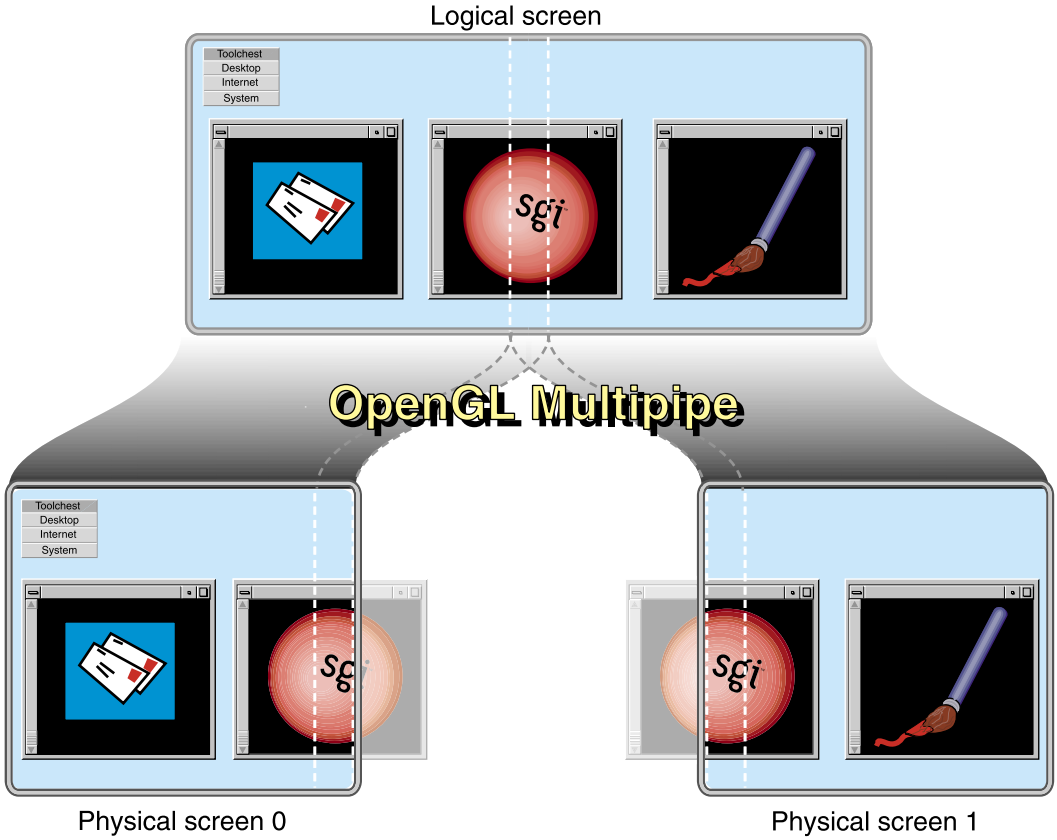

Figure 1-2 OpenGL Multipipe with Overlapping Screens

Note: OpenGL Multipipe does not require you to modify or recompile your application.

Architecture of OpenGL Multipipe

OpenGL Multipipe provides the illusion that an application is rendering 2D (X perspective) and 3D (OpenGL perspective) on a single local pipe when it is actually using one or more pipes. This illusion, or logical display, is created by a set of protocols and proxies coupled with clients and servers, all of which are hidden from the application.

OpenGL Multipipe uses an X proxy layer to hide the physical screen layout from the application. This X proxy layer presents a single logical pipe or meta screen to all applications and allows their windows to be freely moved across or to span any set of pipes. OpenGL Multipipe supports two proxy layers, the Distributed Multihead X (DMX) proxy server, and SGI Xinerama, which is only available on older IRIX platforms.

OpenGL Multipipe also uses an OpenGL proxy library and render servers to send OpenGL calls to each real pipe. Having 3D render servers separate from the 3D proxy library allows the application processing and the rendering to occur in separate processes. This separation aides application compatibility.

Components of OpenGL Multipipe

OpenGL Multipipe has the following components:

- An X proxy layer (the SGI Xinerama extension or the DMX proxy server)
- A 3D proxy render library
- 3D render servers

The X Proxy Layer

For pure X applications—that is, applications that do not use other graphics libraries (such as OpenGL) to draw into their windows—the X proxy layer is all that is needed to enable such applications to run transparently over multiple pipes. This means that windows of applications that are based on the X protocol and that use X extensions can be dragged from one pipe to another and even span multiple pipes. The applications behave as if they are running on a single, large virtual pipe. The X proxy layer hides the real screens from the client applications connecting to it. It distributes to all pipes the X requests from the clients but only sends the clients information about the large virtual display.

The X proxy layer can be either the SGI Xinerama extension or the DMX proxy server. In the case of the SGI Xinerama extension, the X proxy layer is a part of the X server. However, the DMX proxy server is a separate entity. The following sections describe the two options.

The SGI Xinerama Extension

SGI Xinerama is an enhanced version of the standard Xinerama X extension, which groups all screens managed by the X server into one logical screen that it exposes to applications. You must have administrative privileges to enable or disable the SGI Xinerama extension. The SGI Xinerama proxy layer is only available on older platforms, as described in Chapter 5, “Limitations”.

For more information about the SGI Xinerama extension, see the `Xinerama`(3X11) man page.

The DMX Proxy Server

The DMX proxy server, unlike SGI Xinerama, is not part of the X server; it is an X application that behaves like an X server to other X applications. DMX is more flexible than SGI Xinerama both in its supported display geometries and in its ability to act as a proxy for many different X servers. DMX also has built-in support for OpenGL applications through its support of the GLX X extension. This means that DMX will enable X and OpenGL applications to run transparently across multiple pipes. However, DMX’s GLX extension is limited in performance. Hence, it is best to run graphics-intensive applications under the full OpenGL Multipipe environment.

Unlike SGI Xinerama, administrative privileges are not required to start and stop DMX. For more information about the DMX proxy server, see the `Xdmx`(1) man page, which is installed in `/usr/share/omp/doc/user/Xdmx.1.html`.

The 3D (Master) Proxy Render Library

OpenGL applications are X applications that use another graphics library (namely the OpenGL library) to draw into their windows. OpenGL applications open a direct connection to a graphics pipe. This means that the application is bypassing the X protocol (and the X proxy layer, which replicates the the X protocol stream to each pipe) in order to draw in the windows through this direct connection. The X proxy layer, which accounts only for the X protocol, is unable to handle this case.

The *master* proxy library in OpenGL Multipipe handles the OpenGL side of any application. It intercepts OpenGL calls to enable distribution to multiple pipes. To distribute the OpenGL calls, the master library encodes each command using an OpenGL stream protocol and sends the command stream to the *slave* render processes, which render to local pipes on behalf of the application.

Figure 1-3 illustrates the functions of the master proxy library.

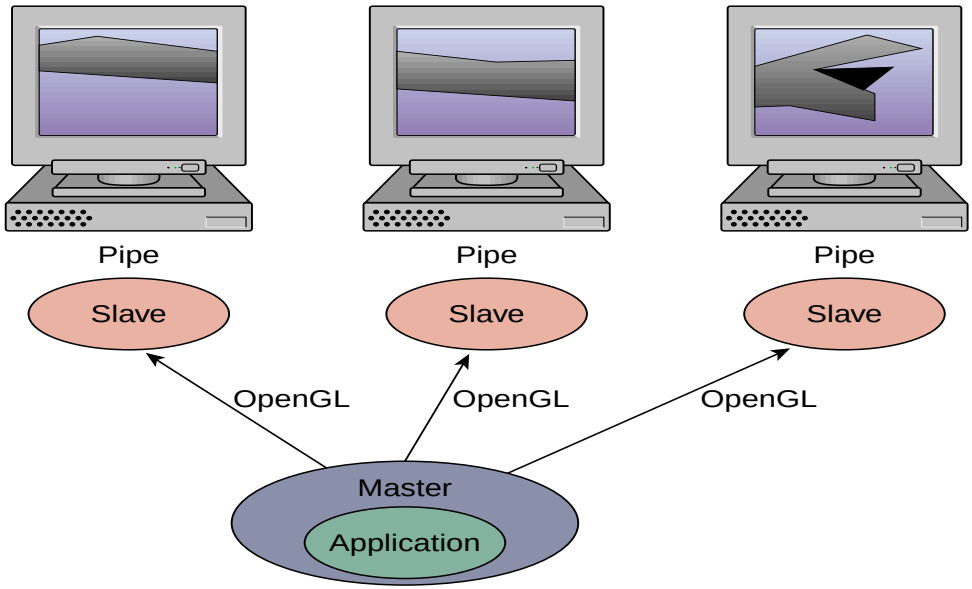

Figure 1-3 Master Proxy Library Functions

In addition to sending an OpenGL stream to each slave, the master also has the capability of rendering directly to a single local pipe in place of a single slave render process (for faster GL state queries), or it may use a local pipe only to track OpenGL state while a slave process renders to that pipe (for improved parallelism). For more information about performance and the master library modes, see Chapter 4, “Optimizing Performance”.

3D (Slave) Render Servers

An application running under the master render library of OpenGL Multipipe communicates its OpenGL commands to slave render server processes. Each slave process parses the OpenGL command stream and executes the commands on the application's behalf. For each rendering application thread of execution, one slave process exists for each screen of the display.

Supported Platforms

Table 1-1 shows the requirements for OpenGL Multipipe 2.5.2.

Table 1-1 Supported Platforms

Server	Processor	Operating System
Silicon Graphics Onyx	MIPS R10000 or later	IRIX 6.5.15 or later
Silicon Graphics Onyx2	MIPS R10000 or later	IRIX 6.5.15 or later
SGI Onyx 3000 with InfiniteReality graphics	MIPS R10000 or later	IRIX 6.5.15 or later
SGI Onyx 3000 with InfinitePerformance graphics	MIPS R10000 or later	IRIX 6.5.15 or later
SGI Onyx 350	MIPS R10000 or later	IRIX 6.5.15 or later
Silicon Graphics Onyx4 UltimateVision	MIPS R10000 or later	IRIX 6.5.24, IRIX 6.5.22 or later plus patch 5448, or IRIX 6.5.27 (for the optional use of `omprun`)
Silicon Graphics Octane2	MIPS R10000 or later	IRIX 6.5.15 or later
Silicon Graphics Tezro	MIPS R10000 or later	IRIX 6.5.15 or later
Silicon Graphics Prism	Intel Itanium 2	SGI ProPack 3 Service Pack 6

Installing OpenGL Multipipe

This chapter describes how you install OpenGL Multipipe on the following two platforms:

Installing on IRIX

This section lists information supplemental to the guide *IRIX Admin: Software Installation and Licensing*. The information listed here is product-specific; use it with the installation guide to install this product.

The following are the prerequisites for installing OpenGL Multipipe on your system:

- Hardware: an Octane2, Onyx, Onyx2, Onyx 3000, Onyx 350, Onyx4, or Tezro system with MIPS R10000 or later processors
- Software:
  - IRIX 6.5.15 or later, generally
  - IRIX 6.5.24 or
    IRIX 6.5.22 or later plus patch 5448 for Onyx4 platforms
    IRIX 6.5.27 for optional use of `omprun` on Onyx4 platforms
  - C++ Standard Execution Environment (c++_eoe)

Note: IRIX 6.5.18 or later is required to use the aware window management feature under SGI Xinerama. IRIX 6.5.20 or later is required to enable duplicate cursor images in screen overlap regions under SGI Xinerama.

To install OpenGL Multipipe, follow these steps:

1. Go to the following URL:

   http://www.sgi.com/software/multipipe/

2. Click on the **Download** button and follow the instructions to download OpenGL Multipipe.

   This installer includes the OpenGL Multipipe libraries and tools described in Table 2-1.

3. Use `inst` or `swmgr` to install OpenGL Multipipe.

   The libraries are provided in two versions:

n32	The new 32-bit libraries. Located in the `/usr/lib32` directory. Usable only on IRIX 6.2 and later operating system releases. The n32 libraries operate at increased efficiency in many situations.
64	The 64-bit libraries. Located in the `/usr/lib64` directory.

   This step installs the file subsystems shown in Table 2-1.

Table 2-1 File Subsystems for OpenGL Multipipe 2.5.2

Subsystem	Description
`omp_eoe.sw.base`	Contains the actual OpenGL Multipipe binaries as well as the script for starting OpenGL applications in OpenGL Multipipe mode.
`omp_eoe.sw64.base`	Contains the OpenGL Multipipe 64-bit software.
`omp_eoe.man.relnotes`	Contains the release notes, located in the following directory: `/usr/share/omp/release_notes/user`
`omp_eoe.man.base`	Contains man pages and other documentation located in the `/usr/share/omp/doc/user` directory.
`omp_eoe.sw.wm`	Contains the aware window manager for multipipe-aware window support under SGI Xinerama.
`omp_eoe.sw.wmp`	Contains the aware window manager proxy for multipipe-aware window support under DMX.
`omp_eoe.sw.xdmx`	Contains the DMX proxy server, `Xdmx`, configuration utilities, and documentation related to the DMX X proxy layer.

Table 2-1 File Subsystems for OpenGL Multipipe 2.5.2 **(continued)**

Subsystem	Description
`omp_eoe.sw.mpc`	Contains libraries, header files, and example code for integrating multipipe applications with the X proxy layer.
`omp_eoe.sw64.mpc`	Contains 64-bit libraries for integrating multipipe applications with the X proxy layer.
`omp_eoe.man.mpc`	Contains documentation related to the OpenGL Multipipe MPC API, which is located in the directory `/usr/share/MPC/doc/developer`.

For any critical updated information, you can check the release notes, `/usr/share/omp/release_notes/user/relnotes.html`, and the OpenGL Multipipe website, http://www.sgi.com/software/multipipe/ .

Installing on 64-Bit Linux

The following are the prerequisites for installing OpenGL Multipipe on your system:

- Hardware: Silicon Graphics Prism visualization system
- Software: SGI Advanced Linux Environment with SGI ProPack 3 Service Pack 6

OpenGL Multipipe for Silicon Graphics Prism systems is installed by default when installing the SGI ProPack package. OpenGL Multipipe contains the following file subsystem:

`sgi-omp-2.`*x*`-`*yyyyy*`.ia64.rpm`

Detailed documentation of the `rpm` utility for managing software installation on Linux is available in the `rpm`(8) man page.

For any critical updated information, you can check the release notes, `/usr/share/omp/release_notes/user/relnotes.html`, and the OpenGL Multipipe website, http://www.sgi.com/software/multipipe/ .

Using OpenGL Multipipe

As described in Chapter 1, OpenGL Multipipe consists of three main components: an X proxy layer, a proxy 3D render library, and 3D render servers. This chapter describes how to effectively use these components with your graphics applications. The following sections describe the pertinent tasks:

For information about other features of OpenGL Multipipe specific to this release, see the release notes in the following file:

```
/usr/share/omp/release_notes/user/relnotes.html
```

Setting up the OpenGL Multipipe Environment

To begin using OpenGL Multipipe, you must enable an X proxy layer. This will cause all applications to see a single logical pipe. To deactivate OpenGL Multipipe, just disable the X proxy layer. Some of the steps required to enable or disable OpenGL Multipipe may require `root` access. This section notes this requirement in the applicable steps.

This section describes the following tasks:

Configuring OpenGL Multipipe with DMX as the X Proxy Layer

DMX will group multiple screens into a logical display. This section describes how you initialize DMX and how to create DMX configuration files.

Initializing DMX

To initialize DMX, do the following:

1. If your system supports SGI Xinerama, ensure that it is not currently enabled.

   To determine if SGI Xinerama is enabled, see the section "Verifying That the OpenGL Multipipe Environment is Enabled" on page 32. If enabled, disable SGI Xinerama and restart the X server.

   Enter the following to disable SGI Xinerama:

   ```
   # chkconfig xinerama off
   ```

   Enter the following as `root` in a command shell to restart the X server:

   ```
   # (/usr/gfx/stopgfx; /usr/gfx/gfxinit; /usr/gfx/startgfx) &
   ```

   The X server has to be restarted for the `chkconfig` change to take effect. With SGI Xinerama disabled, the X server will manage pipes as separate screens.

2. Run DMX on top of the existing X server(s).

   You may do this manually after logging into your desktop or you may configure an `.xsession` script to run DMX immediately upon login.

   To manually initialize DMX, enter the following (`root` access not needed) in a command shell:

   ```
   $ ompstartdmx
   ```

   You can use the flag –`help` for more information about the starting options. If you specify no flags, DMX starts on top of the existing X server and will configure a single large screen that overlays the existing *n* screens such that screen 0 will be the leftmost and screen *n* –1 will be the rightmost. To use a different configuration, such as a vertical configuration, you must provide a DMX configuration file. The following section "Creating DMX Configuration Files"describes how to create such a configuration file.

   By default on IRIX, `ompstartdmx` will run the following clients unless a session script has been specified: `4Dwm`, `toolchest`, and `winterm`. On Linux, only xterm will be started as the session. For more details, see the later subsection "Differences between the IRIX and Linux Versions of ompstartdmx".

   You can configure DMX to run automatically upon login. How you do so depends on your operating system.

   Configuring DMX to Run Automatically on IRIX

   To configure DMX to run automatically upon login, you need to start with an `.xsession` file in your `$HOME` directory. If you do not already have one, you may copy one of the example `.xsession` files provided from the `/usr/share/omp/examples/X11` directory, or you may copy the main system `Xsession` file from `/var/X11/xdm/Xsession` by entering the following:

   ```
   $ cp /var/X11/xdm/Xsession $HOME/.xsession
   $ chmod +w $HOME/.xsession
   ```

   The second command ensures the file is writeable. If not already present, add the following lines at the beginning of your `$HOME/.xsession` file:

   ```
   if [ -x /usr/bin/ompstartdmx -a -z "$SGIOMP_META_DISPLAY" ]; then
       /usr/bin/ompstartdmx -wm none -session /var/X11/xdm/Xsession
       exit
   fi
   ```

   Note that the `if` clause is necessary to prevent infinite recursion. Also, note that –`wm none` is only needed if your `.xsession` script starts a window manager by itself (which is the case if you copied the system `Xsession` file). Only one window

manager can be started on a display, and without the –wm none argument, ompstartdmx would try to start a window manager by default, which would result in an error.

For more information about .xsession files, see the X(1) and xdm(1) man pages.

Configuring DMX to Run Automatically on Linux

This following describes how you configure DMX to run automatically upon login using GDM, KDM, or XDM.

Using GDM

At the login screen, choose one of the following from the the **Session** menu:

```
GNOME-OpenGL-Multipipe
```

```
KDE-OpenGL-Multipipe
```

Choosing one of those will start DMX with a GNOME or KDE session, respectively.

The session will use the default DMX configuration. If the display is managing only one screen, an appropriate message will be displayed.

If you want to customize the ompstartdmx options (for example, to use a special DMX configuration file), modify the script(s) in the directory /etc/X11/gdm/Sessions/GNOME-OpenGL-Multipipe or /etc/X11/gdm/Sessions/KDE-OpenGL-Multipipe. You can also copy and add your own session scripts in the /etc/X11/gdm/Sessions directory.

Using KDM

You will need to configure KDM to add more session type options to the **Session type** menu of the KDM login screen. OpenGL Multipipe does not provide any scripts or automation for doing this. You must also configure the new session type to start ompstartdmx or to use a session script provided by OpenGL Multipipe in directory /etc/X11/gdm/Sessions.

The following steps describe one way to add a new session type, to set the associated session script, and to write that script:

1. In file /etc/X11/xdm/kdmrc, add the label for your new session type to the comma-separated list for the variable SessionTypes.

   The label (for example, KDE-OpenGL-Multipipe) will be displayed in the option menu on the login screen and passed as the first argument to the session script.

2. In `file /etc/X11/xdm/kdmrc`, set your own session script by replacing the following line with the file path to your script:

   ```
   Session=/usr/share/config/kdm/Xsession
   ```

   For example, you might choose the following:

   ```
   Session=omp_kdm_session
   ```

3. Write your session script to look something like the following:

   ```
   #!/bin/bash
   if [ “$1” = KDE-OpenGL-Multipipe ]; then
      exec /etc/X11/gdm/Sessions/KDE-OpenGL-Multipipe
   elif [ “$1” = GNOME-OpenGL-Multipipe ]; then
      exec /etc/X11/gdm/Sessions/GNOME-OpenGL-Multipipe
   else
      exec /usr/share/config/kdm/Xsession $1
   fi
   ```

   Using XDM

   You can add a few lines to your `~/.xsession` script, see `ompstartdmx –help` for more details. The Linux procedure is slightly different than that of IRIX.

After DMX has initialized, you will see a new session covering all the screens. At this point, you can start using OpenGL Multipipe by running applications with `omprun` (see “Running Applications with OpenGL Multipipe” on page 33).

Differences between the IRIX and Linux Versions of `ompstartdmx`

This section lists differences (and one noteworthy similarity) in the behavior and command-line options of the `ompstartdmx` script between IRIX and Linux:

- On Linux, there is no `–wm` option. The window manager is the default window manager for GNOME or KDE. Use their standard tools to set your preferred window manager.
- On Linux, the `ompstartdmx` script will return control only when the session ends. On IRIX, it returns control once the session starts.
- On Linux, the valid options for the `–session` flag are as follows:

  `[ -session {gnome | kde | noaware |` *scriptname*`} ]`

Option	Behavior
`–session` unspecified	Starts a single xterm as the session. The session ends when this xterm dies.

`gnome`	Starts a GNOME session with aware support if GNOME is installed; otherwise, xterm will be used.
`kde`	Starts a KDE session with aware support if KDE is installed; othersise, xterm will be used.
`noaware`	Launches **without** aware window management the default session that would run with the `startx` command.
scriptname	Runs the script as the session. When the script returns control, the session ends. **Note:**The `ompstartdmx` script does not launch any window manager; the session script must do so.

- The way to end the session remains the same: logout or `Ctrl-Alt-q`.

DMX Limitations on Linux

When using a GNOME session with an aware window manager, the session will not be saved; the default session will be used at each login. This happens because the `gnome-wm` script does not pass the session ID parameter to any aware window manager, only to those which are supported by the GNOME environment (`sawfish`, `metacity`, `twm`, and their like). When using a KDE or GNOME session without aware window management, the session will be saved and restord at the next login.

Creating DMX Configuration Files

A DMX configuration file is simply a text file that describes the configuration of a virtual display, the real displays it manages, and the geometry of the virtual screen. This section provides some short examples of configuration files. These and other example configuration files may be found in the directory `/usr/share/omp/examples/dmx` .

To start DMX with one of these configurations, do the following:

1. Save the configuration to a text file with any name—for example, `updown.dmx`.
2. Invoke `ompstartdmx` with the option –`cfgfile`, as shown in the following entry:

   ```
   $ ompstartdmx -cfgfile updown.dmx
   ```

   It is also possible to place many configurations in a single file. In this case, you can choose one configuration from the file by specifying both the –`cfgfile` and –`cfgname` options, as shown in the following:

```
$ ompstartdmx -cfgfile allmyconfigs.dmx -cfgname updown
```

The following example configuration file specifies a vertical layout:

```
virtual updown 1280x2048 {
    display :0.0 1280x1024;
    display :0.1 1280x1024 @0x1024;
}
```

This configuration file defines a virtual screen configuration named `updown` of size 1280x2048. The virtual screen includes the following two real back-end displays:

Display `:0.0` It has a size of `1280x1024` and is located at location `0x0` in the virtual screen space.

Display `:0.1` It has a size of `1280x1024` and is located at `0x1024` in the virtual screen space.

You may also define some overlap between each of the screens, as in the following horizontal layout:

```
virtual overlap 2460x1024 {
    display :0.0 1280x1024;
    display :0.1 1280x1024 @1180x0;
}
```

This configuration file defines two screens of `1280x1024`, each with 100 pixels of overlap, resulting in a virtual screen size of `2460x1024`.

The display value specified can be any valid display value, including a display value that specifies a remote machine, as in the following example:

```
virtual remote 2560x1024 {
    display localhost:0.0 1280x1024;
    display remotehost:0.1 1280x1024 @1280x0;
}
```

There is also a graphical tool to create and edit configuration files. You can find documentation for this tool in `/usr/share/omp/doc/user/xdmxconfig.1.html`. The tool is installed in `/usr/share/omp/X11/bin/xdmxconfig`.

More information about the configuration file format can be found in the file `/usr/share/omp/doc/user/Xdmx.1.html`.

Configuring OpenGL Multipipe with SGI Xinerama as the X Proxy Layer

Note: SGI Xinerama is not supported on Onyx4 or Silicon Graphics Prism servers. Only the DMX proxy layer is supported on these servers.

SGI Xinerama groups multiple screens of a single X server into a logical screen. This section describes how you initialize and configure SGI Xinerama.

Initializing SGI Xinerama

To configure OpenGL Multipipe with SGI Xinerama as the X proxy layer, perform the following steps:

1. Enable SGI Xinerama.

   If you are enabling SGI Xinerama on your system for the first time, enter the following as `root` in an IRIX shell:

   ```
   # chkconfig -f xinerama on
   ```

   Otherwise, enter the following to enable SGI Xinerama:

   ```
   # chkconfig xinerama on
   ```

   On systems having only one graphics pipe or in the case where the X server is directed to handle only one pipe (see the `Xsgi`(1) man page), enabling SGI Xinerama has no effect. In these cases, SGI Xinerama will be disabled, regardless of the value of the `xinerama` flag supplied on the `chkconfig` command.

2. Restart the X server.

   The X server has to be restarted for the `chkconfig` change to take effect. Enter the following as `root` in an IRIX shell to restart the X server:

   ```
   # (/usr/gfx/stopgfx; /usr/gfx/gfxinit; /usr/gfx/startgfx) &
   ```

After the X server is started with SGI Xinerama enabled and you have logged in to the system, you can start using OpenGL Multipipe by running applications with `omprun` (see “Running Applications with OpenGL Multipipe” on page 33).

Configuring Overlapping Screens with SGI Xinerama

Reality Center environments with multiple projectors and multiple graphics pipes often have overlapping screens. To allow seamless alignment of these screens, projectors typically have edge blending capability.

You control the amount of overlap or gap space between screens by specifying the `xoffset` and `yoffset` arguments (in units of pixels) of the –`hw` parameters in the file `/var/X11/xdm/Xservers`. For example, if you have three pipes, each with a resolution of 1280 x 1024 and a horizontal overlap of 256 pixels between each screen, you would place the following –`hw` arguments together on one line in the `/var/X11/xdm/Xservers` file:

```
:0 secure /usr/bin/X11/X
-boards 4,5,6
-hw board=4,right=1
-hw board=5,left=0,right=2,xoffset=-256
-hw board=6,left=1,xoffset=-256
... other X server arguments ...
```

Notes:

- The –`boards` numbers are physical pipe numbers, but the indexes given to the `right`, `left`, `above`, and `below` parameters refer to the logical order of the –`hw` parameters.
- The lines are separated in the example only for readability.

See the `Xsgi`(1) and `xdm`(1) man pages for more information.

Setting Other Configuration Options

Starting with version 2.5, OpenGL Multipipe uses X11 resources to specify configuration options. This approach allows you to define values per application, per user, and system-wide. This section describes setting these options in the following subsections:

- "Resource Descriptions" on page 25

Specifying Resource Names

The full name of each OpenGL Multipipe resource has the following format:

`OMP.`*app_name*`.`*resource_name*

The *app_name* field is the application name, defined to be the last component in the running executable path name. The *resource_name* is one of the available OpenGL Multipipe resources.

You can specify a resource with *loose binding*—that is, matching the resource for all applications—by using an asterisk in the following manner:

`OMP*`*resource_name*

For more information on the X11 resources and their scope, see the `X`(1) and `XrmInitialize`(3) man pages.

The Resource Search Path

The following ordered list of resource files indicates the search path OpenGL Multipipe uses to assign values to the resources:

1. Resources set with `omprun`
2. Resources set with `xrdb`
3. `$HOME/.Xdefaults`
4. `/usr/share/omp/app-defaults/app_name`
5. `/usr/share/omp/config`

The first match found will be used. However, resources set with the full-name format will always precede resources set with loose binding even if the full-name format is found later in the search path. Resources set with the `omprun` command always have precedence. For information of the use of the `omprun` command, see section "Setting Run-Time Options" on page 34.

If a resource is not specified in the preceding search path, then the defined default value for that resource will be used.

When you set the environment variable `SGIOMP_PRINT_CONFIG` to `1`, OpenGL Multipipe prints the configuration values it uses to `stdout` when an application runs through OpenGL Multipipe. This is useful to ensure that any overrides you have specified are being honored.

Resource Types

Each resource value can have one of the following types:

Type	Description
`Bool`	A boolean value. Valid values are `0`, `1`, `off`, or `on`.
`Enum`	Enumeration: one of a specified list of tokens.
`Float`	A floating point value.
`FloatList`	A comma-separated list of floating point values.
`Int`	An integer number.
`IntList`	A comma-separated list of integer values.
`Enum`	An enumeration: one of a specified list of tokens.

Resources and Their Default Values

Table 3-1 lists all resources with their default values.

Notes:

- A resource name is not identical to the name of the associated command-line option of `omprun`.
- The default values might be changed in future releases.

Table 3-1 X11 Resources and Defaults

Resource[a]	Value Type	Default Value
`aa2Jitter.X`	`FloatList`	`0.24649, -0.24649`
`aa2Jitter.Y`	`FloatList`	`0.249999, -0.249999`
`aa4Jitter.X`	`FloatList`	`-0.208147, 0.203849, -0.292626, 0.296924`

Table 3-1 X11 Resources and Defaults **(continued)**

Resource[a]	Value Type	Default Value
aa4Jitter.Y	FloatList	0.353730, -0.353780, -0.149945, 0.149994
activeScreens	IntList	The list of all back-end screens of the meta X server
culling (cull)	Bool	off
culling.cullUserClipPlanes	Bool	off
culling.minPixels (minpixels)	Int	1
culling.showStat (cullshow)	Enum: off bbox stats all	off
culling.texCulling	Bool	off
dlSplit (dlsplit)	Bool	off
dlSplit.maxDepth	Int	8
dlSplit.maxTotalTris	Int	400000
dlSplit.maxTris (dlsplitmaxtris)	Int	800
dlSplit.showRandomColors (dlsplitshow)	Bool	off
drawPixelsClipping	Bool	on
masterMode (mstrmode)	Enum: render track off	track when omprun is used off, otherwise
masterScreen	Int	0

Table 3-1 X11 Resources and Defaults **(continued)**

Resource[a]	Value Type	Default Value
`maxFramesLatency` (`latency`)	`Int`	`4`
`pbuffers.disable`	`Bool`	`off`
`pbuffers.layout`	`Enum:` `duplicate` `horizSplit` `vertSplit` `rectSplit`	`duplicate`
`shmQueueSize`	`Int`	`DisplayWidth * DisplayHeight * 4`
`slaveCPUs`	`IntList`	Empty
`swapSyncMode` (`nosync`,`swapready`)	`Enum:` `none` `soft` `barrier`	`soft`
`texShm`	`Bool`	`off` (unless `culling.texCulling` is on)
`viewportClippingMode` (`novpclip`)	`Enum`: `none` `scissor` `viewport` `subwin`	`scissor` for SGI Xinerama displays and `subwin` for `Xdmx` displays

a. The associated `omprun` command-line options are shown in parentheses, where applicable.

Resource Descriptions

This section briefly describes each of the OpenGL Multipipe tunable resources:

`aa2Jitter.X` and `aa2Jitter.Y`
Controls the jittering offset for each input pipe when the hardware compositor configured for pixel averaging with two pipes. The value must include two values for both X and Y to specify the frustum offset in pixels on the x and y axes to be applied for the first and second pipe participating in the pixel averaging.

`aa4Jitter.X` and `aa4Jitter.Y`
Controls the jittering offset for each input pipe when the hardware compositor configured for pixel averaging with four pipes. The value must include four values for both X and Y to specify the frustum offset in pixels on the x and y axes to be applied for the first, second, third, and fourth pipes participating in the pixel averaging.

`activeScreens`
Specifies the back-end screens to be active. OpenGL Multipipe will render OpenGL primitives only on those pipes. For example, when `Xdmx` (or SGI Xinerama) manages three physical screens (pipes), setting the `activeScreens` resource to `0,1` will make OpenGL rendering to be visible on screens 0 and 1 only. On screens 2 and 3, the OpenGL part of the application windows will be either black or garbage. By default, all managed screens will be active with respect to OpenGL.

`culling`
Specifies the enable switch for the geometry culling feature of OpenGL Multipipe. It can be either `off` or `on`. When this resource is enabled OpenGL Multipipe will draw to each pipe only the geometry primitives that are visible on that pipe. The default value for this switch is `off`. For more information, see section “Geometry Culling” on page 50.

`culling.cullUserClipPlanes`
Enables culling against user-defined clip planes when one of `GL_CLIP_PLANE`*i* is enabled. The default value is `off`, in which case OpenGL Multipipe will cull only against the viewing frustum.

`culling.minPixels`
Specifies approximate culling—that is, the minimum object size in pixels that should be drawn. All objects smaller than this size will be culled. For more information, see section “Small Object Culling” on page 51.

`culling.showStat`
Enables the drawing of culling statistics. The following are the possible values:

`off`	No statistics are drawn (the default).
`bbox`	The bounding box for each geometry object is drawn in blue.
`stats`	Cull percentage statistics are drawn for each screen (only for doubly buffered windows).
`all`	Both bounding boxes and culling percentage statistics are drawn.

`culling.texCulling`
Enables or disables texture culling. When texture culling is enabled, texture loads will be postponed until the texture is really viewable on each specified pipe. In that case only, the relevent part from the texture will be downloaded to the pipe. The resource `OMP*culling` should be turned on for this feature. For more information and current limitations of this feature, see the release notes.

`dlSplit`
Allows OpenGL Multipipe to split a display list into smaller display lists based on a geometry spatial sort to acheive better culling. The smaller display lists can be distributed among the rendering pipes. This feature is usually used when culling is turned on. The valid values for `dlSplit` are `on` and `off`; the default is `off`. For more information, see section “Display List Partitioning” on page 51.

`dlSplit.maxDepth`
Limits the number of recursive subdivisions by the `dlSplit` algorithm, which is an adaptive subdivision algorithm. After the specified number of recursive subdivisions are made, the process will end even if it does not meet the `dlSplit.maxTris` criterion. The default value is `8`.

`dlSplit.maxTotalTris`
Limits the number of triangles that will be considered by the `dlSplit` algorithm at each iteration. If a display list contains more triangles than the specified number, then the splitting algorithm will be applied multiple times, once for each group that contains the specified number of triangles. The default value is `400000`.

`dlSplit.maxTris`
Limits the number of triangles to the specified maximum for each bounding box of subordinate display lists. Display lists that have fewer triangles (or other primitives) than the specified number will not be split by the `dlSplit` algorithm. The default value is `800`.

`dlSplit.showRandomColors`
Allows OpenGL Multipipe to randomly assign a different color for each divided geometry partition. For testing and debugging purposes, this coloring allows you to better see the divisions made by the `dlSplit` algorithm. This may not work for all applications as it simply uses **glColor()** to set the color for each partition. The default is `off`.

`drawPixelsClipping`
Controls whether the **glDrawPixels()** operation will be clipped such that only the viewable rectangle of pixels will be sent to each pipe.

`masterMode`
Specifies if and how the master process needs to use one of the graphics pipes. The following are the valid values:

`render`	The master process will render to the master local pipe and no slave draw process will be forked for this pipe.
`track`	The master process will use its local pipe for state tracking only so that each **glGet()** call will be executed on the local pipe.
`off`	The master will not use any physical pipe for either rendering or state tracking. Some OpenGL states are being tracked by the master in software while other states will be queried from one of the slaves when a **glGet()** call is being executed. For more detailed description of the different master modes, see section "Master Rendering Modes" on page 51. When the `omprun` command is not being used on platforms where the `omprun` command is optional, the master mode is forced to be `off`, regardless of this resource value. Therefore, this resource is usually set only by the `omprun` command using its `–mstrmode` option.

`masterScreen`
Specifies which managed screen (pipe) will be used by the master application for either rendering or state tracking when running OpenGL Multipipe in master mode render or track (available only when using the `omprun` command). By default, it will be the first managed screen of the `Xdmx` or SGI Xinerama server.

`maxFramesLatency`
Specifies the maximum latency (in frames) allowed between the application and rendering slaves. The application process might pack a few frames ahead in the framebuffer while the slave processes are still drawing the previous frames. This introduces some latency between the application and the real drawing by the slave but helps overall throughput of master and slaves. The latency is always 0 when using `masterMode` render and is undefined if `swapSyncMode` is set to `none`. The default value for this resource is 4. For more information, see section "Frame Latency Control" on page 55.

`pbuffers.disable`
Disables pbuffers support. When this resource is enabled, all pbuffer-capable framebuffer configurations will be filtered out and will not be exposed to the application. The default value is `off`.

`pbuffers.layout`
Controls how the rendering into a pbuffer is split among the rendering slaves. Each rendering slave can be configured to draw only to a sub-region of each pbuffer. The following are the valid values:

`duplicate`
All slaves renders to all regions of the pbuffer, That makes the content of the pbuffer to be duplicated on all rendering pipes. Use this configuration when your application uses **glCopy*()** to copy the pixels from the pbuffer to some other pixel buffer on the pipe. The other options might be better if you are using **glReadPixels()** to read back the pbuffer content into your process memory. This is the default value.

`horizSplit`
The area of the pbuffer is split into *N* horizontal strips, where *N* is the number of pipes. Each slave then limits the rendering to only one strip.

`vertSplit`
The area of the pbuffer is split into *N* vertical strips, where *N* is the number of pipes. Each slave then limits the rendering to only one strip.

`rectSplit`
The area of the pbuffer is split into *N* even rectangular areas as much as possible. For example, for a four-pipe configuration, it will be split into 2x2 tiles, where each tile is of size ($w/2$)x($h/2$) and w and h are the pbuffer width and height, respectively. Each pipe then renders to only one tile.

`shmQueueSize`
Sets the size in bytes for the shared memory block that will be used for master-to-slaves communication. When this resource is not specified, the default value depends on the overall resolution of the meta display as determined by the following:

(`DisplayWidth` * `DisplayHeight` * 4) + 32

This allows a single full-screen **glDrawPixels()** command to fit in the shared memory since one single OpenGL command cannot be split into multiple packets. Depending on your application, you may want to set this value to be smaller or larger. When a shared

memory size larger than the default or the specified size is needed, the application will exit with an error message specifiying the minimum shared memory size needed for that application.

`slaveCPUs`
Specifies the CPUs to assign to the draw slave processes for each of the active screens. If there are more active screens than the size of the specified CPU list, then the remaining draw processes will not be assigned to any particular CPU. By default, the draw processes will not be assigned to any particular CPU.

`swapSyncMode`
Specifies what method OpenGL Multipipe will use to synchronize swap buffers on all pipes. The following are the valid values:

`none`	No swap synchronization is being performed. Each rendering slave runs freely.
`soft`	Software synchronization is performed using a software barrier to force all slaves to issue the **glXSwapBuffers()** request at the same time.
`barrier`	OpenGL Multipipe uses the hardware ImageSync or SwapReady line whenever possible (when no other application is using the hardware barrier). Otherwise, software synchronization will be used. For more information on the synchronization of swap buffers, see section “Buffer Swap Synchronization” on page 56.

`texShm`
Controls the caching of textures in shared memory. When this switch is turned on, OpenGL Multipipe will cache a copy of each texture in shared memory that remains valid until the texture is used by all pipes. Once the texture has been downloaded to driver memory of all pipes, the shared memory cache is purged. This feature works together with texture culling (see the `culling.texCulling` resource). When `culling.texCulling` is turned on, the `texShm` resource is implicitly turned on. The default for the `texShm` resource is `off`.

`viewportClippingMode`
Specifies viewport clipping mode. Each rendering slave may limit the rendering region to only the portion of the window that is visible on its rendering pipe. This is required to support large viewport areas, which are allowed by a single pipe, and it also helps to scale fill rate performance. The `viewportClippingMode` resource selects the method OpenGL Multipipe uses to limit this rendering region. The following are the valid values:

`none`	No viewport clipping is performed. Each hardware pipe clips the window to the pipe's boundary.
`scissor`	OpenGL Multipipe uses **glScissor()** to limit the rendering to the viewing area only.
`viewport`	OpenGL Multipipe defines a different viewport for each rendering slave.
`subwin`	OpenGL Multipipe creates a separate child window for each rendering slave. The window size is the viewable region of the application window on the slave's pipe. This option is supported on DMX configurations only.

When this resource is not specified, the default depends on the meta X server that is being used. For `Xdmx` the default is `subwin`. For `Xsgi` with SGI Xinerama enabled, the default is `scissor`. For more information, see section "Viewport Clipping" on page 49.

Verifying That the OpenGL Multipipe Environment is Enabled

The OpenGL Multipipe environment is enabled if an X proxy layer is enabled (DMX or SGI Xinerama).

To verify that an X proxy layer is enabled, ensure that your `DISPLAY` environment variable is pointing to the correct display (usually `:0.0` for SGI Xinerama or `:1.0` for DMX) and enter the following commands in a command shell:

```
$ xdpyinfo | grep SGI-XINERAMA
```

If `SGI-XINERAMA` appears as the result of the prior commands, SGI Xinerama is enabled.

```
$ xdpyinfo | grep DMX
```

If `DMX` appears as the result of the prior commands, DMX is enabled.

Note: If `DMX-DRI` appears as the result of the prior commands, not only is DMX enabled but the use of the `omprun` command will be optional as well. See section "Running Applications with OpenGL Multipipe" on page 33.

Disabling the OpenGL Multipipe Environment

To disable the OpenGL Multipipe environment, do the following:

- To end a DMX session, simply log out.

  You may also force the DMX server to exit by pressing `Ctrl+Alt+q`.

  If you ran `ompstartdmx` from the command line, you will be returned to your regular X session after the session ends.

  If you chose DMX as a login option or modified your `.xsession` file to start your DMX session, you will return to the login screen after the DMX session ends.

  To permanently disable DMX from starting upon login, reverse the DMX-related changes you made to your `.xession` file or delete (or rename) your `$HOME/.xsession` file. On Linux systems, simply select another default login option.

- If enabled, disable SGI Xinerama and restart the X server.

  Enter the following to disable SGI Xinerama:

```
# chkconfig xinerama off
```

Enter the following as `root` in an IRIX shell to restart the X server:

```
# (/usr/gfx/stopgfx; /usr/gfx/gfxinit; /usr/gfx/startgfx) &
```

The X server has to be restarted for the `chkconfig` change to take effect.

Running Applications with OpenGL Multipipe

Plain X applications will generally run under an X proxy layer without assistance. OpenGL (3D) applications need to use the OpenGL Multipipe 3D proxy library to handle 3D rendering correctly and efficiently on all screens.

To use the OpenGL Multipipe 3D proxy library with OpenGL applications, just run the program using the `omprun` script:

```
$ omprun app_name app_args
```

This will preload the OpenGL Multipipe proxy libraries to intercept OpenGL calls.

The following is an example:

```
$ omprun ivview /usr/share/data/models/Banana.iv
```

Note: The use of the `omprun` script is optional on Oynx4 and Silicon Graphics Prism systems running the latest recommended operating system versions (IRIX 6.5.27 or later and SGI ProPack 3 Service Pack 4 or later, respectively). On these platforms, you can invoke the application in the normal fashion and the application will automatically utilize the full OpenGL Multipipe environment (master-slave 3D proxy layer) if the DMX proxy layer is detected. Later references to the use of the `omprun` script in this guide should be considered optional on these platforms, unless otherwise noted.

On systems that do no meet these requirements, failure to use the `omprun` command under DMX will cause the application to use the slower GLX indirect rendering support in DMX to draw OpenGL to all screens.

OpenGL Multipipe causes an OpenGL application to use the intermediate 3D proxy libraries instead of the normal OpenGL library. This enables the OpenGL application to behave correctly when its windows are moved across parts of the logical screen. Such an

application is considered to be started in multipipe-unaware mode (or simply, *unaware* mode), since it is not aware of the underlying graphics hardware structure.

Technically, the `omprun` command sets the link editor environment variable (`_RLDN32_LIST` or `_RLD64_LIST` on IRIX and `LD_PRELOAD` on Linux) to use the `libOMPmaster.so` library of matching format prior to using the `libGL.so` library. When the `omprun` command is not used on systems where it is is optional, OpenGL Multipipe utilizes the X server's DRI extension, a mechanism in the `libGL.so` library, and a special `SGIOMPdmx_dri.so` module to intercept and route an application's OpenGL calls to the OpenGL Multipipe library `libOMPmaster.so`.

For more information on using `omprun`, see the `omprun`(1) man page or use the `-help` command-line option of `omprun` as follows:

```
$ omprun -help
```

The following sections describe run-time options and how to best run different types of graphics applications:

- "Setting Run-Time Options"
- "Running OpenGL Single-Pipe Applications"
- "Running Pure X Applications"
- "Running IRIS GL Applications"
- "Running IRIX o32 Applications"
- "Running Multipipe Applications in Multipipe-Aware Mode"

For more information on running applications with OpenGL Multipipe, see the release notes, `/usr/share/omp/release_notes/user/relnotes.html`.

Setting Run-Time Options

Even though the use of the omprun script is optional on Onyx4 and Silicon Graphics Prism platforms, you may want to use the command to invoke the application to set run-time options, some of which can override configuration options set otherwise. The configuration options are described in section "Setting up the OpenGL Multipipe Environment" on page 14.

Table 3-2 shows the run-time options for `omprun` along with the related configuration resources. For more information about the various options, see the `omprun`(1) man page or use the `-help` command-line option of `omprun`.

Table 3-2 `omprun` Command-Line Options

omprun Option	Configuration Resource	Description
`-cull`	`culling: on`	Enable geometry culling.

`-cullshow`
`bbox`
`stats`
`all`
	`culling.showStat`
	`bbox`
	`stats`
	`all`
		Show culling information.

`-dlsplit`	`dlSplit: on`	Enable display list spatialization.
`-dlsplitmaxtris`	`dlsplit.maxTris`	Set the display list spatialization threshold.
`-dlsplitshow`	`dlsplit.showRandomColors: on`	Show random colors.
`-dmxglx`	N/A	Use GLX instead of master and slaves.
`-latency`	`maxFramesLatency`	Set latency (how far ahead of the slaves the master works).
`-minpixels`	`culling.minPixels`	Set the small object culling threshold.

`-mstrmode`
`render`
`track`
`off`
	`masterMode`
	`render`
	`track`
	`off`
		Set the master rendering mode.

`-nodlopen`	N/A	Disable `dlopen/dlsym` overriding.
`-nosync`	`swapSyncMode: none`	Disable all slave synchronization.
`-novpclip`	`viewportClippingMode: off`	Disable viewport clipping.
`-pthread`	N/A	Force the master to use the pthread API.
`-swapready`	`swapSyncMode: barrier`	Use hardware swap barrier.

Running OpenGL Single-Pipe Applications

OpenGL single-pipe applications are the targeted applications for OpenGL Multipipe. To run such applications, simply enable the OpenGL Multipipe environment and invoke the application using the `omprun` script.

Under SGI Xinerama, any OpenGL application started without the `omprun` script will not behave correctly. In that case, OpenGL drawings will appear only in the part of the window overlapping screen 0. On the other screens, the application will display a random image that corresponds to the current content of the framebuffer on that pipe.

Under DMX, OpenGL applications started without the `omprun` script will display correctly on all screens, using the GLX indirect rendering support in DMX. However, using the `omprun` script will provide better performance for OpenGL applications.

Hint: For an easy way to run a number of single-pipe OpenGL applications under OpenGL Multipipe without the need to always explicitly invoke `omprun`, start a command shell under `omprun`, as shown in the following :

```
$ omprun xwsh
<omprun xwsh>$ app_name app_args
```

Any application started from this new command shell will be started automatically in transparent multipipe mode.

Running Pure X Applications

As noted in Chapter 1, “OpenGL Multipipe Overview”, the X proxy layer enables pure X applications to run transparently over multiple pipes. To run pure X applications, simply enable SGI Xinerama or DMX before invoking them and they will run correctly. You do not need to use the `omprun` script for these applications.

Running IRIS GL Applications

On IRIX, there are applications that use the older IRIS GL graphics library instead of that of OpenGL. OpenGL Multipipe does not support IRIS GL. To check whether your current application is attempting to use IRIS GL, enter the following:

```
$ elfdump -Dl app_name | grep libgl.so
```

The following is an example:

```
$ elfdump -Dl /usr/sbin/showcase | grep libgl.so
[11]    Jun  6 22:31:51 1996    0xe9155925 ----- libgl.so    sgi1.0
```

The `omprun` script does this check and prevents OpenGL Multipipe from executing IRIS GL applications.

If your system supports IRIS GL, you can still run IRIS GL applications, but not using the OpenGL Multipipe `omprun` layer. Under SGI Xinerama, they will run correctly only on screen 0. Also, IRIS GL applications will run correctly in multipipe-aware mode, which is described in the subsequent section "Running Multipipe Applications in Multipipe-Aware Mode".

Only when DMX is used as the X proxy layer, will IRIS GL applications run correctly on all screens without using the `omprun` script. This happens through the GLX indirect rendering support in DMX. Consequently, performance decreases.

Running IRIX o32 Applications

On IRIX, there are applications that use the older o32 application binary interface (ABI) instead of the newer n32 or 64-bit ABIs. OpenGL Multipipe does not support applications that were built using the o32 ABI. To check whether your current application was built with the o32 ABI, enter the following:

```
$ file app_name | grep 32-bit
```

If the text `ELF 32-bit` ... is printed as a result, it is an o32 application.

The following is an example:

```
$ file /usr/sbin/showcase | grep 32-bit
/usr/sbin/showcase: ELF 32-bit MSB mips-2 dynamic executable MIPS -
version 1
```

The `omprun` script does this check and prevents OpenGL Multipipe from executing o32 applications.

If your system supports the o32 ABI, you can still run o32 applications, but not using the OpenGL Multipipe `omprun` layer. Under SGI Xinerama, they will run correctly only on screen 0. Also, o32 applications will run correctly in multipipe-aware mode, which is

described in the subsequent section “Running Multipipe Applications in Multipipe-Aware Mode”.

Only when you use DMX as the X proxy layer, will o32 applications run correctly on all screens without using the `omprun` script. This happens through the GLX indirect rendering support in DMX. Consequently, performance decreases.

Running Multipipe Applications in Multipipe-Aware Mode

Multipipe applications are intentionally written to take advantage of systems that have multiple graphics pipes. If they know about the underlying graphics hardware, they can explicitly address and take advantage of the individual graphics pipes. Typically, multipipe applications are written using OpenGL Performer or OpenGL Multipipe SDK.

It is best not to run such applications under OpenGL Multipipe, which hides the hardware configuration of the system from the applications. To run at full potential, these applications should be aware of the different graphics pipes in the system. To allow such applications to see the real graphics hardware does not require you to disable OpenGL Multipipe.

The OpenGL Multipipe layer may be bypassed on a per-application basis. This allows a multipipe application to be fully aware of the multipipe environment while other applications, aware of only a single large pipe, continue to run under OpenGL Multipipe. Applications that bypass the OpenGL Multipipe layer are said to run in multipipe-aware mode (or simply, *aware mode*), because they are aware of the different graphics pipes handled by the X server.

To run a multipipe application in aware mode while other single-pipe applications run concurrently in unaware mode, set the `DISPLAY` environment variable to point to the desired backend display that is managed by the X proxy layer (for example, `:0.1`), and then start the multipipe application with the `-aware` command-line option of the `omprun` script, as in the following example:

```
$ setenv DISPLAY :0.0
$ omprun -aware perfly
```

Under DMX, it is especially important to set the `DISPLAY` environment variable because the DMX meta display has a completely different display name than its component backend displays. By default, the DMX display is `:1` and the backend aware display is `:0`. In the case of SGI Xinerama, the SGI Xinerama meta display and its component backend displays are referred to by the same display name (for example, `:0`).

Note: Under SGI Xinerama, applications started in aware mode will be under window manager control only with `omp4Dwm`. Under DMX, other window managers may be used. See the subsequent section "Managing Windows for Aware Applications" for more information about aware window management.

Hint: For an easy way to run a number of commands in aware mode, start a command shell in aware mode.

```
$ setenv DISPLAY :0.0
$ omprun -aware xwsh
<aware xwsh>$ app_name app_args
```

Any application started from this new command shell will be started automatically in aware mode.

Using SGI Scalable Graphics Hardware with OpenGL Multipipe

You may configure SGI Scalable Graphics Compositor hardware for use with OpenGL Multipipe to improve geometry and fill performance for an application. This requires no changes to the application. Using the following topics, this section describes how to configure SGI Xinerama or DMX for this purpose as well as settings for OpenGL Multipipe to improve performance and usability in composited logical screen mode:

Configuring Composited Screens with SGI Xinerama

Composited screens are a special case of overlapped screens in which each compositor input screen completely overlaps the other compositor input screens. That is, each compositor input pipelet displays the same area of the logical screen.

To configure totally overlapped pipes, specify negative `xoffset` and `yoffset` parameters that equal the width (and height) of the screens you are overlapping. For example, if you have four pipelets, each with a resolution of 1280 x 1024, connected to a single compositor, you would place the following `-hw` arguments together on one line in the `/var/X11/xdm/Xservers` file:

```
:0 secure /usr/bin/X11/X
-boards 4,5,6,7
-hw board=4,right=1
-hw board=5,left=0,right=2,xoffset=-1280
-hw board=6,left=1,right=3,xoffset=-1280
-hw board=7,left=2,xoffset=-1280
... other X server arguments ...
```

Notes:

- The `-boards` numbers are physical pipe numbers, but the indexes given to the `right`, `left`, `above`, and `below` parameters refer to the logical order of the `-hw` parameters.
- The lines are separated in the example only for readability.

Configuring Composited Screens with DMX

To configure completely overlapped screens under DMX, simply create a DMX configuration file to manage the screens of the backend X server in the desired order. Do not specify an offset or a 0x0 offset after the screen specifications. The following is an example configuration file:

```
virtual totaloverlap 1280x1024 {
    display :0.0 1280x1024;
    display :0.1 1280x1024;
    display :0.2 1280x1024 @0x0; # “@0x0” is optional
    display :0.3 1280x1024;
}
```

For more information on DMX configuration files, see section "Creating DMX Configuration Files" on page 18.

Specifying Static Composited Regions with `ompstartdmx`

You can specify a static tiling mode for a hardware compositor during DMX startup. The `ompstartdmx` script will start both the hardware compositor and DMX with the specified tiling mode.

The `–compositor` option on the `ompstartdmx` script allows you to specify the desired static tiling mode, as shown in the following example:

```
$ ompstartdmx -compositor mode
```

The following values are available for *mode*:

Mode	Description
`quad`	A quad tiling
`vert2`	Two vertical stripes
`vert3`	Three vertical stripes
`vert4`	Four vertical stripes
`horiz2`	Two horizontal stripes
`horiz3`	Three horizontal stripes
`horiz4`	Four horizontal stripes

You can also set up the video format using the `–compositor` option, as shown in the following entry:

```
$ ompstartdmx -compositor quad,1280x1024_75
```

The video format must be supported by both the hardware compositor and the pipes, and all pipes must have the same video format. Otherwise, `ompstartdmx` exits with an error. The default video format is the current one for the pipes.

Using Pixel Averaging Composition Mode for Full-Scene Antialiasing

You can set up the hardware compositor for pixel averaging during DMX startup. In pixel averaging mode, the following occurs:

1. OpenGL Multipipe applies a different sub-pixel jittering offset for each pipe's frustum.
2. The hardware compositor averages the pixel values.
3. The resulting image on the compositor output is antialiased.

The hardware compositor can be configured for pixel averaging mode with either two or four input pipes. For proper operation, configure all pipes connected to the compositor to be fully overlapped.

Note: When using two pipes, connect them to compositor channels 0 and 2, not 0 and 1.

If you are starting a session with only one compositor attached, then the simplest way to use pixel averaging is to use the `-compositor` option on the `ompstartdmx` command, as follows:

```
$ ompstartdmx -compositor {aa2 | aa4}
```

This command entry sets up a DMX configuration file with full overlap for two or four pipes and configures the compositor and OpenGL Multipipe for pixel averaging.

If you are using more than one compositor or you are use your own DMX configuration file, you must use a compositor configuration file, described in the "Using Multiple Compositors" section in the release notes.

You can configure the jittering offset values per application using the following X resources:

- `OMP*aa2Jitter.X` and `OMP*aa2Jitter.Y` (for two-pipe pixel averaging)
- `OMP*aa4jitter.X` and `OMP*aa4Jitter.Y` (for four-pipe pixel averaging)

For more information on the resources, see section "Resource Descriptions" on page 25.

Using Multiple Compositors in an OpenGL Multipipe Session

OpenGL Multipipe allows you to configure multiple compositors by specifing a special compositor configuration file with the `-compositor` option for the `ompstartdmx` command, as shown in the following:

```
$ ompstartdmx -compositor config-file
```

The compositor configuration file *config-file* describes the desired mode of operation for each compositor in the session. The format and use of the configuration file is described in the "Using Multiple Compositors" section in the release notes.

Specifying Static Composited Regions for Fully Overlapped Screens

To establish static composited regions, do the following:

1. Set up fully overlapping screens with SGI Xinerama or DMX.

   The sections "Configuring Composited Screens with SGI Xinerama" on page 40 and "Configuring Composited Screens with DMX" on page 40 describe how you do this.

2. Configure the SGI Scalable Graphics Compositor hardware and OpenGL Multipipe to constrain drawing to areas of the physical pipes that will provide the best static load balancing for your application with `sgcombine` (on IRIX) or `sgcmb` (on Linux).

   Note that `sgcombine` must be run in multipipe-aware mode:

   ```
   $ env DISPLAY=:0.0 omprun -aware /usr/gfx/sgcombine
   ```

   For more information about setting up composited screens with `sgcombine` or `sgcmb`, see the *SGI Scalable Graphics Compositor User's Guide*.

3. Configure OpenGL Multipipe to match the static compositor configuration set with `sgcombine`.

   For example, if you specify a compositor tiling of four input rectangles of 640x512 pixels in `sgcombine`, setting the environment variable `SGIOMP_SCREEN_RECTS` to the following string provides the matching settings to OpenGL Multipipe:

   ```
   "640x512+0+0 640x512+640+0 640x512+0+512 640x512+640+512"
   ```

   OpenGL Multipipe will then clip drawing to these subregions on back-end screens `:0.0`, `:0.1`, `:0.2`, and `:0.3` (pipes 4, 5, 6, and 7), respectively, if DMX or SGI Xinerama are configured as in the previous sections. The rectangles are

specified in pipe-relative coordinates, one per backend X screen, using the format described in the `XParseGeometry`(3X11) man page. Extra rectangles are ignored and screens for which a rectangle is not specified will have clipping performed at the screen borders.

Enabling the `omprun -cull` option will cause geometry to be culled to these areas as well.

Enabling Duplicate Cursor Images in Overlap Regions

When an X proxy layer is used to overlap screen regions on an edge-blended display or a compositor-based system, the cursor will seem to disappear when it enters the overlapped or uncomposited regions of the display. In X, a mouse belongs to one screen of the X server at a time. Therefore, it is normally not possible to draw the mouse multiple times (on different screens) in the overlap region.

To prevent the cursor from disappearing in these cases, you need to create additional or *duplicate cursor* images (not real cursors) where two or more screens overlap. There is still just one real cursor position on the display. The way you do this depends on your X proxy layer, SGI Xinerama or DMX.

Under SGI Xinerama

In IRIX 6.5.20 or later, you enable duplicate cursors by adding the `-phantomcursors` flag to the X server command line in the `/var/X11/xdm/Xservers` file. For more information about the `-phantomcursor`s option, see the `Xsgi`(1) man page.

Under DMX

There are two ways to enable duplicate cursors under DMX:

- Managing an appropriate subarea of each composited input pipe instead of the whole screen area
- Managing multiple backend X servers

Managing Screen Subregions with DMX

You can use a DMX configuration file to cause DMX to manage less than a whole backend screen. When the subregions of each screen are visually assembled using a congruent static compositor tiling, the resulting logical screen will look and behave identically to

the DMX or SGI Xinerama displays with fully overlapping screens described previously. The following configuration file entry illustrates this feature:

```
virtual quadtilesubregions 1280x1024 {
    display :0.0 640x512+0+0     @0x0;
    display :0.1 640x512+640+0   @640x0;
    display :0.2 640x512+0+512   @0x512;
    display :0.3 640x512+640+512 @640x512;
}
```

Note: The `SGIOMP_SCREEN_RECTS` environment variable, described in section "Specifying Static Composited Regions for Fully Overlapped Screens" on page 43, is not necessary here and should not be used with a DMX configuration that manages screen subregions.

Managing Multiple Backend X Servers with DMX

The X mouse cursor can be made visible in the partially or fully overlapped screen regions needed for edge-blended or composited displays. As long as the logically overlapping areas of the DMX display are composed by screens of different backend X servers, DMX can utilize the mouse cursor that is available on each backend X server.

Instead of having DMX manage multiple screens of a single backend X server, you can start multiple backend X servers and specify a DMX configuration that logically overlaps areas from different backend X servers, as shown in the following:

```
virtual totaloverlapmulti 1280x1024 {
   display :0.0 1280x1024;
   display :1.0 1280x1024;
   display :2.0 1280x1024;
   display :3.0 1280x1024;
}
```

Note: The multiserver technique can be combined with the technique of managing screen subregions to create a logical display on which mouse cursors are visible in all overlapped and composited regions.

Managing Windows for Aware Applications

To allow window manager support for applications started in aware mode, OpenGL Multipipe provides aware window management. Under SGI Xinerama, the window manager `omp4Dwm`, a specialized version of the SGI standard window manager (`4Dwm`), is used to manage aware windows. Under DMX, a window manager wrapper is provided that is much like `omprun` for GL applications.

When `omp4Dwm` or the window manager wrapper is not used, applications started in aware mode (using `omprun –aware` *app_name*) will bypass the window manager. This means that their windows cannot be moved, resized, iconified, or otherwise managed. This includes the regular decoration provided by the window manager. The windows will not have this decoration. This occurs because an unaware window manager does not know about all of the real screens that the X proxy layer is managing. It cannot manage aware windows that were not created through X proxy layer.

If you are using DMX as your X proxy layer and you invoke `ompstartdmx` with the standard arguments (or if DMX is configured to start automatically when you log in), a window manager of your choice will be automatically started with the window manager wrapper so that it is able to manage aware windows. You can change the DMX default window manger by using the `ompstartdmx –wm` option. You may follow the steps in the following subsections if you want to run a different window manager to manage aware windows.

This section describes the following topics:

- "Starting an Aware Window Manager"
- "Exiting an Aware Window Manager"
- "Setting an Aware Window Manager as the Default"

Note: The multipipe-aware window manager is currently not supported for compositor-based systems.

Starting an Aware Window Manager

To start an aware window manager, perform the following steps:

1. Exit or kill any unaware window manager that is currently managing the display.

   If you are using `4Dwm` (the default window manager on IRIX), enter the following in a command shell:

   ```
   $ tellwm quit
   ```

   Otherwise, if possible, exit your window manager without logging out. One way to do this is to find the process number for your window manager and kill it manually, as the following illustrates:

   ```
   $ ps -e | grep my_window_manager
      23878 ? 0:42 my_window_manager
   $ kill 23878
   ```

   Some window managers may not allow you to exit the window manager and remain logged in. If this is the case, you will need to start the aware window manager from a `.xsession` file. See the section "Setting an Aware Window Manager as the Default" on page 48 for more information.

2. Start the specialized window manager.

   Under SGI Xinerama, enter the following:

   ```
   $ start_ompwm
   ```

   Under DMX, enter the following:

   ```
   $ ompwrapwm twm
   ```

   The `start_ompwm` script starts `omp4Dwm` after first checking if the display server supports SGI Xinerama. The `ompwrapwm` script starts the window manager `twm` in aware window management mode under DMX. If the display server is determined to be compatible, the script starts the window manager with aware window management support enabled. If the display server is not compatible, the script will exit. The script can be made to start the window manager in unaware mode (with aware window management disabled) as a contingency.

   For more information on using the `start_ompwm` and `ompwrapwm` scripts, see their man pages or use the `-help` command-line option of the scripts as follows:

   ```
   $ start_ompwm -help
   ```

   or

   ```
   $ ompwrapwm -help
   ```

Notes:

- You can use any window manager with the `ompwrapwm` script or with the `ompstartdmx –wm` option on IRIX. Only `4Dwm` and `twm` are supported on IRIX. GNOME is recommended on Linux platforms.
- Starting an application in aware mode and then starting the window manager will result in the application's windows being unmanaged by the aware window manager. An aware window manager must be started prior to running an application in aware mode in order for its windows to be managed.

Exiting an Aware Window Manager

To exit an aware window manager, simply log out and log back in. The default window manager will again manage your display.

If you are running `omp4Dwm` under SGI Xinerama, you may also exit `omp4Dwm` by entering the following:

```
$ tellwm quit
```

Then start your original window manager.

Setting an Aware Window Manager as the Default

An alternate way to run an aware window manager is to invoke the `start_ompwm` or `ompwrapwm` *my_window_mgr* script in your `$HOME/.xsession` file. Which script should be invoked (`start_ompwm` or `ompwrapwm`) depends respectively on whether SGI Xinerama or DMX is running.

For IRIX users, the directory `/usr/share/omp/examples/X11/` contains some example `.xsession` files . For more information about `.xsession` files, see the man pages `X`(1) and `xdm`(1).

For Linux users, you must set up a default aware window manager under DMX. See section "Initializing DMX" on page 14.

Optimizing Performance

OpenGL Multipipe allows single-pipe OpenGL application to be run on multipipe environment in a transparent manner. However, OpenGL Multipipe defines some parameters that control the processing pipeline of the OpenGL stream. In order to acheive the best performance, those parameters will need to be changed depending on the application type and geometry data.

This chapter describes these parameters and some guidelines for using them in the following sections:

The release notes provide a more technical discussion some of these features.

Viewport Clipping

Applications that draw large polygons with complex texturing or shading procedures are likely to be fill-limited—that is, the rasterization stage of the graphics pipeline is the bottleneck to improving performance. If slower performance results in proportion to an increase in the OpenGL window size, this is an indicator that fill performance could be the problem.

To eliminate the pixel fill bottleneck, polygon rasterization work can be divided among multiple graphics pipes. Using OpenGL Multipipe, this can be accomplished by simply

positioning a window so that it spans multiple graphics pipes and each pipe performs an equal fraction of the rasterization work. On each component screen of the logical display, OpenGL Multipipe automatically clips the OpenGL viewport to the physical screen boundaries.

Viewport clipping is enabled by default. It can be disabled with the `omprun -novpclip` option or with the `viewportClippingMode` resource.

The fill performance benefits of viewport clipping can be more fully realized by using an SGI Scalable Graphics Compositor and specifying additional parameters to OpenGL Multipipe. For information on these parameters, see the section "Specifying Static Composited Regions for Fully Overlapped Screens" on page 43.

Geometry Culling

Applications that render large amounts of geometry in display lists can sometimes reach the limit of the polygon processing capabilities of the graphics hardware. Such an application is said to be transform-limited or geometry-limited—that is, the geometry transformation stage of the graphics pipeline is the bottleneck to improving performance. The geometry limit varies for each graphics architecture. For example, an Onyx 3000 pipe can handle millions of polygons per second; an Onyx4 pipe can handle hundreds of millions of polygons per second. If performance remains the same when lighting or textures are disabled by the application, these are indicators that geometry performance is the limiting factor.

To eliminate the geometry transformation bottleneck, OpenGL Multipipe can divide geometry among multiple pipes. By default, OpenGL Multipipe renders all geometry on each graphics pipe, even if not all of the geometry is visible on a given pipe. When geometry culling is enabled, each OpenGL Multipipe slave process renders only the geometry from display lists, vertex arrays, and immediate mode commands that is visible on its pipe.

It is also possible for OpenGL Multipipe to cull geometry to user-specified OpenGL clip planes. See the `culling.cullUserClipPlanes` resource in section "Resource Descriptions" on page 25.

Geometry culling is enabled with the `omprun -cull` command-line option or with the `culling` resource.

Small Object Culling

OpenGL Multipipe can cull objects whose screen-space bounding box is less than a threshold number of pixels. You can specify the threshold with the –`minpixels` command-line option for the `omprun` script or with the `culling.minPixels` resource. Geometry culling must be turned on in order for this feature to take effect.

This feature can increase the rendering speed in applications that have many features that are too small to discern and, therefore, may not be of immediate interest—for example, screws in CAD/CAM applications. With a proper `minpixels` culling threshold, OpenGL Multipipe culls such small features and rendering performance is improved as a result.

Display List Partitioning

When rendering display lists with OpenGL Multipipe's geometry culling option enabled, OpenGL Multipipe culls or renders an entire display list as a unit. When the original display list has a large amount of geometry and spans large areas of the scene, performance scalability can suffer.

OpenGL Multipipe can optionally spatially divide user display lists to break them into smaller ones. After the division, the smaller, more spatially coherent pieces are more friendly to load balancing and display list culling.

This feature is enabled with the `omprun -dlsplit` command-line option or with the `dlSplit` resource. Other options related to display list partitioning are described in the section "Resource Descriptions" on page 25.

Master Rendering Modes

As cited earlier, the OpenGL proxy layer of OpenGL Multipipe has two components: a master render library that intercepts OpenGL calls made by the application and render slave processes that each receive a stream of OpenGL calls from the master and perform OpenGL rendering on the application's behalf.

The master render library functions as part of the application process (that is, the "master process") and can have additional responsibilities besides intercepting, packing, and

distributing OpenGL calls to the slave processes. The master process may also render directly to a single local pipe in place of a single slave process, or it may use a local pipe only to track OpenGL state while a slave process renders to that pipe.

The `omprun –mstrmode` option (or the `masterMode` resource) allows you to specify what functions the master component performs on the single local reference pipe. The master's mode may improve or hinder OpenGL performance depending upon the behavior of a particular application. Therefore, it is important to understand the implications of each mode.

The following master modes are available:

- `off`
- `track`
- `render`

Note: To use the `track` or `render` master mode, you must use the `omprun` command to invoke the application.

Master Mode `off`

The `off` master mode is most efficient for applications that do not perform **glGet***xxx***()** calls (GL state queries) in every frame, because in this mode querying GL state requires round-trip communication to a slave. The master process does not render; it only packs and distributes GL calls to all slave processes. A slave process renders to each pipe and one of the slaves is designated to track state for any occasional **glGet***xxx***()** or **glIs***xxx***()** queries. Figure 4-1 illustrates running in `off` mode.

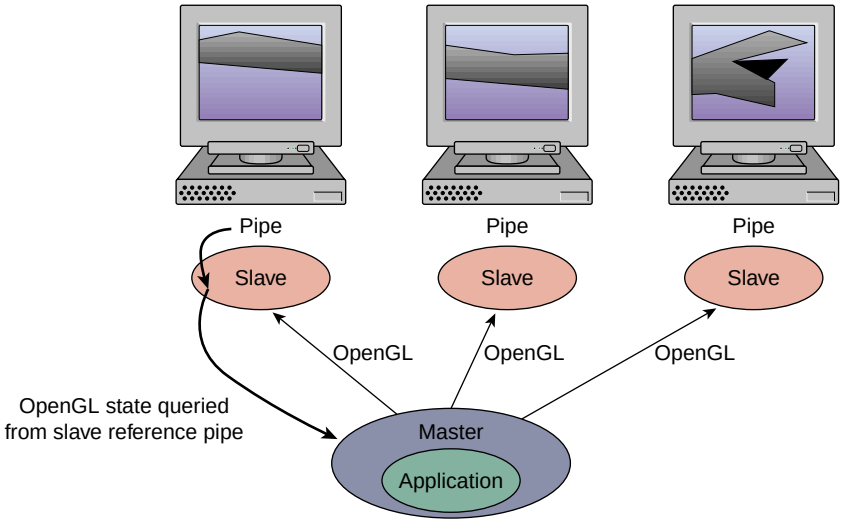

Figure 4-1 Running in Master Mode `off`

Master Mode `track`

The `track` master mode is most efficient for applications that frequently query GL state. The master process does not render but, in addition to packing and sending GL calls to all slave processes, it tracks the GL state on a local reference pipe. Slave processes render on all pipes. Figure 4-2 illustrates running in `track` mode.

Note: To use the `track` master mode, you must use the `omprun` command to invoke the application.

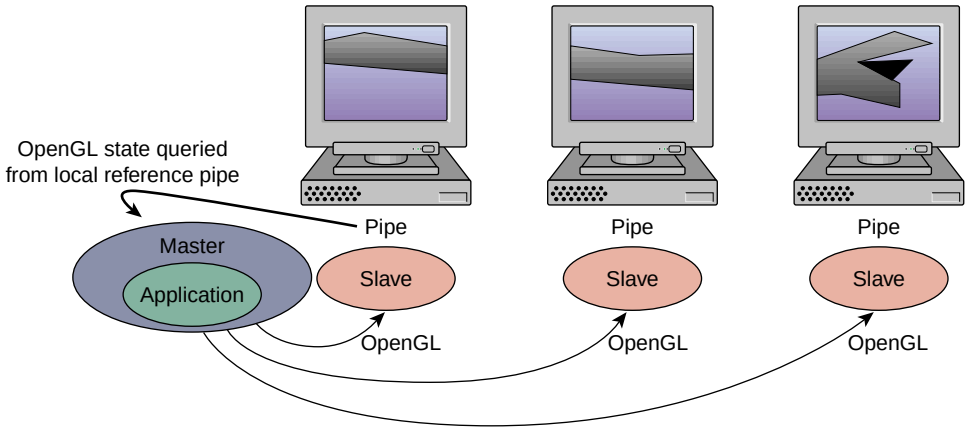

Figure 4-2 Running in Master Mode `track`

Master Mode `render`

The `render` master mode may yield slightly better performance for applications that do not use display lists and that run on systems with only two graphics pipes or with a limited number of processors. The master process renders and tracks GL state on a local reference pipe. One less slave process is needed because the application (master) process renders itself. State queries again are made to the master's local reference pipe. Figure 4-3 illustrates running in `render` mode.

Notes:

- To use the `render` master mode, you must use the `omprun` command to invoke the application.
- Some of the performance features described in this section, including geometry culling, are not available in `–mstrmode render` mode.

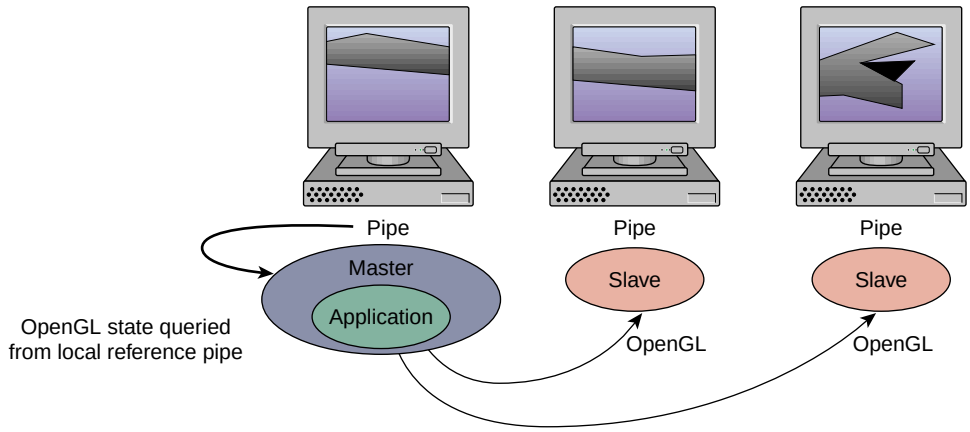

Figure 4-3 Running in Master Mode render

Frame Latency Control

OpenGL Multipipe uses a shared-memory buffer in between the application and the drawing slaves. This buffer can introduce latency—that is, multiple frames can be buffered to be consumed gradually by the slave. If the application does not call **glFinish()** by itself, then OpenGL Multipipe allows the number of buffered frames to reach a small preset limit.

The latency helps smooth out application and drawing speed differences, and thereby increase throughput. However, if the amount of latency is beyond what you can accept, it can be limited by using the `omprun -latency` command-line argument (or the `maxFramesLatency` resource) to specify the maximum latency in number of frames.

The following command forces the master and slaves to have zero frames of latency between them:

`omprun -latency 0` *app_name app_args*

The master will wait for the slaves to finish executing enough of the previous frames so that the latency is below the threshold before beginning to pack data into the buffer for the next frame.

Buffer Swap Synchronization

Variations in pixel fill, geometry load, and many other factors can lead to an unbalanced load among the graphics pipes. Some pipes will render their parts of the scene faster or more slowly than the rest. Synchronization among the pipes is required to prevent one pipe from rendering faster or slower than another, which in some cases can present visible "tearing" in the output image.

This section decribes the following synchronization schemes:

- "Software Swap Synchronization"
- "Hardware Buffer Swap Synchronization"

Software Swap Synchronization

By default, OpenGL Multipipe performs a software synchronization among the slave processes to ensure that they issue their respective swap-buffer commands at the same time. The software synchronization approximates a synchronized swap in hardware.

Software swap synchronization is enabled by default. It can be disabled with the `omprun -nosync` command-line option or with the `swapSyncMode` resource. Note that this also disables any meaningful sense of frame latency.

Hardware Buffer Swap Synchronization

Note: OpenGL Multipipe does not support this feature on Onyx4 and Silicon Graphics Prism servers.

OpenGL Multipipe supports hardware synchronization of **glXSwapBuffers()** across all pipes. Normally, when an application that is run with `omprun` makes a call to **glXSwapBuffers()**, OpenGL Multipipe sends swap-buffer requests to **all** pipes since the application window might be visible on all pipes. When multiple pipes are used to drive a large logical screen (that is, a wall display), ensuring that the actual buffer swaps happen at exactly the same time on every pipe improves the perception that the display is a single logical pipe.

You enable hardware synchronization by using `omprun -swapready` command-line argument (or the `swapSyncMode` resource set to `barrier`).

To use Swap Ready hardware synchronization, follow these steps:

1. Before using the `-swapready` option, make sure you have properly connected Genlock and Swap Ready cables to all pipes and that the pipes are configured to be genlocked.

   Running any OpenGL application that attempts to use Swap Ready hardware without proper configuration can cause a serious graphics failure. This includes all applications started with `omprun -swapready`. For information about wiring the Genlock and Swap Ready cables, see the *POWER Onyx and Onyx Rackmount Owner's Guide*. Also, see the `genlock`(1) man page.

2. When running the application with `omprun`, use the `omprun -swapready` flag:

   ```
   $ omprun -swapready app_name app_args
   ```

Using Swap Ready implicitly disables software swap synchronization. If you are not using Swap Ready hardware to synchronize **glXSwapBuffers()** calls, then OpenGL Multipipe uses a software synchronization that is less accurate.

Note: More than one application may be started with hardware swap synchronization using the `omprun -swapready` option . However, multipipe applications that support Swap Ready natively will conflict with OpenGL Multipipe if an application that was started with `omprun -swapready` is running. Likewise, if a multipipe application is already using the Swap Ready line, the `omprun -swapready` option will revert to software swap synchronization.

Limitations

OpenGL Mulitpipe allows single-pipe applications to run in a multipipe environment without any modification and without the need to recompile the application. It also allows single-pipe and multipipe applications to run concurrently on the same X server. However, OpenGL Multipipe has limitations and the following sections describe them:

For release-dependent limitations, see the OpenGL Multipipe release notes. For supported platforms and configurations, see "Supported Platforms" in Chapter 1.

Performance Enhancement

OpenGL Multipipe does not replace performance-focused multipipe APIs—such as OpenGL Performer or OpenGL Multipipe SDK—or any other custom multipipe solution. Using OpenGL Multipipe results in some minimal overhead (performance loss) for traditional single-pipe applications. This is due to the inherent cost of distributing the X and OpenGL commands among the graphics pipes.

It should be noted that process placement—that is, the way in which processes are assigned to processors—can significantly affect performance. For the affect of process placement on performance, see the OpenGL Multipipe release notes.

X Extensions

Some X extensions are not supported by SGI Xinerama, and others are not supported by DMX. For example, SGI Xinerama does not support the `SGI-VIDEO-CONTROL` extension, which permits control of video operations on the base graphics hardware, and `SCREEN-SAVER`, which is used by some screen saver programs. Applications using these X extensions may not function properly. The behavior of these applications started in unaware mode is undefined, though they will generally behave correctly on screen 0 or in aware mode. To determine whether a particular extension is supported, see the section "X Application Uses Unsupported X Extension" on page 76.

The Multipipe-Aware Window Manager

Due to the nature of the screen overlapping required for composited displays, the aware window manager is currently limited to managing aware windows on noncomposited displays only. Unaware windows will be managed properly on composited displays.

OpenGL Window Size Constraints

The hardware graphics pipes have a hardware-dependent limit on the size of the region into which an OpenGL application renders. The consequence is that an OpenGL application is constrained to draw into a limited area. Under SGI Xinerama, enlarging an OpenGL window beyond this size limit results in undefined behavior. An OpenGL

window may be placed anywhere within the the total area managed by the X server. Only the size of the region into which OpenGL renders is restricted.

Under DMX, however, OpenGL Multipipe allows OpenGL rendering to be unaffected by the window size limit of the graphics hardware. If the OpenGL application is invoked using `omprun`, the application may render into windows of any size.

Processor Requirements

On IRIX, OpenGL Multipipe requires that you use a MIPS R10000 processor or later. The following example shows how you check for the processor type:

```
$ hinv -t cpu
CPU: MIPS R16000 Processor Chip Revision: 2.1
```

SGI ProPack and OpenGL Multipipe Versions

Versions of OpenGL Multipipe that ship with SGI Propack for Linux are only supported on the version of SGI Propack with which they ship due to backwards compatibility issues on Linux.

Overlay Windows Support in DMX

DMX supports overlay windows—that is, windows that use *overlay visuals*—only if overlay visuals are available on each of the underlying X servers that DMX manages. This is the case on most SGI systems. On Onyx4 and Silicon Graphics Prism platforms, you must explicitly enable support of overlay visuals in the configuration file(s) of the underlying XFree86 server(s).

SGI Xinerama Not Supported on Onyx4 or Silicon Graphics Prism Platforms

Due to changes in the X server on Onyx4 and Silicon Graphics Prism platforms, SGI Xinerama will not be supported on these platforms. Only the DMX proxy layer is

supported on these graphics platforms. DMX and SGI Xinerama are available on all other platforms supported by OpenGL Multipipe.

Graphics Pipe Requirements

Each of the graphics pipes managed by SGI Xinerama or DMX must have identical capabilities. Each pipe must provide the same set of X visuals and GLX FBConfigs, have the same amount of texture memory, and so on.

For example, if a system has three InfiniteReality3 graphics pipes where two pipes have two raster managers (RMs) and one pipe has four RMs, the pipe with four RMs must be configured to look as if it has two RMs. This can be done by ensuring the maximum pixel depth setting on each pipe is consistent. You can inspect the pixel depth by using the command `/usr/gfx/gfxinfo`. Depending on your system type, you can adjust the pixel depth on a particular pipe with one of the following commands: `/usr/gfx/ircombine`, `/usr/gfx/sgcombine`, or `xsetmon`. See the man pages for `gfxinfo`(1), `ircombine`(1), `sgcombine`(1), and `xsetmon`(1) for more information.

Some Features Not Available on Linux

All OpenGL Multipipe features are available on IRIX platforms. However, some of these features are not available on Linux platforms. For a list of such features, see the OpenGL Multipipe release notes.

Older Platforms Require the `omprun` Command

Using the `omprun` command is only optional on Oynx4 systems running IRIX 6.5.27 or later and Silicon Graphics Prism systems running SGI ProPack 3 Service Pack 4 or later.

On systems that do not meet these requirements, failure to use the `omprun` command under DMX will cause the application to use the slower GLX indirect rendering in DMX to draw OpenGL to all screens.

Troubleshooting

This chapter describes some problems you might encounter and what to do to solve them. For additional considerations, see the OpenGL Multipipe release notes, `/usr/share/omp/release_notes/user/relnotes.html`.

This chapter documents the following problems:

Cannot Connect to the `ompslave` or `ompcull` Daemon

You will see a cannot-connect-to-daemon error if the `ompslave` or `ompcull` daemon is disabled or missing. OpenGL Multipipe installs and enables these daemons by default; therefore, this error could indicate an installation problem. If reinstalling OpenGL Multipipe does not solve the problem, the following instructions demonstrate how to manually correct the configuration problem.

On IRIX, the following commands should produce the output displayed:

```
$ grep sgi-omp /etc/services
sgi-ompslave 60000/tcp
sgi-ompswapready 60001/tcp
sgi-ompcull 60002/tcp
$ grep sgi-omp /etc/inetd.conf
sgi-ompslave stream tcp nowait root /usr/bin/ompslave ompslave
sgi-ompcull stream tcp nowait root /usr/bin/ompcull ompcull
sgi-ompswapready stream tcp wait root /usr/bin/ompswapready
 ompswapready
```

If the preceding commands do not produce the expected output, either reinstall the `omp_eoe` images or edit the `/etc/services` and `/etc/initd.conf` files to make the corrections.

If the preceding commands produce the expected output and the slaves still fail to run, restart the `inetd` daemon with the following command and look for any error messages in the file `/var/adm/SYSLOG`:

```
$ killall -HUP inetd
```

On Linux, the following commands should produce the output displayed:

```
$ chkconfig --list | grep sgi-omp
```

```
sgi-ompcull: on
sgi-ompslave: on
```

If either of the services is labeled `off`, turn them on by running the following command as `root`:

```
# chkconfig sgi-ompcull on
# chkconfig sgi-ompslave on
# killall -HUP xinetd
```

If the services do not exist, then the `sgi-omp` RPM needs to be reinstalled. Remove the installed RPM using the command `rpm -e sgi-omp` and re-install the package from the SGI ProPack CD.

Problems Enabling SGI Xinerama

On systems having only one graphics pipe or in the case where the X server is directed to handle only one pipe (see the `Xsgi`(1) man page), enabling SGI Xinerama has no effect. In these cases, SGI Xinerama will be disabled, regardless of the value of the `xinerama` flag supplied on the `chkconfig` command.

Note: SGI Xinerama is not available on Onyx4 and Silicon Graphics Prism platforms. Only the DMX proxy layer is supported on these platforms.

Problems Starting DMX

If there is a problem starting the DMX proxy server, you may see output such as the following after running `ompstartdmx`:

```
ompstartdmx fatal: An error occured when starting Xdmx
Check the Xdmx log file for details: /tmp/Xdmx.log.xxxxx
```

This can result from a number of conditions, some of which have workarounds that are described in the following paragraphs. Inspect the `Xdmx.log.`*xxxxx* file, especially toward the end of the log, for messages that indicate one of the following conditions:

- Incompatible screens, no common visuals

DMX will not create a logical display from graphics pipes with differing graphics capabilities. If the DMX proxy server detects that there are no common X visuals on the backend screens it tries to manage, DMX will abort with an error to this effect.

- Only one screen on display

  On systems having only one graphics pipe or in the case where the X server is directed to handle only one pipe, ompstartdmx will exit with an error such as the following:

  ```
  ompstartdmx fatal: Display :0.0 has only one screen.
  DMX was not started
  ```

  In these cases, it does not make sense to start DMX since there is only one pipe. However, specifying a configuration file with the `ompstartdmx –cfgfile` option will not prevent DMX from running on a single backend screen. Use the `ompstartdmx –help` option for more information.

- `ompstartdmx` fails to start or hangs with a GLX error.

  If a `GLX BadMatch` error occurs when starting DMX, the back-end screens may not have matching sets of GLX visuals. This is not a supported configuration and may be caused by a misconfiguration of the underlying X server(s), whose screens DMX manages.

  The configuration of each screen DMX manages should be identical. To verify that all screens have identical sets of visuals, inspect the output of the `glxinfo` command for each of the back-end screen(s) managed by DMX, as shown in the following:

  ```
  $ env DISPLAY=:0.0 glxinfo
  $ env DISPLAY=:0.1 glxinfo
  ```

  Piping the output through `wc` provides a quick comparison.

  For XFree86 servers, per-screen visual attributes and capabilities can be adjusted by modifying the file `/etc/X11/XF86Config-4` on Onyx4 systems and the file `/etc/X11/XF86Config` on Silicon Graphics Prism systems. On IRIX systems other than Onyx4 systems, per-screen visual attributes are configured with the `xsetmon` application.

Problems Starting Applications with omprun

If an application will not start when using the omprun command, one likely scenario is that the DISPLAY environment variable does not point to a meta display. If the DISPLAY environment variable does not point to a meta display, ensure that the following conditions true (check them in the order listed):

1. The DISPLAY environment variable points to the correct display.
2. An X proxy layer is enabled.

   See the section "Verifying That the OpenGL Multipipe Environment is Enabled" on page 32. An X proxy layer must be enabled or when you invoke an application with omprun, the application will exit with the following error:

   ```
   SGIomp fatal: ":0.0" is not a meta display
   ```

3. The application was not run from a shell that was started with the omprun -aware command or from a script that used the omprun -aware command to start the application.

   The omprun -aware command effectively disables the X proxy layer for any programs it invokes.

4. Your application does not use either the OpenGL Multipipe SDK or OpenGL Performer multipipe API.

   Recent versions of these APIs may have integrated with SGI Xinerama or DMX and may not run under OpenGL Multipipe. The solution is to run these applications as is or to simply ensure that they are run in aware mode (with omprun -aware). Another alternative is to use older versions of these APIs that do not contain the X proxy aware code.

Setting OpenGL Multipipe Resources Has No Effect

If you have made changes to an X resource file but the changes do not appear to be having an effect, set the environment variable SGIOMP_PRINT_CONFIG to 1 so that OpenGL Multipipe prints the configuration values it uses to stdout when an application runs through OpenGL Multipipe.

Changing resources in an X resource file does not have an effect until the resources are merged into the active database in the X server with the xrdb command. For more information, see the xrdb(1) man page or use the command xrdb --help.

Shared Memory Failure

You might see a shared memory failure if the slave processes cannot open a connection to a back-end display server (for example, `:0.0`). Look in the file /tmp/ompslave.log to verify that this is the problem. A message indicating the connection was refused may indicate you need to set the `XAUTHORITY` environment variable or, otherwise, run `xhost +` on the X server.

If you run the application from within the DMX session, then `XAUTHORITY` will be set correctly for you. If you are running a command from a remote shell, you might have to set the `XAUTHORITY` environment variable to point to the correct X authority file before running the application. Otherwise, you will not have permission to open a connection to `:1` or `:0`.

In some X sessions, the `XAUTHORITY` variable will point to a temporary X authority file, but in sessions where `XAUTHORITY` is not set, `xauth` defaults to `$HOME/.Xauthority`. If `XAUTHORITY` is not defined in your session, set it to the following before running your application with `omprun`:

```
$ setenv XAUTHORITY $HOME/.Xauthority
```

For more information, see the `xauth`(1) man page.

Problems Running IRIS GL Applications

OpenGL Multipipe does not support IRIS GL applications. In some cases (when the application started with the `omprun` script is an executable and not a script), `omprun` can determine if the application is based on IRIS GL. In such a case, a warning message is generated and the application will not be started, as shown in the following example:

```
$ omprun clock
omprun warning: clock is an IRIS GL application
OMP Library does not support IRIS GL applications
```

For information about how to run an IRIS GL application when an X proxy layer is enabled, see the section “Running IRIS GL Applications” on page 36.

Problems Running o32 Applications

OpenGL Multipipe does not support o32 applications. In some cases (when the application started with the `omprun` script is an executable and not a script), `omprun` can determine if the application was built using the o32 application binary interface (ABI). In such a case, a warning message is generated and the application will not be started, as shown in the following example:

```
$ omprun showcase
omprun warning: showcase is an O32 application
OMP Library does not support O32 applications
```

For information about how to run an o32 application when an X proxy layer is enabled, see the section "Running IRIX o32 Applications" on page 37.

Graphics Do Not Display Correctly on All Screens

If a graphics window displays correctly on some screens only, there are several likely scenarios, which are described in the following subsections:

- "Coding Problem in the Application"
- "You Did Not Use the omprun Script"
- "A User-Defined Script Invokes an IRIS GL or o32 Application"
- "You Are Using the Aware Window Manager"
- "Set-User-ID ("s-bit") Applications"

Coding Problem in the Application

If you are using the `omprun –cull` feature and you resize or move the application window to different screens, some applications may not draw an image properly on all screens. This can occur if an application does not call **glClear()** at the beginning of each frame (that is, it "builds up" an image, relying on a sort of rendering history from past frames). When culling is enabled, applications that do not call **glClear()** at the beginning of each new frame may have unusual rendering artifacts when they are moved from their initial window position. The culling feature by nature eliminates drawing commands that would otherwise be rendered into an off-screen region. To avoid this behavior, do not use the `–cull` option.

You Did Not Use the omprun Script

Note: This section pertains to you only if you use the SGI Xinerama X proxy layer.

If a graphics window displays correctly on one screen only (usually screen 0), ensure that you start the application with the `omprun` script. If the same behavior persists when you invoke the application using the `omprun` script, ensure that one of the other conditions described in the following subsections does not exist.

A User-Defined Script Invokes an IRIS GL or o32 Application

Note: This section pertains to you only if you use the SGI Xinerama X proxy layer.

The `omprun` script cannot detect IRIS GL or o32 applications if it starts another script that in turn starts the target application. The following shell session illustrates this case:

```
$ cd /usr/demos/General_Demos/atlantis
$ omprun ./atlantis
omprun warning: ./atlantis is an IRIS GL application
OMP Library does not support IRIS GL applications
$ omprun ./RUN
```

In the preceding session, `RUN` is a script that invokes Atlantis. `RUN` does start the application, but it will be displayed correctly on one screen only.

If you start an application by using a user-defined script, ensure that the application is not an IRIS GL or o32 application. The following session shows how to test for an application that uses IRIS GL:

```
$ elfdump -Dl /usr/sbin/clock | grep libgl.so
[1]   Oct 20 20:39:53 2000    0xe5383809   ----- libgl.so  sgi1.0
```

If there is no output, the application does not use IRIS GL.

The following demonstrates how to test for an application that uses the o32 ABI:

```
$ file /usr/sbin/iconsmith | grep '32-bit'
/usr/sbin/iconsmith: ELF 32-bit MSB mips-2 dynamic executable MIPS -
version 1
```

If there is no output, the application does not use the o32 ABI.

You Are Using the Aware Window Manager

If you started an application in aware mode (that is, by running the script `omprun -aware` *app_name*...), the application running in aware mode only draws to one screen. If you are running an aware window manager, it is possible that the window manager may position the window on a screen other than the one on which the application is drawing. To see the window rendered correctly, move the application's window to the correct screen.

Set-User-ID ("s-bit") Applications

If the `omprun` command is required to utilize the full OpenGL Multipipe environment on your platform, set-user-ID applications may not be able to utilize the full master-slave environment of OpenGL Multipipe. This is because the `omprun` script makes use of the following environment variable(s) to force an application to load the OpenGL Multipipe library `libOMPmaster.so` instead of the standard OpenGL library, `libGL.so`:

- `_RLDN32_LIST` and `_RLD64_LIST` (on IRIX)
- `LD_PRELOAD` (on Linux)

For security reasons, IRIX and Linux may ignore these environment variables for set-user-ID programs. Therefore, OpenGL Multipipe is not able to intercept and distribute OpenGL calls to all pipes. As a workaround, you may run the application under the DMX proxy layer to use its native support for GLX indirect rendering. To do so, simply run the application under DMX without using the `omprun` command. If this is not feasible, the application should run under `omprun` if you invoke the application while logged in as the user that owns or created the executable (or as `root` if administrative access is available).

Mouse Behavior Offset by a Screen

Note: This section pertains to you only if you use the SGI Xinerama X proxy layer.

If logical pipe `0` is not in the top left screen position, mouse events (such as clicks) are offset by one screen. Logical pipe `0` can be any physical pipe; it is the physical pipe specified by the first `-hw` argument in the X server configuration file, `/var/X11/xdm/Xservers`.

To work around this problem, list the graphics pipe of the monitor that is in the top left position first in the list of `-hw` arguments in the `Xservers` file. For example, in a configuration where pipes `5`, `3`, and `4` are in a linear array in that order, you would use the following –`boards` and –`hw` parameters together on one line in the `/var/X11/xdm/Xservers` file:

```
:0 secure /usr/bin/X11/X
-boards 5,3,4
-hw board=5,right=1
-hw board=3,left=0,right=2
-hw board=4,left=1
 ... other X server args ....
```

Notes:

- The –`boards` numbers are physical pipe numbers, but the indexes given to the `right`, `left`, `above`, and `below` parameters refer to the logical order of the –`hw` parameters.
- The first –`hw` parameter is that of the top, leftmost pipe and should never have a left or top neighbor.
- The lines are separated in the example only for readability.

See the `Xsgi`(1) and `xdm`(1) man pages for more information about the `-hw` options and the `Xservers` file.

Cursor Movement Anomaly When Using a DMX Configuration File

This condition may occur when the DMX screen configuration differs from the screen connectivity of the back-end X server(s) and the mouse cursor is moved quickly from one screen to another.

When a back-end X server is configured so that some of its screens are neighbors, the DMX configuration for these screens should match the back-end neighbor topology for proper mouse cursor movement. Otherwise, the mouse cursor could jump or move to

unexpected locations when it crosses a screen boundary where the backend screen layout does not match the DMX screen layout.

To avoid this condition for XFree86 backend X servers, make sure the screen arrangement in the `/etc/X11/XF86Config` file matches the screen arrangement in the DMX configuration file. By default, DMX and XFree86 manage screens left-to-right in a horizontal row. For more information about configuring XFree86, see the `XF86Config`(5x) man page.

Problems Running Inherently Multipipe Applications

Applications—such as `ircombine`, `sgcombine`, `xsetmon`, `setmon`, and `gamma`—which are designed to manage graphics hardware pipes individually, are inherently multipipe applications. Therefore, they should be started in multipipe-aware mode, as shown in the following examples:

```
$ setenv DISPLAY :0.0
$ omprun -aware gamma
```

```
$ setenv DISPLAY :0.0
$ omprun -aware sgcombine
```

This allows the application to run as if the X proxy layer were disabled. Applications with a GUI—such as `sgcombine`, `ircombine`, and `xsetmon`—will not be under window manager control unless an aware window manager is active.

Multipipe-Aware Applications Fail to Receive Events on Screen `0`

Windows of applications that are run in aware mode are not handled by ordinary window managers. This can cause some problems on screen `0` for keyboard events.

Moving away all the windows that are overlapping the aware window (even if these windows are displayed behind the aware window) will set the correct focus. The aware window will then receive the keyboard events.

Alternately, running the aware window manager will also fix the focus problem.

Nothing Displays or the Graphic Stalls or Hangs

If you start an OpenGL application with `omprun` and it does not display anything or the graphic stalls or even hangs, the source of the problem might be one of the following:

- "Coding Problem in the Application"
- "Window Exceeds Maximum OpenGL Window Size"
- "Improperly Wired Genlock or Swap Ready Cables"

Coding Problem in the Application

You may see a blank display or experience stalls or hangs for OpenGL applications that do not call **glFlush()**, **glFinish()**, or **glXSwapBuffers()** at the end of each frame. This causes OpenGL Multipipe to draw only when its internal buffer overflows. It can happen that the buffer never fills, in the case of an event-driven application—that is, the application draws one frame and waits for an event before drawing the next frame. Unfortunately, there is no workaround for applications that are not frame-based because OpenGL Multipipe relies on regular calls to the functions just cited.

Window Exceeds Maximum OpenGL Window Size

Note: This section pertains to you only if you use the SGI Xinerama X proxy layer.

If the size of a window with OpenGL content is larger than the maximum width or height allowed by the graphics hardware, undefined drawing behavior will result. Although this limit can vary depending on the graphics configuration, it is typically around 4000 pixels. Logical screen configurations with a width or height larger than the maximum OpenGL window size are particularly susceptible to this behavior. As long as an OpenGL window is smaller than the maximum size, it may be placed anywhere within the total area managed by the X server. Only the size of the region into which OpenGL renders is restricted. For more information, see "OpenGL Window Size Constraints" on page 60.

Improperly Wired Genlock or Swap Ready Cables

If you are experiencing long delays between frames of an OpenGL application (whether or not it was started with `omprun`), the condition may have resulted from using the `omprun -swapready` option with improperly configured or improperly wired Genlock or Swap Ready cables.

For more information about this problem and a workaround, see the OpenGL Multipipe release notes.

OpenGL Graphics Render Slowly

While there could be many reasons for slow rendering, ensure that you used the `omprun` command to start your OpenGL application, in particular, if you are using the DMX X proxy layer. Failure to use the `omprun` command under DMX will cause the application to draw to all screens using the slower GLX indirect rendering support in DMX instead of the OpenGL direct rendering provided through the `omprun` command. Use the `omprun` command to achieve the best rendering performance for single pipe ("multipipe-unaware") applications under DMX.

For other performance optimization suggestions, see Chapter 4, "Optimizing Performance".

To achieve the best performance with multipipe applications, see the section "Running Multipipe Applications in Multipipe-Aware Mode" on page 38.

X Applications Are Not Behaving Correctly or Fail to Start

If X applications are not behaving correctly or fail to start, consider the cases described in the following subsections:

- "X Application Uses Unsupported X Extension"
- "SGI Xinerama Client or Server Uses Nonstandard Protocol"
- "Application Window Disappears"
- "Application Explicitly Opens a Display Connection to :0.0"

X Application Uses Unsupported X Extension

Verify that the application is not using unsupported X extensions. Unfortunately, there is no way to accurately list the extensions an application uses. However, the following examples using the `nm` command give some hints about the extensions used. If an application is using an X extension, this can usually be detected by searching for occurrences of the string `extension` or for the name of a particular extension. The `xdpyinfo` command lists the names of extensions supported by the X server.

Indicating the use of the DOUBLE-BUFFER extension (DBE), the following example shows that command `gmemusage` calls `XdbeQueryExtension`:

```
# nm /usr/sbin/gmemusage | grep -i extension
[116] |2143299120| 436|FUNC |GLOB |DEFAULT  |UNDEF| XdbeQueryExtension
```

For a list of X extensions supported by SGI Xinerama, see the `Xinerama`(3X11) man page. For a list of X extensions supported by DMX, see the `Xdmx`(1) man page in the directory `/usr/share/omp/doc/user`. Applications that use unsupported X extensions may be run in aware mode by running them with the `omprun –aware` option so that they bypass the X proxy layer.

SGI Xinerama Client or Server Uses Nonstandard Protocol

The SGI Xinerama versions in IRIX 6.5.11 and earlier use a protocol that is incompatible with versions of SGI Xinerama released in IRIX 6.5.12 and later. If an application links (dynamically at run time or statically at compile time) with X client libraries that came with IRIX 6.5.11 and earlier and then attempts to make SGI Xinerama calls to an X server from IRIX 6.5.12 or later, the behavior will be undefined. Similarly, linking with X client libraries from IRIX 6.5.12 or later and connecting to an X server from IRIX 6.5.11 or earlier will also yield undefined behavior.

Since the OpenGL Multipipe layer calls `XineramaQueryVersion`, it is able to reliably detect and report this server-client version incompatibility.

If you encounter this protocol incompatibility, the workaround is to use a client library and server that both support the same SGI Xinerama version—that is, use a client library and X server from the same IRIX version. More simply stated, if you are connecting to a remote display that has SGI Xinerma enabled, ensure that both the local and remote hosts are running the same version of IRIX.

See the `Xinerama`(3X11) and `XineramaQueryVersion`(3X11) man pages for more details.

Application Window Disappears

Note: This section pertains to you only if you use the DMX X proxy layer.

This can occur if you start an application on one of DMX's backend displays without using `omprun –aware`—for example, if your `DISPLAY` environment variable is set to `:0.0` and the DMX proxy is managing display `:1`. The application will run on `:0.0`, a backend ("aware") display, but because it was not started properly in aware mode (using `omprun –aware`), the window will not be managed by the window manager running under DMX. Consequently, it may be "pushed" behind the DMX root window. Exiting the DMX session will reveal the application window. To avoid this problem, open the application in aware mode using `omprun –aware` *app_name*.

Application Explicitly Opens a Display Connection to `:0.0`

Note: This section pertains to you only if you use the DMX X proxy layer.

This problem can manifest itself in many ways:

- An apparent X error may appear in the output of the application.
- A window may suddenly disappear behind the DMX root window.
- The command `omprun` may output the following message:

  ```
  DISPLAY does not point to a meta display.
  ```

This problem originates from a program explicitly requesting an X display connection to `:0.0` even if the `DISPLAY` environment variable contains a different default display string, which is usually the case when running DMX.

The workaround is for DMX to run as display `:0` and to have the backend X servers use other display numbers so that when the application tries to open `:0.0`, it correctly connects to the DMX meta display on `:0` instead of a backend display by accident. To

have other X servers use other display numbers so that `:0` is available for DMX, do the following:

1. Open file `/var/X11/xdm/Xservers`.
2. Replace the instance of `:0` with another display number (for example, `:1` or `:5`).
3. Restart graphics.
4. Remove the `/tmp/.X11-unix/X0` Unix socket file if it exists.

   The existence of this file will cause `ompstartdmx` to start DMX on a display other than :0.

Now when you run `ompstartdmx`, by default it will use the first available X display number, which should now be `:0`.

Simultaneously Running X Servers with and without SGI Xinerama Enabled

To run X servers with SGI Xinerama enabled simultaneously with regular X servers (that is, with SGI Xinerama disabled) on the same machine, add `+xinerama` or `–xinerama` to the existing arguments in the file `/var/X11/xdm/Xservers`. This allows you to override the `chkconfig xinerama` flag. Refer to the `Xsgi`(1) and `xdm`(1) man pages for more information.

Tiled Background Image

Note: This section pertains to you only if you use the IRIX window manager `4Dwm`.

Setting a large image as the root window background image will result in having a tile image displayed across the screens. You can overcome this problem by the using `4Dwm` desktop features as follows:

- Set the following line in the `$HOME/.Sgiresources` file:

  ```
  4Dwm*SG_useBackgrounds: True
  ```

- Create the background image in the `xpm` (X Pixmap) file format. The fewer colors used in that image, the less impact it will have on the colormaps used by other applications.
- Copy the `/usr/lib/X11/system.backgrounds` file to `$HOME/.backgrounds`.
- Edit `$HOME/.backgrounds` and, using the syntax of `/usr/lib/X11/system.backgrounds` as a template, add a new setting for your image.
- Select your background from the GUI `background` program.

See the `4Dwm`(1X) man page for more information.

Flickering Grey Rubberband During Window Movement

Note: This section pertains to you only if you use the DMX X proxy layer.

This occurs as a result of the lack of overlay visual support in DMX. See "Overlay Windows Support in DMX" on page 61.

Mouse Disappears on Composited or Edge-Blended Display

When an X proxy layer is used to overlap screen regions on an edge-blended display or a compositor-based system, the cursor will seem to disappear when it enters the overlapped or uncomposited regions of the display. In X, a mouse belongs to one screen of the X server at a time. Therefore, it is normally not possible to draw the mouse multiple times (on different screens) in the overlap region.

To prevent the cursor from disappearing in these cases, you need to create additional or *duplicate cursor* images (not real cursors) where two or more screens overlap. There is still just one real cursor position on the display. The way you do this depends on your X proxy layer, SGI Xinerama or DMX. See the section "Enabling Duplicate Cursor Images in Overlap Regions" on page 44.

Problems Running Multithreaded Applications

If the application supports the use of POSIX threads (*pthreads*), use the pthread threading model with OpenGL Multipipe.

To force the use of the pthread threading model, use the `-pthread` option when starting the application, as shown in the following:

```
$ omprun -pthread app_name
```

`ompstartdmx` Does Not Start a Window Manager (Linux only)

With no arguments, `ompstartdmx` starts only an xterm on Linux. To start GNOME or KDE under DMX, use the `ompstartdmx –session` option. OpenGL Multipipe also installs some special GNOME and KDE login options that will appear in the `gdm` login screen. See `ompstartdmx –help` and section "Initializing DMX" on page 14.

Problems with Aware Window Management

The following subsections detail problems with aware window management and workarounds:

- "Windows of Some Aware Applications are Not Managed"
- "Problems with Desktop Background Images"
- "Mouse Events Sometimes Register on the Wrong Screen"
- "Ghost Windows Appear In Overlap Regions on Edge-Blended Displays"

Windows of Some Aware Applications are Not Managed

For windows of aware applications to be managed, first start the aware window manager, then start the desired application in aware mode. If the reverse is done, the windows will not be able to be manipulated.

Problems with Desktop Background Images

Note: This section pertains to you only if you use the IRIX window manager `4Dwm` or its Xinerama-aware variant `omp4Dwm`.

Under SGI Xinerama, if desktop background images do not appear when the `start_ompwm` script is used to start `omp4Dwm`, you must enable the `SG_UseBackgrounds` resource for `4Dwm`. This can be done through an X resource file or on the `start_ompwm` command line, as shown in the following command entries:

```
$ tellwm quit
$ start_ompwm -xrm "*SG_UseBackgrounds: True"
```

If you started DMX as the X proxy layer using `ompstartdmx`, `4Dwm` is started by default. If you need to explicitly start `4Dwm` with proper background images under DMX, you may do so by entering the following:

```
$ tellwm quit
$ ompwrapwm 4Dwm -xrm "*SG_UseBackgrounds: True"
```

Mouse Events Sometimes Register on the Wrong Screen

Note: This section pertains to you only if you use the SGI Xinerama X proxy layer.

A number of known mouse pointer and keyboard focus issues arise with aware windows that reside on screens other than physical screen `0`—that is, on any of the physical screens `1`.. `n` that SGI Xinerama is managing. Events (mouse clicks, keyboard strokes, etc.) are sometimes generated as though the mouse were on screen `0` instead of on the correct screen (`1`.. `n`). This frequently occurs when a popup menu of an aware window is open and the mouse is clicked outside the menu.

To avoid this particular problem, close the popup menu first (for example, by using a keyboard shortcut to close the menu) then click the mouse.

Ghost Windows Appear In Overlap Regions on Edge-Blended Displays

Aware windows bypass the X proxy layer and are only created on one physical screen, but when an aware window manager manages an aware window, it creates window frames on each screen. The frames are multipipe-transparent—that is, drawn on every screen. However, the application window within the frame is multipipe-aware—that is, drawn only on one screen.

Since an aware application window is not drawn on every screen, the multipipe-transparent frame behind the application window will "show through" in screen-overlap regions on an edge-blended display.

To work around this problem, you may want to run your application in a window of a size and position such that it does not overlap any of the screen-overlap regions of the display. Alternatively, you may want to temporarily quit the aware window manager.

Applications Do Not Behave Correctly in Aware Mode

If you are using DMX as your proxy X layer, note that the `ompstartdmx` script sets the environment variable `SGIOMP_META_DISPLAY`, which is important for running applications in aware mode under DMX. If you are starting an application in aware mode from a remote shell or other login shell, this variable will not be set. If unexpected results occur when you try to run an application in aware mode, ensure that the `SGIOMP_META_DISPLAY` variable is set to the DMX display (often `:1`).