SGI® OpenGL Multipipe™ User's Guide

Version 2.3

007-4318-012

CONTRIBUTORS

Written by Ken Jones and Jenn Byrnes

Illustrated by Chrystie Danzer

Production by Karen Jacobson

Engineering contributions by 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 2.3 contains the following changes:

- Performance enhancement options including (disabled by default):
  - Geometry culling to optional, additional OpenGL clip planes
  - OpenGL viewport clipping and geometry culling to improve fill and geometry performance when using a compositor
  - Spatial partitioning of large display lists for efficient geometry culling
  - Run-time hint to improve vertex array performance
- Baseline performance enhancements over OMP 2.2
  - Swap synchronization and frame latency control improvements between slave rendering processes
  - Capablility to expand the OpenGL viewport size beyond the single-pipe limit
  - Improved performance when using geometry culling
- Support for Onyx4 OpenGL extensions
- Support for additional servers. The list of supported servers now includes the following:

  - SGI Onyx 3000 series with InfinitePerformance graphics
  - SGI Onyx 3000 series with InfiniteReality graphics
  - SGI Onyx 350
  - Silicon Graphics Octane2
  - Silicon Graphics Onyx
  - Silicon Graphics Onyx2
  - Silicon Graphics Onyx4 UltimateVision
  - Silicon Graphics Tezro

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
  - o Run-time hint to improve vertex array performance
- Baseline performance enhancements over OMP 2.2
  - o Swap synchronization and frame latency control improvements between slave rendering processes
  - o Capablility to expand the OpenGL viewport size beyond the single-pipe limit
  - 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

- Silicon Graphics Onyx4 UltimateVision
- Silicon Graphics Tezro

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 InfinitePerformance: Scalable Graphics Compositor User's Guide*
- *IRIX Admin: Software Installation and Licensing*

These books might also be helpful:

- Neider, Jackie, Tom Davis, and Mason Woo, *OpenGL Programming Guide*. Reading, Mass.: Addison-Wesley Publishing Company, Inc., 1993. A comprehensive guide to learning OpenGL.
- Nye, Adrian, *Volume One: Xlib Programming Manual*. Sebastopol, California: O'Reilly & Associates, Inc., 1991.

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.
- 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 and Configurations"

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.

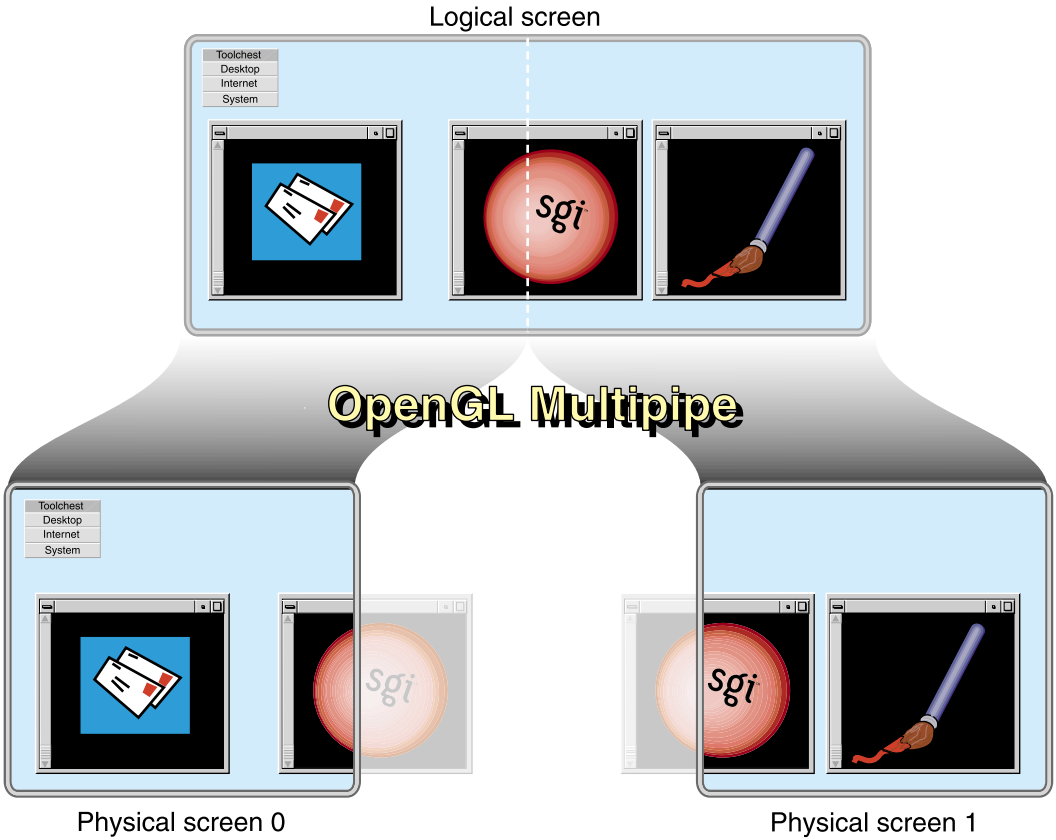

Figure 1-1 OpenGL Multipipe with Non-Overlapping Screens

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.

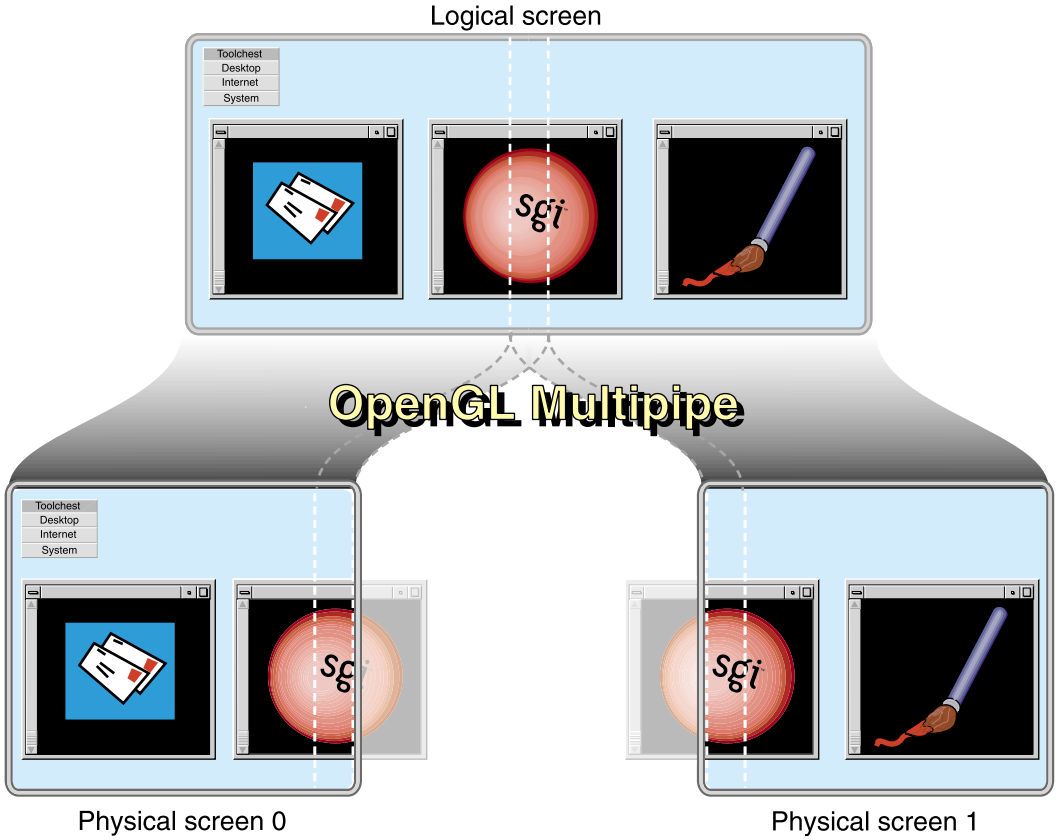

Figure 1-2 OpenGL Multipipe with Overlapping Screens

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. In this regard, a single logical display, OpenGL Multipipe 2 is similar to the first-generation product OpenGL Multipipe 1.

Unlike OpenGL Multipipe 1, however, the architecture of OpenGL Multipipe 2 allows the application processing and the rendering to occur in separate processes instead of separate threads of the same process. The separation improves application compatibility and is a step toward providing better scalability.

Like OpenGL Multipipe 1, OpenGL Multipipe 2 is a set of protocols and proxies coupled with clients and servers. Both versions of the product use an X proxy layer to hide the physical screen layout. 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. In OpenGL Multipipe 1, the X proxy layer is the SGI Xinerama X extension. In OpenGL Multipipe 2 (OpenGL Multipipe 2.2 and later), the X proxy layer may be SGI Xinerama or the new Distributed Multihead X (DMX) proxy server.

OpenGL Multipipe 1 uses the Transparent OpenGL (TGL) library to send OpenGL calls to each real pipe. In contrast, OpenGL Multipipe 2 uses a 3D proxy library and render servers for this purpose.

Table 1-1 charts the primary interfaces of OpenGL Multipipe 1 and OpenGL Multipipe 2.

Table 1-1 Primary OpenGL Multipipe Interfaces

Product	X Server Interface	Interface with OpenGL and Graphics Pipes
OpenGL Multipipe 1	SGI Xinerama extension	Transparent OpenGL (TGL)
OpenGL Multipipe 2	SGI Xinerama extension or Distributed Multihead X (DMX) proxy server	Proxy render library and render servers

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. 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. In OpenGL Multipipe 1, the Transparent OpenGL (TGL) library was designed to handle the OpenGL side of any application.

The "master" proxy library corresponds to the TGL layer in OpenGL Multipipe 1. It intercepts OpenGL calls to enable distribution to multiple pipes, but it does this by sending an OpenGL wire protocol stream to separate slave render processes rather than by spawning threads within the application to render to local pipes as does TGL. Figure 1-3 illustrates the functions of the master proxy library.

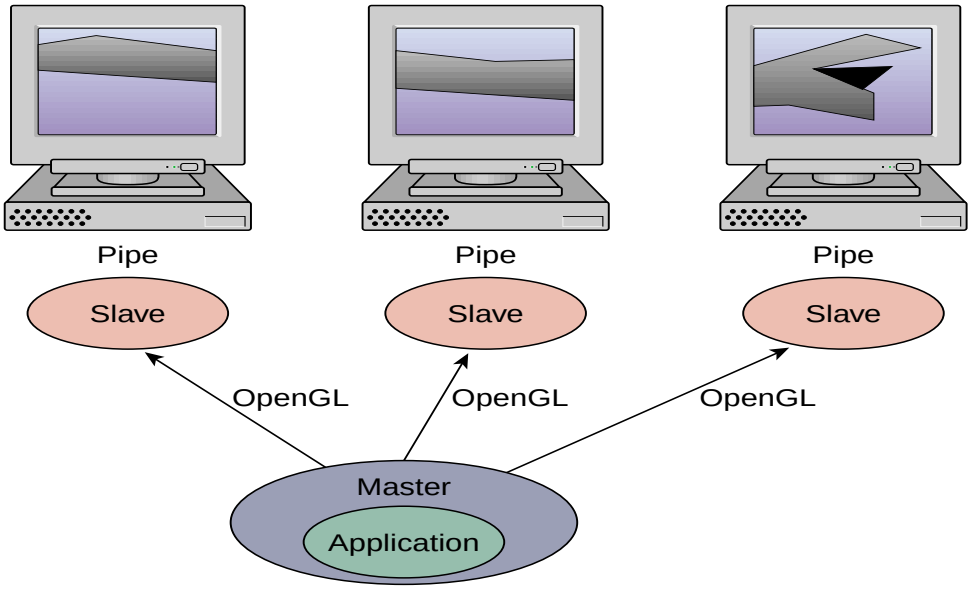

Figure 1-3 Master Proxy Library Functions

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).

3D (Slave) Render Servers

A “slave” render server receives connections from applications running under the “master” render library of OpenGL Multipipe. It translates the OpenGL wire protocol stream into OpenGL commands that are executed locally.

The single `ompslave` process is responsible for spawning multiple slave render servers. It does this for each screen per application that connects to it through the master render library.

Supported Platforms and Configurations

OpenGL Multipipe 2.3 requires the following:

- IRIX 6.5.15 or later operating system
  (IRIX 6.5.22 or later plus patch 5336 for Onyx4 platforms)
- One of the following servers:
  - Silicon Graphics Onyx
  - Silicon Graphics Onyx2
  - SGI Onyx 3000 with InfiniteReality graphics
  - SGI Onyx 3000 with InfinitePerformance grephics
  - SGI Onyx 350
  - SGI Onyx4 UltimateVision
  - Silicon Graphics Octane2
  - Silicon Graphics Tezro
- MIPS R10000 or later processor

Installing OpenGL Multipipe

This chapter 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 6.5.22 or later plus patch 5336 for 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 2.3

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.
`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`.

You may check the release notes on the SGI website cited in step 1 for any critical updated information between releases.

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 and ensure that the `ompslave` 3D render server daemon is running. This will cause all applications to see a single logical pipe. To deactivate OpenGL Multipipe, disable the X proxy layer and (optionally) stop the `ompslave` daemon. Some of the steps required to enable or disable OpenGL Multipipe requires `root` access. This section notes this requirement in the applicable steps.

This section describes the following tasks:

Ensuring That the `ompslave` Render Server is Running

If the `ompslave` render server is running, the following two daemons should be running:

- `ompslave`
- `ompswapready`

You only need to explicitly start or restart these daemons after installing or upgrading OpenGL Multipipe. The daemons will be started automatically when the system reboots if the `chkconfig` flag `omp` is set to `on` (which is the default value).

To verify that the two daemons have been started, enter the following:

```
$ ps -e | grep omp
```

If output similar to the following appears, the `ompslave` render server and `ompswapready` daemons are running:

```
1099 ?       0:00 ompslave
1101  ?       0:00 ompswapre
```

To start the two daemons, enter the following as `root` in an IRIX shell:

```
# /etc/init.d/omp stop; /etc/init.d/omp start
```

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. Ensure that SGI Xinerama is not currently enabled.

   To determine if SGI Xinerama is enabled, see the section "Verifying That the OpenGL Multipipe Environment is Enabled" on page 17. 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. 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 an IRIX 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, `ompstartdmx` will run the following clients unless a session script has been specified: `4Dwm`, `toolchest`, and `winterm`.

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.

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 18).

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 servers. Only the DMX proxy layer is supported on Onyx4 servers.

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 18).

Verifying That the OpenGL Multipipe Environment is Enabled

The OpenGL Multipipe environment is enabled if the following two conditions are true:

- An X proxy layer is enabled (DMX or SGI Xinerama).
- The `omp` daemons `ompslave` and `ompswapready` are running.

To verify that an X proxy layer is enabled, ensure that your `DISPLAY` environment variable is pointing to the correct display and enter the following commands in an IRIX 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.

To verify that the two `omp` daemons are running, enter the following:

```
$ ps -e | grep omp
```

If output similar to the following appears, the `ompslave` and `ompswapready` daemons are running:

```
1099 ?       0:00 ompslave
1101 ?        0:00 ompswapre
```

Disabling the OpenGL Multipipe Environment

To disable the OpenGL Multipipe environment, do the following:

- If enabled, disable DMX.

  To end the DMX session, select **Desktop > Logout** from the Toolchest or run `/usr/bin/X11/endsession`. You may also force the DMX server to exit by pressing `Ctrl+Alt+q`.

  After the session has ended, you will be returned to your regular X session.

If you have configured DMX to run automatically upon login and you end the DMX session, it will automatically end the regular X session as well and you will return to the login screen. To permanently disable DMX from starting upon login, revert the DMX-related changes you made to your `.xsession` file, or delete or rename your `$HOME/.xsession` file.

- 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.

- Optionally, enter the following as `root` in an IRIX shell to stop the `ompslave` render server:

  ```
  # /etc/init.d/omp stop
  ```

  Stopping the `ompslave` daemon is optional because it sits idle unless an OpenGL program is started with `omprun`. The `ompslave` daemon may be left running even when the X proxy layer is disabled.

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: Failure to use the `ompun` command under DMX will cause the application to use the slower GLX indirect rendering support in DMX to draw OpenGL to all screens.

The `omprun` script 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` script sets the `_RLD_LIST` environment variable (actually `_RLDN32_LIST` and `_RLD64_LIST`) to use the `libOMPmaster.so` library of matching format prior to using the `libGL.so` library.

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 how to best run different types of graphics applications:

- "Running OpenGL Single-Pipe Applications"
- "Running Pure X Applications"
- "Running IRIS GL Applications"
- "Running o32 Applications"
- "Running Multipipe Applications in Multipipe-Aware Mode"

For more information on running applications with OpenGL Multipipe, see the OpenGL Multipipe release notes, which are in the following file:

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

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 an IRIX 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

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 o32 Applications

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 an IRIX 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.

Performance Enhancing Features

OpenGL Multipipe has several methods of optimizing OpenGL applications to work correctly and efficiently across multiple pipes. This section provides an overview of the following performance features and some guidelines for their use:

The release notes provide a more technical discussion of each 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.

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" on page 33.

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.

This feature is enabled with the `omprun -cull` command-line option. Other options related to geometry culling are described in the `omprun -help` text and in the Usage Tips and Tricks section of the release notes.

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. Other options related to display list partitioning are described in the `omprun -help` text and in the Usage Tips and Tricks section of the release notes.

Static Vertex Arrays

Unlike display lists, which reside on the graphics pipe and cannot be changed once they are created, vertex arrays reside in memory that the application controls, and their contents can be changed at any time.

Since the application controls the memory in which the vertex arrays are held, OpenGL Multipipe has no way of knowing if the array contents have been changed. Consequently, at a high performance cost, OpenGL Multipipe must pack and send the vertex array memory to the slave processes each time the application issues a **glDrawArrays()** or **glArrayElement()** call.

To help OpenGL Multipipe avoid packing and resending vertex arrays every time one is rendered, you can specify a hint to OpenGL Multipipe that specifies that all vertex arrays that the application uses are "static", meaning that the application does not change the content of a vertex array between any two calls that reference that particular array. With this assumption, OpenGL Multipipe transfers a vertex array only once to the slaves, which then cache it. If the application does try to change the contents of a vertex array, the change will not be visible and incorrect drawing may result.

Other options related to static vertex arrays are described in the `omprun -help` text and in the Usage Tips and Tricks section of the release notes.

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 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:

- `–mstrmode off`
- `–mstrmode track`
- `–mstrmode render`

`–mstrmode off` Mode

The `–mstrmode off` mode is most efficient for applications that do not perform **glGet***xxx***()** calls (GL state queries) in every frame since 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. Slave processes render and one special slave process tracks GL state for any occasional **glGet***xxx***()** calls. Figure 3-1 illustrates running in `–mstrmode off` mode.

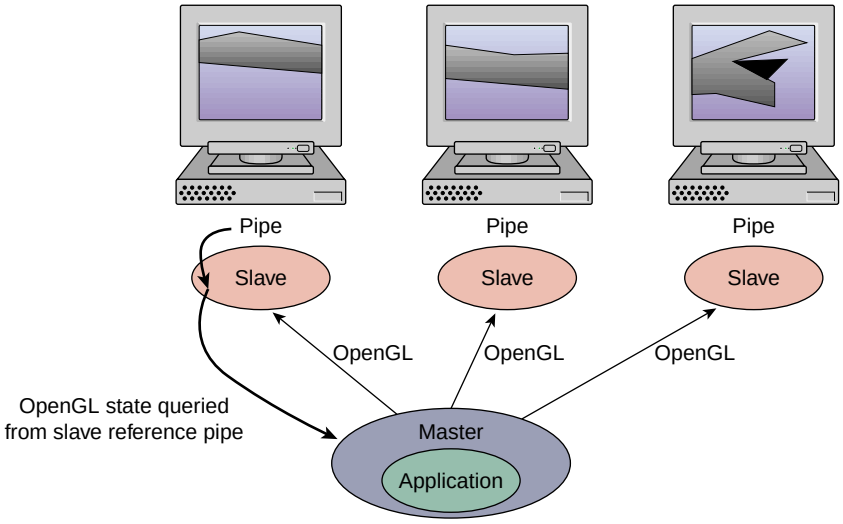

Figure 3-1 Running in –`mstrmode off` Mode

–`mstrmode track` Mode

The –`mstrmode track` 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 3-2 illustrates running in –`mstrmode track` mode.

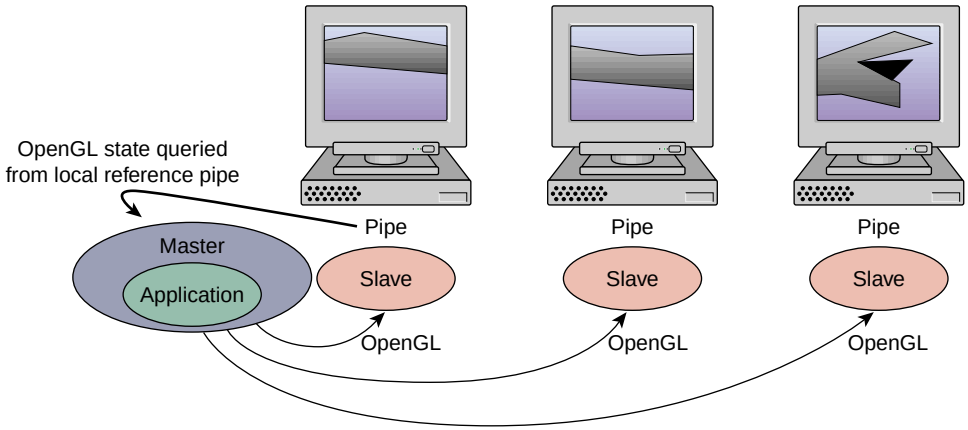

Figure 3-2 Running in –`mstrmode track` Mode

`–mstrmode render` Mode

The –`mstrmode render` 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 3-3 illustrates running in –`mstrmode render` mode.

Some of the performance features described in this section, including geometry culling, are not available in -`mstrmode render` mode.

Note: Use the `omprun –cull` option to enable culling for an application. This option is available with the –`mstrmode off` and –`mstrmode track` options. You cannot use culling with the –`mstrmode render` option.

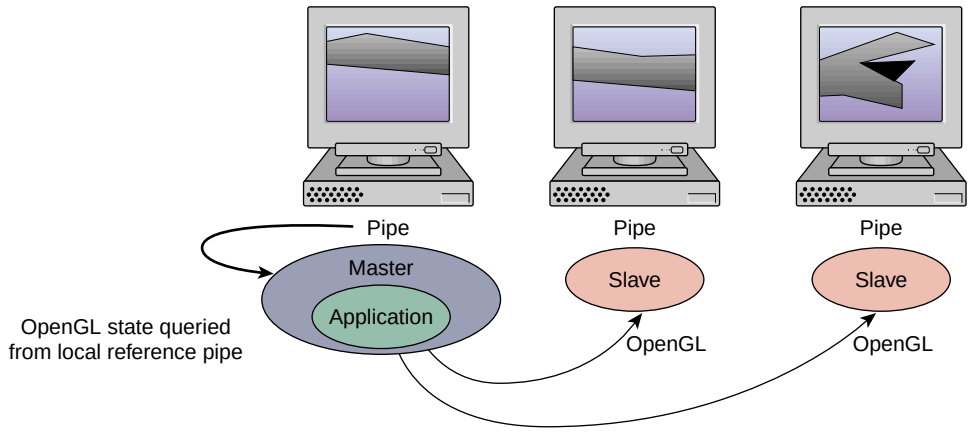

Figure 3-3 Running in –`mstrmode` render Mode

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 to specify the maximum latency in number of frames. For more information related to frame latency control, see the `omprun -help` text and the Usage Tips and Tricks section of the release notes.

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. Note that this also disables any meaningful sense of frame latency.

Hardware Buffer Swap Synchronization

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.

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.

Using an SGI Scalable Graphics Compositor 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.

Enabling Duplicate Cursor Images in Overlap Regions

Note: This feature is available under SGI Xinerama and is not available on Onyx4 platforms.

When SGI Xinerama is used to overlap screen regions on an edge-blended display or compositor-based system, the cursor will seem to disappear when it enters the overlapped or uncomposited regions of the display.

In IRIX 6.5.20 or later, you can use a new X server feature that prevents the cursor from disappearing in these cases. It causes additional **cursor images** (not real cursors) to appear on all pipes contributing to the overlapped regions. To enable this feature, add 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.

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 14.

Specifying Static Composited Regions

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 31 and “Configuring Composited Screens with DMX” on page 32 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`.

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

   ```
   $ setenv DISPLAY :0.0;omprun -aware /usr/gfx/sgcombine
   ```

   For more information about setting up composited screens with `sgcombine`, see the *SGI InfinitePerformance: 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 backend 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.

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), `4Dwm` 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 an IRIX 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 36 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, but currently only `4Dwm` and `twm` are officially supported.
- 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.

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).

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 overlapping by specifying the `xoffset` and `yoffset` arguments (in units of pixels) of the `-hw` parameters in the file `/var/X11/xdm/Xservers`. See the `Xsgi`(1) and `xdm`(1) man pages for a detailed description.

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 and Configurations" 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.

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 53.

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 continue to be managed properly.

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. For InfiniteReality2 graphics subsystems, it depends on the number of raster managers and on the selected pixel depth. Typically, it is limited to a 3840 x 3840 pixel area. OpenGL Multipipe does not allow you to extend this hardware limit. The consequence is that an OpenGL application is constrained to draw into a limited area. 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.

Processor Requirements

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
```

Overlay Windows Support in DMX

DMX currently does not support overlay windows--that is, windows that use *overlay visuals*. These are special windows that are supported on most SGI systems, but they are not currently supported in DMX. The consequences of this will vary from application to application. Some applications may not run, others may not draw properly, and some may function normally. One example of the lack of overlay support is the red "rubberband" that appears around windows managed by `4Dwm` when `Opaque Window Move` is turned off. The rubberband is normally drawn into an overlay window, and without overlay windows it appears as a flickering gray rubberband instead.

SGI Xinerama Is Not Supported on Onyx4 Platforms

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

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, which are in the following file:

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

This chapter documents the following problems:

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.

SGI Xinerama is not available on Onyx4 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.

Problems Starting Applications with omprun

If an application will not start when using the omprun command, there are several likely scenarios, which are described in the following subsections:

- "DISPLAY Does Not Point to a Meta Display"
- "Using omprun without the ompslave Render Server"
- "Shared Memory Failure"

DISPLAY Does Not Point to a Meta Display

If the DISPLAY environment variable does not toa 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 17. 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: DISPLAY does not point to 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.

Using omprun without the ompslave Render Server

The OpenGL Multipipe layer needs to have the ompslave render server running. If none is available on the host you are using and you invoke an application using omprun, the application will exit. The following examples of shell sessions show the result of trying to run the application ideas on a host that does not have the ompslave render server running:

- Example 1:

  ```
  $ omprun /usr/demos/General_Demos/ideas/ideas
  omprun fatal: ompslave daemon is missing
   please run /etc/init.d/omp stop; /etc/init.d/omp start
  ```

  Stop and restart the ompslave render server, or simply run the application without the omprun script. If the error persists, make sure that the chkconfig flag omp is set to on; otherwise, /etc/init.d/omp start will have no effect. To do this, type the following command as root:

  ```
  # chkconfig omp on
  ```

- Example 2:

  ```
  $ omprun /usr/demos/General_Demos/ideas/ideas
  SGIomp fatal: failed to connect to <hostname>: Connection refused
  ```

  The <*hostname*> field is the display server where OpenGL Multipipe expects to find an ompslave daemon.

Shared Memory Failure

After upgrading to a newer version of OpenGL Multipipe, running an application with omprun may produce the following error:

```
SGIomp fatal: Failed to attach to shared memory object: No such file or
directory
```

This message indicates that the ompslave process of the older version is still running. Be sure to run the following commands as root before using a new version of OpenGL Multipipe for the first time:

```
# /etc/init.d/omp stop; /etc/init.d/omp start
```

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 20.

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 o32 Applications" on page 21.

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 problem 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 problem 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

OpenGL Multipipe cannot override the OpenGL calls of set-user-ID applications. This is because the `omprun` script makes use of the `_RLDN32_LIST` and `_RLD64_LIST` environment variables to cause an application to load OpenGL Multipipe's `libOMPmaster.so` instead of the real OpenGL library, `libGL.so`. For security reasons, IRIX may ignore the `_RLDN32_LIST` and `_RLD64_LIST` 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 problem 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.

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

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 40.

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 when using the `omprun` command, see the section "Performance Enhancing Features" on page 23.

To achieve the best performance with multipipe applications, see the section "Running Multipipe Applications in Multipipe-Aware Mode" on page 22.

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 problem 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 problem 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 manage `:0` and to have the backend X servers manage 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 manage 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.

Now when you run `ompstartdmx`, by default it will manage 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

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 problem 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 41.

Mouse Disappears in Overlap Region

Environments such as SGI Reality Center facilities with overlapping projection systems typically use projectors with edge blending capability. The mouse will fade away when dragged towards the edge of a screen unless the “duplicate cursors” feature is enabled. This feature is described in the section “Enabling Duplicate Cursor Images in Overlap Regions” on page 32 and requires IRIX 6.5.20 or later.

Note: This feature is not available on Onyx4 platforms.

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.

Adding the `-phantomcursors` option to the X server command line in the `/var/X11/xdm/Xservers` file tells the X server to draw extra cursor images when the mouse is in a region where two or more screens overlap. There is still only one real cursor position on the display.

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
```

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

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 problem 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.