SGI® OpenGL Multipipe™ User's Guide

007-4318-009 Version 2.1

CONTRIBUTORS

Written by Ken Jones and Jenn Byrnes

Illustrated by Chrystie Danzer

Edited by Susan Wilkening

Production by Glen Traefald

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

New Features in This Release

OpenGL Multipipe 2.1 contains the following changes:

- 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

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-aware 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

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*
- *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, programming language structures, and URLs.
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, and menus. Functions are also denoted in bold with following parentheses.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names.

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
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043-1351

- Send a fax to the attention of “Technical Publications” at +1 650 932 0801.

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 2"
- "Components of OpenGL Multipipe 2"
- "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 all 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.

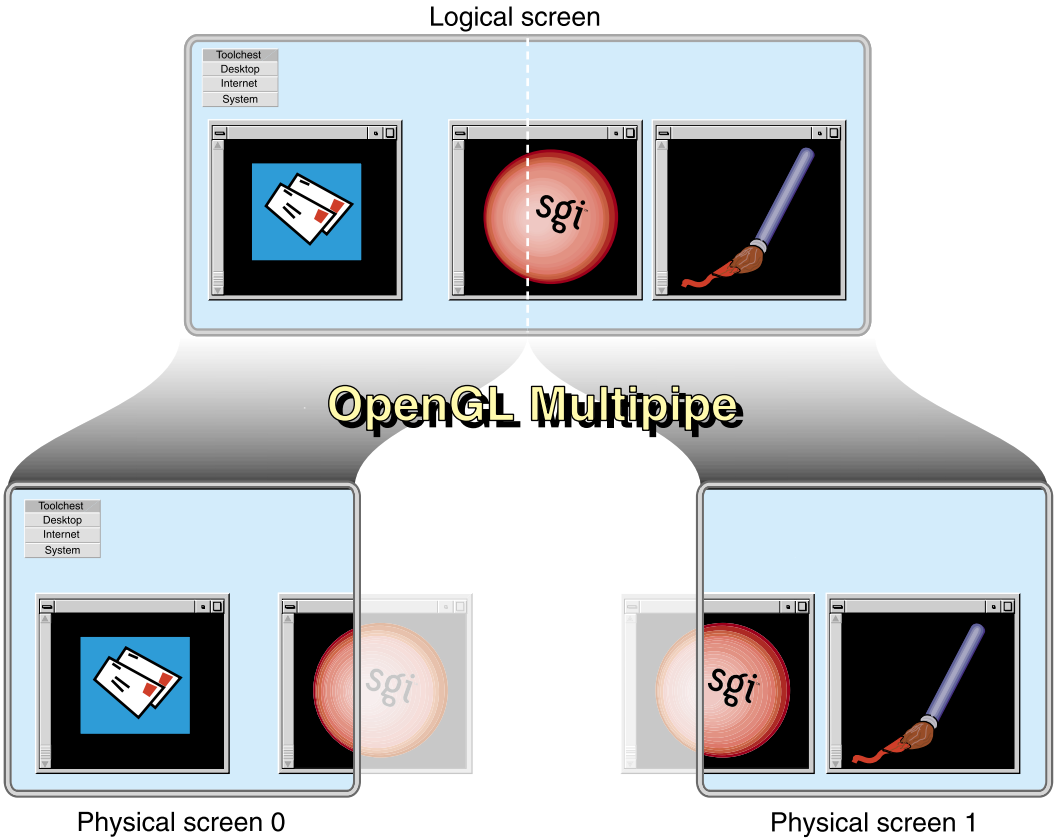

Figure 1-1 OpenGL Multipipe with Non-Overlapping Screens

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.

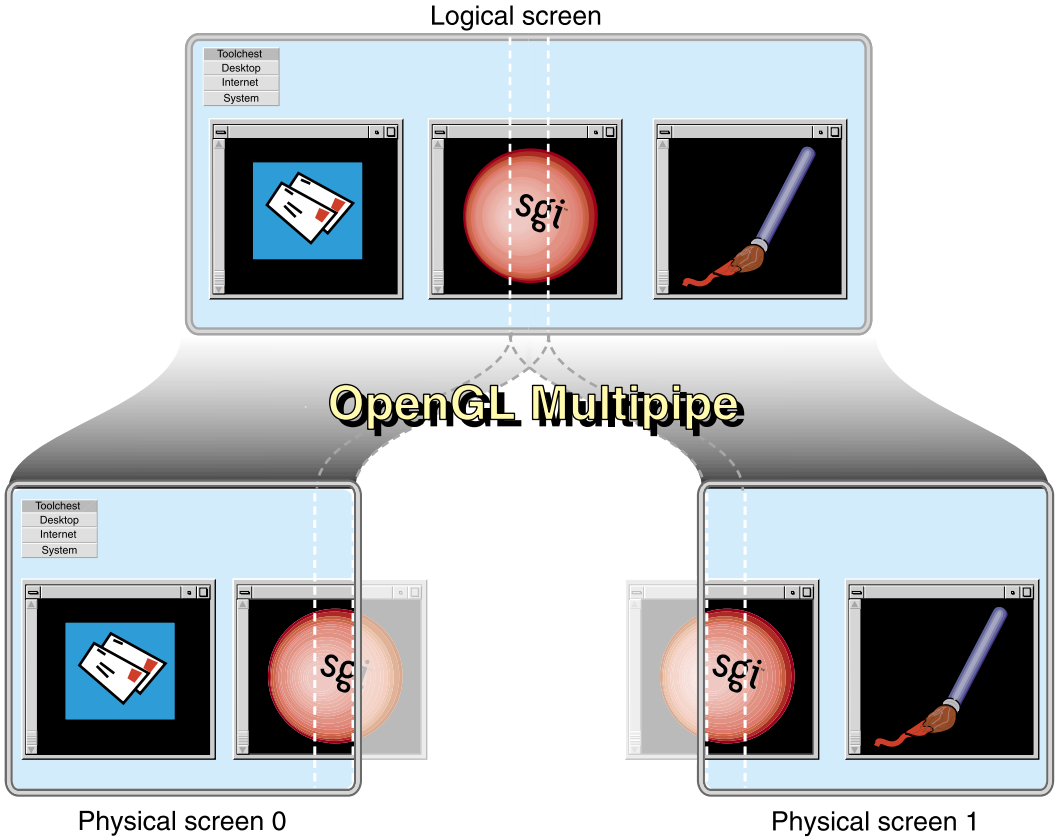

Figure 1-2 OpenGL Multipipe with Overlapping Screens

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

Architecture of OpenGL Multipipe 2

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 the SGI Xinerama extension of the X server to hide the physical screen layout. This X server interface presents a single logical pipe or "meta screen" to all applications and allows their windows to be freely moved across or to span any set of pipes.

OpenGL Multipipe 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	Proxy render library and render servers

Components of OpenGL Multipipe 2

OpenGL Multipipe 2 has the following components:

- X server with the SGI Xinerama extension
- 3D proxy render library
- 3D render servers

SGI Xinerama

SGI Xinerama is an enhanced version of the Xinerama extension of the X server. For pure X applications—that is, applications that do not use other graphics libraries (such as OpenGL) to draw into their windows—the Xinerama extension 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 SGI Xinerama X server 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.

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 can open a direct connection to a graphics pipe. This means that the application is bypassing the X protocol (and SGI Xinerama) in order to draw in the windows through this direct connection. SGI Xinerama, 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 slave render servers rather than by spawning threads to render to local pipes as does TGL. In addition to sending an OpenGL stream to each slave, the master also has the capability of rendering directly to a single local pipe.

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 2.1 requires the following:

- IRIX 6.5.18 or later operating system
- One of the following servers:
  - Silicon Graphics Onyx
  - Silicon Graphics Onyx2
  - SGI Onyx 3000, InfiniteReality
  - SGI Onyx 3000, InfinitePerformance
  - Silicon Graphics Octane2
- MIPS R10000 or later processor

OpenGL Multipipe 2.1 supports logical screen configurations of no more than four pipes. The pipes used to make up the logical screen must all be identical.

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 Onyx, Onyx2, Onyx 3000, or Octane2 system with MIPS R10000 or later processors
- Software:
  - IRIX 6.5.18 or later
  - C++ Standard Execution Environment (c++_eoe)

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:

New-32	32-bit. Located in the `/usr/lib32` directory. Usable only on IRIX 6.2 and later operating system releases. New-32 operates at increased efficiency in many situations.
New-64	64-bit. 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 2.1

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/doc/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.
`omp_eoe.man.wm`	Contains documentation related to the multipipe-aware window manager.

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 server, 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:

- "Setting up the OpenGL Multipipe Environment"
- "Running Applications with OpenGL Multipipe"
- "Managing Windows for Aware Applications"
- "Configuring Overlapping Screens"

Setting up the OpenGL Multipipe Environment

To begin using OpenGL Multipipe, you must enable the SGI Xinerama extension of the X server and start the `ompslave` 3D render server. This will cause all applications to see a single logical pipe. To deactivate OpenGL Multipipe, stop the `ompslave` render server and disable the SGI Xinerama extension. Both enabling and disabling OpenGL Multipipe requires `root` access to restart the X server. This section describes the following tasks:

- "Enabling the OpenGL Multipipe Environment"
- "Verifying the OpenGL Multipipe Environment is Enabled"
- "Disabling the OpenGL Multipipe Environment"

Enabling the OpenGL Multipipe Environment

To enable the OpenGL Multipipe environment, 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
```

2. Start the `ompslave` render server.

   Enter the following as root in an IRIX shell:

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

3. 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) &
   ```

A `chkconfig` flag, `omp`, is created and set to true upon installation. This flag allows or prevents the `ompslave` render server from being started by the command `/etc/init.d/omp start`, which will be executed automatically at boot time. Although having it `on` all the time will not affect system behavior or performance, you may prevent the `ompslave` render server from being started by typing the following as `root`:

```
# chkconfig omp off
```

Thus, to enable the OpenGL Multipipe environment, you will need to execute the following before the previous step 2:

```
# chkconfig omp on
```

This flag affects only the behavior of the `ompslave` render server. It does not in any way affect the behavior of SGI Xinerama.

Verifying the OpenGL Multipipe Environment is Enabled

To verify that the OpenGL Multipipe environment is enabled, enter the following commands in an IRIX shell:

```
$ xdpyinfo -display :0.0 | grep SGI-XINERAMA
```

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

```
$ ps -e | grep ompslave
```

If output similar to the following appears, the `ompslave` render server is started:

```
16639 pts/4     0:00  ompslave
```

Disabling the OpenGL Multipipe Environment

To disable the OpenGL Multipipe environment, perform the following steps:

1. Stop the `ompslave` render server.

   Enter the following as `root` in an IRIX shell:

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

2. Disable SGI Xinerama.

   Enter the following to disable SGI Xinerama:

   # **chkconfig xinerama off**

3. 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) &**

Running Applications with OpenGL Multipipe

Plain X applications will generally run under SGI Xinerama without assistance. OpenGL (3D) applications need to use the OpenGL Multipipe 3D proxy library to handle 3D rendering correctly 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**

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
```

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. Any OpenGL application started with the OpenGL Multipipe environment enabled and 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.

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 automatically started in transparent multipipe mode.

Running Pure X Applications

As noted in Chapter 1, “OpenGL Multipipe Overview”, SGI Xinerama enables pure X applications to run transparently over multiple pipes. To run pure X applications, simply

enable SGI Xinerama 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.

However, you can still run IRIS GL applications, but not with the `omprun` script. In this case, they will draw 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".

Running Multipipe Applications in Multipipe-Aware Mode

Multipipe applications are intentionally written to take advantage of systems having 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.

Specific applications may bypass the OpenGL Multipipe layer so that they can be fully aware of the multipipe environment. Behaving as though they are displaying on a single large pipe, other applications may continue to be run under OpenGL Multipipe.

Applications that bypass the OpenGL Multipipe layer 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, simply start the desired multipipe application with the `-aware` command-line option of the `omprun` script, as in the following example:

```
$ omprun -aware perfly
```

Note: Applications started in aware mode will be under window manager control only with `omp4Dwm`. See the subsequent section "Managing Windows for Aware Applications" for more information about window manager `omp4Dwm`.

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

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

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

Managing Windows for Aware Applications

To allow window manager support for applications started in aware mode, OpenGL Multipipe provides the window manager `omp4Dwm`, a specialized version of SGI's standard window manager (`4Dwm`). With window managers other than `omp4Dwm`, applications started in aware mode will bypass the X 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 SGI Xinerama X server is managing. It cannot manage aware windows that were not created through SGI Xinerama.

This section describes the following topics:

- "Starting omp4Dwm"
- "Exiting omp4Dwm"

- "Defining omp4Dwm as the Default Window Manager"

Starting omp4Dwm

To start omp4Dwm , perform the following steps:

1. Exit or kill any 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, 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 omp4Dwm from a .xsession file. See the subsequent section "Defining omp4Dwm as the Default Window Manager" for more information.

2. Start the specialized window manager by entering the following:

   ```
   $ start_ompwm
   ```

   The start_ompwm script starts omp4Dwm after first checking if the display server supports SGI Xinerama and other features necessary for multipipe-aware window management. If the display server is determined to be compatible, the script starts omp4Dwm with aware window management support enabled. If the display server is not compatible, the script will exit. The script can be made to start omp4Dwm in native 4Dwm mode (with aware window management disabled) as a contingency.

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

   ```
   $ start_ompwm -help
   ```

Note: Starting an application in aware mode and then starting the window manager will result in the application's windows being unmanaged. Window manager `omp4Dwm` must be started prior to running an application in aware mode in order for its windows to be managed.

Exiting `omp4Dwm`

To exit `omp4Dwm`, simply log out and log back in. Your default window manager will again be managing your display.

You may also exit `omp4Dwm` by entering the following:

```
$ tellwm quit
```

Then start your original window manager.

Defining `omp4Dwm` as the Default Window Manager

An alternate way to run the specialized `omp4Dwm` window manager is to invoke the `start_ompwm` script in your `$HOME/.xsession` file. An example `.xsession` file is available in `/usr/share/omp/examples/X11/.xsession`. For more information about `.xsession` files, see the man pages `X`(1) and `xdm`(1).

Configuring Overlapping Screens

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:

- "Performance Enhancement"
- "X Extensions"
- "OpenGL Window Size Constraints"
- "OpenGL Multipipe Does Not Support Processors Prior to MIPS R10000"

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

The major aim of OpenGL Multipipe is not performance enhancement but ease of use:

- 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—for example, the `XSGIvc` extension, which permits control of video operations on the base graphics hardware, and

`XReadDisplay`, which allows a client to read device-independent image information from the screen. Applications using these X extensions, such as `oglsnoop`, will 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.

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.

OpenGL Multipipe Does Not Support Processors Prior to MIPS R10000

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 R12000 Processor Chip Revision: 2.3
```

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/doc/user/release.html
```

This chapter documents the following problems:

- "SGI Xinerama Is Not Enabled"
- "Using omprun without SGI Xinerama"
- "Using omprun without the ompslave Render Server"
- "Problems Running IRIS GL Applications"
- "Problems Running o32 Applications"
- "Graphics Do Not Display Correctly on All Screens"
- "Mouse Behavior Offset by a Screen"
- "Problems Running glxinfo"
- "Multipipe-Aware Applications Fail to Receive Events on Screen 0"
- "Nothing Displays or the Graphic Stalls or Hangs"
- "X Applications Are Not Behaving Correctly or Fail to Start"
- "Simultaneously Running X Servers with and without SGI Xinerama Enabled"
- "Tiled Background Image"
- "Mouse Disappears in Overlap Region"
- "Problems Running Multithreaded Applications"
- "Problems with Aware Window Management"

SGI Xinerama Is Not Enabled

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.

Using `omprun` without SGI Xinerama

The OpenGL Multipipe layer needs to run on an X server with SGI Xinerama enabled. If SGI Xinerama is not enabled on the X server used for display and you invoke an application using `omprun`, the application will exit. The following example shell session shows the result of trying to run the application `atlantis` on the local display, which does not have SGI Xinerama enabled:

```
$ setenv DISPLAY :0.0
$ omprun /usr/demos/General_Demos/atlantis/atlantis
SGIomp fatal: DISPLAY does not point to a meta display
```

You must start SGI Xinerama, or you may simply run the application without the `omprun` script.

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 example shell session shows the result of trying to run the application `ideas` on a host that does not have the `ompslave` render server running:

```
$ 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
```

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
```

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
```

Graphics Do Not Display Correctly on All Screens

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

- "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"

You Did Not Use the omprun Script

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 the application is not an IRIS GL application. See the next subsection for more information.

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

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 the aware widow manager, `omp4Dwm`, 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.

Mouse Behavior Offset by a Screen

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 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 ....
```

Note that 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. Also, note that the first –`hw` parameter is that of the top, leftmost pipe and should never have a left or top neighbor.

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

Problems Running `glxinfo`

An application such as `glxinfo`, which is designed to collect information about the graphics hardware pipes individually, is inherently multipipe-aware. Thus, you need to start it in aware mode:

```
$ omprun –aware glxinfo
```

This allows an application not having a graphical user interface (GUI) to run as if SGI Xinerama were disabled.

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, it might indicate a coding problem in the application. This can occur 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 the 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 since OpenGL Multipipe relies on regular calls to the functions just cited.

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"
- "wts Does Not Display the Main Window"

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. Most extensions can 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
```

The following example indicates that `oglsnoop` is based on the `XReadDisplay` extension. This extension is not supported by SGI Xinerama.

```
# nm /usr/sbin/oglsnoop | grep -i ReadDisplay
[149]   | 268453536| 932|FUNC |GLOB |DEFAULT  |UNDEF| XReadDisplay
```

For a list of extensions supported by SGI Xinerama, see the `Xinerama`(3X11) man page.

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.

`wts` Does Not Display the Main Window

If the `wts` application does not display the main window, it probably indicates that the `Window Size` property is not properly set. When the property `Window Size` is set to `Use Default`, the main window of `wts` does not appear. Setting `Window Size` to a fixed size solves this problem.

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 background image will result in having a tile image displayed across the screens. You can overcome this problem using `4Dwm` 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.

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.

In X, a mouse belongs to one screen of the X server at a time. Therefore, it is not possible to draw the mouse multiple times (on different screens) in the overlap region. The result is that the mouse will fade away when entering an overlap region.

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:

```
$ omprun -pthread app_name
```

Problems with Aware Window Management

The following subsections detail problems with the `omp4Dwm` aware window manager and their 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"
- "Shaped Aware Windows Do Not Appear Correctly"

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

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"
```

Mouse Events Sometimes Register on the Wrong Screen

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 SGI Xinerama and are only created on one physical screen, but when `omp4Dwm` 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.

Shaped Aware Windows Do Not Appear Correctly

Aware windows that are shaped do not display correctly with `omp4Dwm`. There is currently no workaround, but the application will behave correctly if started in unaware mode.