SGI® OpenGL Multipipe™ User's Guide

007-4318-008

Version 1.4.2

CONTRIBUTORS
Written by Ken Jones and Jenn Byrnes
Illustrated by Chrystie Danzer
Edited by Susan Wilkening
Production by Glen Traefald
Engineering contributions by Bill Feth, Christophe Winkler, Claude Knaus, and Alpana Kaulgud

New Features in This Release

OpenGL Multipipe 1.4.2 contains the following changes:

- Broader application support
- Stability improvements to (beta) aware window manager

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

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

To obtain SGI documentation, go to the SGI Technical Publications Library:

```
http://techpubs.sgi.com
```

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, please tell us. 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 us 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 World Wide Web page:

  `http://techpubs.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"
- "Components of OpenGL Multipipe"
- "Supported Platforms"

What OpenGL Multipipe Provides

SGI has always been focused on high-end graphics solutions. The Onyx family of scalable visualization supercomputers allows you to associate 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 to modify the applications.

Many existing graphics applications—like Netscape or applications based on Open Inventor, for example—are constrained to run on one 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 you can directly connect OpenGL 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 application clients are not aware of the underlying hardware configuration. Rather, they only know about a single display and behave accordingly.

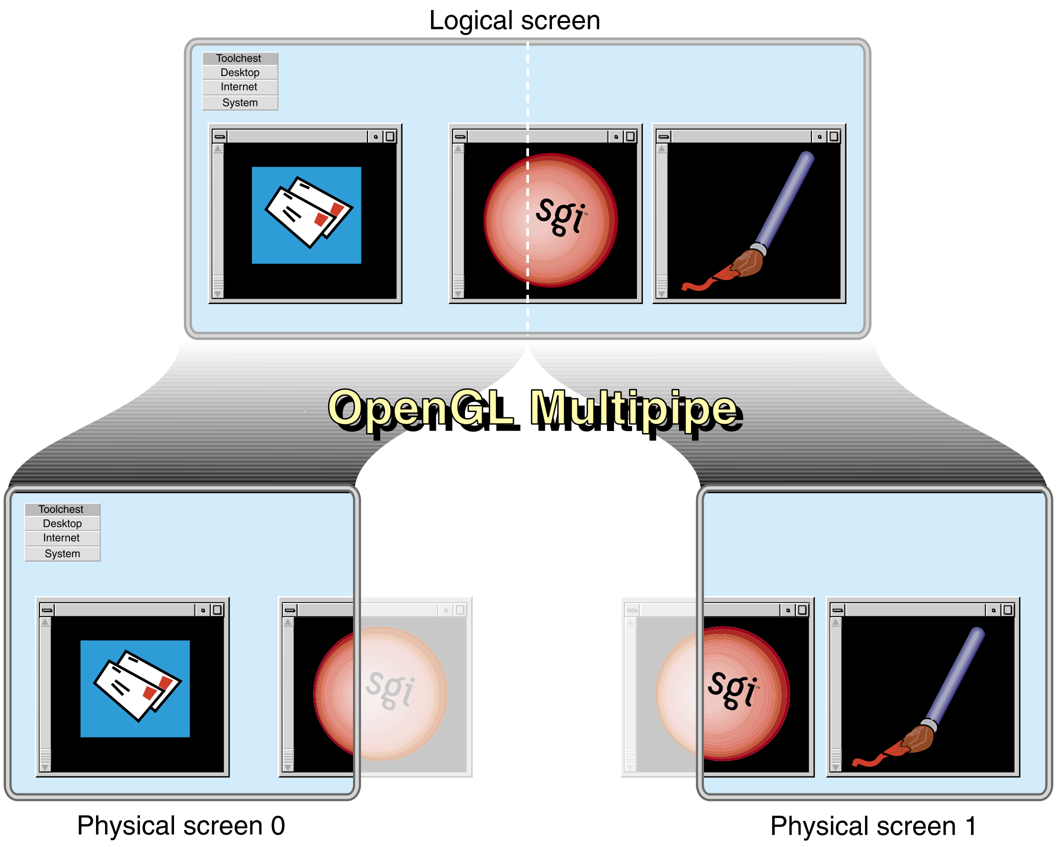

Figure 1-1 OpenGL Multipipe with Non-Overlapping Screens

In contrast to Figure 1-1, if you have screens that overlap each other (like 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.

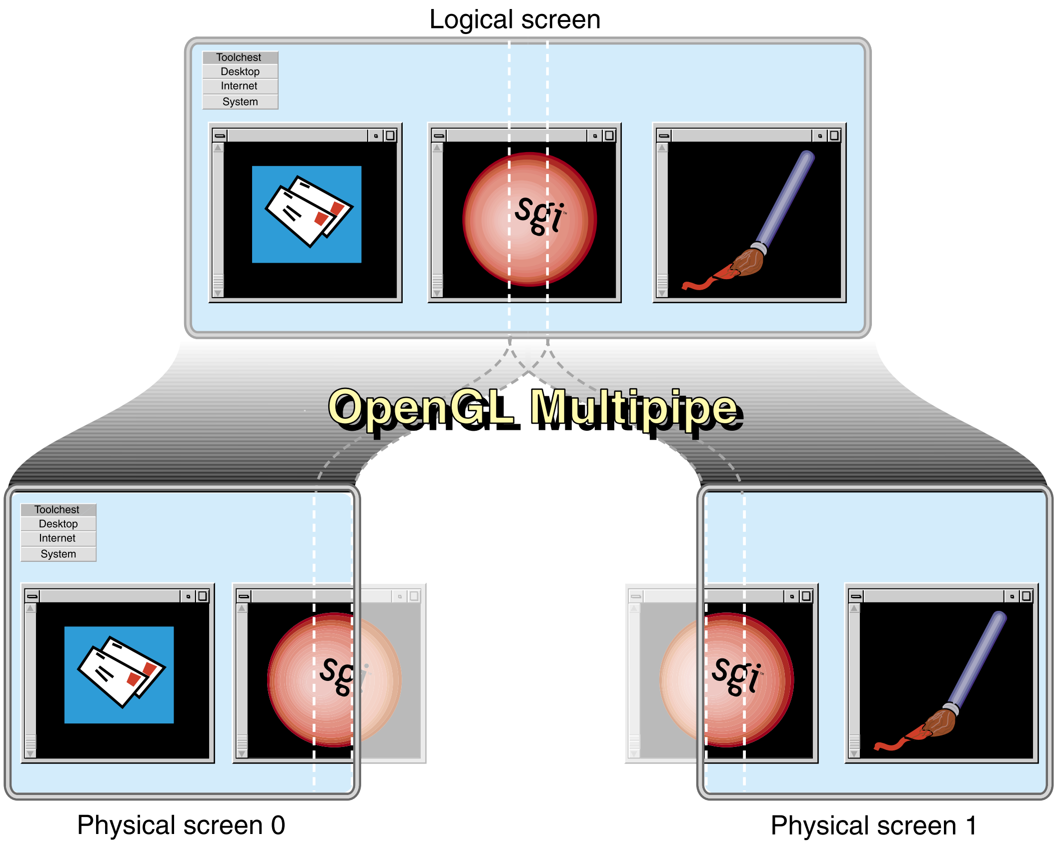

Figure 1-2 OpenGL Multipipe with Overlapping Screens

Components of OpenGL Multipipe

OpenGL Multipipe has the following two components:

- An X server with the SGI Xinerama extension. This component enables the windows to be dragged across and to overlap different pipes. This layer handles the X part of the connection between the client application and the X server.
- Transparent OpenGL. This layer traps the OpenGL calls in order to dispatch them to the underlying graphics pipes. This enables OpenGL applications to run across the graphics pipes. You must enable the SGI Xinerama extension of the X server to use Transparent OpenGL.

SGI Xinerama

SGI Xinerama is an enhanced version of the Xinerama extension of the X server that works properly on only pure X applications—that is, applications that do not use other graphics libraries to draw into their windows. When used alone, SGI Xinerama enables pure X applications to run transparently over multiple pipes. This means that window applications that are based on the X protocol and that use X extensions can be dragged from one pipe to another and even span across multiple pipes. The applications behave as if they are running on a single, large virtual pipe.

The X server hides the real screens from the clients connecting to it. It dispatches to all pipes the X requests from the clients and provides the clients with the information of only one large virtual display.

Transparent OpenGL

An OpenGL application can open a direct connection to a graphics pipe. In this case, the application is bypassing the X protocol 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. Transparent OpenGL has been designed to handle the OpenGL side of any application. It is a layer that traps the OpenGL calls and dispatches them to all the pipes handled by the X server with SGI Xinerama enabled.

Using Transparent OpenGL allows you to run any single-pipe OpenGL application on top of SGI Xinerama transparently. The windows of such an application can be freely dragged across the pipes and overlap multiple pipes. The application will not even be aware that the virtual screen is in fact composed of multiple underlying pipes. The application still correctly displays its graphics with no need of recompilation. It will no longer be constrained to one specific hardware graphics pipe.

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

Supported Platforms

OpenGL Multipipe 1.4.2 requires the following:

- IRIX 6.5.15 or later operating system
- One of the following servers:
  - Silicon Graphics Onyx
  - Silicon Graphics Onyx2
  - SGI Onyx 3000
- 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 Onyx, Onyx2, or Onyx 3000 system with MIPS R10000 or later processors
- Software:
  - IRIX 6.5.15 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 three versions:

Old-32	32-bit MIPS-2. Located in the `/usr/lib` directory. Provided for compatibility with older programs.
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 1.4.2

Subsystem	Description
`multipipe_eoe.sw.base`	Contains the actual OpenGL Multipipe libraries as well as the script for starting OpenGL applications in OpenGL Multipipe mode.
`multipipe_eoe.sw64.base`	Contains the OpenGL Multipipe 64-bit library.
`multipipe_eoe.man.relnotes`	Contains the release notes, which can be accessed through one of the following commands: `grelnotes multipipe_eoe` or `relnotes multipipe_eoe`
`multipipe_eoe.man.pages`	Contains the man pages.
`multipipe_eoe.man.doc`	Contains documentation located in the `/usr/share/multipipe/doc/user` directory.
`multipipe_eoe.sw.wm`	Contains the customized window manager for aware window support.
`multipipe_eoe.man.wm`	Contains documentation related to the customized window manager.

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

Using OpenGL Multipipe

As described in Chapter 1, OpenGL Multipipe consists of two components: SGI Xinerama and Transparent OpenGL. This chapter describes how to effectively use these components with your graphics applications. The following sections describe the pertinent tasks:

- "Enabling SGI Xinerama"
- "Using Transparent OpenGL"
- "Running OpenGL Single-Pipe Applications"
- "Running Pure X Applications"
- "Running IRIS GL Applications"
- "Running Multipipe Applications"
- "Using Remote Display Connections"
- "Configuring Overlapping Screens"
- "TGL Pipe Management"
- "Disabling SGI Xinerama"

Enabling SGI Xinerama

To enable SGI Xinerama and verify that it is enabled, do the following:

1. Enter the `chkconfig` command.

   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. Restart the X server.

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

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

   The X server has to be restarted to account for the SGI Xinerama extension.

3. Verify that SGI Xinerama is enabled.

   Enter the following two commands in an IRIX shell:

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

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

Using Transparent OpenGL

To run any single-pipe OpenGL application, OpenGL Multipipe provides Transparent OpenGL (TGL). To use this layer, the application has to be started with the `tgl` script as follows:

$ **tgl** *app_name app_args*

The following is an example:

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

The `tgl` script forces the OpenGL application to use the intermediate TGL layer instead of OpenGL. This enables the OpenGL applications to behave correctly when their windows are moved across the pipes in SGI Xinerama mode. The applications are then considered to be started in *non-aware mode*, since they are not aware of the underlying graphics hardware structure.

Technically, the `tgl` script sets the `_RLD_LIST` environment variable and related ones (`_RLDN32_LIST` and `_RLD64_LIST`) to use the `libtGL.so` library of matching format prior to using the `libGL.so` library.

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

```
$ tgl –help
```

Running OpenGL Single-Pipe Applications

OpenGL single-pipe applications are the targeted applications for OpenGL Multipipe. Simply enable SGI Xinerama and invoke the application using `tgl`. Any OpenGL application started with SGI Xinerama enabled and without the `tgl` script will not behave correctly. In that case, OpenGL drawings will only show up 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.

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 TGL. Thus, do not use the `tgl` script for these applications.

Running IRIS GL Applications

There are applications that use the 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 `tgl` script does this check and prevents OpenGL Multipipe from executing IRIS GL applications.

However, IRIS GL applications can still be run without using TGL (that is, by not using the `tgl` script). In this case, they will work correctly only on screen 0. On the other screens, 3D drawings will be corrupted.

Running Multipipe Applications

Multipipe applications are written to explicitly take advantage of systems having multiple graphics pipes. They know about the underlying graphics hardware and explicitly address the different graphics pipes. Typically, OpenGL multipipe applications are written using OpenGL Performer or OpenGL Multipipe SDK. Do not try to run such applications transparently and hide the hardware configuration of the system from them; that is, do not use TGL with them. These applications run in *aware mode* because they are aware of the different graphics pipes handled by the X server.

You can still run multipipe applications concurrently with single-pipe applications under OpenGL Multipipe. You do this by using the `–aware` command-line option of the `tgl` script, as in the following example:

```
$ tgl –aware perfly
```

The use of the `XINERAMA_AWARE` environment variable has been deprecated.

Even though you can run multipipe applications on an X server with SGI Xinerama, it introduces a conceptual contradiction. On one side, the SGI Xinerama extension allows for any window of this application to be dragged from the screen where it originally appeared (say screen 0) to any other screen. On the other side, all OpenGL commands issued by this application for that window are issued for screen 0 only. Thus, moving a given window to another pipe will corrupt the visual results of the OpenGL commands.

Multipipe applications started in aware mode can now be under window manager control with `omp4Dwm`, a specialized version of `4Dwm`, the SGI standard widow manager. Window manager `omp4Dwm` is provided with this release of OpenGL Multipipe. See "Window Manager Support for Aware Applications" in Chapter 5 for more information about `omp4Dwm`.

With other window managers, 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.

Using Remote Display Connections

Remote display connections allow you to run an application on a given machine (client machine) and to display the windows of the application on another machine (display server).

If the display server has SGI Xinerama enabled, the X applications started in non-aware mode will run correctly (that is, with the same limitations they would have if started on the display server). In order to start X applications in aware mode, the X libraries on the client machines must be synchronized (that is, the clients must run IRIX 6.5.15 or later) with the X server on the display server.

For remote display under OpenGL Multipipe, OpenGL applications can run only if OpenGL Multipipe is installed on the client machine. Install the same version of OpenGL Multipipe on both the client machine and on the display server.

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.

TGL Pipe Management

By default, TGL renders OpenGL applications using all underlying physical screens. However, it is also possible to specify the subset of the screens to use. You specify this subset with the `–pipes` command-line option of the `tgl` script, as in the following example:

```
$ tgl –pipes pipe1,pipe2,...,pipen ivview
```

The use of the `TGL_PIPES` environment variable has been deprecated.

This feature allows you to minimize the amount of context swapping on the pipes by constraining the applications to different pipes. This also implies that an application will display a random image on every pipe that is not part of the given subset.

In the following example, the application Ideas in Motion runs on the physical screens mapped to pipes 0 and 1 while the Open Inventor viewer `ivview` uses pipe 2 only:

```
$ tgl –pipes 0,1 /usr/demo/GLUT/ideas
$ tgl –pipes 2 ivview Banana.iv
```

Disabling SGI Xinerama

In order to disable the SGI Xinerama extension of the X server, simply set the `chkconfig xinerama` flag to `off` and restart the X server. To do so, issue the following commands as `root`:

```
# chkconfig xinerama off
# (/usr/gfx/stopgfx; /usr/gfx/startgfx) &
```

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 Enhancements"
- "Nontransparent X Extensions"
- "OpenGL Window Size Constraints"

For release-dependent limitations, refer to the OpenGL Multipipe release notes.

Performance Enhancements

The major thrust of OpenGL Multipipe is not performance enhancement, as described in the following items:

- OpenGL Multipipe does not replace performance-aimed multipipe applications—such as those based on OpenGL Performer or OpenGL Multipipe SDK—or any other custom 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.

Nontransparent X Extensions

Some X extensions are inherently nontransparent. For example, `Xvc` permits control of video operations on the base graphics hardware and `XReadDisplay` allows a client to read device-independent image information from the screen. Applications using these X extensions, like `oglsnoop`, will not function properly. The behavior of these applications started in non-aware mode is undefined, though they will generally behave correctly on screen 0.

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.

Beta Features

This chapter describes the following beta features:

- "Curved Screen Support"
- "Window Manager Support for Aware Applications"

Curved Screen Support

The goal of OpenGL Multipipe is to hide the underlying graphics configuration to the client applications and to pretend to the applications that there is only a single *virtual screen*. This allows you to run single-pipe applications on a Reality Center with a planar display layout (an SGI Reality Center wall display).

Support for curved screens in Reality Center environments introduces a conceptual contradiction: single-pipe applications, which are developed to support only a single *planar* screen, are displayed in an environment where the projection of the 3D scene is different for each underlying screen, as shown in Figure 5-1. There is no exact solution to handle these cases, where the 2D X Window System part is simply wrapped on the curved screen while the 3D content is projected according to the screen on which it will appear.

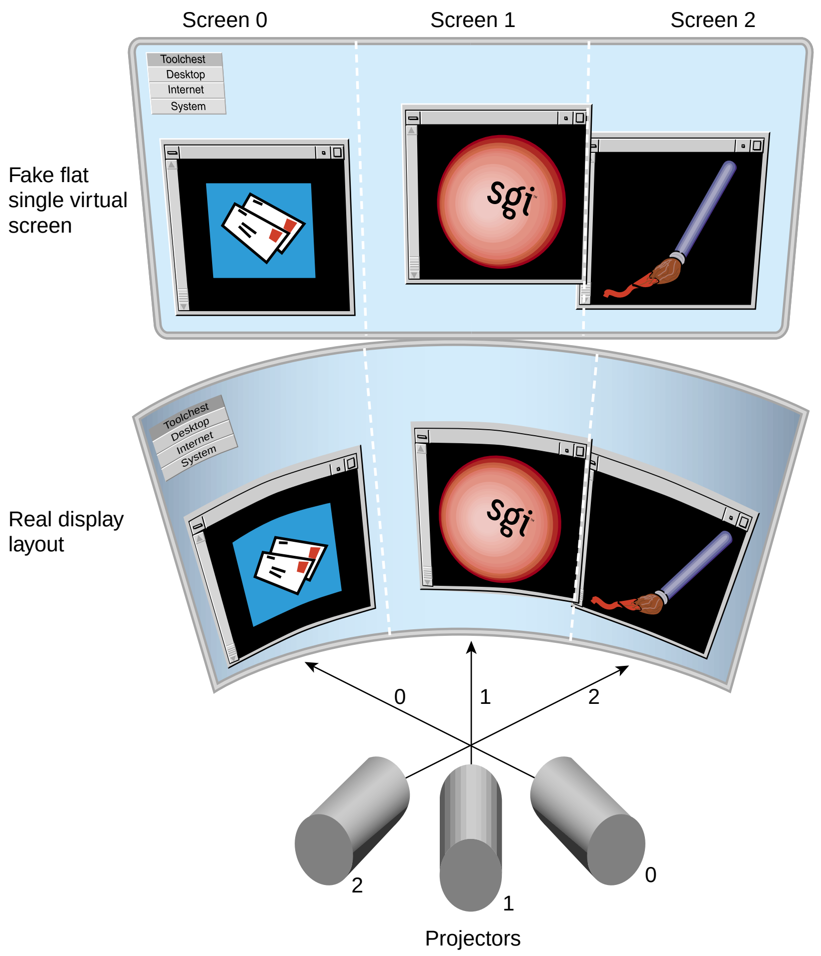

Figure 5-1 Fake Virtual Screen Projected on a Curved Screen Display Environment

A configuration file mechanism has been added in OpenGL Multipipe. It allows you to specify the display geometry in order to adapt the behavior of Transparent OpenGL to the Reality Center facility. Some hints can also be given to customize OpenGL Multipipe. The following subsection, “Configuration File”, describes how you specify the display geometry and the hints.

Configuration File

You need to configure OpenGL Multipipe to account for the display geometry of a particular Reality Center facility. This is done by describing the display layout in a configuration file.

OpenGL Multipipe searches for the configuration file in the following precedence order:

- User-specified file

  Specified by the `-config` argument of the `tgl` script. The following is an example:

  ```
  $ tgl -config ~/cf/rc1.cf ivview model.iv
  ```
- `.tglrc` in the current directory
- `.tglrc` in the home directory

The following are configuration-related arguments of the `tgl` script:

Argument	Description
`-config`	Allows you to specify an alternate configuration file.
`-config_print`	Outputs the parsed configuration.
`-config_debug`	Provides some hints about possible configuration file parsing errors.

Structure of the Configuration File

A configuation file consists of the description of one or more graphics pipes and optionally customization hints for the behavior of OpenGL Multipipe. Example 5-1 illustrates the structure of the file.

Example 5-1 Configuration File Structure

```
pipe {
    name pipe-no
    projection-desc
}
pipe {
    name pipe-no
    projection-desc
}
```

```
hints {
    hint-1
    hint-2
    ...
    hint-n
}
```

Pipe Description

You describe a graphics pipe or, more specifically, the geometry of the screen on which the pipe displays using the following construct in the configuration file:

```
pipe {
    name pipe-no
    projection-desc
}
```

The keyword `name` has the argument *pipe-no*, which corresponds to the pipe number, as managed by the X server. In the following example, the pipe `:0.1` is specified:

```
pipe {
    name 1
    ...
}
```

The argument *projection-desc* is the description of the pipe layout. Two possibilities exist, the *projection* description and the *wall* description.

The projection description uses polar (cylindrical or spherical) coordinates to describe the layout. You use the following construct in the configuration file:

```
projection {
    origin (x, y, z)
    distance d
    fov (hfov,vfov)
    hpr (head, pitch, roll)
}
```

The following are arguments you must supply:

Argument	**Description**
(*x*, *y*, *z*)	The origin of the projection—that is, the apex of the projection pyramid. This 3D coordinate value is usually `(0,0,0)` and corresponds to the center of the projection system.
d	The distance from the origin to the screen.
(*hfov*,*vfov*)	The field of view defining the horizontal and vertical fields of view, given in degrees.
(*head, pitch, roll)*	The *head*, *pitch*, and *roll*, given in degrees. These are used to define the horizontal and vertical offsets of the projection (the roll is usually not used).

The following is an example of a complete projection description of pipe `1` (the # symbol precedes comments):

```
pipe {
   name 1

   # projection description
   projection {
      origin (0, 0, 0)
      distance 3
      fov (54, 47)
      hpr (48, 0, 0)
   }
}
```

The wall description specifies the screen layout as a rectangle in 3D. The rectangle is defined by three vertices and the projection center is the origin of the coordinate system. The following is an example of a complete wall description of pipe `2`:

```
pipe {
   name 2

   # wall description
   wall {
      bottom_left (-2, -1, -3)
      bottom_right (2, -1, -3)
      top_left (-2, 2, -3)
   }
}
```

Both description approaches are equivalent. Any projection description can be expressed in an equivalent wall description and vice versa. Trigonometric rules in 3D allow you to compute one description from the other.

Hints

You can specify some optional hints to customize the behavior of OpenGL Multipipe. The possible hints are the following:

Hint	Description
`mode`	Defines the projection strategy. So far, the only defined modes are `standard`, which does not account for any projection correction, and `immersive`, which discards the projection definition given by the application and replaces it with a description extracted from the display layout. The default value is `standard`.
`ref_pipe`	Defines the reference pipe, which is needed in various cases. The reference pipe is used to return the current projection matrix because this matrix ends up being different for each pipe in a curved screen environment.
`scale_near`	Defines the scaling factor to compute the effective near plane value for every pipe. The effective near plane value is the one given by the application scaled by `scale_near`. The default value is `1`.
`scale_far`	Allows you to adapt the position of the far clipping plane to the application's need, as shown in Figure 5-2. A value of `0` or `no` keeps the same far distance for all the screens. For other values, the far clipping plane is pushed back at the intersection between the axis of the projection pyramid and the clipping plane of the ref pipe (that is, moved from `A` to `A'`). The actual scaling is performed on the resulting clipping plane. For example, a value of `2` will move the far plane at twice the distance of `A'`. The default value is `no`.
`track_modelview`	Indicates the number of stack levels for which the `GL_MODELVIEW` matrix is cached. A value of `0` means no caching at all. The cached matrix is the matrix as known by the application. The actual matrices on each pipe can be different, since the modelview matrix is adapted to the display layout. The default value is `0`.

Since matrix caching has a performance impact, you should avoid its use. Some applications use the matrix `push` and `pop` mechanism in order to reset the matrix to an initial default value instead of calling **glLoadIdentity()**, for example. In that case, the matrix caching allows OpenGL Multipipe to initialize the modelview matrices correctly on each pipe.

Note: Except for the `mode` hint, the hints are enabled only in `immersive` mode.

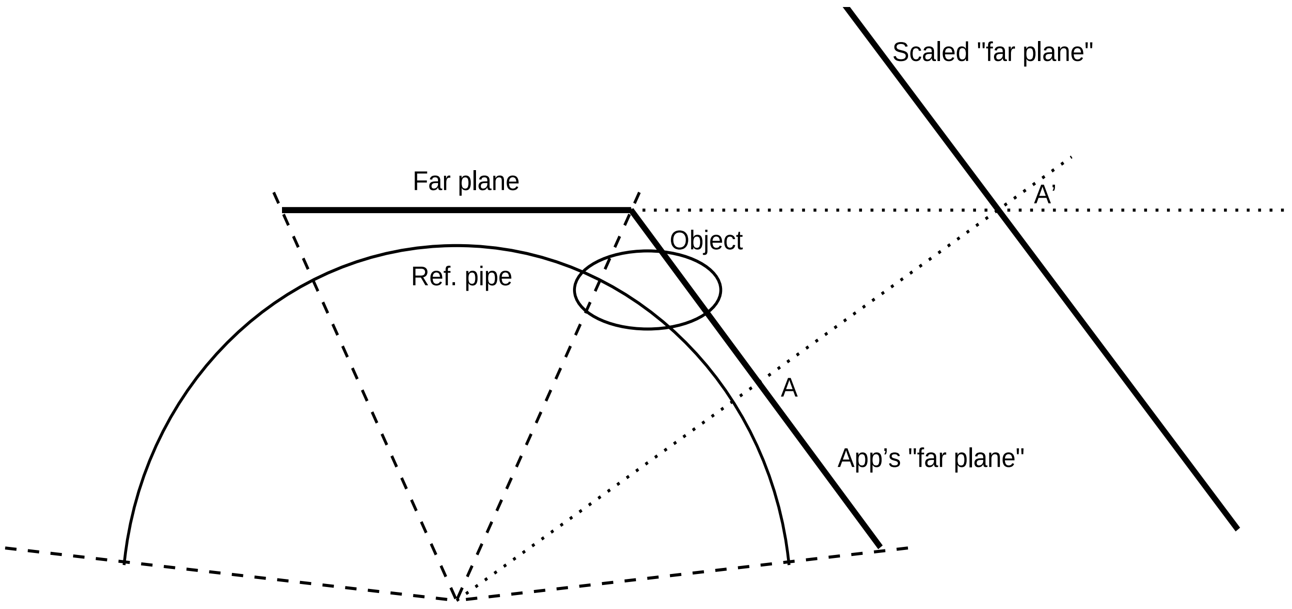

Figure 5-2 Far Plane Scaling Rule (Top View)

The following lines show an example of a hints specification.

```
hints {
   mode immersive
   ref_pipe 1
   scale_near .8
   scale_far 2
   track_modelview 2
}
```

X versus OpenGL Inconsistency

Transparent support for a curved screen comes with an inherent inconsistency between the 2D Window system and the 3D OpenGL graphics.

The 2D X position is *wrapped* on the display layout, as shown in Figure 5-3. The figure represents a curved screen Reality Center facility. The screen shown as a dark arc is driven by three projectors. Any application running transparently in this environment knows only about the single virtual screen, represented by the dark dotted line. Any drawing generated by the X Window System will be displayed on the curved screen (that is, arc `[LA]` for the screen `0`) instead of the virtual screen (that is, segment `[L’A]` for screen `0`). The relation between a virtual 2D pixel `M’` on the virtual screen and its real projection `M` on the curved screen can be approximated by a rotation around the screen edge `A`. This means that if the user clicks on `M`, the application thinks that the click occurred on `M’`.

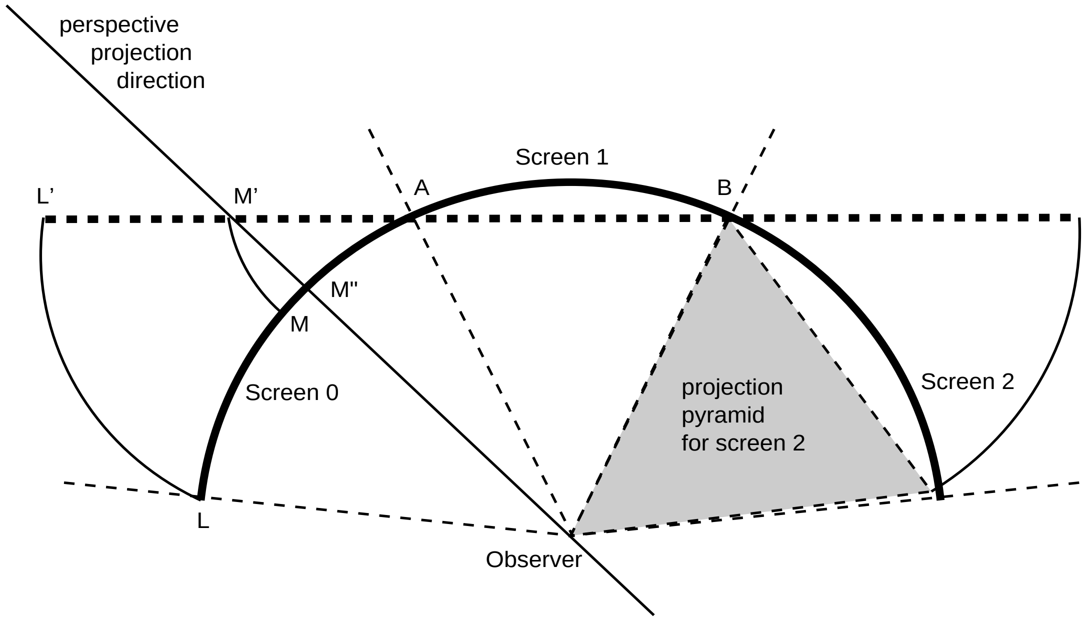

Figure 5-3 2D X Versus 3D OpenGL Inconsistency (Top View)

In order to give the user the feeling of being *immersed* in the 3D scene, the objects must be projected differently according to the screen on which they appear. For example, a 3D object that is meant to be drawn at `M'` by the application will be displayed on `M"` on the real screen in order to have a correct immersion perception (`M"` is the projection of `M'`, according to the observer, on the real screen `0`).

This inconsistency between 2D wrapping and 3D projection can be observed when the user wants to pick a 3D object. The user clicks on the object displayed at `M`, the application is told the click occurred on `M'` and selects the 3D object, which is displayed at `M'`. This object is then drawn at `M"` in order to have the correct perception of immersion.

Window Manager Support for Aware Applications

To allow window manager support for applications started in aware mode, `omp4Dwm`, a specialized version of SGI's standard window manager (`4Dwm`), has been included in this release.

This release of OpenGL Multipipe includes a beta version of `omp4Dwm`, which means that it is still evolving. For more information about bugs, see the Troubleshooting section in the Beta Features chapter of the release notes.

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 “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 typing 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/multipipe/X11/etc/.xsession`. Refer to the man pages for `X`(1) and `xdm`(1) for more information about `.xsession` files.

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. Enter the following command or refer to the `relnotes` (1) man page:

```
$ grelnotes multipipe_eoe
```

This chapter documents the following problems:

- "SGI Xinerama Is Not Enabled"
- "Using Transparent OpenGL without SGI Xinerama"
- "Transparent OpenGL Does Not Support IRIS GL Applications"
- "Transparent OpenGL Does Not Support Processors Prior to MIPS R10000"
- "Graphics Display Correctly on Some Screens Only"
- "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 Multiple X Servers with and without SGI Xinerama Enabled"
- "Application Graphics Not Synchronized across Screens"
- "Tiled Background Image"
- "Mouse Disappears in Overlap Region"
- "Multiple OpenGL Applications Run Slowly"
- "Problems Running Multithreaded Applications"

- "Problems with Curved Screens"
- "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 Transparent OpenGL without SGI Xinerama

The Transparent OpenGL 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 `tgl`, the application will exit. The following example shell session shows the result of trying to run the application `atlantis` on server `blaster`, which does not have SGI Xinerama enabled:

```
$ setenv DISPLAY blaster:0.0
$ tgl /usr/demos/General_Demos/atlantis/atlantis
TGL fatal error: display ‘blaster:0.0’ doesn’t support SGI-XINERAMA
extension
```

In this case, simply run the application without the `tgl` script.

Transparent OpenGL Does Not Support IRIS GL Applications

Transparent OpenGL does not support IRIS GL applications. In some cases (when the application started with the `tgl` script is an executable and not a script), `tgl` 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:

```
$ tgl showcase
tgl warning: showcase is an IRIS GL application
TGL Library does not support IRIS GL applications
```

Transparent OpenGL Does Not Support Processors Prior to MIPS R10000

Transparent OpenGL 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
```

Graphics Display Correctly on Some Screens Only

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

You Failed to Use the `tgl` Script

If a graphics window displays correctly on one screen only (usually screen 0), ensure that you start the application with the `tgl` script. If the same behavior persists when you invoke the application using the `tgl` script, ensure the application is not an IRIS GL application. See the next subsection for more information.

A User-Defined Script Invokes an IRIS GL Application

The `tgl` script cannot detect IRIS GL 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
$ tgl ./atlantis
tgl warning: ./atlantis is an IRIS GL application
TGL Library does not support IRIS GL applications
$ tgl ./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 application. The following session shows you how to generate the desired test string:

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

TGL Pipe Management Is Used

If you started an application with the `tgl` script using the TGL Pipe Management feature (that is, by running `tgl –pipes` *p1*,*p2*... or by specifying the deprecated `TGL_PIPES` environment variable), it is possible that a window is opened on a screen that does not belong to the subset you specified. To see the window rendered correctly, move the application's window to a screen you specified.

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. 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 like `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:

```
$ tgl –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, using the specialized window manager included in this release of OpenGL Multipipe will also fix the focus problem.

Nothing Displays or the Graphic Stalls or Hangs

If you start an OpenGL application with `tgl` 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 are not calling **glFlush()**, **glFinish()**, or **glXSwapBuffers()** at the end of each frame. This causes Transparent OpenGL 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. There is no real workaround since Transparent OpenGL is conformant with the OpenGL specification.

X Applications Are Not Behaving Correctly or Fail to Start

If X applications are not behaving correctly or fail to start, there are several cases to consider and the following subsections describe them.

X Application Uses Unsupported X Extension

Verify that the application is not using unsupported X extensions. There is unfortunately no way to have the exact list of extensions that are used by an application. The following examples using the `nm` command give only a hint 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 with X client libraries (dynamically at run time or statically at compile time) 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.

Only applications that call `XineramaQueryVersion` will be able to reliably detect and report server and client cross-version incompatibilities.

The workaround for this protocol incompatibility 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.

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 Multiple 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.

Application Graphics Not Synchronized across Screens

OpenGL applications may encounter synchronization problems if they are displayed across multiple screens. This occurs because buffer swapping of windows on different pipes must be synchronized explicitly by using the swap barrier. You must wire the `genlock` and Swap Ready cables. For more information, refer to the *POWER Onyx and Onyx Rackmount Owner's Guide* and to the `genlock`(7) man page. In order to force the swap barrier for an OpenGL application, you must use the `–sync` command-line option of the `tgl` script, as shown in the following example:

```
$ tgl –sync /usr/demos/General_Demos/atlantis/atlantis
```

The use of the `TGL_SWAP_BARRIERS` environment variable has been deprecated.

InfiniteReality supports only one Swap Ready line. If you start several applications simultaneously with this environment variable set, the following warning results:

```
TGL warning: swap barrier is already used or not available
Cannot synchronize windows across multiple screens
```

The applications will not benefit from this synchronization mechanism.

Another way to decrease the effect of asynchronized swapping is to place the window so that every pipe has the same graphics load. The pipes will then execute the buffer swapping more or less at the same time.

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 (refer to the `4Dwm`(1X) man page):

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

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

- Create the background image in the `xmp` 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.

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.

Multiple OpenGL Applications Run Slowly

Depending on the OpenGL applications, running several OpenGL applications simultaneously can slow down rendering speed significantly. A workaround is to try to limit the extensions of the application windows to separate subsets of the physical screens. See “TGL Pipe Management” in Chapter 3 for details.

Problems Running Multithreaded Applications

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

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

```
$ tgl -pthread app_name
```

Problems with Curved Screens

For a description of problems related to curved screens, see the Troubleshooting section in the Beta Features chapter of the release notes.

Problems with Aware Window Management

For a description of problems related to aware window management, see the Troubleshooting section in the Beta Features chapter of the release notes.