''%'"

DRI User Guide

Precision Insight, Inc.

18 May 2000

1. Preamble

1.1 Copyright

Copyright © 2000 by Precision Insight, Inc., Cedar Park, Texas. All Rights Reserved.

Permission is granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies.

1.2 Trademarks

OpenGL is a registered trademark and SGI is a trademark of Silicon Graphics, Inc. Unix is a registered trademark of The Open Group. The 'X' device and X Window System are trademarks of The Open Group. XFree86 is a trademark of The XFree86 Project. Linux is a registered trademark of Linus Torvalds. Intel is a registered trademark of Intel Corporation. 3Dlabs, GLINT, and Oxygen are either registered trademarks or trademarks of 3Dlabs Inc. Ltd. 3dfx, Voodoo3, Voodoo4, and Voodoo5 are registered trademarks of 3dfx Interactive, Incorporated. Matrox is a registered trademark of Matrox Electronic Systems Ltd. ATI Rage is a registered trademark of ATI Technologies, Inc. All other trademarks mentioned are the property of their respective owners.

2. Introduction

With XFree86 4.0 and Precision Insight's Direct Rendering Interface (DRI), hardware accelerated 3D graphics can be considered a standard feature on Linux workstations. Support for other operating systems, such as FreeBSD, is underway.

This document describes how to use the DRI system and troubleshoot problems which may occur. Readers should have a basic understanding of Linux, X and OpenGL. See the resources section at the end for more documentation and software downloads.

This document does not cover compilation or installation of XFree86 4.0. It is assumed that you've already installed a Linux distribution which includes XFree86 4.0 or that you're an experienced Linux developer who has compiled the DRI for himself. DRI download, compilation and installation instructions can be found at http://dri.sourceforge.net/DRIcompile.html

3. Supported Hardware

3D acceleration is currently only available for systems with Intel-compatible CPUs. Support for Alpha, and perhaps other CPUs, should be available in the future.

XFree86 4.0 (or later versions) includes 3D acceleration for the following graphics hardware:

- 3dfx:
  - Voodoo3 3500 TV
  - Voodoo3 3000 AGP
  - Voodoo3 3000 PCI
  - Voodoo3 2000 AGP
  - Voodoo3 2000 PCI
  - Voodoo Banshee
  - Velocity 100/200

  There are many configurations of 3dfx cards on the market. Not all have been tested.
- Matrox:
  - Matrox G200
  - Matrox G400
- Intel i810
  - i810
  - i810-dc100
  - i810e
- ATI Rage 128
  - Rage Fury AGP
  - Rage Magnum AGP
  - XPERT 2000 AGP
  - XPERT 128 AGP
  - XPERT 99 AGP
  - All-in-Wonder 128 AGP

  The PCI versions of these cards also have minimal support. Note that there are also Rage 128 Pro based boards on the market, and these are not yet supported.
- 3Dlabs Oxygen GMX 2000 (MX/Gamma based)

Support for the following hardware is underway:

- 3dfx Voodoo4 and Voodoo5 series

4. Prerequisite Software

- XFree86 4.0

- For the 3dfx Voodoo3 driver, Linux kernel 2.2.x (later kernels will be supported in the near future, and may be required for some chipsets)
- For the Matrox G200/G400, Linux kernel 2.3.51, with AGP support
- For the Intel i810, Linux kernel 2.3.99-pre6, with AGP support

Mesa 3.3 (beta) is included with XFree86 4.0; there is no need to download the stand-alone Mesa distribution.

5. X Server Start-up

This section describes the steps needed to start the X server with 3D acceleration support.

5.1 Kernel module

XFree86 4.0.1 added automatic kernel module loading to the X server. On Linux, the X server uses modprobe to load kernel modules. The DRM kernel modules should be in /lib/modules/KERNEL-VERSION/misc/ for automatic loading to work.

Optionally, DRM kernel modules can be loaded manually with insmod prior to starting the X server.

You can verify that the kernel module was installed with lsmod, checking the X server startup log, and checking that /proc/dri/0 exists.

5.2 XF86Config file

First, the XF86Config file must load the GLX and DRI modules:

```
Section "Module"
...
# This loads the GLX module
    Load       "glx"
# This loads the DRI module
    Load       "dri"
EndSection
```

Next, the DRI section can be used to restrict access to direct rendering.

If you want all of the users on your system to be able to use direct-rendering, then use a simple DRI section:

```
Section "DRI"
     Mode 0666
EndSection
```

This section will allow any user with a current connection to the X server to use direct rendering.

If you want to restrict the use of direct-rendering to a certain group of users, then create a group for those users by editing the `/etc/group` file on your system. For example, you may want to create a group called `xf86dri` and place two users (e.g., `fred` and `jane`) in that group. To do that, you might add the following line to `/etc/group`:

```
xf86dri:x:8000:fred,jane
```

You have to be careful that the group id (8000 in this example) is unique.

Then you would use the following DRI section:

```
Section "DRI"
     Group "xf86dri"
     Mode 0660
EndSection
```

This would limit access to direct-rendering to those users in the `xf86dri` group (`fred` and `jane` in this example). When other users tried to use direct rendering, they would fall back to unaccelerated indirect rendering.

[Note that there is a known bug in XFree86 4.0 that prevents some changes to the DRI section from taking effect. Until this bug is fixed, if you change the DRI section, please also remove the `/dev/dri` directory with the `rm -rf /dev/dri` command.]

Next, the Device section of the XF86Config file must describe your particular hardware.

For example, here's the Device section for a 3dfx Voodoo3 card:

```
Section "Device"
    Identifier  "Voodoo3"
    VendorName  "3dfx"
    Driver      "tdfx"
EndSection
```

Here's the Device section for a Matrox G400 card:

```
Section "Device"
    Identifier  "G400"
    VendorName  "Matrox"
    Driver      "mga"
    VideoRam    32768
EndSection
```

Here's the Device section for an ATI Rage 128 card:

```
Section "Device"
    Identifier  "Rage128"
    VendorName  "ATI"
    Driver      "r128"
EndSection
```

Here's the Device section for an Intel i810 motherboard:

```
Section "Device"
    Identifier  "i810"
    VendorName  "Intel"
    Driver      "i810"
    VideoRam    10000
EndSection
```

Finally, the Screen section of the XF86Config file may have to be specially configured as well. For example, Voodoo3 hardware acceleration is only available in 16bpp mode.

```
Section "Screen"
    Identifier  "Screen 1"
    Device      "Voodoo3"
    Monitor     "High Res Monitor"
    DefaultDepth 16
    Subsection "Display"
       Depth       16
       Modes       "1280x1024" "1024x768" "800x600" "640x480"
       ViewPort    0 0
    EndSubsection
EndSection
```

Replace *Voodoo3* with *G400* for Matrox G400.

Replace *Voodoo3* with *Rage128* for ATI Rage 128.

If there are errors in the XF86Config file, the X server will log errors to the file /var/log/XFree86.0.log

5.3 Memory usage

Using the 3D features of a graphics card requires more memory than when it's just used as a 2D device. Double buffering, depth buffering, stencil buffers, textures, etc. all require extra graphics memory. These features may require four times the memory used for a simple 2D display.

If your graphics card doesn't have a lot of memory (less than 16MB, for example), you may have to reduce your screen size and/or color depth in order to use 3D features.

The documentation included with your card should have information about maximum screen size when using 3D.

6. Using 3D Acceleration

This section describes how to link your application with libGL.so and verify that you are in fact using 3D acceleration.

6.1 libGL.so

Your OpenGL program must link with the libGL.so.1.2 library provided by XFree86 4.0. The libGL.so.1.2 library contains a GLX protocol encoder for indirect/remote rendering and DRI code for accessing hardware drivers. In particular, be sure you're not using libGL.so from another source such as Mesa or the Utah GLX project.

Unless it was built in a special way, the libGL.so library does not contain any 3D hardware driver code. Instead, libGL.so dynamically loads the appropriate 3D driver during initialization.

Most simple OpenGL programs also use the GLUT and GLU libraries. A source for these libraries is listed in the Resources section below.

6.2 Compiling and linking an OpenGL program

A simple GLUT/OpenGL program may be compiled and linked as follows:

```
gcc program.c -I/usr/local/include -L/usr/local/lib -L/usr/X11R6/lib -lglut -lGLU -lGL -o program
```

The `-I` option is used to specify where the GL/glut.h (and possibly the GL/gl.h and GL/glu.h) header file may be found.

The `-L` options specify where the libglut.so, libGLU.so and X libraries are located.

The `-lglut -lGLU -lGL` arguments specify that the application should link with the GLUT, GLU and GL libraries.

6.3 Running your OpenGL program

Simply typing ./program in your shell should execute the program.

If you get an error message such as

```
gears: error in loading shared libraries: libGL.so.1: cannot
open shared object file: No such file or directory
```

if means that the libGL.so.1 file is not the right location. Proceed to the trouble shooting section.

6.4 libOSMesa.so

OSMesa (Off-Screen Mesa) is an interface and driver for rendering 3D images into a user-allocated block of memory rather than an on-screen window.

libOSMesa.so implements the OSMesa interface and must be linked with your application if you want to use the OSMesa functions. You must also link with libGL.so. For example:

```
gcc osdemo.c -L/usr/X11R6/lib -lOSMesa -lGLU -lGL -o osdemo
```

In stand-alone Mesa this interface was compiled into the monolithic libGL.so (formerly libMesaGL.so) library. In XFree86 4.0.1 and later this interface is implemented in a separate library.

6.5 glxinfo

glxinfo is a useful program for checking which version of libGL you're using as well as which DRI-based driver. Simply type `glxinfo` and examine the OpenGL vendor, renderer, and version lines. Among the output you should see something like this:

```
OpenGL vendor string: Precision Insight, Inc.
OpenGL renderer string: Mesa DRI Voodoo3 20000224
OpenGL version string: 1.2 Mesa 3.3 beta
```

or this:

```
OpenGL vendor string: Precision Insight, Inc.
OpenGL renderer string: Mesa GLX Indirect
OpenGL version string: 1.2 Mesa 3.3 beta
```

The first example indicates that the 3dfx driver is using Voodoo3 hardware. The second example indicates that no hardware driver was found and indirect, unaccelerated rendering is being used.

If you see that indirect rendering is being used when direct rendering was expected, proceed to the troubleshooting section.

`glxinfo` also lists all of the GLX-enhanced visuals available. Here you can determine which visuals may have depth buffers, stencil buffers, accumulation buffers, etc.

6.6 Environment Variables

The libGL.so library recognizes three environment variables. Normally, none of them need to be defined. If you're using the csh or tcsh shells, type `setenv VARNAME value` to set the variable. Otherwise, if you're using sh or bash, type `export VARNAME=value`.

1. `LIBGL_DEBUG`, if defined will cause libGL.so to print error and diagnostic messages. This can help to solve problems. Setting `LIBGL_DEBUG` to `verbose` may provide additional information.

2. LIBGL_ALWAYS_INDIRECT, if defined this will force libGL.so to always use indirect rendering instead of hardware acceleration. This can be useful to isolate rendering errors.
3. LIBGL_DRIVERS_PATH can be used to override the default directories which are searched for 3D drivers. The value is one or more paths separated by colons. In a typical XFree86 installation, the 3D drivers should be in /usr/X11R6/lib/modules/dri/ and LIBGL_DRIVERS_PATH need not be defined. Note that this feature is disabled for set-uid programs. This variable replaces the LIBGL_DRIVERS_DIR env var used in XFree86 4.0.

Mesa-based drivers (this includes most of the drivers listed above) also observe many of the existing Mesa environment variables. These include the MESA_DEBUG and MESA_INFO variables.

7. General Trouble Shooting

This section contains information to help you diagnose general problems. See below for additional information for specific hardware.

7.1 Starting the X server

1. Before you start the X server, verify the appropriate 3D kernel module is installed. Type lsmod and look for the appropriate kernel module. For 3dfx hardware you should see tdfx, for example.
2. Verify you're running XFree86 4.0 and not an older version. If you run xdpyinfo and look for the following line near the top:

```
vendor release number:    4000
```

3. Verify that your XF86Config file (usually found at /etc/X11/XF86Config) loads the glx and dri modules and has a DRI section.

   See the Software Resources section below for sample XF86Config files.
4. Examine the messages printed during X server startup and check that the DRM module loaded. Using the Voodoo3 as an example:

```
(==) TDFX(0): Write-combining range (0xf0000000,0x2000000)
(II) TDFX(0): Textures Memory 7.93 MB
(0): [drm] created "tdfx" driver at busid "PCI:1:0:0"
(0): [drm] added 4096 byte SAREA at 0xc65dd000
(0): [drm] mapped SAREA 0xc65dd000 to 0x40013000
(0): [drm] framebuffer handle = 0xf0000000
(0): [drm] added 1 reserved context for kernel
(II) TDFX(0): [drm] Registers = 0xfc000000
(II) TDFX(0): visual configs initialized
(II) TDFX(0): Using XFree86 Acceleration Architecture (XAA)
        Screen to screen bit blits
        Solid filled rectangles
        8x8 mono pattern filled rectangles
        Indirect CPU to Screen color expansion
        Solid Lines
        Dashed Lines
        Offscreen Pixmaps
        Driver provided NonTEGlyphRenderer replacement
        Setting up tile and stipple cache:
                10 128x128 slots
(==) TDFX(0): Backing store disabled
(==) TDFX(0): Silken mouse enabled
(0): X context handle = 0x00000001
(0): [drm] installed DRM signal handler
(0): [DRI] installation complete
(II) TDFX(0): direct rendering enabled
```

5. After the X server has started, verify that the required X server extensions are loaded. Run `xdpyinfo` and look for the following entries in the extensions list:

```
GLX
SGI-GLX
XFree86-DRI
```

7.2 Linking, running and verifying 3D acceleration

After you've verified that the X server and DRI have started correctly it's time to verify that the GL library and hardware drivers are working correctly.

1. Verify that you're using the correct libGL.so library with `ldd`. The /usr/lib and /usr/X11R6/lib directories are expected locations for libGL.so.

   Example:

```
% ldd /usr/local/bin/glxinfo
        libglut.so.3 => /usr/local/lib/libglut.so.3 (0x40019000)
        libGLU.so.1 => /usr/local/lib/libGLU.so.1 (0x40051000)
        libGL.so.1 => /usr/lib/libGL.so.1 (0x40076000)
        libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x402ee000)
        libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x40301000)
        libm.so.6 => /lib/libm.so.6 (0x40309000)
        libc.so.6 => /lib/libc.so.6 (0x40325000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40419000)
        libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x404bd000)
        libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x40509000)
        libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x40512000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40529000)
        libvga.so.1 => /usr/lib/libvga.so.1 (0x40537000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x4057d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

2. You may also double check that libGL.so is in fact DRI-capable. Run `strings libGL.so.1.2 | grep DRI` and look for symbols prefixed with "XF86DRI", such as "XF86DRIQueryExtension".
3. To be safe one should run `ldconfig` after installing libGL.so to be sure the runtime loader will find the proper library.
4. Verify that the appropriate 3D driver is in /usr/X11R6/lib/modules/dri/ For example, the 3dfx driver will be named `tdfx_dri.so`.
5. Set the `LIBGL_DEBUG` environment variable. This will cause libGL.so to print an error message if it fails to load a DRI driver. Any error message printed should be self-explanatory.
6. Run `glxinfo`. Note the line labeled "OpenGL renderer string". It should have a value which starts with "Mesa DRI" followed by the name of your hardware.
7. Older Linux OpenGL applications may have been linked against Mesa's GL library and will not automatically use libGL.so. In some cases, making symbolic links from the Mesa GL library to libGL.so.1 will solve the problem:

   ```
   ln -s libGL.so.1 libMesaGL.so.3
   ```

   In other cases, the application will have to be relinked against the new XFree86 libGL.so.

   It is reported that part of the problem is that running `ldconfig` will silently rewrite symbolic links based on the SONAME field in libraries.

If you're still having trouble, look in the next section for information specific to your graphics card.

8. Hardware-Specific Information and Troubleshooting

This section presents hardware-specific information for normal use and troubleshooting.

8.1 3dfx Voodoo3

8.1.1 Configuration

Your XF86Config file's device section must specify the `tdfx` device:

```
Section "Device"
    Identifier  "Voodoo3"
    VendorName  "3dfx"
    Driver      "tdfx"
EndSection
```

The Screen section should then reference the Voodoo3 device:

```
Section "Screen"
    Identifier  "Screen 1"
    Device      "Voodoo3"
    Monitor     "High Res Monitor"
    DefaultDepth 16
    Subsection "Display"
       Depth       16
       Modes       "1280x1024" "1024x768" "800x600" "640x480"
       ViewPort    0 0
    EndSubsection
EndSection
```

The kernel module for the Voodoo3 is named `tdfx.o` and should be installed in /lib/modules/KERNEL-VERSION/misc/. It will be automatically loaded by the Xserver if needed.

The DRI 3D driver for the Voodoo3 should be in `/usr/X11R6/lib/modules/dri/tdfx_dri.so`. This will be automatically loaded by libGL.so.

8.1.2 Troubleshooting

- 3D acceleration for Voodoo3 is only supported in the 16 bit/pixel screen mode. Use `xdpyinfo` to verify that all your visuals are depth 16. Edit your XF86Config file if needed.
- The `/dev/3dfx` device is not used for DRI; it's only for Glide on older 3dfx hardware.

8.1.3 Performance

- Normally, buffer swapping in double-buffered applications is synchronized to your monitor's refresh rate. This may be overridden by setting the `FX_GLIDE_SWAPINTERNVAL` environment variable. The value of this variable indicates the maximum number of swap buffer commands can be buffered. Zero allows maximum frame rate.
- The `glTexEnv` mode `GL_BLEND` is not directly supported by the 3dfx hardware. It can be accomplished with a multipass algorithm but it's not implemented at this time. Applications which use that mode, such as the Performer Town demo, may become sluggish when falling back to software rendering to render in that mode.

8.1.4 Known Problems

- Glide cannot be used directly; only OpenGL-based programs are supported on the Voodoo3.
- SSystem has problems because of poorly set near and far clipping planes. The office.unc Performer model also suffers from this problem.

8.2 Intel i810

8.2.1 Configuration

Your XF86Config file's device section must specify the `i810` device, and specify a usable amount of video ram to reserve.

```
Section "Device"
    Identifier  "i810"
    VendorName  "Intel"
    Driver      "i810"
    VideoRam    10000
EndSection
```

The Screen section should then reference the i810 device:

```
Section "Screen"
    Identifier  "Screen 1"
    Device      "i810"
    Monitor     "High Res Monitor"
    DefaultDepth 16
    Subsection "Display"
        Depth       16
        Modes       "1280x1024" "1024x768" "800x600" "640x480"
        ViewPort    0 0
    EndSubsection
EndSection
```

The kernel module for the i810 is named `i810.o` and should be installed in /lib/modules/KERNEL-VERSION/misc/. It will be automatically loaded by the Xserver if needed.

The DRI 3D driver for the i810 should be in `/usr/X11R6/lib/modules/dri/i810_dri.so`. This will be automatically loaded by libGL.so.

8.2.2 Troubleshooting

- 3D acceleration for the i810 is only available in the 16 bit/pixel screen mode at this time. 32bpp acceleration is not supported by this hardware. Use `xdpyinfo` to verify that all your visuals are depth 16. Edit your XF86Config file if needed.
- The i810 uses system ram for video and 3d graphics. The X server will ordinarily reserve 4mb of ram for graphics, which is too little for an effective 3d setup. To tell the driver to use a larger amount, specify a VideoRam option in the Device section of your XF86Config file. A number between 10000 and 16384 seems adequate for most requirements. If too little memory is available for DMA buffers, back and depth buffers and textures, direct rendering will be disabled.

8.3 Matrox G200 and G400

8.3.1 Configuration

Your XF86Config file's device section must specify the `mga` device:

```
Section "Device"
    Identifier  "MGA"
    VendorName  "Matrox"
    Driver      "mga"
EndSection
```

The Screen section should then reference the MGA device:

```
Section "Screen"
    Identifier  "Screen 1"
    Device      "MGA"
    Monitor     "High Res Monitor"
    DefaultDepth 16
    Subsection "Display"
       Depth       16
       Modes       "1280x1024" "1024x768" "800x600" "640x480"
       ViewPort    0 0
    EndSubsection
EndSection
```

The kernel module for the G200/G400 is named `mga.o` and should be installed in /lib/modules/KERNEL-VERSION/misc/. It will be automatically loaded by the Xserver if needed.

The DRI 3D driver for the G200/G400 should be in `/usr/X11R6/lib/modules/dri/mga_dri.so`. This will be automatically loaded by libGL.so.

8.3.2 Troubleshooting

- 3D acceleration for the G200 and G400 is only supported in the 16 bit/pixel screen mode at this time. 32bpp will be supported in the future. Use `xdpyinfo` to verify that all your visuals are depth 16. Edit your XF86Config file if needed.

8.3.3 Performance

No data at this time.

8.3.4 Known Problems

- Multitexture is currently disabled on the G400 to work around a hardware lockup bug. This should be restored in a subsequent release.

8.4 ATI Rage 128

8.4.1 Configuration

Your XF86Config file's device section must specify the `r128` device:

```
Section "Device"
    Identifier  "Rage128"
    VendorName  "ATI"
    Driver      "r128"
EndSection
```

The Screen section should then reference the Rage 128 device:

```
Section "Screen"
    Identifier  "Screen 1"
    Device      "Rage128"
    Monitor     "High Res Monitor"
    DefaultDepth 16
    Subsection "Display"
       Depth       16
       Modes       "1280x1024" "1024x768" "800x600" "640x480"
       ViewPort    0 0
    EndSubsection
    Subsection "Display"
       Depth       32
       Modes       "1280x1024" "1024x768" "800x600" "640x480"
       ViewPort    0 0
    EndSubsection
EndSection
```

The kernel module for the Rage 128 is named `r128.o` and should be installed in /lib/modules/KERNEL-VERSION/misc/. It will be automatically loaded by the Xserver if needed.

The DRI 3D driver for the Rage 128 should be in `/usr/X11R6/lib/modules/dri/r128_dri.so`. This will be automatically loaded by libGL.so.

You may also set your screen depth to 32 for 32bpp mode.

8.4.2 Performance

While PCI Rage 128 based cards are supported, they do not yet support PCI GART, so they will not perform as well as their AGP counterparts.

8.4.3 Known Problems

DGA is not yet supported in the ATI Rage 128 X server. This feature will be added in a future release.

8.5 3DLabs Oxygen GMX 2000

The driver for this hardware was experimental and is no longer being developed or supported.

9. Limitations and Known Bugs

9.1 OpenGL

The following OpenGL features are not supported at this time: overlays, stereo, hardware-accelerated indirect rendering.

OpenGL-like functionality is provided with the Mesa library. XFree86 4.0 uses a beta version Mesa 3.3. When newer versions of Mesa are available, the 3D drivers can be updated without reinstalling XFree86 or libGL.so.

9.2 GLX

The GLX 1.3 API is exported but none of the new 1.3 functions are operational.

The new `glXGetProcAddressARB` function is fully supported.

9.3 Signal Handling

There are several understood, but unresolved problems relating to hardware locking and signal handling. Hitting CTRL-z to suspend a 3D application can sometimes cause the X server to lock-up if executing device driver code at that moment. Also, using a debugger to step through OpenGL/Mesa device driver functions code could cause a lock-up. These problems will be fixed in the future.

9.4 Scheduling

When you run multiple GL applications at once you may notice poor time slicing. This is due to an interaction problem with the Linux scheduler which will be addressed in the future.

9.5 libGL.so and dlopen()

A number of popular OpenGL applications on Linux (such as Quake3, HereticII, Heavy Gear 2, etc) dynamically open the libGL.so library at runtime with dlopen(), rather than linking with -lGL at compile/link time.

If dynamic loading of libGL.so is not implemented carefully, there can be a number of serious problems. Here are the things to be careful of in your application:

- Specify the RTLD_GLOBAL flag to dlopen(). If you don't do this then you'll likely see a runtime error message complaining that _glapi_Context is undefined when libGL.so tries to open a hardware-specific driver. Without this flag, *nested* opening of dynamic libraries does not work.
- Do not close the library with dlclose() until after XCloseDisplay() has been called. When libGL.so initializes itself it registers several callbacks functions with Xlib. When XCloseDisplay() is called those callback functions are called. If libGL.so has already been unloaded with dlclose() this will cause a segmentation fault.
- Your application should link with -lpthread. On Linux, libGL.so uses the pthreads library in order to provide thread safety. There is apparently a bug in the dlopen()/dlclose() code which causes crashes if the library uses pthreads but the parent application doesn't. The only known work-around is to link the application with -lpthread.

Some applications don't yet incorporate these procedures and may fail. For example, changing the graphics settings in some video games will expose this problem. The DRI developers are working with game vendors to prevent this problem in the future.

9.6 Bug Database

The DRI bug database which includes bugs related to specific drivers is at the SourceForge DRI Bug Database

Please scan both the open and closed bug lists to determine if your problem has already been reported and perhaps fixed.

10. Resources

10.1 Software

A collection of useful configuration files, libraries, headers, utilities and demo programs is available from http://dri.sourceforge.net/resources/resources.html

10.2 Documentation

- General OpenGL information is available at the OpenGL Home Page
- XFree86 information is available at the XFree86 Home Page
- Information about the design of the DRI is available from Precision Insight, Inc.
- Visit the DRI project on SourceForge.net for the latest development news about the DRI and 3D drivers.
- The DRI Compilation Guide explains how to download, compile and install the DRI for yourself.

10.3 Support

- The DRI-users mailing list at SourceForge is a forum for people to discuss DRI problems.
- In the future there may be IHV and Linux vendor support resources for the DRI.

DRI User Guide

CONTENTS

1. Preamble 1
   1.1 Copyright 1
   1.2 Trademarks 1
2. Introduction 1
3. Supported Hardware 2
4. Prerequisite Software 2
5. X Server Start-up 3
   5.1 Kernel module 3
   5.2 XF86Config file 3
   5.3 Memory usage 5
6. Using 3D Acceleration 5
   6.1 libGL.so 5
   6.2 Compiling and linking an OpenGL program 5
   6.3 Running your OpenGL program 6
   6.4 libOSMesa.so 6
   6.5 glxinfo 6
   6.6 Environment Variables 6
7. General Trouble Shooting 7
   7.1 Starting the X server 7
   7.2 Linking, running and verifying 3D acceleration 8
8. Hardware-Specific Information and Troubleshooting 9
   8.1 3dfx Voodoo3 9
   8.2 Intel i810 10
   8.3 Matrox G200 and G400 11
   8.4 ATI Rage 128 12
   8.5 3DLabs Oxygen GMX 2000 13
9. Limitations and Known Bugs 13
   9.1 OpenGL 13
   9.2 GLX 13
   9.3 Signal Handling 13
   9.4 Scheduling 14
   9.5 libGL.so and dlopen() 14
   9.6 Bug Database 14
10. Resources 14
   10.1 Software 14
   10.2 Documentation 14
   10.3 Support 15

$XFree86: xc/programs/Xserver/hw/xfree86/doc/sgml/DRI.sgml,v 1.5 2000/06/17 00:03:17 martin Exp $