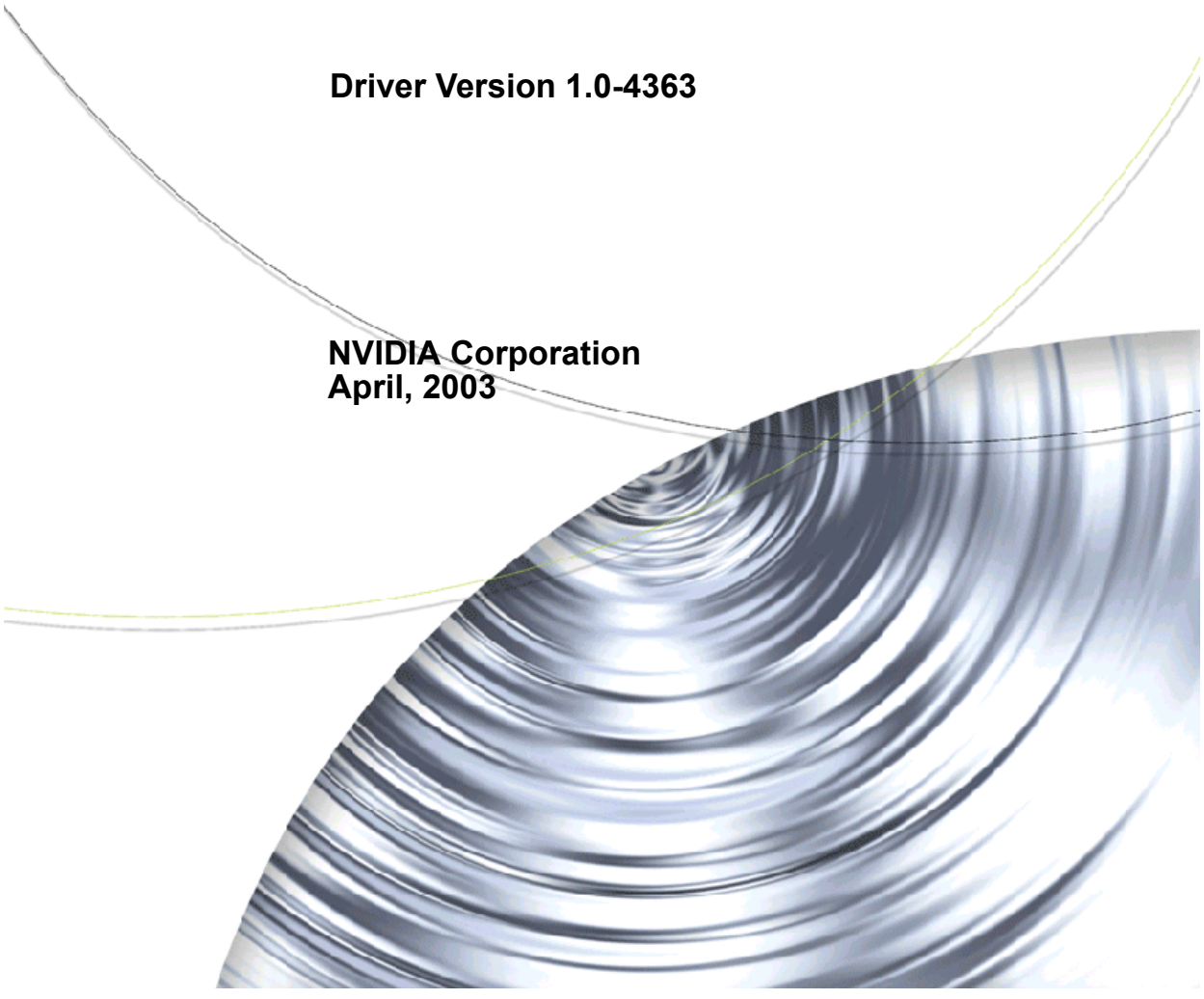
NVIDIA Accelerated Linux Driver Set
Release 40 Notes

Driver Version 1.0-4363

NVIDIA Corporation
April, 2003



Confidential Information

Published by
NVIDIA Corporation, Inc.
2701 San Tomas Expressway
Santa Clara, CA 95050

Table of Contents

List of Tables

CHAPTER 1

INTRODUCTION

This chapter contains the following major sections:

About Release Notes

These Release Notes contain information about the current Release 40 NVIDIA® Accelerated Linux® Driver Set. NVIDIA provides these notes to enable add-in-card (AIC) producers and original equipment manufacturers (OEMs) to monitor performance improvements and software problem (bug) resolutions in each documented version of the driver.

This guide explains how to install, configure, and use the NVIDIA Accelerated Linux Driver Set. It also describes current and historic software problem resolutions and software enhancements *and* contains Troubleshooting, Frequently Asked Questions, and other contact and support information.

Note: The content of this guide is also available in a `README` file, which is posted on the NVIDIA web site (www.nvidia.com) and is installed in `/usr/share/doc/NVIDIA_GLX-1.0/`.

About the NVIDIA Accelerated Linux Driver Set

The NVIDIA Accelerated Linux Driver Set brings both accelerated 2D functionality and high performance OpenGL® support to Linux XFree86 with the use of NVIDIA graphics processing units (**GPUs**).

The NVIDIA Accelerated Linux Driver Set contains the following key features:

- Optimized hardware acceleration of OpenGL applications through a direct-rendering X Server
- Accelerated 2D functionality and high performance OpenGL® support to Linux XFree86 with the use of NVIDIA graphics processing units (**GPUs**).
- NVIDIA Unified Driver Architecture (**UDA**) model. This means that one NVIDIA driver set can be used with all supported NVIDIA GPUs. Supported NVIDIA GPUs are listed in Table 1.2 and supported features (such as TwinView configuration and TV-Out and flat panel displays) are discussed in "NVIDIA Linux Driver History" on page 16.)

Choosing the NVIDIA Packages for Your System

The current version of the Release 40 NVIDIA Accelerated Linux Driver Set introduces a new packaging and installation mechanism that greatly simplifies the installation process.

You now need to download only the following file, which contains everything previously in the `NVIDIA_kernel` and `NVIDIA_GLX` packages:

Note: For the NVIDIA Linux driver release 1.0-4363, note the following:

- `NVIDIA_kernel` and `NVIDIA_GLX` tar files are available from the NVIDIA FTP site.
- `NVIDIA_kernel` and `NVIDIA_GLX` source RPM (SRPMS) files are not available.

For details on how to use the NVIDIA Linux Driver installer utility, see "Installing the NVIDIA Linux Driver" on page 32.

Minimum Operating System Requirements

This release includes drivers for the Linux operating systems listed in Table 1.1.

Table 1.1 Minimum Operating System Requirements

Module	Version	Determining the Version
Linux Kernel	2.2.12	`# cat /proc/version`
XFree86	4.0.1	`# XFree86 -version`
Kernel modutils	2.1.121	`# insmod -V`
If you need to build the NVIDIA kernel module, use the following components:		
`binutils`	2.9.5	`# size --version`
`GNU make`	3.77	`# make --version`
`gcc`	2.91.66	`# gcc --version`
If you build the NVIDIA kernel module from source RPMs (Red Hat Linux Package Manager utility), use the following component:		
`spec-helper RPM`		`# rpm -qi spec-helper`

Notes and Tips on Operating Systems

- XFree86 can be retrieved from `www.xfree86.org`. Software packages may also be available through your Linux distributor.
- All official stable kernel releases from version 2.2.12 and higher are supported.

  Prerelease versions such as "2.4.3-pre2" *and "*development" kernels such as 2.3.*x* or 2.5.*x* are *not supported*.

  The Linux kernel sources can be downloaded from `www.kernel.org` or one of its mirrors.
- `binutils` and `gcc` can be retrieved from `www.gnu.org` or one of its mirrors.
- If you are using XFree86, but do not have a file `/var/log/XFree86.0.log`, then you probably have a version 3.*x* of XFree86 and *must* upgrade.
- If you are setting up `XFree86 4.x` for the first time, it is often easier to begin with one of the open source drivers that ships with XFree86 (either "nv", "vga", or "vesa"). Once XFree86 is operating properly with the open source driver, it is easier to switch to the NVIDIA driver.

Supported NVIDIA Products

Table 1.2 lists the NVIDIA products supported by the current version of the NVIDIA Accelerated Linux Driver Set. The white and grey backgrounds indicate separate families of GPUs.

Table 1.2 Supported NVIDIA Products

Desktop Product Name	Workstation Product Name	Device PCI ID (SSDID)	Number of Displays Supported Per Card
nForce™2 (GeForce™4 MX Integrated GPU)		0x01F0	2
nForce 420/420D (GeForce2 Integrated GPU)		0x01A0	1
nForce 220/220D (GeForce2 Integrated GPU)		0x01A0	1
	Quadro® FX 2000	0x0309	2
	Quadro FX 1000	0x032B	2
GeForce FX 5800 Ultra		0x0301	2
GeForce FX 5800		0x0302	2
GeForce FX 5600 Ultra		0x0311	2
GeForce FX 5800		0x0312	2
GeForce FX 5200 Ultra		0x0321	2
GeForce FX 5200		0x0322	2
	Quadro4 980 XGL	0x0288	2
	Quadro4 780 XGL	0x0289	2
	Quadro4 700 Go GL	0x028C	2
GeForce4 4200 Go		0x0286	2
GeForce4 Ti 4800 SE		0x0282	2
GeForce4 Ti 4200 with AGP 8X		0x0281	2
GeForce4 Ti 4800		0x0280	2
	Quadro4 900 XGL	0x0258	2
	Quadro4 750 XGL	0x0259	2
	Quadro4 700 XGL	0x025B	2
GeForce4 Ti 4200		0x0253	2
GeForce4 Ti 4400		0x0251	2
GeForce4 Ti 4600		0x0250	2
	Quadro DCC	0x0203	1
GeForce3 Ti 500		0x0202	1
GeForce3 Ti 200		0x0201	1
GeForce3		0x0200	1

Table 1.2 Supported NVIDIA Products (continued)

Desktop Product Name	Workstation Product Name	Device PCI ID (SSDID)	Number of Displays Supported Per Card
	Quadro4 580 XGL	0x0188	2
	Quadro4 380 XGL	0x018B	2
	Quadro NVS 280	0x0181	2
GeForce4 448 Go		0x0186	2
GeForce4 488 Go		0x0187	2
GeForce4 MX 420 with AGP 8X		0x0183	2
GeForce4 MX 440 SE with AGP 8X		0x0182	2
GeForce4 MX 440 with AGP 8X		0x0181	2
	Quadro4 500 Go GL	0x017C	2
	Quadro NVS	0x017A	2/4
	Quadro4 500/550 XGL	0x0178	2
GeForce4 410 Go 16M		0x017D	2
GeForce4 440 Go 64M		0x0179	2
GeForce4 460 Go		0x0177	2
GeForce4 420 Go 32M		0x0176	2
GeForce4 420 Go		0x0175	2
GeForce4 440 Go		0x0174	2
GeForce4 MX 440 SE		0x0173	2
GeForce4 MX 420		0x0172	2
GeForce4 MX 440		0x0171	2
GeForce4 MX 460		0x0170	2
	Quadro2 Pro	0x0153	1
GeForce2 Integrated		0x01A0	1
GeForce2 Ultra		0x0152	1
GeForce2 Ti		0x0151	1
GeForce2 GTS/Pro		0x0150	1
	Quadro2 MXR/Go	0x0113	2
	Quadro2 EX	0x0113	1
	Quadro	0x0103	1
GeForce2 Go		0x0112	2
GeForce2 MX 100/200		0x0111	2
GeForce2 MX/MX 400		0x0110	2
GeForce DDR		0x0101	1
GeForce 256		0x0100	1

Table 1.2 Supported NVIDIA Products (continued)

Desktop Product Name	Workstation Product Name	Device PCI ID (SSDID)	Number of Displays Supported Per Card
RIVA TNT™2™ Ultra		0x0029	1
RIVA TNT2		0x0028	1
RIVA TNT2 M64		0x002D	1
RIVA TNT2 Vanta™		0x002C	1
RIVA TNT2 (Integrated)		0x00A0	1
RIVA TNT™		0x0020	1

Notes and Tips on Supported NVIDIA Products

- RIVA 128/128ZX products are supported by the open source “nv” driver for XFree86 but not by the NVIDIA Accelerated Linux Driver Set.
- Some NVIDIA Linux Driver features support only certain NVIDIA products, which is indicated, where applicable.
- If you want to check the Device PCI IDs for comparison with the values shown in Table 1.2, use one of these methods:
  - “`cat /proc/pci`” *or*
  - “`lspci -n`” . With this method, locate the device with the vendor ID of “`10de`”, as in the following example:

    ```
    02:00.0 Class 0300: 10de:0010 (rev 10)
    ```

    The non-NVIDIA information presented in the above line will vary depending of type of card you are using.

Features and Enhancements

For current and earlier history of new features and enhancements, see relevant sections, categories by driver version number, in Chapter 2: “NVIDIA Linux Driver History” on page 16.

Known Product Limitations

Software Issues

Athlon Processors: Support for Page Size Extension

For details, see "Support for the Processor Page Size Extension on Athlon Processors" on page 39 under "Troubleshooting AGP Stability Problems" on page 39.

X Server and Changing AGP Drivers

Under Linux32, after starting X twice with different AGP support, the X Server may crash causing the system to become unusable. To work around this problem, if you change your AGP driver, be sure to restart the system before restarting X.

OpenGL + Xinerama

Currently, OpenGL will not display to anything other than the **first display device** in an Xinerama environment.

OpenGL and dlopen()

There are some issues with the older version of the `glibc` dynamic loader (e.g., the version that shipped with Red Hat Linux 7.2) and applications such as Quake3 and Radiant that use the `dlopen()` function.

For further details, see "Frequently Asked Questions, Troubleshooting, & Other Resources" on page 68.

DPMS and TwinView

DPMS (Display Power Management System) modes “suspend” and “standby” do not work correctly on a second CRT when using TwinView. The screen becomes blank instead of the monitor being set to the requested DPMS state.

DPMS and Flat Panel

The DPMS modes “suspend” and “standby” do not work correctly on a flat panel display. The screen becomes blank instead of the flat panel being set to the requested DPMS state.

Multi-Card, Multi-Display

In certain cases under a multi-card/multi-display configuration, the secondary card may not be initialized correctly by the NVIDIA kernel module. You can work around this by enabling the `"UseInt10Module"` option in the XFree86 configuration file to soft-boot all secondary cards.

For further information, *see* “Option “UseInt10Module” “boolean”” on page 49.

Laptop Issues

For a list of current known issues with the NVIDIA driver and laptop computers, refer to “Known Laptop Computer Issues” on page 67.

Full-Scene Antialiasing (FSAA)

When FSAA is enabled (i.e., the `__GL_FSAA_MODE` environment variable is set to a value that enables FSAA and a multisample visual is selected), the rendering may be corrupted when resizing the window.

For details on setting the `__GL_FSAA_MODE` environment variable, see “OpenGL Environment Variable Settings” on page 61.

Interaction with Pthreads

Single threaded applications that perform `dlopen()` on the NVIDIA `libGL` library and then perform `dlopen()` on any other library that is linked against pthreads will crash with the NVIDIA `libGL` library. The problem does not occur with the current NVIDIA ELF TLS OpenGL libraries. (See “After Completing Driver Installation” on page 36 for a description of the ELF TLS OpenGL libraries.)

You can work around this problem by following one of these steps:

- Load the library that is linked with pthreads before loading libGL.so.
- Link the application with pthreads.

Hardware Issues

This section describes problems that will not be fixed. Usually, the source of the problem is beyond the control of NVIDIA.

Gigabyte GA-6BX Motherboard

This motherboard uses a LinFinity regulator on the 3.3-V rail that is rated to only 5 A — less than the AGP specification, which requires 6 A. When diagnostics or applications are running, the temperature of the regulator rises, causing the voltage to the NVIDIA chip to drop as low as 2.2 V. Under these circumstances, the regulator cannot supply the current on the 3.3-V rail that the NVIDIA chip requires.

This problem does not occur when the graphics card has a switching regulator or when an external power supply is connected to the 3.3-V rail.

VIA KX133 and 694X Chipsets With AGP 2X

On Athlon motherboards with the VIA KX133 or 694X chipset, such as the ASUS K7V motherboard, NVIDIA drivers default to AGP 2x mode to work around insufficient drive strength on one of the signals.

You can force AGP 4x by setting `NVreg_EnableVia4x` to `1`. However, note that doing so may cause the system to become unstable.

Irongate Chipsets With AGP 1X

AGP 1X transfers are used on Athlon motherboards with the Irongate chipset to work around a problem with the signal integrity of the chipset.

ALi (Acer Laboratories Inc.) Chipsets: ALi1541 and ALi1647

On ALi1541 and ALi1647 chipsets, NVIDIA drivers disable AGP to work around timing issues and signal integrity issues. You can force AGP to be enabled on these chipsets by setting `NVreg_EnableALiAGP` to `1`. However, note that doing so may cause the system to become unstable.

I/O APIC (SMP)

If you are experiencing stability problems with a Linux SMP system and seeing I/O APIC warning messages from the Linux kernel, system reliability may be greatly improved by setting the "`noapic`" kernel parameter.

Local APIC (UP)

On some systems, setting the "`Local APIC Support on Uniprocessors`" kernel configuration option can have adverse effects on system stability. If you are experiencing lockups with a Linux UP system where this option is set, you may want to disable local APIC support to resolve this problem.

CHAPTER 2

NVIDIA LINUX DRIVER HISTORY

This chapter contains the following major sections:

NVIDIA Linux Driver Versions

Release 40 is the latest release of the NVIDIA Accelerated Linux Driver Set. Table 2.1 contains a summary of driver releases and the versions associated with them. Some versions listed may not have been released outside of NVIDIA.

Table 2.1 NVIDIA Linux Driver Versions

Driver	Versions	Comments
Release 40	1.0-4000 through 1.0-4363	Releases ongoing
Release 25	1.0-2802 through 1.0-3123	
Release 20	1.0-2312 through 1.0-2313	
Release 10	1.0-1101 through 1.0-1541	
Release 6	0.9-1 through 0. 9-769	

New Features, Resolved Issues, and Enhancements

This section outlines new features, software enhancements, and resolved issues categorized by NVIDIA Linux driver releases:

Release 40: New Features

NVIDIA Linux Driver Installer

Features
The Linux `nvidia-installer` utility contains the following features:

- Uninstallation
- Auto-Update
- Multiple User Interfaces
- Updated Kernel Interfaces

For complete details on using the `nvidia-installer` utility, see "Installing the NVIDIA Linux Driver" on page 32.

Acknowledgements

- The Linux `nvidia-installer` utility was inspired by the `loki_update` tool.

  Reference: `http://www.lokigames.com/development/loki_update.php3`.
- The FTP and HTTP support in `nvidia-installer` is based on snarf 7.0.

  Reference: `http://www.xach.com/snarf/`.
- The self-extracting archive (`.run file`) in `nvidia-installer` is generated using `makeself.sh`.

  Reference: `http://www.megastep.org/makeself/`.

"TVOverscan" Option

The "TVOverscan" configuration option lets you enable and disable overscan on TV displays. For details, see "Option "TVOverscan" "integer"" on page 56.

"IgnoreDisplayDevices" Option

This `"IgnoreDisplayDevices"` configuration option communicates to the NVIDIA kernel module to ignore the indicated classes of display devices when checking those that are connected. For details, see "Option "IgnoreDisplayDevices" "string"" on page 42.

Support for NVIDIA GeForce FX and Quadro FX GPUs

The current NVIDIA Release 40 Linux driver supports the NVIDIA GeForce FX and Quadro FX series of GPUs. See "Supported NVIDIA Products" on page 9.

Release 40: Enhancements and Resolved Issues

Resolved issues in this section are categorized by the following versions:

Version 1.0-4363 *(current release)*

- Installing the NVIDIA driver while running X with the `'nv'` driver causes a system hang followed by installation failure.
- FSAA 8x and 16x modes do not work.
- System hangs on SMP systems when driving digital flat panels.
- Native resolution on Dell laptop internal flat panels may cause the flat panel to turn off.
- The installer installs the NVIDIA OpenGL header files in `/usr/include/GL` by default.
- System hangs on KT400 with AGP 8X.
- On TNT and TNT2, X may hang when viewing Web pages with large images.

Version 1.0-4349

- Changing from 'nv' to 'nvidia' can cause system to hang.
- Window position test fails on conform 1.4 with indirect rendering.
- Point Parameter test fails on conform 1.4 with indirect rendering.
- Added support for the new TLS mechanism in glibc 2.3 and Red Hat Linux 9.0.

Version 1.0-4321

- Cannot compile the driver on RedHat version 8.1 beta.
- Running two X servers results in corrupted display.
- Quadro NVS**:** Display corruption occurs when using multiple X screens.
- GeForce FX: Need to print out "Power supply" warning message.
- BadLength error occurs running conform1.4.
- Indirect rendering of Maya from NVIDIA to non-NVIDIA GPU-based cards causes system crash.
- Visuals in overlay plane are not returned if GLX_TRANSPARENT_TYPE_EXT is specified.
- Segfault occurs in _glXGetFBConfigFromVisual.
- Segfault occurs with glxinfo rendering to non-NVIDIA GPU-based cards.
- Quadro 980 XGL: Stability issues occur when running certain OpenGL programs and KDE.
- glXCreateWindow segfaults in libGL.so.
- Indirect rendering glXGetFBConfigsSGIX request fails from SGI Indy.
- Driver installation alters the permissions of /etc/modules.conf.
- In certain cases, glXGetProcAddressARB picks up wrong symbols.
- GLXBadRenderRequest occurs when running ac3d indirectly.
- Slow 2D performance using NVIDIA driver 1.0-4191. (Fixed by improving 2D performance over 1.0-4191.)
- There is no overlay or native DVD modes support on CX871 (or NVIDIA TV Output)
- With certain configurations, X can take from five to seven minutes to start.
- Cannot use Valgrind memory checker

- Quadro4 XGL: Cannot get overlay visual on TwinView.

Version 1.0-4191

- Support for OpenGL 1.4 with CineFX™ architecture
- Support for GLX 1.3
- Support for AGP 8X (AGP 3.0)
- Improved support for nForce2 IGP
- Support for TwinView Clone Mode stereo
- Added support for FSAA with UBB.
- Fixed bug where there was corruption at the borders of a PBuffer
- More complete acceleration for the `XRENDER` extension. Due to the experimental nature, `RENDER` acceleration is now disabled by default (`Option "RenderAccel"` enables `RENDER` acceleration. (See "Option "RenderAccel" "boolean"" on page 53 for details.)
- The driver no longer uses the XFree86 Acceleration Architecture (XAA), but provides its own 2D acceleration architecture to better accommodate the needs of simultaneous 3D and 2D rendering.
- Added color index overlay emulation on Quadro4 GPUs.
- Added support for separate X screens on a single GPU-based graphics card. This feature is supported by all GPUs that support TwinView.
- Fixed: PBuffer X crash
- Fixed: Floating point application crashing
- Fixed: Performance drops on pixel zoom value of less than 1.0.
- Fixed: GLX_SGI_video_sync functions differently on NVIDIA hardware compared to SGI hardware.
- Fixed: Corruption occurs at borders of data read from PBuffer

Release 25: New Features, Enhancements, and Resolved Issues

Note: The NVIDIA Linux Release 20 and earlier drivers do not support features that are new in Release 25.

Resolved issues in this section are categorized by the following versions:

Version 1.0-3123

- Added support for the NVIDIA product category that features the Quadro4 580 XGL and GeForce4 MX and GeForce4 4200
- Added support for the family of NVIDIA products that includes Quadro4 580 XGL and GeForce4 MX and the product line that includes Quadro4 980 XGL, GeForce4 4200 Go, and GeForce4 Ti 4800 SE. (See Table 1.2 in Chapter 1 for the categories of NVIDIA products.)
- Added Quadbuffered stereo visuals support for Quadro2-based, Quadro4-based, and Quadro DCC graphics cards.
- Improved Viewperf numbers.
- Added support for up to sixteen display devices (monitors).
- Added support for the IBM T221 digital flat panel.
- Added support for RGB (red-green-blue) OpenGL overlays in TwinView mode for graphics cards based on the most recent two NVIDIA product families. They include Quadro4 980 XGL, Quadro4 700 Go GL, GeForce 4200 Go, GeForce4 Ti 4200, Quadro4 900 XGL, GeForce4 Ti 4400, and others. See Table 1.2 in for reference.
- Added support for hardware clip IDs on NVIDIA Quadro4-based products. The configuration option `"UseClipIDs"` enables them, as explained in "Option "UseClipIDs" "boolean"" on page 45.

Version 1.0-2960

- Fixed problem with loading the GLX extension in multi-head environments with non-NVIDIA-based graphics cards.
- Significant performance improvements in Viewperf on Quadro-based graphics cards.
- Added the configuration option "NoRenderExtension" to disable the RENDER extension. This is useful when running in 8 bpp (bits per pixel) where the RENDER extension preallocates a large portion of the default colormap and thus corrupts many legacy applications. See "Option "NoRenderExtension" "boolean"" on page 44.
- Fixed a regression where I420 XvImages had the chroma planes swapped.
- Fixed some problems with moving overlay windows.
- Added a dynamic XvMC library "libXvMCNVIDIA_dynamic.so" so that applications can dynamically load the vendor-specific core XvMC support.
- Added XvMC motion-compensation acceleration for the Quadro4 XGL (700/750/900)-based and GeForce4 Ti (4200/4600)-based cards.
- Fixed certain issues with XvMC support under the NVIDIA GeForce4/Quadro4-based graphics cards.
- Fixed some problems with FSAA modes failing.

Version 1.0-2880

- Fixed rendering problems which occurred in some cases when GL_SYNC_TO_VBLANK was enabled.
- Fixed problem where the maximum pixel clock was set too low.
- Fixed a problem with image flicker when running full-screen applications with GeForce4-based graphics cards.
- Fixed a pixmap cache corruption problem when using GeForce3.
- Fixed some issues running multiple NVIDIA graphics cards simultaneously.

Version 1.0-2802

- Improved support for NVIDIA nForce.
- Added support for the GeForce4-based and Quadro4-based family of NVIDIA GPUs
- Added support for anisotropic filtering.

- Added the configuration option "DigitalVibrance" to enable Digital Vibrance Control™, which is a mechanism for controlling color separation and intensity that boosts the color saturation of an image. For details, see "Option "DigitalVibrance" "integer"" on page 46.
- Added the configuration option "Flat Panel Properties" to adjust dithering and scaling when X is started. See "Option "FlatPanelProperties" "string"" on page 47.
- Added support for SoftEDIDs. When this option is enabled, the driver generates an EDID based on the video BIOS instead of performing a table lookup. For further information, see "Standard Functionality" on page 64 under "Configuring a Laptop Computer" on page 64.
- Added libXvMCNVIDIA.a, which is an implementation of XvMC 1.0. This allows MPEG acceleration on NVIDIA GeForce4-based and Quadro4-based graphics cards.
- Added RGB workstation overlays for Quadro4-based graphics cards. These are double-buffered, Z-buffered 16-bit visuals. The transparency key is 0x0000 hex.
- Added an 8:8:8:8 XRGB XvImage format to the video blitter.
- Fixed problem of using SGIX_fbconfig and SGIX_PBuffer with indirect rendering, which was caused by an incorrect protocol.
- Fixed problem where the driver would fail on systems with 1GB or more of memory and a kernel configured to use all the memory.

Release 20: New Features, Enhancements, and Resolved Issues

Note: The NVIDIA Linux Release 10 and earlier drivers do not support features that are new in Release 20.

Resolved issues in this section are categorized by the following versions:

- "Version 1.0-2313" on page 23
- "Version 1.0-2312" on page 24

Version 1.0-2313

- Support for NVIDIA nForce (IGP 220D/IGP 420D)
- UBB support. See "Option "UBB" "boolean"" on page 48.
- Page Flipping. See "Option "PageFlip" "boolean"" on page 48.
- Window Flipping. See "Option "WindowFlip" "boolean"" on page 49.

- Support for OpenGL 1.3 (the current ARB-approved version of OpenGL)
- `PBuffer` and `fbconfig` extensions provide support for accelerated off-screen rendering.

Version 1.0-2312

- Improved performance of SPECviewperf, Quake, and immediate mode applications.
- Improved driver stability on AMD platforms.
- Fixed TwinView problem that caused garbage to appear on the screen when starting X with a null option on head 0.
- Added workaround for XAA bug that caused systems to hang when Lisa Screensaver is run. The workaround is to add the following line to the "`Device`" section of the XF86Config file:

  ```
  Option "XaaNoSolidFillTrap".
  ```

  This option prevents XAA from breaking wide lines (and polygons) into trapezoids and avoids an XAA clipping problem.
- Fixed a system hang that occurred on some GPUs when taking the X server down (after having run once successfully) and restarting it while using the DVI-I connected to a flat panel.
- Fixed problem where the performer application Perfly would hang when run in forked-draw mode.
- Fixed problem where the X driver would segfault when given an invalid MetaMode.
- Fixed problem where the console was not restored properly after entering X and returning to the console.
- Fixed problem where X did not redraw completely after a screen blank on GeForce2 Go and Quadro2 Go.
- Fixed problem with initializing the secondary card. In most cases, the secondary card is posted correctly. In the cases where it isn't, a workaround was created to initialize the card via X. (*See* "Multi-Card, Multi-Display" on page 13.)
- Added workarounds to enable AGP on ALi chipsets and enable 4X AGP on VIA chipsets. (*See* "VIA KX133 and 694X Chipsets With AGP 2X" on page 14 and "ALi (Acer Laboratories Inc.) Chipsets: ALi1541 and ALi1647" on page 14.)

Release 10: New Features

Note: The NVIDIA Linux Release 6 and earlier drivers do not support features that are new in Release 10.

GeForce 3

Release 10 is the first driver release to fully support the GeForce3 nfiniteFX™ engine in OpenGL using Texture and Vertex programs. Developers can now expose the full functionality of GeForce3 under Linux.

GeForce2 Go and Quadro2 Go

The Release 10 driver also added support for mobile (laptop) platforms with the GeForce2 Go and Quadro2 Go products, hot key switching, and improved driver stability on mobile platforms.

"TV Output" Option

Graphics cards based on an NVIDIA GPU with a TV-Out (S-Video) connector can be used to send the display to a television as another display device, such as a CRT (monitor) or a digital flat panel (DFP) display. The TV can be used by itself, or (on appropriate graphics cards) in conjunction with another display device in a TwinView™ configuration.

If a TV is the only display device connected to your graphics card, it will be used as the primary display when you start up your system; that is, the display will come up on the TV just as if it was a CRT.

Release 10: Resolved Issues and Enhancements

Resolved issues and enhancements are categorized by the following versions:

"Version 1.0-1541" on page 25

"Version 1.0-1512" on page 26

"Version 1.0-1450" on page 26

"Version 1.0-1420" on page 26

"Version 1.0-1251" on page 27

Version 1.0-1541

Fixed problem where starting X on GeForce3 caused screen corruption (e.g., red vertical lines).

Version 1.0-1512

- Fixed problem where garbage appears on the screen and the LCD blooms when X is started on the Toshiba 3000 series laptops.
- Changed behavior of the X server so that the NVIDIA splash screen only appears on the first run of X. The splash screen can also be disabled by setting an option in the XF86Config file; *see* "Option "NoLogo" "boolean"" on page 53 for details.
- Fixed problem where OpenGL applications would sometimes leave portions of their rendering behind when the window was closed using the "x" button on the window banner.
- Fixed problem on mobile where X would respond to the wrong hot key event under certain conditions.
- Fixed several more problems with indirect rendering.
- Fixed problem on SMP machines that occurred when VT switching while running gloss and gears with indirect rendering.
- Fixed problem where `/proc/nv/card0` did not report Quadro DCC correctly.

Version 1.0-1450

- Fixed problem on TNT2, where the driver would only support up to four threads per process.
- Fixed X Server crash that occurred when running two X Servers with AGPGART.
- Fixed some problems in GLX that occurred when running multi-threaded applications.
- Fixed problems with window borders picking up color values when moved across active OpenGL applications.
- Fixed problems so that the X Server detects a Quadro DCC-based card and properly initializes.
- Fixed problem so that Red Hat Linux 7.1 SMP RPMs can correctly uninstall with the "`rpm -e`" command while X Server is running.
- Corrected default OpenGL state when indirect rendering.

Version 1.0-1420

- Added `xf86XVOffscresenImage` support so the V4L module can use the hardware scaler on YUV surfaces.

- Added support for hot key switching on mobile platforms, i.e., laptop computers.
- Fixed a hang on mobile that occurred after starting, stopping, and then restarting X.
- Fixed a problem on mobile platforms that prevented DVDs from displaying.
- Fixed a problem that caused OpenGL programs to segfault when using a graphical login with xdm/kdm, and doing the following sequence: login, mode switch, logout, log in and run an OpenGL application.
- Fixed some indirect rendering problems.
- Fixed a problem that caused XF86Config file to fail on NVIDIA drivers.
- Fixed a crash that occurred when X forwarding over SSH.
- Fixed OpenGL front buffer clipping bug.
- Improved X-Render acceleration.
- Fixed a problem that prevented X-Render acceleration on GeForce3.
- The `NvAGP` option now defaults to"`3`", which causes the driver to use AGP GART, if it is available, and NVIDIA AGP, otherwise.
- Fixed issue with GeForce 256/DDR where Linux PCI adapter did not correctly initialize with AGP card installed.
- Fixed typo in error format string which caused error messages to report "%" when it should have printed one of several error messages.

Version 1.0-1251

- Added preliminary GeForce2 Go support.
- Added support for GeForce3 OpenGL and GLX extensions.
- Fixed many SMP issues.
- Added TV-Out support.
- Fixed DGA depth change problem.
- Rewrote 2D off screen memory allocation.
- Fixed X-Video in TwinView.
- Added acceleration for X-Render extension.
- Fixed `GLXPixmap` rendering.
- Fixed problem with `glXMakeCurrent()` to same drawable but different display.

- Fixed problem in which OpenGL caused a segfault when reading X atoms.
- Fixed issues so that X now gets the **dots per inch (dpi)** from the monitor's **EDID (Extended Display Identification Data)** instead of defaulting to 75 dpi.
  - All DPMS modes are now supported. Some DPMS issues remain for flat panels and the second display in a TwinView configuration.
  - Fixed support for AGP on systems with 1 GB or more of memory.

Release 6: New Features

TwinView

Note: The TwinView feature is only supported on NVIDIA products that support dual-display functionality, such as GeForce2 MX, GeForce2 Go, Quadro2 MXR, Quadro2 Go, and any of the GeForce4 or Quadro4 family of GPUs. You may want to consult with your graphics card vendor to confirm that TwinView is supported on your card.

TwinView is a mode of operation where two display devices can display the contents of a single X screen in any arbitrary configuration. TwinView supports a variety of display options, such as digital flat panels, RGB monitors, TVs, and analog flat panels. This method of using multiple displays has several distinct advantages over other techniques (such as Xinerama), as outlined here:

- A single X screen is used. The NVIDIA driver conceals all information about multiple display devices from the X Server, which only acknowledges one screen.
- Both display devices share one frame buffer. Thus, all the functionality present on a single display (e.g. accelerated OpenGL) is available in TwinView.
- No additional overhead is needed to emulate having a single desktop.

Note: If you are interested in using each display device as a separate X screen, see “Configuring Multiple Screens on One Graphics Card” on page 102.

Release 6: Resolved Issues and Enhancements

Resolved issues and enhancements are categorized by the following versions:

“Version 0.9-769” on page 29

“Version 0.96” on page 30

“Version 0.95” on page 30

Version 0.9-769

- Fixed problem where an old version of the release documentation was being installed instead of the current one.
- Fixed problem where direct rendering applications were allowed to continue rendering after "`xkill`" was called.
- Fixed problem where Tribes 2 crashed when compressed (s3tc) textures were used.
- Some drawable leaks were fixed in X and GLX.
- Fixed problem where the application would hang when calling "`glXMakeCurrent()`" while holding the X Server grab.
- BIOS posting problems with GeForce2 GTS and GeForce Ultra were fixed. These problem caused a significant performance loss.
- Added support for the X Render extension.
- TwinView functionality was enhanced for each display to pan independently.
- Fixed problem on TNT and TNT2 where "`Xv(Shm)PutImage`" returned "`BadAlloc`" in high resolutions when there was not enough video bandwidth to correctly display the YUV video overlay. This works now but the resulting display has artifacts.
- Fixed problem with cursor hangs in X.
- Fixed problem with X console not restoring on some monitors.
- Fixed problem with `fork()` and OpenGL rendering
- Fixed problem with X driver module, `nvidia_drv.o`, being stripped when RPM was rebuilt.
- Added missing PCI device IDs for some TNT2 variants and GeForce3.
- Fixed problem where the kernel would often hang during X and/or OpenGL operation when on an SMP machine and using the version 2.4 kernel.
- Fixed `SYNC_TO_VBLANK` hang with 2.4 kernels.
- Fixed DPMS so that it is possible to set the "off" option. DPMS options "suspend" and "standby" are not fully supported; these options simply blank the screen.

Note: Be sure to include `Option "DPMS" i`n your XF86Config file. Refer to the XF86Config man page for detailed information.

Version 0.96

- Fixed many SMP problems.
- Fixed memory management problems that arose with large RAM systems (500 MB plus).
- Added multi-monitor OpenGL support.
- Added TwinView support.
- Fixed more mode-line handling issues and added double-scan support.
- Fixed BIOS-posting problems with TNT2 M64 and GeForce2 MX.
- Added dynamic run-time selection between NVAGP and AGPGART.
- Fixed TNT2 OpenGL slowdowns, which were noticeable in UT.

Version 0.95

- Improved XFree86 version 4.0.1 support.
- Re-fixed console switch lockup.
- Fixed some AGP regressions resulting in better detection/support of AGPGART.
- Fixed color palette problems (Xgamma, direct color visuals).
- Added BIOS-posting override to help with NVIDIA products such as TNT2 M64.
- Update included version 2.4 support to newest test kernels.

Version 0.94

- Added support for XFree86 version 4.0.1.
- Fixed mode-setting problem.
- Added AGPGART support (NVAGPGART version 0.5-5).
- Added GeForce2 MX support.
- Fixed various hangs.
- Added FSAA support.
- Fixed problem where an OpenGL application malfunctioning during a console switch would crash the X Server.

Version 0.9-3

- Allowed mode-line directives in the XF86Config file to override NVIDIA auto-detection of monitor resolutions and refresh rates.
- Implemented “correct” fix for TNT memory-type problems.
- Fixed VT switch lockups.
- Fixed general ALi chipset lockups.
- Added and documented some registry keys. Check `os-registry.c` in the kernel source directory for more details and options.
- Implemented workaround for Quake3 mode switch problem that caused system to crash. Note that this was a problem in the `dlopen()` function.
- Implemented major improvement in multi-threading behavior.
- Display list sharing with `glXCreateContext` now works.
- Added faster implementation of `glTexImage/glTexSubImage` and `glCopyTexImage/glCopytexSubImage` calls.
- Fixed kernel memory leak, which was related to threaded OpenGL This problem was most noticeable with XMMS.
- Fixed build problems with older version 2.2.*x* kernels (Red Hat Linux 6.0)

Version 0.9-2

- Fixed problem initializing TNT with SGRAM.
- Added better logging and messages for tracking problems.
- Added dynamic, rather than static, allocation of client data in kernel.
- Incorporated *unsupported* version 2.3 kernel changes for completeness.
- Makefile updates add “`-D_LOOSE_KERNEL_NAMES`” and default to “`make install`”.
- Improved mode switching in Quake3.
- Changed installation name of libraries. Added revision .1.0.1 to the libraries.
- Temporarily forced disabling of AGP fast writes for all chips.
- Fixed monitor issues and allowed overriding of synchronization polarities.

Version 0.9-1

Initial Release

CHAPTER 3

INSTALLING THE NVIDIA LINUX DRIVER

This chapter contains the following major topics:

Note: See Appendix A "Installed NVIDIA Linux Driver Components" on page 86 for descriptions of the NVIDIA Linux driver components.

Before Starting NVIDIA Driver Installation

Follow these steps before beginning the driver installation:

1. Exit the X server.
2. Set your default run level so you will start up on a VGA console and directly into X. Doing so will make it easier to recover if there is a problem during the installation.

   Note: This procedure is normally done by modifying your `/etc/inittab` file. If you need help with this procedure, consult the documentation that came with your Linux distribution.

Using the NVIDIA Driver Installer

Follow these steps to use the NVIDIA Linux driver installations utility:

1. Download the following file:

   `NVIDIA-Linux-x86-1.0-4363.run`.

2. Exiting the X server.

3. Change directory to the directory the contains the downloaded file.

4. Run the following command:

   `sh NVIDIA-Linux-x86-1.0-4363.run`

   The `.run` file is a self-extracting archive. Executing the `.run` file with the above command installs and extracts the contents of the archive and runs the `nvidia-installer` utility. The utility prompts you through the NVIDIA driver installation process.

   You can also use the `nvidia-installer` utility to uninstall and auto-download updated NVIDIA Linux drivers. For details, see "Using the NVIDIA Driver Installer Features" on page 35 later in this chapter.

5. For troubleshooting information and answers to frequently asked questions about NVIDIA driver installation, see "NVIDIA Installer: Common Questions and Problems" on page 68 in Chapter 5.

.Run File Command Line Options

The `.run` file accepts many command line options. Table 3.1 lists a few of the more common options.

Table 3.1 `Run` File Command Line Options

Option	Command Line Example and Description
`--info`	`sh NVIDIA-Linux-x86-1.0-4363.run --info` Print embedded information about the .run file and exit
`--check`	`sh NVIDIA-Linux-x86-1.0-4363.run --check` Check the integrity of the archive and exit.
`--extract-only`	`sh NVIDIA-Linux-x86-1.0-4363.run --extract-only` Extract the contents of: `./NVIDIA-Linux-x86-1.0-4363.run` but do not run `nvidia-installer`.

Table 3.1 Run File Command Line Options (continued)

Option	Command Line Example and Description
--help	sh NVIDIA-Linux-x86-1.0-4363.run --help Print usage information for the common command line options and exit.
--advanced-options	sh NVIDIA-Linux-x86-1.0-4363.run --advanced-options Print usage information for the common command line options as well as the advanced options, and then exit.

Kernel Interfaces

The NVIDIA kernel module has a kernel interface layer that must be compiled specifically for the configuration and version of the kernel you are running. NVIDIA distributes the source code to this kernel interface layer as well as a precompiled version for many of the kernels distributed by some popular distributions.

When you run the NVIDIA installer, it determines if it has a precompiled kernel interface for the kernel you are running. If the installer does not have this kernel interface, it checks for availability of the interface on the NVIDIA FTP site and then downloads it (*assuming you are connected to the internet*), if available.

- If a precompiled kernel interface that matches your kernel is found, then that kernel interface will be linked (see **Note:** below) against the binary portion of the NVIDIA kernel module. The result of this operation is a kernel module appropriate for your kernel.

  Note: The NVIDIA installation process requires an installed linker. The linker (usually /usr/bin/ld) is part of the binutils package. Please confirm that you have this package installed before you install the NVIDIA Linux driver.

- If no matching precompiled kernel interface is found, then the NVIDIA installer compiles the kernel interface for you. However, the installer first checks that you have the correct kernel headers installed on your system. If the installer must compile the kernel interface, then you must install the kernel-sources package for your kernel.

Using the NVIDIA Driver Installer Features

Uninstallation

The NVIDIA Linux driver installation process automatically backs up any conflicting files and records any new files that are installed on the system. Note that when you run the NVIDIA installer to install a new NVIDIA Linux driver, the process automatically uninstalls any previous drivers.

If you want to manually uninstall the current NVIDIA Linux driver, run the following command:

```
nvidia-installer --uninstall
```

This process removes any files that were installed on your system and restores any files that may have been backed up.

Auto-Update

Table 3.2 lists commands that you can use to update your NVIDIA Linux driver.

Table 3.2 Auto-Update Command Line Options

Command	Description
`nvidia-installer --latest`	Connects to the NVIDIA FTP site and reports the latest driver version and the URL to the latest driver file.
`nvidia-installer --update`	Connects to the NVIDIA FTP site, download the most recent driver file, and installs it on your system.

Multiple User Interfaces

The NVIDIA Linux driver installer will use an ncurses-based user interface if it can find the correct ncurses library. Otherwise, it will fall back to a simple command line user interface.

To disable use of the ncurses user interface, use the option "`--ui=none`".

Updated Kernel Interfaces

The NVIDIA Linux driver installer can download updated precompiled kernel interfaces from the NVIDIA FTP site (for kernels that were released after the NVIDIA driver release).

After Completing Driver Installation

After you have completed installing the NVIDIA Linux driver on your computer, you must edit your XF86Config file before the newly installed driver can be used. For details, see the next chapter, "Configuring the NVIDIA Linux Driver" on page 37.

.

CHAPTER 4

CONFIGURING THE NVIDIA LINUX DRIVER

This chapter contains the following major topics:

Editing Your XF86Config File

When XFree86 4.0 was released, it used a slightly different XF86Config file syntax than the 3.*x* series used. Therefore, to allow both 3.*x* and 4.*x* versions of XFree86 to coexist on the same system, it was decided that XFree86 4.*x* would use the configuration file `'/etc/X11/XF86Config-4'`, if it existed; otherwise, XFree86 4.*x* would use `'/etc/X11/XF86Config'`.

Note: X searches a large path to find the Config (configuration) files. For a complete description of the search path, it is strongly recommended that you refer to the XF86Config man page.

Follow this procedure to work with your XF8Config file:

1. Verify the configuration file that XFree86 is using.

   To do so, you can locate a line beginning with "`==) Using config file:`" in your XFree86 log file (`/var/log/XFree86.0.log`).

2. **If you do not have a working XF86Config file**, here are some suggested resources you can use for reference:

   - A sample config file is included with XFree86.
   - A sample config file is also included with the NVIDIA driver package — the file is installed in '`/usr/share/doc/NVIDIA_GLX-1.0/`'.
   - You can also use a program such as XF86Config. Some distributions provide their own tool for generating an XF86Config file. For additional details on XF86Config file syntax, refer to the man page.

3. If you encounter any problems working with the XF86Config file, refer to relevant questions in "Frequently Asked Questions, Troubleshooting, & Other Resources" on page 68.

XF86Config Options: Configuring AGP

The "`NvAgp`" option in your XF86Config file provides several ways to configure the use of AGP by the NVIDIA kernel module.

1. You can use either the NVIDIA AGP module (`NvAGP`) or the AGP module that comes with the Linux kernel (AGPGART).

   `Option "NvAgp" "0"` ... disables AGP support

   `Option "NvAgp" "1"` . . . use NVAGP, if possible

   `Option "NvAgp" "2"` ... use AGPGART, if possible

   `Option "NvAgp" "3"` ... try AGPGART; if that fails, try NVAGP

   Default: "`3`"

   (The default was "`1`" through NVIDIA Linux driver version 1.0-1251.)

2. It is recommended that you use the AGP module that works best with your AGP chipset. **If you are experiencing problems with stability**, you may want to start by disabling AGP and observing if that solves the problems. You can then experiment with either of the other AGP modules.

3. You can query the current AGP status at any time using the `/proc` file system interface. For further details, see "Proc Filesystem Interface" on page 98.

4 To use the Linux AGPGART module, you must compile it with your kernel, either statically linked in or built as a module.

Note: NVIDIA AGP support cannot be used if AGPGART is loaded in the kernel. It is recommended that you compile AGPGART as a module and verify that it is not loaded when trying to use NVIDIA AGP. ***Note*** that changing AGP drivers generally requires restarting your computer before the changes actually take effect.

5 Rebuild and reinstall the new driver using the "`make`" command, which forces the driver to ignore the BIOS of the NVIDIA product and use your values.

AGP Chipsets Supported by NVIDIA AGP

The following AGP chipsets are supported by the NVIDIA AGP module; for all other chipsets, it is recommended that you use the AGPGART module,

ALi 1621	ALi 1631	ALi 1647	ALi 1651	ALi 1671
AMD 751 ("Irongate")	AMD 761 ("IGD4")	AMD 762 ("IGD4 MP")	Intel 440LX	Intel 440BX
Intel 440GX	Intel 815 ("Solano")	Intel 820 ("Camino")	Intel 830	Intel 840 ("Carmel")
Intel 845 ("Brookdale")	Intel 845G	Intel 850 ("Tehama")	Intel 860 ("Colusa")	Micron SAMDDR ("Samurai")
Micron SCIDDR ("Scimitar")	nForce AGP nForce2 AGP	RCC 6585HE	VIA 8371	VIA 82C694X
VIA KT133	VIA KT266	SiS 630	SiS 633	SiS 635
SiS 645	SiS 730	SiS 733	SiS 735	SiS 745

Troubleshooting AGP Stability Problems

If you are experiencing AGP stability problems, note the following issues:

Support for the Processor Page Size Extension on Athlon Processors

Some Linux kernels have a conflicting cache attribute bug that is exposed by advanced speculative caching in newer AMD Athlon family processors (AMD Athlon XP, AMD Athlon 4, AMD Athlon MP, and Models 6 and above AMD Duron). This kernel issue is usually encountered under heavy use of accelerated 3D graphics with an AGP graphics card.

Linux distributions based on kernel 2.4.19 and later "should" incorporate the fix for this issue. However, older kernels require help from the user in ensuring that

a small portion of advanced speculative caching is disabled (normally done through a kernel patch) and a boot option is specified in order to apply the entire fix.

Note: The NVIDIA driver automatically disables the small portion of advanced speculative caching for the affected AMD processors without the need to patch the kernel; it can be used on kernels that already incorporate the kernel bug fix.

Additionally, for older kernels, the user needs to perform the startup (boot) option portion of the fix by explicitly disabling 4MB pages. This can be done from the startup command line by specifying:

```
mem=nopentium
```

Or by adding the following line to `'etc/lilo.conf'`:

```
append = "mem=nopentium"
```

AGP Drive Strength BIOS Setting (Via-based Motherboards)

Many Via-based motherboards allow adjusting the AGP drive strength in the system BIOS.

Note: The setting of this option largely affects system stability; the range between 0xEA and 0xEE seems to work best for NVIDIA hardware. Setting either nibble to 0xF generally results in severe stability problems.

Caution: If you decide to experiment with this setting and, as a result, **use improper settings, be advised that you may cause your system to be unbootable**. In that case, you will need to reset to a working value by using either a PCI graphics card or by resetting the BIOS to its default values.

System BIOS Version

Make sure you have the latest system BIOS provided by the graphics card manufacturer.

AGP Rate

You may want to decrease the AGP rate setting if you are seeing lockups with the value you are currently using. You can do so by following these steps:

1. Extract the `.run` file using the following commands:

```
sh NVIDIA-Linux-x86-1.0-4363.run --extract-only
cd NVIDIA-Linux-x86-1.0-4363/usr/src/nv/
```

2. Edit `os-registry.c` and make the following changes:

```
- static int NVreg_ReqAGPRate = 7;
+ static int NVreg_ReqAGPRate = 4;   /* force AGP Rate
to 4x */
```

or

```
+ static int NVreg_ReqAGPRate = 2;   /* force AGP Rate
to 2x */
```

or

```
+ static int NVreg_ReqAGPRate = 1;   /* force AGP Rate
to 1x */
```

3. Remove the two leading underscores:

```
- { "__ReqAGPRate", &NVreg_ReqAGPRate },
+ { "ReqAGPRate", &NVreg_ReqAGPRate },
```

4. Recompile and load the new kernel module.

Athlon Motherboards with the VIA KX133 or 694X Chipset

Athlon motherboards with the VIA KX133 or 694X chip set such as the ASUS K7V motherboard, NVIDIA drivers default to AGP 2x mode to work around insufficient drive strength on one of the signals. You can force AGP 4x by setting "`NVreg_EnableVia4x`" to 1.

Note: This procedure may cause the driver to become unstable.

ALi1541 and ALi1647 Chipsets

On ALi1541 and ALi1647 chipsets, NVIDIA drivers disable AGP to work around timing issues and signal integrity issues. You can force AGP to be enabled on these chipsets by setting "`NVreg_EnableALiAGP`" to 1.

Note: This procedure may cause the driver to become unstable.

XF86Config Options: Introduced in Release 40

The following options are supported by the NVIDIA XFree86 driver for Release 40. To enable any of these options, you must include them in your XF86Config configuration file:

Note: These Release 40 features are not supported by Release 25 or older NVIDIA Linux drivers.

Option "TVOverscan" "*integer*"

This option was introduced in version 1.0-4349.See "Option "TVOverscan" "integer"" on page 56.

Option "IgnoreDisplayDevices" "*string*"

This option was introduced in version 1.0-4349.

This option communicates to the `NVIDIA_kernel` module to completely ignore the indicated classes of display devices when checking those display devices that are connected. You may specify a comma-separated list containing any of "CRT", "DFP", and "TV".

Example: `Option "IgnoreDisplayDevices" "DFP,TV"`

will cause the NVIDIA driver to not attempt to detect the DFP and TV display devices.

Note: Any display device attached to a 15-pin VGA connector is recognized by the NVIDIA Linux driver as a CRT. "DFP" should only be used to refer to flat panels connected through a DVI port.

Use of this option is normally unnecessary. However, some video BIOSs contain incorrect information about the display devices that may be connected or the i2c port that should be used for detection. These errors can cause long delays in starting X.

If you are experiencing such delays, you may be able to avoid the delay by using this option to ignore display devices that you know are not connected.

Option “NoBandWidthTest” “*boolean*”

As part of mode validation, the X driver tests whether a given mode fits within the constraints of the hardware's memory bandwidth. This option disables the test.

Default: The memory bandwidth test is performed.

Option “Dac8Bit” “*boolean*”

Most NVIDIA Quadro-based processors, by default, use a 10-bit color look-up table (LUT).

Enabling this option (specifying `"on"` in the below line) forces the NVIDIA GPU to use an 8-bit LUT.

Default: `Option "Dac8Bit" "off"` forces the use of a 10-bit LUT, when available.

Option “CIOverlay” “*boolean*”

When this option is enabled, the X server provides Color Index workstation overlay visuals (depth 8 `PseudoColor` visuals) both with and without a transparency key.

Note: When enabled, the Color Index workstation overlay visuals have the same limitations as described in “Option “Overlay” “boolean”” on page 46.

Default: `Option "CIOverlay" "off"` disables Color Index workstation overlay visuals.

Option “TransparentIndex” “*integer*”

When color index overlays are enabled, this option allows the user to choose the pixel that is used for the transparent pixel in visuals featuring transparent pixels. This value is clamped between 0 and 255.

Note: Some applications such as Alias/Wavefront’s Maya require this option to be set to “0” (zero) in order to work correctly.

Default: `Option "TransparentIndex" "0"` disables `TransparentIndex`.

Option “OverlayDefaultVisual” “*boolean*”

When overlays are used and this option is enabled (the above statement is set to "on"), this option sets the default visual to an overlay visual thereby putting the root window in the overlay.

Note: This option is not recommended for RGB overlays.

Default: `Option "OverlayDefaultVisual" "off"`.

XF86Config Options: Introduced in Release 25

The following options are supported by the NVIDIA XFree86 driver for Release 25 and need to be included in your XF86Config configuration file:

Note: These Release 25 features are not supported by Release 20 or older NVIDIA Linux drivers.

Option “NoRenderExtension” “*boolean*”

This option, when specified, disables the `Render` extension.

Other than recompiling the X server, XFree86 doesn't seem to have another way of disabling the `Render` extension. Because the NVIDIA Linux driver can control this functionality, this option is exported, which is useful in depth 8 where `Render` would normally utilize most of the default color map.

Default: `RENDER` is offered, when possible.

Option “NoTwinViewXineramaInfo” “*boolean*”

When in TwinView, the NVIDIA X driver normally provides a Xinerama extension that allows X clients (such as window managers) to call `XineramaQueryScreens()` to detect the current TwinView configuration. Because some window mangers cannot properly this process, this option is provided to disable this process.

Default: `Option "NoTwinViewXineramaInfo" "on"` means that TwinView Xinerama information is *not* provided.

Option “UseClipIDs” “*boolean*”

Note: This feature is only supported on Quadro4 and Quadro FX families of GPUs when UBB is enabled.

Enabling this feature sets aside a small amount of video RAM for the clip ID surfaces, which is typically less than two megabytes.

Example: `Option "UseClipIDs" "on"` *enables* usage of hardware clip ID buffers to improve rendering performance to drawables that are clipped in a complex way.

Default: `Option "UseClipIDs" "off"` means that clip ID surfaces are not used.

Option “Stereo” “*integer*”

This option enables support of quadbuffered stereo visuals on Quadro-based graphics cards.The “integer” indicates the type of stereo glasses being used, as explained below.

Default: `Option "Stereo" "0"` means that stereo is not enabled.

Example: `Option "Stereo" "1"`

1 - DDC glasses: The synchronization signal is sent to the glasses through the DDC signal to the display device (monitor), which usually involve a pass-through cable between the display device and the graphics card.

2 - “Blueline” glasses: These usually involve a pass-through cable between the display device and the graphics card. The glasses can detect the eye to display based on the length of a blue line visible at the bottom of the screen. When in this mode, the root window dimensions are one pixel shorter in the Y dimension than requested. This mode does not work with virtual root window sizes larger than the visible root window size (desktop panning).

3 - Onboard stereo support: This feature is usually only found on professional graphics cards. The glasses connect through a DIN connector on the back of the graphics card.

4 - TwinView Clone Mode Stereo: On NVIDIA GPU-based graphics cards that support TwinView, the left eye is displayed on the first display and the right eye is displayed on the second display. This is normally used in conjunction with special projectors to produce two polarized images, which can then be viewed with polarized glasses.

Note: To use this stereo mode, you must also configure TwinView in Clone mode with the same resolution, panning offset, and panning domains on each display.

Note: Stereo is only available on Quadro-based cards and is not supported in TwinView (with the exception of TwinView Clone Mode Stereo - option #4 above).

Note: Currently, stereo operation may not work properly on the original Quadro (NV10) processor and left-right flipping may be erratic. This issue will be resolved in a future release of the NVIDIA Linux driver.

Option "DigitalVibrance" "*integer*"

Note: This feature is not supported on NVIDIA products that are older than GeForce2.

Digital Vibrance, a mechanism for controlling color separation and intensity, boosts the color saturation of an image.

The *integer* value can be must be in the range from 0 through 25, where 25 is the highest level of digital vibrance.

Example: `Option "DigitalVibrance" "20"`

Default: `Option "DigitalVibrance" "0"` disables Digital Vibrance.

Option "Overlay" "*boolean*"

Note: This feature is supported only on NVIDIA Quadro4 and Quadro FX (excludes Quadro NVS) family of GPUs in depth 24.

This option performs the following functions:

- Enables RGB workstation overlay visuals.
- Causes the X server to advertise the `SERVER_OVERLAY_VISUALS` root window property and GLX to report single and double-buffered Z-buffered 16-bit overlay visuals.

Also note the following:

- The transparency key is pixel 0x0000 (hex).
- There is no gamma correction support in the overlay plane.
- This feature requires XFree86 version 4.1.0 or later version.
- NVIDIA GPUs in the Quadro4 2*xx*/3*xx*/5*xx* XGL family have additional limitations — overlays are not supported in TwinView mode or with virtual desktops larger than 2046 x 2047 in any dimension. For example, overlays are not supported in 2048 x 1536 screen resolution.

  Note: NVIDIA GPUs in the Quadro4 7*xx*/9*xx* XGL and the Quadro FX families *do not have this limitation*.

Example: `Option "Overlay" "on"` enables overlay.

Default: `Option "Overlay" "off"` disables overlay.

Option "FlatPanelProperties" "*string*"

This options requests particular properties of any connected flat panels as a comma-separated list of "property = value" pairs.

Currently, the only two available properties are "Scaling" and "Dithering", which can have any of the following values:

- **Scaling**
  - *default*: The driver will use that scaling state that is current.
  - *native:* The driver will use the flat panel's scaler, if there is one.
  - *scaled:* The driver will use the NVIDIA scaler, if possible.
  - *centered:* The driver will center the image, if possible.
  - *aspect-scaled*: The driver will scale with the NVIDIA scaler, but keep the aspect ratio correct.
- **Dithering**
  - *default*: The driver determines when to dither.
  - *native:* The driver will always dither, when possible.
  - *disabled:* The driver will never dither.
- **Example:** An example properties string is:

```
Option "FlatPanelProperties" "Scaling = centered,
Dithering = enabled"
```

XF86Config Options: Introduced in Release 20

Note: These Release 20 features are also supported by Release 25 and newer NVIIDA Linux driver releases.

The following options are supported by the NVIDIA XFree86 driver for Release 20 and need to be included in your XF86Config configuration file:

UBB, Page Flipping, and Window Flipping features provide performance gains under certain conditions.

Option "PageFlip" "*boolean*"

Note: This options is available on all NVIDIA GeForce and newer GPUs, which excludes TNT/TNT2 products.

Page Flipping is enabled (by default) in the case of a single full-screen unobscured OpenGL application when synchronizing to `VBLANK`. Buffer swapping is done by changing the buffer rather than copying the back buffer contents to the front buffer.

This mechanism allows for a much higher performance and non-tearing swapping during the retrace (when `__GL_SYNC_TO_VBLANK` is set).

Default: `Option "PageFlip" "on"` enables Page Flipping.

Example: `Option "PageFlip" "off"` disables Page Flipping.

Option "UBB" "*boolean*"

Note: The Unified Back Buffer (UBB) option is only available with the Quadro (with exception of Quadro NVS) family of NVIDIA GPUs,

UBB is enabled by default when there is sufficient video memory available so that all windows share the same back, stencil, and depth buffer. When there are many windows, the back, stencil, and depth buffer usage will never exceed the size of that used by a full-screen window.

Note: For a single small window, the back, stencil, and depth buffer usage equals that of a full-screen window. In this case, the video RAM may be used more efficiently when UBB is disabled.

Default: `Option "UBB" "on"` enables UBB.

Example: `Option "UBB" "off"` disables UBB.

Option "UseInt10Module" "*boolean*"

Default: `Option "UseInt10Module" "off"`

In this "default" condition, POSTing the cards is done through the NVIDIA kernel module.

To enable use of the XFree86 Int10 module to soft-boot all secondary cards rather than POSTing the cards through the NVIDIA kernel module, specify `"on"` in the above statement.

Option "WindowFlip" "*boolean*"

Note: Window Flipping is a feature that requires UBB and therefore only available on Quadro and Quadro2 family products.

To enable Window Flipping:

1. Confirm that UBB is enabled. (*See* "Option "UBB" "boolean"" on page 48 in the previous section.)
2. Include this line in your configuration file:

```
Option "WindowFlip" "on"
```

Window Flipping only works when there is a single OpenGL window. This OpenGL windows buffers can be swapped by changing the buffer rather than copying the contents of the back buffer to the front buffer. This is similar to the Page Flipping functionality but removes the restriction that the window must be unobscured and full-screen in order for the feature to work.

Default: `Option "WindowFlip" "off"` disables Window Flipping.

XF86Config Options: Introduced in Release 10

The following options are supported by the NVIDIA XFree86 driver and should be specified in your XF86Config file.

Note: These Release 10 options are also supported by Release 20 and newer NVIDIA Linux driver releases.

Option "ConnectedMonitor" "*string*"

This option lets you to override the display device that the NVIDIA kernel module detects as connected to your graphics card. Using this option may be useful under the following circumstances:

- Some of your display devices are not being detected using Display Data Channel **(DDC)** protocols
- You are using a keyboard/video/mouse **(KVM)** switch and you are switched away when X is started. In such a situation, the NVIDIA kernel module cannot detect the display devices that are connected and the NVIDIA X driver assumes you have a single CRT connected.

Default: String is NULL.

Values for this option are:

- `"CRT"` (cathode ray tube / analog monitor)
- `"DFP"` (digital flat panel)
- `"TV"` (television)

If using a **TwinView** configuration, this option may be a comma-separated list of display devices; for example:

- `"CRT,CRT"`
- `"CRT,DFP"`
- `"CRT,TV"`

Note: Any display device attached to a 15-pin VGA connector is recognized by the NVIDIA Linux driver as a CRT. “DFP” should only be used to refer to flat panels connected through a DVI port.

Examples:

- `Option "ConnectedMonitor" "TV"`
- `Option "ConnectedMonitor" "CRT"`
- `Option "ConnectedMonitor" "CRT,DFP"`
- `Option "ConnectedMonitor" "CRT,TV"`

As in all XF86Config entries, spaces are ignored and all entries are case insensitive.

For details on configuring TwinView, see “Enabling and Configuring TwinView” on page 57.

Option “CursorShadow” “*boolean*”

Note: This option is only available with GeForce2 and newer GPUs, which excludes TNT/TNT2, GeForce 256, GeForce DDR, and Quadro.

This option enables (`"on"`) or disables (`"off"`) use of a shadow with the hardware accelerated cursor. The shadow is a black translucent replica of your cursor shape at a given offset from the real cursor.

Default: `Option "CursorShadow" "off"`

Option “CursorShadowAlpha” “*integer*”

This option defines the alpha value to use for the cursor shadow and is applicable *only* if “`CursorShadow`” is enabled.

Default: `Option "CursorShadowAlpha" "64"`

Integer must be in the range [0, 255].

0: Completely transparent
255: Completely opaque

Option "CursorShadowXOffset" "*integer*"

This option defines the offset, in pixels, that the shadow image will be shifted to the right from the real cursor image. This option is only applicable if "CursorShadow" is enabled.

Default: Option "CursorShadowXOffset" "4"

Integer must be in the range [0, 32].

Option "CursorShadowYOffset" "*integer*"

This option defines the offset, in pixels, that the shadow image will be shifted down from the real cursor image. This option is only applicable if "CursorShadow" is enabled.

Default: Option "CursorShadowYOffset" "2"

Integer must be in the range [0, 32].

Option "HWCursor" "*boolean*"

This option enables ("on") or disables ("off") hardware rendering of the X cursor.

Default: Option "HWCursor" "on"

Option "IgnoreEDID" "*boolean*"

This option disables ("on") or enables ("off") probing of EDIDs from your monitor. Requested modes are compared against values obtained from your monitor EDIDs (if any) during mode validation.

Caution: Some monitors may not provide accurate information about its physical capabilities. However, ignoring the values that the monitor provides (i.e., setting the option to "on") may help to validate a certain mode but *could result in serious problems* and, therefore, should be used with extreme care.

Default: Option "IgnoreEDID" "off" means EDIDs *are used* to validate modes.

Option “NoDDC” “*boolean*”

Synonym for Option “IgnoreEDID” “boolean”.

Option “NoLogo” “*boolean*”

Default: `Option "NoLogo" "off"` *enables* the NVIDIA logo splash screen when the X server is started.

To disable the splash screen during X start-up, specify the `"on"` setting.

Option “NvAGP” “*integer*”

This option configures AGP support; *integer* value can be one of:

0: Disable AGP.

1: Use the NVIDIA internal AGP support, if possible.

2: Use AGPGART, if possible.

3: Use any AGP support (try AGPGART, then the NVIDIA AGP).

Default: `Option "NvAGP" "3"`

Note: The default was **1** through NVIDIA Linux driver version 1.0-1251.

Note: The NVIDIA internal AGP support cannot work if AGPGART is either statically compiled into your kernel or is built as a module and loaded into your kernel. (Some distributions load AGPGART into the kernel during system startup.)

Option “RenderAccel” “*boolean*”

Default: `Option "RenderAccel" "off"` *disables* hardware acceleration of the `"RENDER"` extension.

To enable hardware acceleration of the `"RENDER"` extension, specify `"on"` in the above statement.

Option “SWCursor” “*boolean*”

Default: `Option "SWCursor" "off"` disables software rendering of the X cursor.

To enable software rendering of the X cursor, specify `"on"` in the above statement.

Option “UseEdidFreqs” “*boolean*”

Default: `Option "UseEdidFreqs" "off"` means that the frequencies provided by the EDID provided by the display device will *not* be used.

When `"on"` is specified in the above line, the X server uses the `HorizSync` and `VertRefresh` ranges given in a display device's EDID, if any. EDID-supplied range information will override the `HorizSync` and `VertRefresh` ranges specified in the “`Monitor`” section of the XFree86 configuration file.

If a display device does not provide an EDID, or the EDID does not specify an HSync or VRefresh range, then the X server will default to the `HorizSync` and `VertRefresh` ranges specified in the “`Monitor`” section of the XFree86 configuration file.

XF86Config TV Options: Introduced in Release 10

Note: The `"TVOverscan"` option is a new feature in Release 40. The other options were introduced in Release 10 of the NVIDIA Linux driver.

Enabling and Configuring TV

This sections contains the following topics:

A TV display can be connected to an NVIDIA GPU-based graphics card with a TV-Out (S-Video) connector so that the TV functions as any other display device, such as a CRT or digital flat panel (DFP). The TV can be used by itself, or (on appropriate graphics cards) in conjunction with another display device in a TwinView configuration. (For TwinView features, see the section “Configuring a Laptop Computer” on page 64).

If a TV is the only display device connected to your graphics card, the TV will be used as the primary display when you start up your computer — that is, the console will come up on the TV just as if it were a CRT.

XF86Config File Settings

To use your TV with X, note the following settings in your XF86Config file:

- `VertRefresh` and `HorizSync` values in the "`Monitor`" section of the XF86Config file; confirm that the values are appropriate for your television. Values are generally:
  - `HorizSync 30-50`
  - `VertRefresh 60`
- Valid modes for TV in the "`Screen`" section of the XF86Config file are:
  - `640x480`
  - `800x600`
  - `1024x768` (if the TV encoder on your graphics card is a BrookTree 871)

  Note: Your XFree86 log file should indicate the encoder that you have; locate the line: '`(--) NVIDIA(0): TV Encoder detected as`'
- "`TV Standard`" is an option that you need to *add* to the "`Screen`" section of your XF86Config file. *See* "Option "TVStandard" "string"" on page 56.
- Use the "`ConnectedMonitor`" option ("Option "ConnectedMonitor" "string"" on page 61) to tell X to use the TV for display. This option is required *only* if your TV is not detected by the graphics card, or you normally use a CRT or digital flat panel as your start-up display but want to redirect X to use the TV.

  Example command line in your XF86Config file:

  ```
  Option "ConnectedMonitor" "TV"
  ```
- "`TVOutFormat`" is an option that you can use to force S-Video or Composite output.

Option "TVOutFormat" "*string*"

Add the "`TVOutFormat`" option to force S-Video or Composite output.

Note: Without this option, the driver auto-detects the output format but may not do so correctly. The output format can be forced with *one* of these options:

```
Option "TVOutFormat" "SVIDEO"
```

or

```
Option "TVOutFormat" "COMPOSITE"
```

Option "TVOverscan" "*integer*"

Note: This option is new in the NVIDIA Linux Release 40 driver and is currently only applicable to NVIDA GeForce4 or newer NVIDIA GPUs with either NVIDIA or Conexant TV encoders.

`"TVOverscan"` can be used to enable Overscan, where supported.

Default: `"TVOverscan" "0.0"` disables TVOverscan.

Values are decimal values in the range of 0.0 to 1.0, where:

- **1.0** means overscan as much as possible — make the image as large as possible.
- **0.0** means disable overscanning — make the image as small as possible.

Option "TVStandard" "*string*"

Add the `"TVStandard"` option to the "`Screen`" section of your XF86Config file. Replace "*string*" with a valid TV output format, as listed in Table 4.1. A sample line in the XF86Config file is:

```
Option "TVStandard" "NTSC-M"
```

If you don't specify a TVStandard, or you specify an invalid value, the *default* "`NTSC-M`" value is used.

Default: `"NTSC-M"`

Note: If your country is not in the list of countries, select the country closest to your location.

Table 4.1 TV Output Formats by Country

TV Output Format	Country Where Used
`PAL-B`	Belgium, Denmark, Finland, Germany, Guinea, Hong Kong, India, Indonesia, Italy, Malaysia, The Netherlands, Norway, Portugal, Singapore, Spain, Sweden, and Switzerland
`PAL-D`	China and North Korea
`PAL-G`	Denmark, Finland, Germany, Italy, Malaysia, The Netherlands, Norway, Portugal, Spain, Sweden, and Switzerland
`PAL-H`	Belgium
`PAL-I`	Hong Kong and United Kingdom
`PAL-K1`	Guinea
`PAL-M`	Brazil
`PAL-N`	France, Paraguay, and Uruguay
`PAL-NC`	Argentina

Table 4.1 TV Output Formats by Country (continued)

TV Output Format	Country Where Used
NTSC-J	Japan
NTSC-M	Canada, Chile, Colombia, Costa Rica, Ecuador, Haiti, Honduras, Mexico, Panama, Puerto Rico, South Korea, Taiwan, United States of America, and Venezuela

XF86Config TwinView Options: Introduced in Release 6

Enabling and Configuring TwinView

This section contains the following topics:

To enable TwinView, you must specify the following options in the "Screen" section of your XF86Config file.

```
Option "TwinView"
Option "SecondMonitorHorizSync"     "<hsync range(s)>"
Option "Second Monitor              "<vrefresh range(s)>"
VertRefresh"
Option "MetaModes"                  "<list of metamodes>"
```

You may also use any of the following options, though they are *not required*.

```
Option                      "<relationship of head
"TwinViewOrientation"       (display) 1 to head (display)
                            0>"
Option "ConnectedMonitor"   "<list of connected display
                            devices>"
```

Descriptions of the options are provided in the sections that follow.

Option “TwinView” “*boolean*”

Default: `Option "TwinView" "off"` means TwinView is disabled.

Note: To enable TwinView, enable this option with the `"on"` setting in the above line; otherwise all other TwinView related options are ignored.

Option “SecondMonitorHorizSync” “*range(s)*”

Default: None

This option is similar to the `"HorizSync"` entry in the “`Monitor`” section of your XF86Config file, but applies to the second monitor when using TwinView.

According to the XF86Config man page, the “ranges” may be a comma-separated list of distinct values and/or ranges of values, where a range is given by two distinct values separated by a dash. The `HorizSync` value is given in KHz.

If you trust the EDID of your display device, you may want to use the `"UseEdidFreqs"` option instead of this option. (For details, *see* “Option “UseEdidFreqs” “boolean”” on page 54.)

Option “Second Monitor VertRefresh” “*range(s)*”

Default: None

This option is similar to the `"VertRefresh"` entry in the “`Monitor`” section of your XF86Config file, but applies to the second monitor when using TwinView.

According to the XF86Config man page, the “ranges” may be a comma-separated list of distinct values and/or ranges of values, where a range is given by two distinct values separated by a dash. The `VertRefresh` value is given in Hz.

If you trust the EDID of your display device, you may want to use the `"UseEdidFreqs"` option instead of this option. For details, *see* “Option “UseEdidFreqs” “boolean”” on page 54.

Option “MetaModes” “*string*”

Default: None

A single MetaMode describes the mode that should be used on each display device at a given time. Multiple MetaModes list the combinations of modes and the sequence in which they should be used. When the NVIDIA driver communicates the available modes to X, it is really the minimal bounding box of the MetaMode that is communicated, while the “per display device” mode is kept internal to the NVIDIA driver. In MetaMode syntax, modes within a MetaMode are separated by a comma and multiple MetaModes are separated by semicolons. For example:

```
“<mode name 0>, <mode name 1>; <mode name 2>, <mode name 3>; ...”
```

Where “`<mode name 0>`” is the name of the mode to be used on display device 0 concurrently with “`<mode name 1>`” used on display device 1. A mode switch will then cause “`<mode name 2>`” to be used on display device 0 and “`<mode name 3>`” to be used on display device 1.

- **An actual MetaMode entry** from the XF86Config TwinView sample configuration file is as follows:

  ```
  Option “MetaModes” “1280x1024,1280x1024; 1024x768,1024x768”
  ```

- **If you do not want a display device to be active for a certain MetaMode**, you can use the mode name “`NULL`”, or simply omit the mode name entirely, as follows:

  ```
  “1600x1200, NULL; NULL, 1024x768”
  ```

  or

  ```
  “1600x1200; , 1024x768”
  ```

- **Optionally, mode names can be followed by “offset” information** to control the positioning of the display devices within the virtual screen space; for example:

  ```
  “1600x1200 +0+0, 1024x768 +1600+0; ...”
  ```

  Offset descriptions follow the conventions used in the X “`-geometry`” command line option; i.e. both positive and negative offsets are valid, though negative offsets are only allowed when a virtual screen size is explicitly given in the XF86Config file.

  When no offsets are given for a MetaMode, the offsets are computed following the value of the “`TwinViewOrientation`” option. (*See* “Option “TwinViewOrientation” “string”” on page 61.)

Note: If offsets are given for any one of the modes in a single MetaMode, then offsets are expected for all modes within that single MetaMode; in such a case, offsets are assumed to be +0+0 when not given.

When not explicitly given, the virtual screen size is computed as the bounding box of all MetaMode bounding boxes.

MetaModes with a bounding box larger than an explicitly given virtual screen size are discarded.

- **A MetaMode string can be further modified with a** "**panning domain" specification**; for example:

  `"1024x768 @1600x1200, 800x600 @1600x1200"`

  A panning domain is the area in which a display device's viewport is panned to follow the mouse.

  Panning takes place on two levels under TwinView:

  - *First*, an individual display device's viewport is panned within its panning domain, as long as the viewport is contained by the bounding box of the MetaMode.
  - *Second***,** once the mouse leaves the bounding box of the MetaMode, the entire MetaMode (i.e., all display devices) is panned to follow the mouse within the virtual screen.

  Note: The panning domains of the display devices default to being clamped to the position of the display devices' viewports. Therefore, the default functionality is that viewports remain "locked" together and only perform the second type of panning.

  The most beneficial use of panning domains is to eliminate dead areas, which are regions of the virtual screen that are inaccessible due to display devices with different resolutions. For example:

  - Specifying `"1600x1200, 1024x768"` produces an inaccessible region below the 1024x768 display.
  - Specifying `"1600x1200, 1024x768 @1024x1200"` as a panning domain for the second display device: provides access to that dead area by allowing you to pan the 1024x768 viewport up and down in the 1024x1200 panning domain.

- **Offsets can be used in conjunction with panning domains** to position the panning domains in the virtual screen space. The offset describes the panning domain and only affects the viewport in that the viewport must be contained within the panning domain. For example, the following line describes two modes, each with a panning domain width of 1900 pixels where the second display is positioned below the first:

  `"1600x1200 @1900x1200 +0+0, 1024x768 @1900x768 +0+1200"`

Note: If a MetaMode string is not specified, then the X driver uses the modes listed in the relevant "`Display`" subsection of the XF86Config file, attempting to place matching modes on each display device.

Option "TwinViewOrientation" "*string*"

Default: `Option "TwinViewOrientation" "RightOf"`

Valid values are:

- `"RightOf"`
- `"LeftOf"`
- `"Above"`
- `"Below"`
- `"Clone"`

This option controls the positioning of the second display device relative to the first within the virtual X screen, when offsets are not explicitly given in the MetaModes.

Note: Under the TwinView Clone option, both display devices are assigned offsets of 0,0.

Option "ConnectedMonitor" "*string*"

See "Option "ConnectedMonitor" "string"" on page 61 for application to TwinView.

OpenGL Environment Variable Settings

The following topics are discussed in this section:

Full-Scene Antialiasing (FSAA)

Note: FSAA is supported on GeForce2 (Quadro2) family and newer NVIDIA GPUs.

Anti-aliasing is a technique used to smooth the edges of objects in a scene to reduce the jagged "stairstep" effect that sometimes appears in images.

Several antialiasing methods are available and you can select among them by setting the `__GL_FSAA_MODE` environment variable. Table 4.2 describes the possible values for `__GL_FSAA_MODE` and their effect on various NVIDIA GPUs.

Table 4.2 Values for the `__GL_FSAA_MODE` Environment Variable

Value	GeForce, GeForce2, Quadro, and Quadro2 Pro	GeForce4 MX, GeForce4 4*xxx* Go, Quadro4 380/550/ 580 XGL, Quadro NVS	GeForce3, Quadro DCC, GeForce4 Ti, GeForce4 4200 Go, and Quadro4 700, 750, 780, 900, 980 XGL	GeForce FX and Quadro FX
0	FSAA is disabled.	FSAA is disabled.	FSAA is disabled.	FSAA is disabled.
1	FSAA is disabled.	2x Bilinear Multisampling	2x Bilinear Multisampling	2x Bilinear Multisampling
2	FSAA is disabled.	2x Quincunx Multisampling	2x Quincunx Multisampling	2x Quincunx Multisampling
3	1.5 x 1.5 Supersampling	FSAA is disabled.	FSAA is disabled.	FSAA is disabled.
4	2 x 2 Supersampling	2 x 2 Supersampling.	4x Bilinear Multisampling	4x Bilinear Multisampling
5	FSAA is disabled.	FSAA is disabled.	4x Gaussian Multisampling	4x Gaussian Multisampling
6	FSAA is disabled.	FSAA is disabled.	2x Bilinear Multisampling by 4x Supersampling	2x Bilinear Multisampling by 4x Supersampling
7	FSAA is disabled.	FSAA is disabled.	FSAA is disabled.	4x Bilinear Multisampling by 4x Supersampling

Notes

- "2x Bilinear Multisampling by 4x Supersampling" and "4x Bilinear Multisampling by 4x Supersampling" are not available when using the UBB configuration option. (For details on using the UBB option, see "Option "UBB" "boolean"" on page 48.)

- Increasing the number of samples taken during FSAA rendering may decrease performance.
- When FSAA is enabled (i.e., the `__GL_FSAA_MODE` environment variable is set to a value that enables FSAA and a multisample visual is chosen), the rendering may be corrupted when resizing the window.

Anisotropic Texture Filtering

Automatic anisotropic texture filtering can be enabled by setting the environment variable `__GL_DEFAULT_LOG_ANISO`.

Table 4.3 describes the possible values for `__GL_DEFAULT_LOG_ANISO` and their effect on various NVIDIA GPUs.

Table 4.3 Values for the `__GL_DEFAULT_LOG_ANISO` Environment Variable

Value	GeForce/GeForce2/GeForce4 MX Description	GeForce3/GeForce4 Ti/GeForce FX Description
0	No anisotropic filtering.	No anisotropic filtering.
1	Enable automatic anisotropic filtering.	Low anisotropic filtering
2		Medium anisotropic filtering
3		Maximum anisotropic filtering.

VBLANK Synchronizing

Note: This option is available only on NVIDIA “3D” products, such as those in the GeForce/GeForce2/GeForce3 and equivalent workstation family.

Setting the environment variable `__GL_SYNC_TO_VBLANK` to a non-zero value forces `glXSwapBuffers` to synchronize to your monitor's vertical refresh rate. This implies that the code performs a swap only during the vertical blanking period on GeForce or newer GPUs, which excludes those in the TNT and TNT2 families.

Disabling CPU-Specific Features

- Setting the following environment variable:

  `__GL_FORCE_GENERIC_CPU`

  to a non-zero value will inhibit the use of CPU-specific features such as MMX, SSE, or 3DNOW!.
- Use of this environment variable may result in performance loss.

- Use of this environment variable may be useful in conjunction with software such as the Valgrind memory debugger.

Configuring a Laptop Computer

This section contains the following topics:

Standard Functionality

Installation and configuration of the NVIDIA Accelerated Linux Driver Set on a laptop computer is the same as for any desktop environment, with a few minor exceptions, listed below.

Starting with the NVIDIA Linux driver version 1.0-2802, information about the internal flat panel for use in initializing the display is, by default, generated on the fly from data stored in the Video BIOS.

- This information can be disabled by setting the "`SoftEDIDs`" kernel option to "`0`".
- If "`SoftEDIDs`" is turned off, then hardcoded data will be chosen from a table, based on the value of the "`Mobile`" kernel option.
- The "`Mobile`" kernel option can be set to any of the following values:

Value	Description
`0xFFFFFFFF`	Lets the kernel module auto detect the correct value
`1`	Dell laptops
`2`	Toshiba laptops that are *not* Compal
`3`	All other laptops
`4`	Compal Toshiba laptops
`5`	Gateway laptops

Again, the "`Mobile`" kernel option is only needed if "`SoftEDIDs`" is disabled. When the "`Mobile`" kernel option is used, it is usually safest to let the kernel module auto-detect the correct value, which is the default functionality.

If you need to alter either of these options, you can use any of the following methods:

- Edit `os-registry.c` in the `'usr/src/nv/'` directory of the `.run` file.
- Set the value on the `modprobe` command line. For example:

  ```
  modprobe nvidia NVreg_SoftEDIDs=0 NVreg_Mobile=3
  ```

- Add an `options` line to your module configuration file (usually `'/etc/modules.conf'`). For example:

  ```
  options nvidia NVreg_Mobile=5
  ```

TwinView Functionality

All mobile NVIDIA GPUs support TwinView.

TwinView on a laptop (mobile computer) can be configured in the same way as on a desktop computer. (*See* "Configuring a Laptop Computer" on page 64.)

In a TwinView configuration, when using the laptop's internal flat panel and an external CRT:

- The CRT is the primary display device. Specify the CRT's `HorizSync` and `VertRefresh` values in the "`Monitor`" section of your XF86Config file.
- The flat panel is the secondary display device. Specify the flat panel's `HorizSync` and `VertRefresh` values using the `"SecondMonitorHorizSync"` and `"SecondMonitorVertRefresh"` options. (*See* "Option "SecondMonitorHorizSync" "range(s)"" on page 58 and "Option "Second Monitor VertRefresh" "range(s)"" on page 58.)

  You can also use the `"UseEdidFreqs"` option to obtain the `HorizSync` and `VertRefresh` values from the EDID of each display device, in which case, you don't have to set these values in your XF86Config file.

  Note: Use this method of obtaining `HorizSync` and `VertRefresh` values *only* if you if you trust the reported EDIDs of your display device. For details, *see* "Option "UseEdidFreqs" "boolean"" on page 54.

Using Hot Keys to Switch Display Devices

Laptop computers using GeForce2 Go (or higher level product) can react to an LCD/CRT hot key event, toggling between each of the connected display devices and each possible combination of the connected display devices

Note: Only two display devices may be active at a time.

Note: TwinView, as configured in your XF86Config file, and hot key functionality are mutually exclusive; that is, if you enable TwinView in your XF86Config file, then the NVIDIA X driver ignores hot key events.

Besides TwinView, mobile NVIDIA products can also react to an LCD/CRT hot key event, toggling between each of the connected display devices and each possible combination of the connected display devices; note that only two display devices may be active at a time.

The hot key functionality lets you dynamically connect and remove display devices to/from your laptop and hot key to them without restarting X.

When using this feature, it is a good idea to use the "`UseEdidFreqs`" option so that the `HorizSync` and `VertRefresh` values for each display device can be retrieved from its EDID. Otherwise, the "`Monitor`" section of the XF86Config file will be interpreted differently with each hot key event.

When X is started, or when a change is detected in the list of connected display devices, a new hot key sequence list is constructed, which shows the display devices to be used with each hot key event.

When a hot key event occurs, then the next hot key state in the sequence is chosen. Each mode requested in the XF86Config file is validated against each display device's constraints, and the resulting modes are made available for that display device.

If multiple display devices are to be active at once, then the modes from each display device are paired together. If an exact match (same resolution) is not found, then the closest fit is used and the display device with the lower resolution is panned within the resolution of the other display device.

When VT-switching away from X, the VGA console will always be restored on the display device on which it was present when X was started. Similarly, when VT-switching back into X, the same display device configuration will be used

Non-Standard Modes on LCD Displays

Some users have had difficulty programming a 1400x1050 mode (the native resolution of some laptop LCDs). In version 4.0.3, XFree86 added several 1400x1050 modes to its database of default modes, but if you're using an older version of XFree86, here is a modeline that you can use.

```
# -- 1400x1050 --

# 1400x1050 @ 60Hz, 65.8 kHz hsync

Modeline  "1400x1050" 129   1400  1464  1656  1960

                            1050  1051  1054  1100  +HSync   +VSync
```

Known Laptop Computer Issues

The following issues currently exist when NVDIA drivers are used with laptop computers:

- Power Management is currently not supported on laptop computers.
- The TwinView feature is currently not supported on Toshiba Satellite 2800 series laptops.
- LCD/CRT hot key switching is not currently functioning on any Toshiba laptops, with the exception of the Toshiba Satellite 3000 series.
- The video overlay only works on the first display device on which you started X.

  For example, if you perform these steps, the video will not appear on the second display device:

  a. Start X on the internal LCD.

  b. Run a video application that uses the video overlay (i.e., the “Video Overlay” adaptor advertised through the XV extension).

  c. Hot key switch to add a second display device.

  Note: To work around this problem, you can either configure the video application to use the “Video Blitter” adaptor advertised through the XV extension (this is always available) or hot key switch to the display device on which you want to use the video overlay *before* starting X.

CHAPTER 5

FREQUENTLY ASKED QUESTIONS, TROUBLESHOOTING, & OTHER RESOURCES

This chapter contains the following major sections:

NVIDIA Installer: Common Questions and Problems

How do I extract the contents of the `.run` file without actually installing the driver?

Enter the following command to create the directory name `NVIDIA-Linux-x86-1.0-4363`, which contains the uncompressed contents of the `.run` file.

```
sh NVIDIA-Linux-x86-1.0-4363.run --extract-only.
```

How can I see the source code to the kernel interface layer?

The source files to the kernel interface layer are in the `usr/src/nv` directory of the extracted `.run` file. To access these sources, run the following commands:

```
sh NVIDIA-Linux-x86-1.0-4363.run --extract-only
cd NVIDIA-Linux-x86-1.0-4363/usr/src/nv/
```

I just upgraded my kernel, and now the NVIDIA kernel module will not load. What's wrong?

The kernel interface layer of the NVIDIA kernel module must be compiled specifically for the configuration and version of your kernel. If you upgrade your kernel, then the simplest solution is to reinstall the driver.

Advanced Use: You can install the NVIDIA kernel module for a non-running kernel (for example, in a situation where you just built and installed a new kernel but haven't yet rebooted) with a command line such as the following:

```
sh NVIDIA-Linux-x86-1.0-4363.run --kernel-name='KERNEL_NAME'
```

where `'KERNEL_NAME'` is what `` `uname -r` `` would report if the target kernel were running.

Why doesn't NVIDIA provide RPMS anymore?

Not every Linux distribution uses RPM and NVIDIA wanted a single solution that would work across all Linux distributions. As indicated in the NVIDIA Software License, Linux distributions are free to repackage and redistribute the NVIDIA Linux driver in any desired format.

`nvidia-installer` doesn't work on my computer. How can I install the driver contained in the `.run` file?

To install the NVIDIA driver contained in the `.run` file without using `nvidia-installer`, you can use the included Makefile with the following commands:

```
sh ./NVIDIA-Linux-x86-1.0-4363.run --extract-only
cd NVIDIA-Linux-x86-1.0-4363
make install
```

Note: This method of installation is *not recommended* and is only provided as a last resort in case the `nvidia-installer` does not work correctly on your computer.

Can the nvidia-installer use a proxy server?

Yes. Because the FTP support in nvidia-installer is based on snarf, it will honor the `FTP_PROXY`, `SNARF_PROXY` and `PROXY` environment variables.

Where can I find the source code for the nvidia-installer utility?

The nvidia-installer utility is released under the **GPL** (General Public License). The latest source code for it is available at:

`ftp://download.nvidia.com/XFree86/nvidia-installer/`

Miscellaneous: Common Questions and Problems

Where should I start when diagnosing display problems?

One of the most useful tools for diagnosing problems is the `XFree86` log file in `/var/log`; the file is named:

```
/var/log/XFree86.<#>.log
```

where `<#>` is the server number -- usually "0" (zero).

Note the following about this log file:

- Lines that begin with "`(II)`" are informational.
- Lines that begin with "`(WW)`" are warnings.
- Lines that begin with "`(EE)`" are errors.
- Verify that the correct configuration file (i.e., the configuration file you are editing) is being used; find the line that begins with:

  ```
  (==) Using config file:
  ```

- **Important:** Verify that the NVIDIA driver is being used instead of the "nv" or "vesa" driver by locating the line:

  ```
  (II) LoadModule: "nvidia"
  ```

  Lines from the driver should begin with: `(II) NVIDIA(0)`

How can I increase the amount of data printed in the XFree86 log file?

By default, the NVIDIA X driver prints relatively few messages to `stderr` and the `XFree86` log file.

If you need to troubleshoot, it may be helpful to enable more verbose output by using the XFree86 command line options "`-verbose`" and "`-logverbose`",

which can be used to set the verbosity level for the `stderr` and log file messages, respectively.

The NVIDIA X driver outputs more messages when the verbosity level is at or above 5. (XFree86 defaults to verbosity level 1 for `stderr` and level 3 for the log file.) Therefore, to enable verbose messaging from the NVIDIA X driver to both the log file and `stderr`, you can start X by using the following command:

```
startx -- -verbose 5 -logverbose 5
```

My X server fails to start and my XFree86 log file contains the error: "`(EE) NVIDIA(0): Failed to initialize the NVIDIA kernel module!`"

Nothing will work if the NVIDIA kernel module does not function properly.

- If you see an error line in the X log file such as: "`EE) NVIDIA(0): Failed to initialize the NVIDIA kernel module!`" then, most likely, a problem exists with the NVIDIA kernel module. Follow these troubleshooting guidelines:

  a. Verify that, if you installed from RPM, that the RPM was built specifically for the kernel you are using.

  b. Check that the module "`/sbin/lsmod`" is loaded. If it is not loaded, try loading it explicitly with "`insmod`" or "`modprobe`". (Be sure to exit the X Server before installing a new kernel module.)

- If you receive errors about unresolved symbols, the kernel module has most likely been built using header files for a different kernel revision than what you are running.

  You can explicitly control the kernel header files that are used when building the NVIDIA kernel module with the "`--kernel-include-dir`" option.

  See the description of "`--advanced-options`" in ".Run File Command Line Options" on page 33.

  Note: The convention for the location of kernel header files is in a state of transition, as is the location of kernel modules. If the kernel module fails to load properly, `modprobe/insmod` may be attempting to load an older kernel module, assuming you've upgraded. Changing to the directory with the new kernel module and using the command "`insmod ./nvidia.o`" may help.

- Another cause of the problem could be that the `/dev/nvidia*` device files may be missing.

- Finally, the NVIDIA kernel module may print error messages indicating a problem. To view these messages, check /`var/log/messages` or where

syslog has been directed to place kernel messages. These messages are prepended with "NVRM".

My X Server starts but why do OpenGL applications terminate immediately?

Most likely a problem exists with other libraries that are in the way or the presence of obsolete symlinks. (*See* "After Completing Driver Installation" on page 36.)

Note: You may be able to fix this problem by rerunning `ldconfig`.

1. Use `xdpyinfo` to verify that the following (correct) extensions are present:

   ```
   GLX
   NV-GLX
   NVIDIA-GLX
   ```

   If these three extensions are not present, then there is most likely a problem with the loading of the `glx` module or its inability to implicitly load `GLcore`.

2. Check your XF86Config file to verify that you are loading `glx`. See "OpenGL and GLX Header Files" on page 86.

3. If your XF86Config file is correct, then check the XFree86 log file for warnings and errors pertaining to `glx`.

4. Also check that all of the necessary symlinks are in place. *See* "About Installation of Libraries" on page 87.

Why does installing the NVIDIA kernel module result in an error message such as:

```
#error Modules should never use kernel-headers system headers
#error but headers from an appropriate kernel-source
```

You need to install the source for the Linux kernel. In most cases, your can fix this problem by installing the kernel-source package for your distribution.

Why do OpenGL applications exit with this error message?

```
Error: Could not open /dev/nvidiactl because the permissions are
too restrictive. Please see the TROUBLESHOOTING section of /usr/
share/doc/NVIDIA_GLX-1.0/README for steps to correct.
```

It is likely that a security module for the **Linux-PAM** (Pluggable Authentication Modules for Linux) system may be changing permissions on the NVIDIA device files.

To correct the problem, it is recommended that you disable this security feature by following the steps below. Different Linux distributions have different files to control. Please consult with your distributor for the correct method of disabling this security feature.

For example:

1. If your system has the file `'/etc/security/console.perms'`, edit the file by removing the line that starts with `<dri>`.
2. If your system has the file `'/etc/logindevperms'`, edit the file by removing the line that lists `'/dev/nvidiactl'`.
3. The above steps will prevent the PAM security system from modifying the permissions on the NVIDIA device files.
4. Reset the permissions on the device files back to their original permissions and owner. You can use the following command:

```
chmod 0666 /dev/nvidia* chown root /dev/nvidia*
```

Why do OpenGL applications crash and print out the following warning:

```
WARNING: Your system is running with a buggy dynamic loader. This
may cause crashes in certain applications. If you experience
crashes you can try setting the environment variable
__GL_SINGLE_THREADED. For more information please consult (sec-
04) TROUBLESHOOTING in the file /usr/share/doc/NVIDIA_GLX-1.0/
README.
```

The dynamic loader on your system has a software problem, which will cause applications to crash under these conditions:

- The application is linked with the pthreads library *and*
- The application calls the `dlopen()` function to open the library `libGL.so` multiple times during its execution *and*
- The system has a version of the Linux loader with the software problem.

This problem is present in older versions of the dynamic loader. Distributions that shipped with this loader include but are not limited to Red Hat Linux 6.2 and Mandrake Linux 7.1.

If the crashing application is single-threaded then setting the environment variable `__GL_SINGLE_THREADED` to any value will prevent the crash.

- In the "`bash`" shell, use the "`export __GL_SINGLE_THREADED`" command**.**

- In "csh" and derivatives, use the "setenv __GL_SINGLE_THREADED" command.

Previous releases of the NVIDIA Accelerated Linux Driver Set attempted to work around this problem; however, the workaround caused problems with other applications and was removed after version 1.0-1541.

Why does Quake3 crash when changing video modes?

You are probably experiencing the problem described above. Check the text output for the WARNING message shown in the previous page.

To resolve the problem, *before* running Quake3, set "__GL_SINGLE_THREADED" as described above.

My system runs but seems unstable.

Your stability problems may be AGP-related. (*See* "XF86Config Options: Configuring AGP" on page 38 for details.)

The kernel module doesn't get loaded dynamically when X starts; I always have to do "modprobe nvidia" first.

Verify that the line "alias char-major-195 nvidia" appears in your module configuration file, which usually is one of the following:

- /etc/conf.modules
- /etc/modules.conf
- /etc/modutils/alias

For details, consult the documentation that came with your distribution.

I can't build the NVIDIA kernel module, or I can build the NVIDIA kernel module but modprobe/insmod fails to load the module into my kernel?

These problems are generally caused by the build using the wrong kernel header files; i.e., header files for a different kernel version than the one you are running.

The previous convention was that kernel header files are stored in:

```
/usr/include/linux/
```

but that is being deprecated in favor of:

```
/lib/modules/`uname -r`/build/include.
```

The `nvidia-installer` can determine the location on your system. However, if you encounter a problem, you can force the build to use certain header files by using the "`-kernel-include-dir`" option.

Of course, for this process to work, you need the appropriate kernel header files installed on your system.

Note: *Consult the documentation that came with your distribution;* some distributions do not install the kernel header files by default, or they install headers that do not work properly with the kernel you are running.

Why do OpenGL applications run so slowly?

The application is probably using a different library (that still remains on your system) instead of the NVIDIA-supplied OpenGL library.

For details, *see* "After Completing Driver Installation" on page 36.

Why are there are problems running Quake2.

Quake2 requires minor setup to get it started.

1. First, in the Quake2 directory, the installation program creates a symlink called "`libGL.so`" that points to "`libMesaGL.so`." *Remove* or *rename* this symlink.
2. Then, to run Quake2 in OpenGL mode, use this command:

```
quake2 +set vid_ref glx +set gl_driver libGL.so
```

   Quake2 does not seem to support any kind of full-screen mode but you can run your X Server at the resolution on which Quake2 runs in order to emulate full-screen mode.

There are problems running Heretic II. What can I do?

Heretic II also installs, by default, a symlink called libGL.so in the application directory. You can remove or rename this symlink, since the system will then find the default `libGL.so`, which the NVIDIA drivers install in `'/usr/lib'`. From within Heretic II, you can then set your render mode to OpenGL in the video menu.

A patch is also available for Heretic II from lokigames at this location:

```
http://www.lokigames.com/products/heretic2/updates.php3
```

Where can I get `gl.h` or `glx.h` so I can compile OpenGL programs?

Most systems come with these headers installed. However, NVIDIA has provided its own `gl.h` and `glx.h` files in case your system does not have them or in case you want to develop OpenGL applications that use the new OpenGL extensions.

These files are installed by default by `nvidia-installer`.

You can override the default by passing the "`--no-opengl-headers`" option to "`NVIDIA-Linux-x86-1.0-4363.run`" during driver installation.

Can I receive E-mail notification of new NVIDIA Accelerated Linux Driver Set releases?

Yes. Complete the form at this site:

```
http://www.nvidia.com/view.asp?FO=driver_update
```

Why does my system hang when VT-switching if I have `rivafb` enabled?

Currently, you cannot use both `rivafb` and the NVIDIA kernel module simultaneously due to software limitations.

Note: In general, using two independent software drivers to drive the same piece of hardware *is not recommended*.

Compiling the NVIDIA kernel module gives the following error:

```
You appear to be compiling the NVIDIA kernel module with a
compiler different from the one that was used to compile the
running kernel. This may be perfectly fine, but there are cases
where this can lead to unexpected behavior and system crashes.

If you know what you are doing and want to override this check,
you can do so by setting IGNORE_CC_MISMATCH.

In any other case, set the CC environment variable to the name
of the compiler that was used to compile the kernel.
```

You should compile the NVIDIA kernel module with the same compiler version that was used to compile your kernel. Some Linux kernel data structures are dependent on the version of gcc used to compile them; for example, in `include/linux/spinlock.h`:

```
.....
* Most gcc versions have a nasty bug with empty initializers.
*/
#if (__GNUC__ > 2)
```

```
typedef struct { } rwlock_t;
#define RW_LOCK_UNLOCKED (rwlock_t) { }
#else
typedef struct { int gcc_is_buggy; } rwlock_t;
#define RW_LOCK_UNLOCKED (rwlock_t) { 0 }
#endif
```

If the kernel is compiled with gcc 2.*x*, but gcc 3.x is used when the kernel interface is compiled (or vice versa), the size of `rwlock_t` will vary and internal function calls such as `ioremap` will fail.

To verify the version of gcc that was used to compile your kernel, you can examine the output of:

```
cat /proc/version
```

To verify the version of gcc that is currently in your `$PATH`, you can examine the output of:

```
gcc -v
```

Why does X fail with error "`Failed to allocate LUT context DMA`"?

This is one of the possible consequences of compiling the NVIDIA kernel interface with a different gcc version than used to compile the Linux kernel. (See previous answer for reference).

What is the NVIDIA policy towards development series Linux kernels?

NVIDIA does not officially support development series kernels. However, all the kernel module source code that interfaces with the Linux kernel is available in the `'usr/src/nv/'` directory of the `.run` file.

NVIDIA encourages members of the Linux community to develop patches to these source files to support development series kernels. A "`Google.com`" search will most likely yield several community-supported patches.

I recently updated various libraries on my system using my Linux distributor's update utility. Now the NVIDIA graphics driver no longer works.

Conflicting libraries may have been installed by your distribution's update utility.

For further details on how to diagnose this problem, see "After Completing Driver Installation" on page 36.

"`rpm --rebuild`" results in the error "`unknown option`".

Recent versions of rpm no longer support the "`--rebuild`" option. If you have such a version of rpm, use the following command instead:

```
rpmbuild --rebuild
```

The `rpmbuild` executable is provided by the rpm-build package.

I am using either nForce or nForce2 internal graphics, and I see warnings such as the following in my `XFree86.0.log` file:

```
Not using mode "1600x1200" (exceeds valid memory bandwidth
usage)
```

Integrated graphics have stricter memory bandwidth limitations that restrict the resolution and refresh rate of the modes you request.

To work around this limitation, you can reduce the maximum refresh rate by lowering the higher value of the "`VertRefresh`" range in the "`Monitor`" section of your XF86Config file.

Note: Though *not recommended*, you can disable the memory bandwidth test with the "`NoBandWidthTest`" XF86Config file option.

I've rebuilt the NVIDIA kernel module, but when I try to insert it, a message indicates that I have unresolved symbols.

Unresolved symbols are most often caused by a mismatch between your kernel sources and your running kernel. They must match for the NVIDIA kernel module to build correctly. Please make sure your kernel sources are installed and configured to match your running kernel.

How do I know whether I have my kernel sources installed?

If you are running on a distro that uses RPM (Red Hat Linux, Mandrake, SuSE, etc), then you can use RPM to indicate whether you have the kernel sources installed. Follow this procedure:

At a shell prompt, type: `` `rpm -qa | grep kernel` `` and look at the output.

You should see a package that corresponds to your kernel (often named something like kernel-2.4.18-3) and a kernel source package with the same version (often named something like kernel-source-2.4.18-3).

- If none of the lines seem to correspond to a source package, then you'll probably need to install it.

- If the versions listed mismatch (ex: kernel-2.4.18-10 vs. kernel-source-2.4.18-3), then you'll need to update the kernel-source package to match the installed kernel.
- If you have multiple kernels installed, you need to install the kernel-source package that corresponds to your *running* kernel (or make sure your installed source package matches the running kernel). You can do this by looking at the output of '`uname -r`' and matching versions.

Why am I unable to load the NVIDIA kernel module that I compiled for the Red Hat Linux 7.3 2.4.18-3bigmem kernel?

The kernel header files Red Hat Linux distributes for Red Hat Linux 7.3 2.4.18-3bigmem kernel are incorrectly configured. The NVIDIA precompiled kernel module for this kernel can be loaded, but if you want to compile the NVIDIA kernel interface files yourself for this kernel, then you must follow this procedure:

```
cd /lib/modules/`uname -r`/build/
cp configs/kernel-2.4.18-i686-bigmem.config .config
make mrproper oldconfig dep
```

Note: Red Hat Linux ships kernel header files that are simultaneously configured for *all* of their kernels for a particular distribution version. A header file generated at boot time sets up a few parameters that select the correct configuration. *Rebuilding the kernel headers with the above commands will create header files suitable for the Red Hat Linux 7.3 2.4.18-3bigmem kernel configuration only, thus corrupting the header files for the other configurations.*

X takes a long time to start (possibly several minutes).

Most of the startx delay problems we have found are caused by incorrect data in video BIOSs about connected display devices or determining the i2c port to use for detection.You can work around these problems with the XF86Config option `"IgnoreDisplayDevices"`. See "Option "IgnoreDisplayDevices" "string"" on page 42 for details.

Why does X use so much memory?

When measuring any application's memory usage, you must be careful to distinguish between physical system RAM used and virtual mappings of shared resources.

For example, most shared libraries exist only once in physical memory but are mapped into multiple processes. This memory should only be counted once when computing total memory usage.

In the same way, the video memory on a graphics card or register memory on any device can be mapped into multiple processes. These mappings do not consume normal system RAM.

This topic has been frequently discussed on XFree86 mailing lists. See, for example:

`http://marc.theaimsgroup.com/?l=xfree-xpert&m=96835767116567&w=2`

The `` `pmap` `` utility described in the above thread and available here:

```
http://web.hexapodia.org/~adi/pmap.c
```

is a useful tool for distinguishing between types of memory mappings. For example, while `` `top` `` may indicate that X is using several hundred megabytes of memory, the last line of output from `` `pmap` ``:

```
mapped 287020 KB writable/private: 9932 KB shared: 264656 KB
```

indicates that X is really only using roughly 10 MB of system RAM (the "writable/private" value).

Note also that X must allocate resources on behalf of X clients — i.e., the window manager, your web browser, and so on. X memory usage increases as more clients request resources (such as pixmaps) and decreases as you close X applications.

OpenGL applications leak significant amounts of memory on my system!

If your kernel is making use of the `-rmap VM`, the system may be leaking memory due to a memory management optimization introduced in `-rmap14a`.

The `-rmap VM` has been adopted by several popular distributions. The memory leak is known to be present in some of the distribution kernels and has been fixed in `-rmap15e`.

If you suspect that your system is affected by the memory leak, try upgrading your kernel or contact the distribution's vendor for assistance.

Some OpenGL applications (such as Quake3 Arena) crash when I start them on Red Hat Linux 9.0.

Some versions of the `glibc` package (shipped by Red Hat Linux) that support TLS do not properly handle using `dlopen()` to access shared libraries that utilize some TLS models. This problem occurs, for example, when Quake3 Arena uses `dlopen()` on the NVIDIA `libGL` library.

Note: Be sure to obtain at least glibc-2.3.2-11.9, which is available as an update from Red Hat Linux.

I've installed the NVIDIA Linux driver but my "Enable 3D Acceleration" check box is still greyed out!

Most distribution-provided configuration applets are not aware of the NVIDIA Linux driver and, consequently, will not update when you install the driver. Your NVIDIA Linux driver, if it has been installed properly, should function without problems.

Where can I find the tarballs?

Visit `ftp://download.nvidia.com/XFree86/1.0-4363/`.

Where can I find older driver versions?

Visit `ftp://download.nvidia.com/XFree86_40/`.

TwinView: Common Questions and Problems

Nothing gets displayed on my second monitor — what's wrong?

Monitors (including most older monitors) that do not support monitor detection using DDC protocols cannot be detected by your NVIDIA product. You need to explicitly tell the NVIDIA XFree86 driver the type of device that is connected by using the `"ConnectedMonitor"` option; for example:

```
Option "ConnectedMonitor" "CRT, CRT"
```

For additional details on using this option, *see* "Option "ConnectedMonitor" "string"" on page 55.

Will window managers be able to appropriately place windows — i.e., avoid placing windows across both display devices or in inaccessible regions of the virtual desktop)?

The NVIDIA X driver provides a Xinerama extension that allows X clients (such as window managers) to call `XineramaQueryScreens()` to detect the current TwinView configuration.

Note: The Xinerama protocol doesn't provide a way to inform clients when a configuration change occurs. So, if you modeswitch to a different MetaMode, your window manager will continue to detect the previous

configuration. By using the Xinerama extension in conjunction with the `XF86VidMode` extension to get modeswitch events, window managers should be able to determine the TwinView configuration at any given time.

Note: The data provided by `XineramaQueryScreens()` appears to confuse some window managers. To workaround such an issue, you can disable communication of the TwinView screen layout with the `"NoTwinViewXineramaInfo"` XF86Config Option. See "Option "NoTwinViewXineramaInfo" "boolean"" on page 39.

Note: Be aware that the NVIDIA driver cannot provide the Xinerama extension if XFree86's own Xinerama extension is being used. Explicitly specifying Xinerama in the XF86Config file or on the XFree86 command line will prevent NVIDIA's Xinerama extension from installing. Therefore, confirm that XFree86's "`/va/log/XFree86.0.log`" is not reporting:

```
(++) Xinerama enabled
```

if you want the NVIDIA driver to provide the Xinerama extension while in TwinView.

Another solution is to use panning domains to eliminate inaccessible regions of the virtual screen. (*See* "Option "MetaModes" "string"" on page 52.)

A third solution is to use two separate X screens, rather than use TwinView. (*See* "Configuring Multiple Screens on One Graphics Card" on page 134.)

Why can't I get a resolution of 1600x1200 on the second display device when using a GeForce2 MX GPU?

Since the second display device was designed to be a digital flat panel, the Pixel Clock for the second display device is only 150 MHz. This effectively limits the resolution on the second display device to somewhere around 1280x1024. Note that this limitation is not present on GeForce4 and GeForce FX GPUs because the maximum pixel clock is the same on both heads.

Note: For a description of how Pixel Clock frequencies limit the programmable modes, see the *XFree86 Video Timings How To* documentation.

Do video overlays work across both display devices?

Hardware video overlays only work on the first display device. The current solution is that blitted video is used on TwinView configuration.

How are virtual screen dimensions determined in TwinView?

After all requested modes have been validated, and the offsets for the viewport for each MetaMode have been computed, the NVIDIA driver computes the bounding box of the viewports for each MetaMode. The maximum bounding box width and height is then figured.

Note: A side effect of this process is that the virtual width and virtual height may come from different MetaModes. Given the following MetaMode string:

"1600x1200,NULL; 1024x768+0+0, 1024x768+0+768",

the resulting virtual screen size will be 1600 x 1536.

Can I play full-screen games across both display devices?

Yes. While the details of configuration will vary among games, the basic idea is that a MetaMode presents X with a mode whose resolution is the bounding box of the viewports for that MetaMode. For example, the following lines:

```
Option "MetaModes" "1024x768,1024x768; 800x600,800x600"
Option "TwinViewOrientation" "RightOf"
```

produce two modes: one with a resolution of 2048x768 and another with a resolution of 1600x600. Games such as Quake 3 Arena use the VidMode extension to discover the resolutions of the modes currently available.

To configure Quake 3 Arena to use the above `MetaMode` string, add the following lines to your `q3config.cfg` file:

```
seta r_customaspect "1"
seta r_customheight "600"
seta r_customwidth "1600"
seta r_fullscreen   "1"
seta r_mode         "-1"
```

Note: Given the above configuration, there is no mode with a resolution of 800x600 (remember that the MetaMode 800x600, 800x600 has a resolution of 1600x600), so if you change Quake 3 Arena to use a resolution of 800x600, it will display in the lower left corner of your screen with the rest of the screen grayed out.

To have single-display modes also available, an appropriate MetaMode string could be:

"800x600,800x600; 1024x768,NULL; 800x600,NULL; 640x480,NULL"

More precise configuration information for specific games is beyond the scope of this document. However, the above examples coupled with numerous online sources can point you in the right direction.

ALi Chipset Users: Troubleshooting

The NVIDIA Driver for Linux, Version 0.9-3, fixed the majority of problems with ALi chipsets, though some problems still exist. The following tips help stabilize problematic systems.

Tips:

- Disable `TURBO AGP MODE` in the BIOS.
- When using a P5A, upgrade to BIOS Revision 1002 BETA 2.
- When using 1007, 1007A, or 1009, adjust the IO Recovery Time to 4 cycles.
- AGP is disabled by default on some ALi chipsets (e.g, ALi1541 and ALi1647) to work around severe system stability problems (e.g., timing and signal integrity issues) that occur when using these chipsets. You can force AGP *on* for these chipsets by enabling `NVreg_EnableALiAGP` in `os-registry.c` and rebuilding the NVIDIA kernel module. (*See* the comments for `NVreg_EnableALiAGP` in `os-registry.c` to force AGP on.).

  Note: Forcing AGP *on* may cause the driver to become unstable on these chipsets.

NVIDIA TNT Users: Troubleshooting

The NVIDIA Linux Driver (Version 0.9-3) corrected the problem with SGRAM/SDRAM TNT products.

Note: In the rare case that your NVIDIA TNT-based card has the wrong BIOS installed, the driver will fail.

If the driver fails, follow these steps:

1. Watch your monitor as the system starts up. The very first, brief screen will identify the type of video memory for your card, which will be either SGRAM or SDRAM.
2. Edit the file `os-registry.c` from the kernel module sources.
3. Locate the variable "`NVreg_VideoMemoryTypeOverride`" and set the value of this variable to the type of memory you have. (Numerically, see the line just above it).

4. Since this variable is not normally used, change the "`#if 0`" (about 10 lines above the variable) to "`#if 1`".
5. Rebuild and reinstall the new driver using the "`make`" command.

Contacting Us

You can access the NVIDIA Linux Driver web forum by going to:

`www.nvnews.net`

and following the "Forum" and "Linux Discussion Area" links. This is the preferable tool for seeking help; users can post questions, answer other users' questions, and search the archives of previous postings.

Note: If all else fails, you can contact NVIDIA for support at:

`linux-bugs@nvidia.com`

But please only send E-mail to this address after you've browsed the "Frequently Asked Questions" sections in this chapter and have requested help on the `nvnews.net` web forum.

Additional Resources

- **Linux OpenGL ABI**

  `http://oss.sgi.com/projects/ogl-sample/ABI/`

- **NVIDIA Linux HowTo**

  `http://www.tldp.org/HOWTO/XFree86-Video-Timings-HOWTO/index.html`

- **OpenGL:** `www.opengl.org`
- **The XFree86 Project:** `www.xfree86.org`
- **`#nvidia`** (`irc.openprojects.net`)

APPENDIX

A

INSTALLED NVIDIA LINUX DRIVER COMPONENTS

This appendix contains the following major topics:

Installed Components

This section describes the following components of the NVIDIA Accelerated Linux Driver Set**:**

The file shown in parenthesis is the full name of the component after installation. `x.y.z` denotes the current version — appropriate symlinks are created during installation.

OpenGL and GLX Header Files

`(/usr/include/GL/gl.h, /usr/include/GL/glx.h)`

ELF TLS OpenGL and OpenGL Core Libraries

`(/usr/lib/tls/libGL.so.x.y.z and /usr/lib/tls/libGLcore.so.x.y.z).`

Linux systems that utilize glibc 2.3 (or higher version) with **TLS** support enabled use a new mechanism for TLS. Because this mechanism is incompatible with the previous NVIDIA TLS support, special ELF TLS libraries are provided and installed in `/usr/lib/tls/` on systems that support these libraries.

The runtime loader will select between the OpenGL libraries installed in (`/usr/lib/`) and those installed in (`/usr/lib/tls/`).

Note: This new TLS mechanism also affects the GLX extension module (`libglx.so.x.y.z`). However, because the XFree86 loader cannot differentiate between TLS and non-TLS libraries, the correct `libglx` library is automatically installed in `/usr/X11R6/lib/modules/extensions/`.

You can determine whether your `glibc` uses the new TLS mechanism by using this command:

```
/lib/libc.so.6 | grep "Thread-local storage support included."
```

The above command will print "`Thread-local storage support included.`" on systems that support the new TLS.

NVIDIA-Installer Utility

The `nvidia-installer` utility (`/usr/bin/nvidia-installer`) is the NVIDIA tool for installing and updating NVIDIA Linux drivers.

About Installation of Libraries

Note: Problems will arise if applications use the wrong version of a library, which can occur if either old libGL libraries or obsolete symlinks exist. If you have reason to believe that your installation may encounter problems, check that the following files are in place; these files are part of the NVIDIA Accelerated Linux Driver Set and include their symlinks:

```
/usr/X11R6/lib/modules/drivers/nvidia_drv.o


/usr/X11R6/lib/modules/extensions/libglx.so.x.y.z
/usr/X11R6/lib/modules/extensions/libglx.so ->
libglx.so.x.y.z


/usr/lib/libGL.so.x.y.z
/usr/lib/libGL.so.x -> libGL.so.x.y.z
/usr/lib/libGL.so -> libGL.so.x
```

```
/usr/lib/libGLcore.so.x.y.z
/usr/lib/libGLcore.so.x -> libGLcore.so.x.y.z

/lib/modules/`uname -r`/video/nvidia.o, or
/lib/modules/`uname -r`/kernel/drivers/video/nvidia.o
```

The installation process also creates the `/dev` files:

```
crw-rw-rw- 1 root root     195,   0 Feb 15 17:21 nvidia0
crw-rw-rw- 1 root root     195,   1 Feb 15 17:21 nvidia1
crw-rw-rw- 1 root root     195,   2 Feb 15 17:21 nvidia2
crw-rw-rw- 1 root root     195,   3 Feb 15 17:21 nvidia3
crw-rw-rw- 1 root root     195, 255 Feb 15 17:21 nvidiactl
```

If there are other libraries with a "soname" that conflicts with that of the NVIDIA libraries, "`ldconfig`" may create the wrong symlinks. In this case, it is recommended that you follow these steps:

1. Manually remove or *rename* conflicting libraries. (*See* the **Note** at the end of these steps.)

Note: Be sure to rename clashing libraries to a name that "`ldconfig`" will not identify; for example, you can prepend "`xxx`" to a library name.)

2. Rerun "`ldconfig`" and check that the correct symlinks were made. Some libraries that often create conflicts are

   `/usr/X11R6/lib/libGL.so*` and
   `/usr/X11R6/lib/libGLcore.so*`.

3. Once you've verified the libraries, then verify that the application is using the correct libraries.

   For example, to check that the application `/usr/X11R6/bin/gears` is using the NVIDIA libraries, you would issue the following command:

```
$ ldd /usr/X11R6/bin/gears
  libglut.so.3 => /usr/lib/libglut.so.3 (0x40014000)
  libGLU.so.1 => /usr/lib/libGLU.so.1 (0x40046000)
  libGL.so.1 => /usr/lib/libGL.so.1 (0x40062000)
  libc.so.6 => /lib/libc.so.6 (0x4009f000)
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x4018d000)
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x40196000)
  libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x401ac000)
  libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x401c0000)
  libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x401cd000)
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x401d6000)
  libGLcore.so.1 => /usr/lib/libGLcore.so.1 (0x402ab000)
```

```
libm.so.6 => /lib/libm.so.6 (0x4048d000)
libdl.so.2 => /lib/libdl.so.2 (0x404a9000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x404ac000)
```

Note: If the files being used for `libGL` and `libGLcore` are not NVIDIA libraries, you need to either remove the libraries that are getting in the way, or adjust your "`ld`" search path. If you are not familiar with this process, you may want to read the man pages for `"ldconfig"` and `"ldd"` for pointers.

For information on troubleshooting the NVIDIA Accelerated Linux Driver Set, *see* "Frequently Asked Questions, Troubleshooting, & Other Resources" on page 68.

APPENDIX

B

PROGRAMMING MODES

This appendix contains the following major topics:

Introduction

The NVIDIA Accelerated Linux Driver Set supports all standard VGA and VESA modes, as well as most user-written custom mode lines.Double-scan modes are supported on all hardware.

In general, your display device (such as a CRT, DFP, or TV) can be a greater constraint on usable modes than either your NVIDIA product-based graphics card *or* the NVIDIA Accelerated Linux Driver Set.

To request one or more standard modes for use in X, you can add a "`Modes`" line, as shown below, in the appropriate "`Display`" subsection of your XF86Config file:

```
Modes "1600x1200" "1024x768" "640x480"
```

(Refer to the XF86Config(4/5) man page for further details.)

Note: The documentation that follows is primarily of interest if you create your own custom mode lines, experiment with `xvidtune(1)`, or are simply interested in further knowledge. This document is neither an explanation nor a guide to crafting custom mode lines for XFree86, which is offered in documents such as the *XFree86 Video Timings HowTo*; refer to the `www.tdlp.org` web site.

Depth, Bits Per Pixel, and Pitch

The bits used per pixel is an important issue when considering the maximum programmable resolution, though not a direct concern when programming modes. A discussion of the terms “depth” vs. “bits per pixel” follows.

Depth is the number of bits of data are stored per pixel. Supported depths are 8, 15, 16, and 24. Most video hardware, however, stores **pixel data** in sizes of 8, 16, or 32 bits; this is the amount of memory allocated per pixel. When you specify the depth, X selects the **bits per pixel (bpp)** size in which to store the data.

Table B.1 lists the bits per pixel used for each supported depth. .

Table B.1 Bits Per Pixel Used for Depth

Depth	Bits Per Pixel
8	8
15	16
16	16
24	32

The **pitch** is the number of bytes in the linear frame buffer between the data of one pixel and the data of the pixel immediately below that one. Pitch can be represented by the formul

Pitch = HR * (bpp/8)

Pitch = Horizontal Resolution (HR) multiplied by the Bytes Per Pixel

Bytes Per Pixel = Bits Per Pixel (bpp) divided by 8

Note: In practice, the pitch may be greater than this product because video hardware often requires the pitch to be a multiple of a certain value.

Maximum Resolutions

The NVIDIA Accelerated Linux Driver Set and NVIDIA product-based graphics cards support resolutions up to 2048x1536, though the maximum resolution that your system can support is also limited by the amount of video memory (referenced in Useful Formulas in the next section) and the maximum supported resolution of your display device (i.e., CRT, DFP, or TV).

Note: While use of a video overlay does not limit the maximum resolution or refresh rate, video memory bandwidth used by a programmed mode does affect the quality of the overlay.

Useful Formulas

Video Memory Used

The maximum resolution is a function of the amount of video memory *and* the bits per pixel (bpp) that you want to use, as represented by the formul

Amount of Video Memory Used = HR * VR * (bpp/8)

Amount of Video Memory Used =	Horizontal Resolution (HR) multiplied by the Vertical Resolution (VR) multiplied by the Bytes Per Pixel
Bytes Per Pixel =	Bits Per Pixel (bpp) divided by 8

Note: Technically, the Video Memory Used = (Pitch * Vertical Resolution) where the Pitch may be slightly greater than "HR * (bpp/8)" to accommodate hardware requirements that specify the pitch to be a multiple of a certain value.

This discussion *only* refers to memory usage for the frame buffer; video memory is also used by other operations, such as OpenGL or pixmap caching.

Resolution, Pixel Clock, and Vertical Refresh Rate

The relationship between resolution, pixel clock (i.e., dot clock), and vertical refresh rate can be described by this formul

RR = PCLK / (HFL * VFL)

Refresh Rate (RR) =	Pixel Clock (PCLK) divided by the Total Number of Pixels
Total Number of Pixels =	Horizontal Frame Length (HFL) multiplied by the Vertical Frame Length (VFL).

The frame length in **VFL** refers to the actual frame length and not simply the visible resolution.

As described in the *XFree86 Video Timings HowTo* documentation, the above formula may be rewritten as:

PCLK = RR * HFL * VFL

Given a maximum pixel clock, you can adjust the **RR**, **HFL** and **VFL** as needed.

The pixel clock is reported in the log file when you run X with verbose logging, as shown below:

```
startx -- -logverbose 5
```

Your XFree86.0.log should contain several lines (as shown below) to indicate the maximum pixel clock at each bit per pixel size.

```
(--) NVIDIA(0): Display Device 0: maximum pixel clock at 8 bpp:
350 MHz
(--) NVIDIA(0): Display Device 0: maximum pixel clock at 16 bpp:
350 MHz
(--) NVIDIA(0): Display Device 0: maximum pixel clock at 32 bpp:
300 MHz
```

Mode Validation

During the pre-initialization phase of the X Server, the NVIDIA X driver validates all requested modes using these steps:

1. Takes the intersection of the `HorizSync` and `VertRefresh` ranges provided by the user in the XF86Config file with the ranges reported by the monitor in the EDID. This function can be disabled by using the `"IgnoreEDID"` option, in which case the X driver will accept the HorizSync and VertRefresh provided by the user.

2. Calls the `xf86ValidateModes()` helper function, which finds modes with the user-specified names in the XF86Config file, pruning out modes with:
   - invalid horizontal sync frequencies or vertical refresh rates,
   - pixel clocks larger than the maximum pixel clock for the graphics card, *and*
   - resolutions larger than the virtual screen size, if a virtual screen size was specified in the XF86Config file.

   Several other constraints are applied to this mode search. (*See* `xc/programs/Xserver/hw/xfree86/common/xf86Mode.c:xf86ValidateModes()`.)

3. All modes returned from `xf86ValidateModes()` are then examined to ensure that their resolutions are not greater than the highest mode reported by the monitor's EDID. (As explained earlier, this function may be disabled with the `"IgnoreEDID"` option.)

   If the display is a TV, each mode is checked to ensure that it has a resolution that is supported by the TV encoder. Usually, only resolutions of 800x600 and 640x480 are supported by the encoder.

4. All modes are also tested to confirm that they fit within the constraints of the hardware's memory bandwidth. You can disable this test with the `NoBandWidthTest` XF86Config file option. See “Option “NoBandWidthTest” “boolean”” on page 43.

5. All remaining modes are then checked to ensure sure they pass the constraints described in “Additional Mode Constraints” on page 95.

Note: In order to catch potentially invalid modes submitted by `XF86VidModeExtension` (e.g., `xvidtune(1)`), the last two steps are also performed when each mode is programmed.

Note: Under TwinView configuration, the above validation is performed for the requested modes for each display device.

Additional Mode Constraints

Following is a list of additional constraints on the mode parameters: In some cases these are specific to particular chips, as shown in Table B.2. The **Mode** column values are explained in the text that follows the table..

Table B.2 Maximum DAC Values

Mode	Family of GPUs		
	GeForce and TNT	**GeForce2 and GeForce3**	**GeForce4 (*or later*)**
HR	4096	4096	8192
HBW	1016	1016	2040
HSS	4088	4088	8224
HSW	256	256	512
HFL	4128	4128	8224
VR	2048	4096	8192
VBW	128	128	256
VSS	2047	4095	8192
VSW	16	16	16
VFL	2049	4097	8192

- The **Horizontal Resolution (HR)** must be a multiple of 8 and be less than or equal to the value in Table B.2.
- The **Horizontal Blanking Width (HBW)** must be a multiple of 8 and be less than or equal to the value in Table B.2.The formula is:

  HBW = max(HFL,HSE) – min(HR,HSS)

  max = maximum of
  HFL, = Horizontal Frame Length *and*
  HSE = Horizontal Sync End
  – = *minus*
  min = minimum of
  HR, = Horizontal Resolution *and*
  HSS = Horizontal Sync Start

- **The Horizontal Sync Start (HSS)** value must be a multiple of 8 and be less than or equal to the value in Table B.2.
- The **Horizontal Sync Width (HSW)** must be a multiple of 8 and less than or equal to the value in Table B.2.The formula is:

HSW = HSE – HSS

HSE =	Horizontal Sync End
– =	*minus*
HSS =	Horizontal Sync Start

- The **Horizontal Frame Length (HFL)** must be a multiple of 8, must be greater than or equal to 40, and must be less than or equal to the value in Table B.2.
- The **Vertical Resolution (VR)** must be less than or equal to the value in Table B.2.
- The **Vertical Blanking Width (VBW)** must be less than or equal to the value in Table B.2. The formula is:

  VBW = max(VFL,VSE) – min(VR,VSS)

max =	maximum of
VFL, =	Vertical Frame Length *and*
VSE =	Vertical Sync End
– =	*minus*
min =	minimum of
VR, =	Vertical Resolution *and*
VSS =	Vertical Sync Start

- The **Vertical Sync Start (VSS) v**alue must be less than or equal to the value in Table B.2.
- The **Vertical Sync Width (VSW)** must be less than or equal to the value in . The formula is:

  VBW = VSE - VSS

VSE =	Vertical Sync End
– =	*minus*
VSS =	Vertical Sync Start

- The **Vertical Frame Length (VFL)** must be greater than or equal to 2 *and* less than or equal to the value in Table B.2.

Example Mode Line

The following is an example mode line that contains each abbreviation that is used in the constraints listed above:

```
# Custom Mode line for the SGI 1600SW Flatpanel
#       name          PCLK  HR   HSS  HSE  HFL  VR   VSS  VSE  VFL
Modeline "sgi1600x1024" 106.9 1600 1632 1656 1672 1024 1027 1030
1067
```

Note: An XFree86 modeline generator conforming to the GTF Standard has been posted to the XFree86 Xpert mailing list:

`http://www.xfree86.org/pipermail/xpert/2001-October/012070.html`

For additional modeline generators, search for "`Modeline`" on `freshmeat.net`.

APPENDIX

PROC FILESYSTEM INTERFACE

The /proc filesystem interface allows you to obtain run-time information about the driver, any installed NVIDIA graphics cards, and the AGP status. This information is contained in several files in `/proc/driver/nvidia`.

The following are brief descriptions of each one of these files:

- **version**

  Lists the installed driver revision and the version of the GNU C compiler used to build the Linux kernel module.

- **cards/0...3**

  Provides information about each of the installed NVIDIA graphics adapters (model name, IRQ, BIOS version, Bus Type). Please note that the BIOS version is only available while X is running.

- **agp/card**

  Provides information about the installed AGP card's AGP capabilities.

- **agp/host-bridge**

  Provides information about the host bridge (model and AGP capabilities).

- **agp/status**

  Provides information about the current AGP status. If AGP support has been enabled on your system, the AGP driver being used, the AGP rate, and information about the status of AGP Fast Writes and Side Band Addressing is shown.

  The AGP driver is either one of NVIDIA (NVIDIA's built-in AGP driver) or AGPGART (the Linux kernel's `agpgart.o` driver). If you see “inactive” next

to AGPGART, then this means that the AGP chipset was programmed by AGPGART, but is not currently in use.

SBA and Fast Writes indicate whether either one of the features is currently in use. Please note that several factors decide if support for either will be enabled. First of all, both the AGP card and the host bridge must support the feature. Even if both do support it, the driver may decide not to use it in favor of system stability. This is particularly true of AGP Fast Writes.

APPENDIX

D

XVMC SUPPORT

Note: This release of the NVIDIA Linux Driver Set includes support for the X-Video Motion Compensation (XvMC) version 1.0 API on GeForce4-based and GeForce FX-based GPUs.

The static library "`ibXvMCNVIDIA.a`" and the dynamic library "`libXvMCNVIDIA_dynamic.so`" are suitable for `dlopening`.

- GeForce4 MX-based and GeForce FX-based GPUs support both the XvMC "IDCT" and "motion-compensation" levels of acceleration.
- GeForce4 Ti GPUs only support the motion-compensation level.
- AI44 and IA44 subpictures are supported.
- 4:2:0 surfaces up to 2032 x 2032 are supported.
- `libXvMCNVIDIA` observes the `XVMC_DEBUG` environment variable and will provide some debug output to `stderr` when set to one of the following integer values:
  - "0" disables debug output.
  - "1" enables debug output for failure conditions.
  - "2" or higher enables output of warning messages.

APPENDIX

E

GLX SUPPORT

This release of the NVIDIA Linux Driver Set supports GLX 1.3 with the following extensions:

- GLX_EXT_visual_info
- GLX_EXT_visual_rating
- GLX_SGIX_fbconfig
- GLX_SGIX_pbuffer
- GLX_ARB_get_proc_address

For a description of these extensions, refer to the OpenGL extension registry at the following web site:

http://oss.sgi.com/projects/ogl-sample/registry/index.html

Note: Some of the above extensions exist as part of core GLX 1.3 functionality and are also exported as extensions for backwards compatibility.

APPENDIX

F

CONFIGURING MULTIPLE SCREENS ON ONE GRAPHICS CARD

NVIDIA GPUs that support TwinView can also be configured to treat each connected display device as a separate X screen.

While there are several disadvantages to this approach as compared to TwinView (e.g., windows cannot be dragged between X screens and hardware accelerated OpenGL cannot span the two X screens), it offers several advantages over TwinView:

- If each display device is a separate X screen, properties (such as, depth, root window size, and so on) that may vary between X screens may also vary between displays.
- Hardware that can only be used on one display at a time (for example, video overlays and hardware accelerated RGB overlays) and, consequently, cannot be used at all in TwinView configuration, can be exposed on the first X screen when each display is a separate X screen.
- Some applications require the one-to-one association of display devices to X screens as a benefit for doing this. I'm not sure the design of X is the right thing to point out here

To configure two separate X screens to share one NVIDIA GPU-based card, follow these steps:

1. First, create two separate Device sections, each listing the `BusID` of the graphics card to be shared and listing the driver as `"nvidia"`. Assign each a separate screen:

```
Section "Device"
  Identifier  "nvidia0"
  Driver      "nvidia"
  # Edit the BusID with the location of your graphics card
  BusID       "PCI:2:0:0"
  Screen      0
EndSection

Section "Device"
  Identifier  "nvidia1"
  Driver      "nvidia"
  # Edit the BusID with the location of your graphics card
  BusId       "PCI:2:0:0"
  Screen      1
EndSection
```

2. Create two Monitor sections, one describing the capabilities of each display device.

```
Section "Monitor"
  Identifier  "Monitor0"
  # Replace these values with those appropriate for your
  monitor.
  HorizSync "50-110"
  VertRefresh "60-120"
EndSection

Section "Monitor"
  Identifier  "Monitor1"
  # Replace these values with those appropriate for your
  monitor.
  HorizSync "50-110"
  VertRefresh "60-120"
EndSection
```

3. Then, create two Screen sections, each using one of the Device sections:

```
Section "Screen"
  Identifier  "Screen0"
  Device      "nvidia0"
  Monitor     "Monitor0"
  DefaultDepth 24
  Subsection "Display"
     Depth       24
     Modes       "1600x1200" "1024x768" "800x600" "640x480"
  EndSubsection
EndSection

Section "Screen"
  Identifier  "Screen1"
  Device      "nvidia1"
  Monitor     "Monitor1"
  DefaultDepth 24
  Subsection "Display"
     Depth       24
     Modes       "1600x1200" "1024x768" "800x600" "640x480"
  EndSubsection
EndSection
```

4. Create a second "`Monitor`" section.

5. Finally, update the `ServerLayout` section to use and position both Screen sections:

```
Section "ServerLayout"
  ...
  Screen         0 "Screen0"
  Screen         1 "Screen1" leftOf "Screen0"
  ...
EndSection
```

6. For further details, refer to the XF86Config manpage.