NVIDIA

NVIDIA ACCLERATED LINUX® DRIVER SET
Release 10 Notes

Version 1.0-1512

NVIDIA Corporation
September 4, 2001

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 topics:

About Release Notes

These Release Notes provides information about the current Release 10 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 document serves several purposes; it

- explains how to install, configure, and use the NVIDIA Accelerated Linux Driver Set,
- describes current and historic software problem resolutions and software enhancements, categorized by Release and Version number, and
- contains Troubleshooting, Frequently Asked Questions, and other contact information.

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 x86 with the use of NVIDIA **graphics processing units (GPUs)**.

NVIDIA has a unified driver architecture model, which means that one driver set can be used with all supported NVIDIA hardware.

These drivers provide optimized hardware acceleration of OpenGL applications via a direct-rendering X Server. The drivers support the NVIDIA chips listed in Table 1.2 and the features discussed in "Supported NVIDIA Features" on page 5

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 NVdriver kernel module, use the following components:		
`binutils`	2.9.5	`# size --version`
`GNU make`	3.77	`# make --version`
`gcc`	2.7.2.3	`# gcc --version`
If you build the NVdriver kernel module from source RPMs, use the following component:		
`spec-helper rpm`		`# rpm -qi spec-helper`

Note: XFree86 can be retrieved from **www.xfree86.org**. Software packages may also available through your Linux distributor.

- All official stable kernel releases from version 2.2.12 and higher are supported.

  Note: "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 services can be retrieved from **www.kernel.org** or one of its mirrors.

- **binutils** and **gcc** are required *only* if you install the `NVIDIA_kernel` package by **SRPM** (Source-RPM) or tar file and can be retrieved from **www.gnu.org** or one of its mirrors.

  Note: binutils and **gcc** are *not* required by binary RPM installations.

- 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.

Note: NVIDIA GPUs may not work with older versions of the "nv" driver that shipped with XFree86. For example, the "nv" driver that shipped with XFree86 (version 4.0.1) did not support video cards based on the NVIDIA GeForce2 family and the Quadro2 MXR GPUs. However, this was fixed in XFree86 (version 4.0.2).

Also note that the GeForce2 Go GPU is *not* supported by the "nv" driver. For complete GeForce2 Go support information, see the first item in "Notes on NVIDIA GPU Support" on page 4 and "Release 10: GeForce2 Go" on page 5.)

Supported NVIDIA GPUs

Table 1.2 lists the NVIDIA GPUs supported by the current version of the NVIDIA Accelerated Linux Driver Set. For additional information on NVIDIA GPU support, see "Notes on NVIDIA GPU Support" on page 4.

Table 1.2 Supported NVIDIA GPUs

NVIDIA Desktop GPU	NVIDIA Workstation GPU	Device PCI ID
	Quadro DCC™	0x0203
GeForce3™		0x0200
	Quadro2 Pro™	0x0153
GeForce2™ Ultra™		0x0152
GeForce2 GTS		0x0151
GeForce2 GTS™		0x0150
GeForce2 Pro™		0x0150
	Quadro2 EX™	0x0113
	Quadro2 MXR™	0x0113
	Quadro2 Go™	0x0113
GeForce2 Go™		0x0112

Table 1.2 Supported NVIDIA GPUs

NVIDIA Desktop GPU	NVIDIA Workstation GPU	Device PCI ID
GeForce2 MX 200		0x0111
GeForce2 MX 100		0x0111
GeForce2 MX 400		0x0110
GeForce2 MX™		0x0110
	Quadro™	0x0103
GeForce DDR™		0x0101
GeForce 256™		0x0100
RIVA™ TNT2™ Family		
RIVA TNT2		0x0028
RIVA TNT2 (Ultra)		0x0029
RIVA TNT2 (Vanta)		0x002C
RIVA TNT2 (M64)		0x002D
RIVA TNT2		0x002E
RIVA TNT2		0x002F
RIVA TNT2 (Integrated)		0x00A0
RIVA TNT™		0x0020

Notes on NVIDIA GPU Support

- GeForce2 Go (GPU for laptop computers) is supported by the NVIDIA Accelerated Linux Driver Set but *not* by the open source "nv" driver for XFree86. *See also* "Release 10: GeForce2 Go" on page 5.
- RIVA 128/128ZX GPUs are supported by the open source "nv" driver for XFree86 but not by the NVIDIA Accelerated Linux Driver Set.
- Some Release 10 and Release 6 features support only certain NVIDIA products, as specified in the "Supported NVIDIA Features" on page 5.
- If you want to check the VGA device IDs for comparison with the values shown in Table 1.2, use one of these methods:
  - "`cat /proc/pci`" *or*
  - "`lspci -n`" to dump hex information. With this method, you need to retrieve the line with the NVIDIA vendor ID of "`Class 0300: 10de`", as in the following example:

    ```
    0x10de: 01:00.0 Class 0300: 10de:0028 (rev 15)
    ```

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

Supported NVIDIA Features

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.

Release 10: GeForce2 Go

Release 10 adds support for mobile platforms using the GeForce2 Go GPU and, additionally, the current version (1.0-1512) of the software adds support for the Quadro2 Go GPU. The current version of the driver improves stability on mobile platforms and adds support for hot key switching.

Release 10: TV Output Feature

NVIDIA GPU-based video cards 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 (computer monitor) or a **digital flat panel (DFP)** display. The TV can be used by itself, or (on appropriate video cards) in conjunction with another display device in a TwinView configuration. (See Release 6: TwinView Feature below for details.)

If a TV is the only display device connected to your video 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 were a CRT.

Release 6: TwinView Feature

The TwinView feature is only supported on NVIDIA GPUs that support dual-display functionality, such as the GeForce2 MX family, GeForce2 Go, and Quadro2 MXR.

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, **red-green-blue (RGB)** monitors, TVs, and analog flat panels. This method of using multiple monitors 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.

Known Product Limitations

Software Issues

The following problems exist in this release and are in the process of being resolved:

TwinView and Hot Key Switching on Toshiba Satellite 2800 Series Laptops

The TwinView feature and hot key switching are currently not supported on Toshiba Satellite 2800 series laptops. For additional information on known laptop issues, see "Known Laptop Issues" on page 40.

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 is not functional with Xinerama.

OpenGL and dlopen()

That there are some issues with the `glibc` dynamic library loading and `libdl.so` that cause problems with applications that use `dlopen()` to load the OpenGL library. Applications that use `dlopen()` include Quake3 and Radiant. A workaround has been implemented that will fix some, but not all, cases where this happens.

glReadPixels and glCopyPixels After Window is Moved

When the window moves, the data that is read back from the back buffer, stencil buffer, and/or depth buffer will be incorrect unless the window is redrawn after the move.

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.

Multicard, Multimonitor

X Server does not work reliably when two cards are used to drive multiple monitors.

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” on page 7
- “VIA KX133 and 694X Chipsets With AGP 2X” on page 8
- “Irongate Chipsets With AGP 1X” on page 8

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 video 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 the ASUS K7V motherboard, NVIDIA drivers default to AGP 2X mode to work around insufficient drive strength on one of the signals.

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.

CHAPTER 2

NVIDIA LINUX DRIVER HISTORY

This chapter contains the following major topics:

NVIDIA Linux Driver Versions

Release 10 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.

Note: Some versions listed may not have been released outside of NVIDIA.

Table 2.1 NVIDIA Linux Driver Versions

Driver	Versions	Comments
Release 10	1.0-1251 through 1.0-1512	Releases ongoing
Release 6	0.9 through 0. 9-769	

Overview of Resolved Issues by Version

This section contains key software issues resolved in the following releases of the NVIDIA Linux drivers:

Release 10

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 28 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 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 NV20 (Quadro DCC) correctly.
- 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 (NV20) card and properly initializes.
- Fixed problem so that Redhat 7.1 SMP RPMS can correctly uninstall with the “`rpm -e`” command while X Server is running.
- Corrected default OpenGL state when indirect rendering.
- Added `xf86XVOffscreenImage` 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 Xconfigure 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

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 GPUs were fixed. This problem had resulted in 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 GPUs 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 GPUs.
- 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. (Be sure to include `Option "DPMS"` in 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 GPUs.
- 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 some NVIDIA GPUs, for example, 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 **full-scene antialiasing (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 `dlopen()`.
- 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 (RedHat 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 in the libraries.
- Temporarily forced disabling of AGP fast writes for all chips.
- Fixed monitor issues and allowed overriding of synch polarities.

Version 0.9-1

Initial Release

CHAPTER 3

INSTALLING THE NVIDIA LINUX DRIVERS

This chapter contains the following major topics:

Choosing the NVIDIA Packages for Your System

The **NVIDIA Accelerated Linux Driver Set** consists of two packages that you need to download and install:

- **`NVIDIA_GLX`** package contains the OpenGL libraries and the XFree86 driver.
- **`NVIDIA_kernel`** package contains the NVdriver kernel module required by the X driver and OpenGL libraries in the `NVIDIA_GLX` package.

For detailed description of the components of each package, see "Installed Components" on page 21.

Note: You must install both packages with matching version numbers; i.e., `NVIDIA_GLX-0.9-6` should only be used with `NVIDIA_kernel-0.9-6` and *not* `NVIDIA_kernel-0.9-3`.

The packages are available in these three formats:

- **RPM**
- **SRPM**
- **Tar file**

Installation of each package type is described in the sections that follow.

The package type is largely a matter of personal preference, though note that the binary RPMs are for use only with the kernel shipped with a particular distribution; i.e., `NVIDIA_kernel-0.9-6.rh62.i386.rpm` should only be used with the uni-processor kernel shipped with RedHat version 6.2.

Where appropriate, NVIDIA has provided separate RPMs for the distinct SMP and uni-processor kernels of each distribution. If you have upgraded your kernel, or a specific `NVIDIA_kernel` RPM is not available for your distribution, then use either the `NVIDIA_kernel` SRPM or tar file.

In the case where distributors ship multiple kernels (as is often the case with uni-processor and SMP systems), multiple RPMs are available, i.e., `NVIDIA_kernel-0.9-7.rh62.i386.rpm` and `NVIDIA_kernel-0.9-7.rh62.smp.i386.rpm`.

The `NVIDIA_GLX RPM`, however, is not dependent upon the kernel version, and therefore an SRPM is not required. Install the `NVIDIA_GLX` package either by RPM or tar file.

Installing the NVIDIA_kernel and NVIDIA_GLX Packages

This section contains the following topics:

Before You Begin Driver Installation

1. Exit the X Server.
2. Set your default run level so you will boot to console and not start up X.

Note: If you are unsure about how to perform this step, refer to the documentation that came with your Linux distribution.

3 The package revision numbers have been omitted in the installation instructions to make them as general as possible.

For example, if the directions are `NVIDIA_kernel.tar.gz`, replace that with the name of the driver version you are installing; for example: `NVIDIA_kernel.0.9-6.tar.gz`.

Installing by RPM

Commands

```
$ rpm -ivh NVIDIA_kernel.i386.rpm
$ rpm -ivh NVIDIA_GLX.i386.rpm
```

Commands Explained

1 Before installing from RPM, be sue you have downloaded the `NVIDIA_kernel RPM` that is appropriate for your kernel.

2 Once you have verified that you have the correct RPM, install `NVIDIA_kernel` with this command:

```
$ rpm -ivh NVIDIA_kernel.i386.rpm
```

3 Next, install the `NVIDIA_GLX RPM` with this command:

```
$ rpm -ivh NVIDIA_GLX.i386.rpm
```

Upgrading by RPM

Commands

```
$ rpm -Uvh NVIDIA_kernel.i386.rpm
$ rpm -e NVIDIA_GLX
$ rpm -ivh NVIDIA_GLX.i386.rpm
```

Commands Explained

Before upgrading from RPM, be sure that you have downloaded the `NVIDIA_kernel RPM` that is appropriate for your kernel. Once you have verified that you have the correct RPM, upgrade the `NVIDIA_kernel` package using this command:

```
$ rpm -Uvh NVIDIA_kernel.i386.rpm
```

Note: You should not use the "`-U`" option to upgrade the `NVIDIA_GLX RPM` because a problem in the uninstall section of older NVIDIA RPMs will cause some files to be incorrectly removed. Instead, use "`-e`" to

remove the old NVIDIA_GLX RPM and then install the new one using these commands:

```
$ rpm -e NVIDIA_GLX
$ rpm -ivh NVIDIA_GLX.i386.rpm
```

Installing & Upgrading by SRPM

Commands

```
$ rpm --rebuild NVIDIA_kernel.src.rpm
$ rpm -ivh /path/to/rpms/RPMS/i386/NVIDIA_kernel.i386.rpm
$ rpm -ivh NVIDIA_GLX.i386.rpm
```

Commands Explained

To build a custom NVIDIA_kernel for your system, pass RPM the "--rebuild" flag:

```
$ rpm --rebuild NVIDIA_kernel.src.rpm
```

Watch for the line that looks something like (the path may be different):

```
Wrote: /usr/src/redhat/RPMS/i386/NVIDIA_kernel.i386.rpm
```

and use that as input for RPM to install:

```
$ rpm -ivh /usr/src/redhat/RPMS/i386/NVIDIA_kernel.i386.rpm
```

or upgrade:

```
$ rpm -Uvh /usr/src/redhat/RPMS/i386/NVIDIA_kernel.i386.rpm
```

To install the NVIDIA_GLX package, follow the instructions above for either installing or upgrading NVIDIA_GLX from RPM.

Installing & Upgrading by Tar File

Commands

```
$ tar xvzf NVIDIA_kernel.tar.gz
$ tar xvzf NVIDIA_GLX.tar.gz
$ cd NVIDIA_kernel
$ make install
$ cd ../NVIDIA_GLX
$ make install
```

Commands Explained

To install from tar file, follow these steps:

1. Unpack each file with these commands:

```
$ tar xvzf NVIDIA_kernel.tar.gz
$ tar xvzf NVIDIA_GLX.tar.gz
```

2. Change to the `NVIDIA_kernel` directory. Then use the "`make install`" command to compile the kernel interface to the NVdriver; link the NVdriver; copy the NVdriver into place; and attempt to insert the NVdriver into the running kernel.

```
$ cd NVIDIA_kernel
$ make install
```

3. Move into the `NVIDIA_GLX` directory by using the "`make install`" command to copy required OpenGL and XFree86 files into place.

```
$ cd ../NVIDIA_GLX
$ make install
```

Editing Your XF86Config File

Note: This documentation uses "XF86Config" to refer to your configuration file; you may have a different name for the 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 co-exist on the same system, it was decided that XFree86 4.x will use the configuration file `/etc/X11/XF86Config-4`, if this file exists. If this file does not exit, XFree86 4.x will use `/etc/X11/XF86Config`.

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

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`.)

If you do not have a working XF86Config file, there are several ways to begin:

- Refer to the sample configuration file is included with XFree86.
- Refer to the sample configuration file is included in the `NVIDIA_GLX` package and 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 details on the XF86Config file syntax, refer to the man page.)

If you already have an XF86Config file working with a different driver, such as the `nv` driver, then follow these steps:

1. Find the relevant Device section and replace the line:

   ```
   Driver "nv"
   ```

   with

   ```
   Driver "nvidia"
   ```

2. In the Module section, verify that you have:

   ```
   Load   "glx"
   ```

3. *Remove* the following lines, if they exist:

   ```
   Load  "dri"
   Load  "GLcore"
   ```

4. There are many options that can be added to the XF86Config file to fine-tune the NVIDIA XFree86 driver. (*See* "XF86Config Options" on page 25 for a complete list of these options.)

5. Once you have configured your XF86Config file, you are ready to restart X and begin using the accelerated OpenGL libraries.

   After you restart X, you can run any OpenGL application, which will automatically use the new NVIDIA libraries. If you encounter any problems, *see* "Troubleshooting, Questions, & Resources" on page 41.

Installed Components

The NVIDIA Accelerated Linux Driver Set consists of several components:

- The **XFree86 driver**, **GLX module**, **libGL**, and **libGLcore** components are included in the `NVIDIA_GLX` package.
- The **NVdriver kernel module** component is included in the `NVIDIA_kern`el package.
- **Documentation and the OpenGL and GLX header files** are part of the `NVIDIA_GLX` package and are installed in:

  `/usr/share/doc/NVIDIA_GLX-1.0`

Note: 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.

- **An XFree86 driver**

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

  is required by XFree86 to make use of your NVIDIA hardware. The `nvidia_drv.o driver` is binary compatible with XFree86 version 4.0.1 and higher.

- **A GLX extension module**

  (`/usr/X11R6/lib/modules/extensions/libglx.so.x.y.z`)

  is used by XFree86 to provide server-side GLX support.

- **An OpenGL library**

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

  provides the API entry points for all OpenGL and GLX function calls. At run-time, OpenGL applications link to this library.

- **An OpenGL core library**

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

  is implicitly used by `libGL` and `libglx` and contains the core accelerated 3D functionality.

  Note: *Do not* explicitly load the OpenGL core library in your XF86Config file; this task is performed by `libglx`.

- **A kernel module**

  (`/lib/modules/'uname -r'/video/NVdriver`) or
  (`/lib/modules/'uname -r'/kernel/drivers/video/NVdriver`)

  provides low-level access to your NVIDIA hardware for all of the above components. It is generally loaded into the kernel when the X Server is started and is used by the XFree86 driver and OpenGL.

  `NVdriver` has two components:

  - the **binary-only core** and
  - a **kernel interface** that must be compiled specifically for your kernel version.

  Note: The Linux kernel does not have a consistent binary interface such as XFree86, so it is important that this kernel interface is matched to the version of the kernel that you are using. You can accomplish this by compiling yourself or using precompiled binaries provided for the kernels shipped with some of the more common Linux distributions.

- **OpenGL and GLX header files**

  (`/usr/share/doc/NVIDIA_GLX-1.0/usr/include/GL/gl.h`)
  (`/usr/share/doc/NVIDIA_GLX-1.0/usr/include/GL/glx.h`)

In most cases, the system-supplied headers in `/usr/include/GL` will suffice for OpenGL development. But NVIDIA provided these headers because they contain the most up-to-date versions of the NVIDIA OpenGL extensions. If you want to use these headers, it is recommended that you copy them to `/usr/include/GL`.

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/NVdriver, or`
- `/lib/modules/‘uname -r‘/kernel/drivers/video/NVdriver`

Installing the `NVIDIA_kernel` package 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 “so* name” 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 "Troubleshooting, Questions, & Resources" on page 41.

CHAPTER 4

CONFIGURING NVIDIA LINUX DRIVERS

This chapter contains the following major topics:

Note: This documentation uses "XF86Config" to refer to your configuration file; you may be using a different file name.

XF86Config Options

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

Option "ConnectedMonitor" "*string*"

This option overrides the device that the NVIDIA kernel module detects as connected to your video card. This may be useful, for example, under these conditions:

- If you use a **KVM (keyboard, video, mouse)** 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.

- If, however, you use a digital flat panel instead of a CRT, use this option to explicitly communicate the connected device to the NVIDIA X driver. This may be useful, for example, if any of your display devices does not support detection using **Display Data Channel (DDC**) protocols.

Valid values for this option are:

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

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

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

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

Option "CursorShadow" "*boolean*"

Default: Off (no cursor shadow)

This options enables or disables use of a shadow with the hardware accelerated cursor; this is a black translucent replica of your cursor shape at a given offset from the real cursor.

Option "CursorShadowAlpha" "*integer*"

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

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

0: Completely transparent
255: Completely opaque

Default: 64

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. It is only applicable if `CursorShadow` is enabled.

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

Default: 4

Option "CursorShadowYOffset" "*integer*"

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

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

Default: 2

Option "HWCursor" "*boolean*"

This option enables or disables hardware rendering of the X cursor.

Default: On (Hardware rendering of the X cursor is enabled.)

Option "IgnoreEDID" "*boolean*"

This option disables probing of **EDID (Extended Display Identification Data)** from your monitor. Requested modes are compared against values obtained from your monitor EDIDs (if any) during mode validation.

Default: Off (Use EDIDs)

Note: Some monitors may not provide accurate information about its physical capabilities. Ignoring the values that the monitor provides may allow X to use modes that exceed the EDID values but **could result in serious problems** and, therefore, should be used with extreme caution.

Option "NoAccel" "*boolean*"

This option disables or enables 2D acceleration using the XAA module.

Note: This is not the same as 3D acceleration.

Default: Off (Acceleration is enabled.)

Option “NoDCC” “*boolean*”

Synonym for “Option “IgnoreEDID” “boolean”” on page 27.

Option “NoLogo” “*boolean*”

To disable the NVIDIA splash screen when the X server is started, specify the following line in the XF86config file:

```
Option "NoLogo" "on"
```

Default: The default setting is `Option "NoLogo" "off"`, which enables the NVIDIA splash screen during X start-up.

Option “NoRenderAccel” “*boolean*”

This option enables or disables experimental hardware acceleration of the `RENDER` extension.

Default: On (`RENDER` is accelerated, when possible.)

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: 3 (The default was **1** up 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 boot up.)

Option “SWCursor” “*boolean*”

This option enables or disables software rendering of the X cursor.

Default: Off (Software rendering of the X cursor is disabled.)

Option “UseEdidFreqs” “*boolean*”

This option causes the X Server to use the `HorizSync` and `VertRefresh` ranges in a display device's EDID, if any. Range information provided by EDID will override the HorizSync and VertRefresh ranges specified in the “Monitor” section. If a display device does not provide an EDID, or the EDID doesn't specify an HorizSync or VertRefresh range, then the X Server will default to the HorizSync and VertRefresh ranges specified in the “Monitor” section of the XF86Config file.

OpenGL Environment Variable Settings

Full-Scene Antialiasing

Antialiasing is a technique used to smooth the edges of objects in a scene to reduce the jagged “stairstep” effect that sometimes appears. **Full-scene antialiasing (FSAA)** is supported on the following NVIDIA GPUs:

GeForce3	GeForce 256
GeForce2 Go	Quadro
GeForce2 Ultra	Quadro2 Go
GeForce2 Pro	Quadro2 MXR/EX
GeForce2 GTS	Quadro DCC
GeForce2 MX	Quadro2 Pro

By setting the `__GL_FSAA_MODE` environment variable described below, you can enable full-scene antialiasing in any OpenGL application when using one of the above NVIDIA GPUs.

Several antialiasing methods are available and you can select among them by setting the `__GL_FSAA_MODE` environment variable appropriately.

Note: Increasing the number of samples taken during FSAA rendering may decrease performance.

___GL_FSAA_MODE

Use one of the values in Table 4.1 to set this environment variable. The values correspond to antialiasing methods.

Table 4.1 Values for the __GL_FSAA_MODE Environment Variable

Value	GeForce/Geforce2/Quadro Description	GeForce3 Description
0	FSAA is disabled.	FSAA is disabled.
1	FSAA is disabled.	2 x 2 oversampling with texture LOD bias
2	FSAA is disabled.	2 x 2 Quincunx
3	1.5 x 1.5 oversampling	FSAA is disabled.
4	2 x 2 oversampling with no texture LOD bias	4 x 4 Bilinear
5	FSAA is disabled.	4 x 4 Gaussian

VBLANK Synchronizing

__GL_SYNC_TO_VBLANK

Setting the variable `__GL_SYNC_TO_VBLANK` to a non-zero value forces `glXSwapBuffers` to synchronize to your monitor's vertical refresh rate; i.e., the code performs a swap only during the vertical blanking period.

Configuring AGP

The `"NvAgp"` option in your XF86Config file provides several ways to configure the use of AGP by the NVdriver 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 check your AGP status with this command:

   ```
   cat /proc/nv/card0
   ```

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.

   Changing AGP drivers generally requires a reboot before the changes actually take effect.

5. Rebuild and reinstall the new driver using "`make`", which forces the driver to ignore the BIOS of the NVIDIA GPUs 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.

- Intel 440LX
- Intel 440BX
- Intel 440GX
- Intel 815 ("Solano")
- Intel 820 ("Camino")
- Intel 840 ("Carmel")
- Intel 845 ("Brookdale")
- Intel 850 ("Tehama")
- Intel 860 ("Colusa")
- AMD 751 ("Irongate")
- AMD 761 ("IGD4")
- AMD 762 ("IGD4 MP")
- VIA 8371
- VIA 82C694X
- VIA KT133
- RCC 6585HE
- Micron SAMDDR ("Samurai")
- Micron SCIDDR ("Scimitar")

Enabling and Configuring TwinView

For a description of the TwinView feature, see “Release 6: TwinView Feature” on page 5.

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 "SecondMonitorVertRefresh"      "<vrefresh range(s)>"
Option "MetaModes"                     "<list of metamodes>"
```

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

```
Option "TwinViewOrientation" "<relationship of head (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: Off (TwinView is disabled.)

This option is required to enable TwinView; without it, 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.

You may, if you trust your display devices' EDIDs, use the `"UseEdidFreqs"` option instead of these options. (For details, see “Option “UseEdidFreqs” “boolean”” on page 29.)

Option “SecondMonitorVertRefresh” “*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.

You may, if you trust your display devices' EDIDs, use the `"UseEdidFreqs"` option instead of these options. (For details, see “Option “UseEdidFreqs” “boolean”” on page 29.)

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 35.)

  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 a panning domain for the second display device:

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

  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"
  ```

If no MetaMode string is 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: RightOf

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.

Valid values are:

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

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

Option "ConnectedMonitor" "*string*"

See "Option "ConnectedMonitor" "string"" on page 25 for application to TwinView.

Enabling and Configuring TV-Out

A TV monitor can be connected to an NVIDIA GPU-based video 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 video cards) in conjunction with another display device in a TwinView configuration. (For TwinView features, see the previous section "Enabling and Configuring TwinView" on page 32).

If a TV is the only display device connected to your video card, the TV will be used as the primary display when you start-up your system; 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

- **Modes** in the "Screen" section of the XF86Config file

  Valid modes for TV are:

  - 640x480
  - 800x600
  - 1024x768 (if the TV encoder on your video 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 37.

- **"ConnectedMonitor"** option ("Option "ConnectedMonitor" "string"" on page 25). You can use this option to tell X to use the TV for display. This options is required *only* if your TV is not detected by the video 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. See the description of this option in the section that follows.

Option "TVOutFormat" "*string*"

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

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 4.2. 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"`

Table 4.2 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
`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

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

Configuring a Laptop

The following topics are explained:

Standard Functionality

Configuring NVIDIA Linux drivers on a laptop computer is similar to the process used in a desktop environment.

To enable drivers for a laptop under NVIDIA Accelerated Linux Driver Set Release 10 (version 1.0-1251), users were required to pass the option `NVreg_Mobile` to the `NVdriver` kernel module using either the `modprobe` command:

```
modprobe NVdriver NVreg_Mobile=X
```

or by adding the following to the kernel module configuration file (usually either `/etc/conf.modules` or `/etc/modules.conf`):

```
options NVdriver NVreg_Mobile=X
```

where `X` is:

- `1` if using a GeForce2 Go or a Quadro2 Go in a Dell laptop
- `2` if using a GeForce2 Go or a Quadro2 Go in a Satellite 2800 series Toshiba laptop
- `4` if using a GeForce2 Go or a Quadro2 Go in a Satellite 3000 series Toshiba laptop

For example, for a Toshiba Satellite laptop with GeForce2 Go, you would add the following line to your kernel module configuration file:

```
options NVdriver NVreg_Mobile=2
```

Note: Versions after 1.0-1251, including the current version of the drivers, no longer require the `NVreg_Mobile` option, though it can be used to override what is detected.

TwinView Functionality

Both GeForce2 Go and Quadro2 Go GPUs support TwinView. TwinView on a laptop can be configured in the same way as on a desktop machine. (Refer to "Enabling and Configuring TwinView" on page 32.)

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 32 and "Option "SecondMonitorVertRefresh" "range(s)"" on page 33.)

  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 29.

Using Hot Keys to Switch Display Devices

Laptop computers using GeForce2 Go 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.

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. This list contains the display devices that will be used with each hot key event.

When a hot key event occurs, the next hot key state in the sequence is chosen. Each mode requested in the XF86Config file is validated against the constraint of each display device 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) can't be found, then the closest match is used and the display device with the smaller 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 as when you VT-switched away from X, regardless of what LCD/CRT hot key activity occurred while VT-switched away.

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 Issues

- Power Management is currently not supported on laptop computers.
- LCD/CRT hot key switching is currently not supported on Toshiba Satellite 2800 series laptops.
- TwinView feature is currently not supported on Toshiba Satellite 2800 series laptops.
- Exiting X and returning to the VGA console after having multiple display devices active may cause the VGA console to restore improperly. To work around this issues, LCD/CRT hot key switch back and forth once or twice.

CHAPTER 5

TROUBLESHOOTING, QUESTIONS, & RESOURCES

This chapter contains the following major topics:

Troubleshooting: General

Here are a few guidelines to keep in mind when troubleshooting problems with the NVIDIA Accelerated Linux Driver Set:

- 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`.
  - 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" driver by locating the line:

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

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

- 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
  ```

- Nothing will work if the NVdriver kernel module doesn't function properly.

  If you see a line in the X log file such as:

  ```
  (EE) NVIDIA(0): Failed to initialize the NVdriver kernel
  module!
  ```

  then, most likely, a problem exists with the NVdriver 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 by building NVdriver from the `NVIDIA_kernel tar` file by using the command:

  ```
  make install SYSINCLUDE=/path/to/kernel/headers
  ```

  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 ./NVdriver`" may help.

Note: The NVdriver may print error messages to indicate a problem. To view these messages, check `/var/log/messages` or the location where syslog is directed to place kernel messages.

- **If X starts, but OpenGL causes problems**, most likely a problem exists with other libraries that are in the way or the presence of obsolete symlinks. (*See* "Installed Components" on page 21.)

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

  a. 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`.

  b. Check your XF86Config file to verify that you are loading `glx`. (*See* "Editing Your XF86Config File" on page 20.)

  If your XF86Config file is correct, then check the XFree86 log file for warnings and errors pertaining to GLX. Also check that all of the necessary symlinks are in place. (*See* "Installed Components" on page 21.).

- **If you are trying to install/upgrade by SRPM** and the command "`rpm --rebuild ...`" only prints out a list of RPM command line options, then you probably do not have the RPM-Development packages installed.

  In most cases, you can fix this problem by installing the RPM-Development package for your distribution. Alternatively, you can install/upgrade by tar file since the tar files do not require RPM.

- **If installing the NVIDIA_kernel module results in an error message** such as the following:

  ```
  #error Modules should never use kernel-headers system headers

  #error but headers from an appropriate kernel-source
  ```

  then 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.

- If your OpenGL applications exit with the following 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.
  ```

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

  To correct the problem, it is recommended that you disable this security feature by following the steps below.

  a. Different Linux distributions have different files to control.

     If your system has the file `/etc/security/console.perms`, you need to edit the file by removing the line that starts with "`<dri>`".

     If your system has the file `/etc/logindevperms`, you need to edit the file by removing the line that lists `/dev/nvidiactl`.

     (The above steps will prevent the PAM security system from modifying the permissions on the NVIDIA device files.)

  b. Next, you need to reset the permissions on the device files to their original settings and owner by using the following commands:

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

Troubleshooting: ALI Chipset Users

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:

- 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.

Troubleshooting: NVIDIA TNT Users

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

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. Obtain the most recent `NVIDIA_kernel tar` file.
3. Edit the file `os-registry.c` from the kernel module sources.
4. 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).
5. Since this variable is not normally used, change the "`#if 0`" (about 10 lines above the variable) to "`#if 1`".
6. Rebuild and reinstall the new driver using the "`make`" command.

Frequently Asked Questions - General

Q = Question; A = Answer

Q: When I start X, it fails and my XFree86 log file contains:

```
II) LoadModule: "nvidia"
II) Loading /usr/X11R6/lib/modules/drivers/nvidia_drv.o
No symbols found in this module
EE) Failed to load /usr/X11R6/lib/modules/drivers/nvidia_drv.o
(II) UnloadModule: "nvidia"
EE) Failed to load module "nvidia" (loader failed, 256) ...
(EE) No drivers available.
```

A: The `nvidia_drv.o` X driver has been stripped of needed symbols; some versions of RPM (incorrectly) strip object files while installing. You probably need to upgrade your version of RPM. Or, you can install the `NVIDIA_GLX` package from tar file.

Q: Why does the NVdriver not work with DevFS?

A: `DevFS` will be supported in a future NVIDIA release. In the meantime, you will need to recreate the NVIDIA device nodes after each reboot. Several patches have been suggested by users to make NVdriver DevFS-aware. You may want to try one of these patches; a Web search can provide you several options.

Q: My system runs but seems unstable. What's wrong?

A: You might be using the wrong AGP module. (*See* "Configuring AGP" on page 30 for details.)

Q: The kernel module doesn't get loaded dynamically when X starts; I always have to do "`modprobe NVdriver`" first. What's wrong?

A: Verify that the line "`alias char-major-195 NVdriver`" 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.

Q: I can't build the NVdriver kernel module, or I can build the NVdriver kernel module but `modprobe/insmod` fails to load the module into my kernel. What's wrong?

A: 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_kernel Makefile` 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 this command:

```
make SYSINCLUDE=/path/to/kernel/headers
```

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.

Q: Why do OpenGL applications run so slowly?

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

For details, *see* "Installed Components" on page 21.

Q: There are problems getting Quake2 going.

A: Quake2 requires a minor setup to get it started.

c. First, in the Quake2 directory, the install creates a symlink called "`libGL.so`" that points to "`libMesaGL.so`." *Remove* or *rename* this symlink.

d. 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.

Q: There are problems getting Heretic II going.

A: 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:

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

Q: Where can I get gl.h or glx.h so I can compile OpenGL programs.

A: 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 NVIDIA OpenGL extensions. These files have been installed in:

`/usr/share/doc/NVIDIA_GLX-1.0/usr/include/GL`

to avoid conflicting with the system-installed versions. To use these headers, copy them to `/usr/include/GL`.

Frequently Asked Questions: TwinView

Q: Nothing gets displayed on my second monitor; what's wrong?

A: Monitors (including most older monitors) that do not support monitor detection using DDC protocols cannot be detected by your NVIDIA GPU. 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 options, see “Option “ConnectedMonitor” “string”” on page 25.

Q: 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)?

A: Not exactly. NVIDIA is considering writing an implementation of the Xinerama extension, which would allow Xinerama-aware window managers to query the screen layout. The other solution is to use panning domains to eliminate inaccessible regions of the virtual screen.

See “Option “MetaModes” “string”” on page 33.

Q: Why can’t I get a resolution of 1600x1200 on the second display device?

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

Q: Do video overlays work across both display devices?

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

Q: How are virtual screen dimensions determined in TwinView?

A: 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.

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

A: 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.

Contacting Us

If, after following the troubleshooting information provided in this chapter, you still encounter problems with the NVIDIA Accelerated Linux Driver Set, you can contact NVIDIA for support at: **linux-bugs@nvidia.com**.

Before you send e-mail to linux-bugs@nvidia.com for assistance, please follow these guidelines:

- Attach a copy of your XFree86 log file (`/var/log/XFree86.0.log`) along with any other information you think may be relevant.
- Send a log file generated with verbose messaging enabled; i.e, "`startx -- -logverbose 5`".

  See "Troubleshooting: General" on page 41 for details on verbose messages.

Additional Resources

- **Linux OpenGL ABI:** http://oss.sgi.com/projects/ogl-sample/ABI/
- **NVIDIA Linux HowTo:** http://www.linuxdoc.org/HOWTO/mini/Nvidia-OpenGL-Configuration/index.html
- **OpenGL:** www.opengl.org
- **The XFree86 Project:** www.xfree86.org
- #nvidia (irc.openprojects.net)

APPENDIX

PROGRAMMING MODES

This chapter contains the following major topics:

Introduction

The NVIDIA Accelerated Linux Driver Set supports all standard VGA and VESA modes, most user-written custom mode lines, as well as double-scan and interlaced modes.

In general, your display device (such as a CRT, DFP, or TV) can be a greater constraint on usable modes than either your NVIDIA GPU-based video 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.linuxdoc.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 A.1 lists the bits per pixel used for each supported depth..

Table A.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 formula:

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 GPU-based video 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 formula:

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 formula:

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)**.

Note: 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 video 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 remaining modes are then checked to ensure sure they pass the constraints described in "Additional Mode Constraints" on page 56.

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:

- The **Horizontal Resolution** (**HR**) must be a multiple of 4 *and* less than or equal to 2048.
- The **Horizontal Blanking Width** (**HBW**) must be a multiple of 4 *and* less than or equal to 1024. 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 4 *and* be less than or equal to 4088.
- The **Horizontal Sync Width (HSW)** must be a multiple of 4 *and* less than or equal to 256. 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 4 *and* less than or equal to 4128 *and* greater than or equal to 40.
- The **Vertical Resolution (VR)** must be less than or equal to 2048.

- The **Vertical Blanking Width (VBW)** must be less than or equal to 128. 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)** value must be less than or equal to 2047.
- The **Vertical Sync Width (VSW)** must be less than or equal to 16. The formula is:

  VBW = VSE - VSS

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

- The **Vertical Frame Length (VFL)** must be less than or equal to 2049 *and* greater than or equal to 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
```