NVIDIA Accelerated FreeBSD Graphics Driver README and Installation Guide

NVIDIA Corporation
Last Updated: Sun Aug 19 20:41:43 PDT 2012
Most Recent Driver Version: 304.43

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

TABLE OF CONTENTS

Chapter 1. Introduction

1A. ABOUT THE NVIDIA ACCELERATED FREEBSD GRAPHICS DRIVER

The NVIDIA Accelerated FreeBSD Graphics Driver brings accelerated 2D functionality and high-performance OpenGL support to FreeBSD x86_64 with the use of NVIDIA graphics processing units (GPUs).

These drivers provide optimized hardware acceleration for OpenGL and X applications and support nearly all recent NVIDIA GPU products (see Appendix A for a complete list of supported GPUs). TwinView, TV-Out and flat panel displays are also supported.

1B. ABOUT THIS DOCUMENT

This document provides instructions for the installation and use of the NVIDIA Accelerated FreeBSD Graphics Driver. Chapter 3, Chapter 5 and Chapter 6 walk the user through the process of downloading, installing and configuring the driver. Chapter 7 addresses frequently asked questions about the installation process, and Chapter 8 provides solutions to common problems. The remaining chapters include details on different features of the NVIDIA FreeBSD Driver. Frequently asked questions about specific tasks are included in the relevant chapters.

1C. ABOUT THE AUDIENCE

It is assumed that the user and reader of this document has at least a basic understanding of FreeBSD techniques and terminology. However, new FreeBSD users can refer to Appendix H for details on parts of the installation process.

1D. ADDITIONAL INFORMATION

In case additional information is required, Chapter 28 provides contact information for NVIDIA FreeBSD driver resources, as well as a brief listing of external resources.

Chapter 2. Minimum Software Requirements

The official minimum software requirements for the NVIDIA FreeBSD Graphics Driver are as follows:

Software Element	Min Requirement
Kernel	FreeBSD 7-STABLE (7.3 or later)
XFree86/X.Org	4.2/6.7.0

Additionally, the kernel source tree must be installed in /usr/src/sys (package 'ssys' installed)

Note that FreeBSD 7-STABLE versions older than FreeBSD 7.3 and FreeBSD 10-CURRENT development snapshots are not supported.

Chapter 3. Installing the NVIDIA Driver

This installation procedure will likely be simplified further in the future, but for the moment you will need to download the NVIDIA FreeBSD Graphics Driver archives from the NVIDIA website, extract them to a temporary location of your choice, and run the following from the root of the extracted directory hierarchy:

```
    % make install
```

This will compile the NVIDIA FreeBSD kernel module, install it, and kldload it. It will also remove any conflicting OpenGL libraries, and install the NVIDIA OpenGL libraries. The '/dev/nvidia' device files will be created (unless the system is using devfs), and your '/boot/loader.conf' file will be updated to automatically load the NVIDIA kernel module on boot, as well as the Linux ABI compatibility module should you not have it compiled into your kernel.

Chapter 4. Installed Components

The NVIDIA Accelerated FreeBSD Graphics Driver consists of the following components.

Installed File	Location
nvidia.ko	/boot/modules
libGL.so	/usr/lib/xorg
libGL.so.1	/usr/lib/xorg
libnvidia-tls.so	/usr/lib/xorg
libnvidia-tls.so.1	/usr/lib/xorg
libnvidia-cfg.so	/usr/lib/xorg
libnvidia-cfg.so.1	/usr/lib/xorg
libnvidia-glcore.so	/usr/lib/xorg
libnvidia-glcore.so.1	/usr/lib/xorg
nvidia_drv.so	/usr/lib/xorg/modules/drivers
libglx.so	/usr/lib/xorg/modules/extensions
libglx.so.1	/usr/lib/xorg/modules/extensions
libvdpau.so	/usr/lib/xorg
libvdpau.so.1	/usr/lib/xorg
libvdpau_trace.so	/usr/lib/xorg
libvdpau_trace.so.1	/usr/lib/xorg
libvdpau_nvidia.so	/usr/lib/xorg
libvdpau_nvidia.so.1	/usr/lib/xorg
nvidia-xconfig	/usr/bin
nvidia-xconfig.1	/usr/man/man1
nvidia-settings	/usr/bin
nvidia-settings.1	/usr/man/man1
nvidia0	/dev
nvidia1	/dev

```
    nvidia2                            /dev
    nvidia3                            /dev
    nvidiactl                          /dev
    libGL.so.304.43                    /compat/linux/usr/lib
    libnvidia-tls.so.304.43            /compat/linux/usr/lib
    libnvidia-glcore.so.304.43         /compat/linux/usr/lib
    libvdpau.so.304.43                 /compat/linux/usr/lib
    libvdpau_trace.so.304.43           /compat/linux/usr/lib
    libvdpau_nvidia.so.304.43          /compat/linux/usr/lib
```

Chapter 5. Using Linux Compatibility Support

If you wish to run Linux OpenGL applications on your FreeBSD computer, you will need to make sure that several prerequisites are met.

First, you should follow the basic Linux compatibility installation guide in the FreeBSD Handbook (install the linux_base package, etc). Once the basic components are in place, you will need to install the NVIDIA Linux OpenGL libraries in '/compat/linux/usr/lib' (do not brandelf them!); if the '/compat/linux/usr/lib/' directory exists when you install the FreeBSD driver, the Linux compatibility OpenGL libraries will automatically be installed.

Additionally, the 'nvidia.ko' kernel module needs to be built with support for the Linux ABI compatibility layer. This is the case by default; as a consequence, the 'nvidia.ko' kernel module requires the 'linux.ko' module to be loaded.

Note: If you have no need for Linux ABI compatibility and do not wish to load 'linux.ko', you can build the 'nvidia.ko' kernel module without support for the Linux ABI compatibility layer (see 'nv-freebsd.h' for details).

Chapter 6. Configuring X for the NVIDIA Driver

The X configuration file provides a means to configure the X server. This section describes the settings necessary to enable the NVIDIA driver. A comprehensive list of parameters is provided in Appendix B.

The NVIDIA Driver includes a utility called nvidia-xconfig, which is designed
to make editing the X configuration file easy. You can also edit it by hand.

6A. USING NVIDIA-XCONFIG TO CONFIGURE THE X SERVER

nvidia-xconfig will find the X configuration file and modify it to use the
NVIDIA X driver. In most cases, you can simply answer "Yes" when the installer
asks if it should run it. If you need to reconfigure your X server later, you
can run nvidia-xconfig again from a terminal. nvidia-xconfig will make a
backup copy of your configuration file before modifying it.

Note that the X server must be restarted for any changes to its configuration
file to take effect.

More information about nvidia-xconfig can be found in the nvidia-xconfig
manual page by running.

```
    % man nvidia-xconfig
```

6B. MANUALLY EDITING THE CONFIGURATION FILE

In April 2004 the X.Org Foundation released an X server based on the XFree86
server. While your release may use the X.Org X server, rather than XFree86,
the differences between the two should have no impact on NVIDIA FreeBSD users
with two exceptions:

- The X.Org configuration file is '/etc/X11/xorg.conf' while the XFree86
  configuration file is '/etc/X11/XF86Config'. The files use the same
  syntax. This document refers to both files as "the X config file".

- The X.Org log file is '/var/log/Xorg.#.log' while the XFree86 log file is
  '/var/log/XFree86.#.log' (where '#' is the server number -- usually 0).
  The format of the log files is nearly identical. This document refers to
  both files as "the X log file".

In order for any changes to be read into the X server, you must edit the file
used by the server. While it is not unreasonable to simply edit both files, it
is easy to determine the correct file by searching for the line

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

in the X log file. This line indicates the name of the X config file in use.

If you do not have a working X config file, there are a few different ways to obtain one. A sample config file is included both with the XFree86 distribution and with the NVIDIA driver package (at '/usr/X11R6/share/doc/NVIDIA_GLX-1.0/'). The 'nvidia-xconfig' utility, provided with the NVIDIA driver package, can generate a new X configuration file. Additional information on the X config syntax can be found in the XF86Config manual page (`man XF86Config` or `man xorg.conf`).

If you have a working X config file for a different driver (such as the "nv" or "vesa" driver), then simply edit the file as follows.

Remove the line:

```
      Driver "nv"
  (or Driver "vesa")
  (or Driver "fbdev")
```

and replace it with the line:

```
    Driver "nvidia"
```

Remove the following lines:

```
    Load "dri"
    Load "GLCore"
```

In the "Module" section of the file, add the line (if it does not already exist):

```
    Load "glx"
```

If the X config file does not have a "Module" section, you can safely skip the last step if the X server installed on your system is an X.Org X server or an XFree86 X release version 4.4.0 or greater. If you are using an older XFree86 X server, add the following to your X config file:

```
Section "Module"
    Load "extmod"
    Load "dbe"
    Load "type1"
    Load "freetype"
    Load "glx"
EndSection
```

There are numerous options that may be added to the X config file to tune the NVIDIA X driver. See Appendix B for a complete list of these options.

Once you have completed these edits to the X config file, you may restart X and begin using the accelerated OpenGL libraries. After restarting X, any OpenGL application should automatically use the new NVIDIA libraries. (NOTE: If you encounter any problems, see Chapter 8 for common problem diagnoses.)

6C. RESTORING THE X CONFIGURATION AFTER UNINSTALLING THE DRIVER

If X is explicitly configured to use the NVIDIA driver, then the X config file should be edited to use a different X driver after uninstalling the NVIDIA driver. Otherwise, X may fail to start, since the driver it was configured to use will no longer be present on the system after uninstallation.

If you edited the file manually, revert any edits you made. If you used the 'nvidia-xconfig' utility, either by answering "Yes" when prompted to configure the X server by the installer, or by running it manually later on, then you may restore the backed-up X config file, if it exists and reflects the X config state that existed before the NVIDIA driver was installed.

If you do not recall any manual changes that you made to the file, or do not have a backed-up X config file that uses a non-NVIDIA X driver, you may want to try simply renaming the X configuration file, to see if your X server loads a sensible default.

Chapter 7. Frequently Asked Questions

This section provides answers to frequently asked questions associated with the NVIDIA FreeBSD x86_64 Driver and its installation. Common problem diagnoses can be found in Chapter 8 and tips for new users can be found in Appendix H. Also, detailed information for specific setups is provided in the Appendices.

NVIDIA DRIVER

Q. Where should I start when diagnosing display problems?

A. One of the most useful tools for diagnosing problems is the X log file in '/var/log'. Lines that begin with "(II)" are information, "(WW)" are warnings, and "(EE)" are errors. You should make sure that the correct config file (i.e. the config file you are editing) is being used; look for the line that begins with:

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

Also make sure that the NVIDIA driver is being used, rather than the "nv" or "vesa" driver. Search for

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

Lines from the driver should begin with:

```
    (II) NVIDIA(0)
```

Q. How can I increase the amount of data printed in the X log file?

A. By default, the NVIDIA X driver prints relatively few messages to stderr and the X log file. If you need to troubleshoot, then it may be helpful to enable more verbose output by using the X 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 will output more messages when the verbosity level is at or above 5 (X defaults to verbosity level 1 for 'stderr' and level 3 for the log file). So, to enable verbose messaging from the NVIDIA X driver to both the log file and 'stderr', you could start X with the verbosity level set to 5, by doing the following

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

Q. I have read that the NVIDIA FreeBSD Driver is not a native driver, but sits on top of the Linux ABI compatibility layer. Is this true?

A. No, the NVIDIA FreeBSD Graphics Driver is a native driver. It does provide
   Linux OpenGL libraries in addition to the native, FreeBSD libraries to
   enable users to run Linux OpenGL applications.

Q. Is the NVIDIA FreeBSD Accelerated Graphics Driver thread-safe?

A. This release is thread-safe on FreeBSD 7.3 or later systems making use of
   the libpthread or libthr KSE threading libraries. On these systems, the
   NVIDIA Linux ABI compatibility libraries are fully thread-safe as well.

Q. Why can't the Linux compatibility libraries correctly determine if they are
   used in a multithreaded application?

A. The Linux compatibility libraries are not able to correctly determine if
   they are used in a multithreaded application because the %gs segment
   register is not initialized correctly for Linux compatibility.

   The '__GL_SINGLE_THREADED' environment variable (set to "1") can be used to
   work around this issue, but at the cost of thread-safeness.

Q. Why does X use so much memory?

A. 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 has been a frequently discussed topic on XFree86 mailing lists; see,
   for example:

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

   Note, also, that X must allocate resources on behalf of X clients (the
   window manager, your web browser, etc); the X server's memory usage will
   increase as more clients request resources such as pixmaps, and decrease as
   you close X applications.

The "IndirectMemoryAccess" X configuration option may cause additional virtual address space to be reserved.

Q. Why do applications that use DGA graphics fail?

A. The NVIDIA driver does not support the graphics component of the XFree86-DGA (Direct Graphics Access) extension. Applications can use the XDGASelectInput() function to acquire relative pointer motion, but graphics-related functions such as XDGASetMode() and XDGAOpenFramebuffer() will fail.

The graphics component of XFree86-DGA is not supported because it requires a CPU mapping of framebuffer memory. As graphics cards ship with increasing quantities of video memory, the NVIDIA X driver has had to switch to a more dynamic memory mapping scheme that is incompatible with DGA. Furthermore, DGA does not cooperate with other graphics rendering libraries such as Xlib and OpenGL because it accesses GPU resources directly.

NVIDIA recommends that applications use OpenGL or Xlib, rather than DGA, for graphics rendering. Using rendering libraries other than DGA will yield better performance and improve interoperability with other X applications.

Q. My kernel log contains messages that are prefixed with "Xid"; what do these messages mean?

A. "Xid" messages indicate that a general GPU error occurred, most often due to the driver misprogramming the GPU or to corruption of the commands sent to the GPU. These messages provide diagnostic information that can be used by NVIDIA to aid in debugging reported problems.

Q. I use the Coolbits overclocking interface to adjust my graphics card's clock frequencies, but the defaults are reset whenever X is restarted. How do I make my changes persistent?

A. Clock frequency settings are not saved/restored automatically by default to avoid potential stability and other problems that may be encountered if the chosen frequency settings differ from the defaults qualified by the manufacturer. You can use the command line below in '~/.xinitrc' to automatically apply custom clock frequency settings when the X server is started:

```
    # nvidia-settings -a GPUOverclockingState=1 -a
GPU2DClockFreqs=<GPU>,<MEM> -a GPU3DClockFreqs=<GPU>,<MEM>
```

Here '<GPU>' and '<MEM>' are the desired GPU and video memory frequencies (in MHz), respectively.

Q. Why is the refresh rate not reported correctly by utilities that use the XRandR X extension (e.g., the GNOME "Screen Resolution Preferences" panel, `xrandr -q`, etc)?

A. The XRandR X extension is not presently aware of multiple display devices on a single X screen; it only sees the MetaMode bounding box, which may contain one or more actual modes. This means that if multiple MetaModes have the same bounding box, XRandR will not be able to distinguish between them.

In order to support DynamicTwinView, the NVIDIA X driver must make each MetaMode appear to be unique to XRandR. Presently, the NVIDIA X driver accomplishes this by using the refresh rate as a unique identifier.

You can use `nvidia-settings -q RefreshRate` to query the actual refresh rate on each display device.

This behavior can be disabled by setting the X configuration option "DynamicTwinView" to FALSE.

For details, see Chapter 12.

Q. Why does starting certain applications result in Xlib error messages indicating extensions like "XFree86-VidModeExtension" or "SHAPE" are missing?

A. If your X config file has a "Module" section that does not list the "extmod" module, some X server extensions may be missing, resulting in error messages of the form:

```
Xlib: extension "SHAPE" missing on display ":0.0"
Xlib: extension "XFree86-VidModeExtension" missing on display ":0.0"
Xlib: extension "XFree86-DGA" missing on display ":0.0"
```

You can solve this problem by adding the line below to your X config file's "Module" section:

```
    Load "extmod"
```

Q. Where can I find older driver versions?

A. Please visit ftp://download.nvidia.com/XFree86/FreeBSD-x86_64/ (new packages for FreeBSD 7.3 and newer) or ftp://download.nvidia.com/freebsd/ (old i386-only packages for FreeBSD 5.3 through 7.2, inclusive).

Q. What is the format of a PCI Bus ID?

A. Different tools have different formats for the PCI Bus ID of a PCI device.

The X server's "BusID" X configuration file option interprets the BusID string in the format "bus@domain:device:function" (the "@domain" portion is only needed if the PCI domain is non-zero), in decimal. More specifically,

```
"%d@%d:%d:%d", bus, domain, device, function
```

in printf(3) syntax. NVIDIA X driver logging, nvidia-xconfig, and nvidia-settings match the X configuration file BusID convention.

The FreeBSD pciconf(8) utility, in contrast, prints and interprets the PCI Bus ID in the format "domain:bus:device.function" (the "domain:" portion is only needed if the PCI domain is non-zero), in decimal. More specifically,

```
"pci%d:%d:%d:%d", domain, bus, device, function
```

in printf(3) syntax.

Q. How do I interpret X server version numbers?

A. X server version numbers can be difficult to interpret because some X.Org X servers report the versions of different things.

In 2003, X.Org created a fork of the XFree86 project's code base, which used a monolithic build system to build the X server, libraries, and applications together in one source code repository. It resumed the release version numbering where it left off in 2001, continuing with 6.7, 6.8, etc., for the releases of this large bundle of code. These version numbers are sometimes written X11R6.7, X11R6.8, etc. to include the version of the X protocol.

In 2005, an effort was made to split the monolithic code base into separate modules with their own version numbers to make them easier to maintain and so that they could be released independently. X.Org still occasionally releases these modules together, with a single version number. These releases are simply referred to as "X.Org releases", or sometimes "katamari" releases. For example, X.Org 7.6 was released on December 20, 2010 and contains version 1.9.3 of the xorg-server package, which contains the core X server itself.

The release management changes from XFree86, to X.Org monolithic releases, to X.Org modular releases impacted the behavior of the X server's "-version" command line option. For example, XFree86 X servers always report the version of the XFree86 monolithic package:

```
XFree86 Version 4.3.0 (Red Hat Linux release: 4.3.0-2)
Release Date: 27 February 2003
X Protocol Version 11, Revision 0, Release 6.6
```

X servers in X.Org monolithic and early "katamari" releases did something similar:

```
X Window System Version 7.1.1
Release Date: 12 May 2006
X Protocol Version 11, Revision 0, Release 7.1.1
```

However, X.Org later modified the X server to start printing its individual module version number instead:

```
X.Org X Server 1.9.3
Release Date: 2010-12-13
X Protocol Version 11, Revision 0
```

Please keep this in mind when comparing X server versions: what looks like "version 7.x" is OLDER than version 1.x.

Chapter 8. Common Problems

This section provides solutions to common problems associated with the NVIDIA FreeBSD x86_64 Driver.

Q. My X server fails to start, and my X log file contains the error:

```
(EE) NVIDIA(0): The NVIDIA kernel module does not appear to
(EE) NVIDIA(0):     be receiving interrupts generated by the NVIDIA
graphics
(EE) NVIDIA(0):     device PCI:x:x:x. Please see the COMMON PROBLEMS
(EE) NVIDIA(0):     section in the README for additional information.
```

A. This can be caused by a variety of problems, such as PCI IRQ routing errors, I/O APIC problems or conflicts with other devices sharing the IRQ (or their drivers).

If possible, configure your system such that your graphics card does not share its IRQ with other devices (try moving the graphics card to another slot if applicable, unload/disable the driver(s) for the device(s) sharing the card's IRQ, or remove/disable the device(s)).

Q. My X server fails to start, and my X log file contains the error:

```
(EE) NVIDIA(0): The interrupt for NVIDIA graphics device PCI:x:x:x
(EE) NVIDIA(0):     appears to be edge-triggered. Please see the COMMON
(EE) NVIDIA(0):     PROBLEMS section in the README for additional
information.
```

A. An edge-triggered interrupt means that the kernel has programmed the interrupt as edge-triggered rather than level-triggered in the Advanced

Programmable Interrupt Controller (APIC). Edge-triggered interrupts are not intended to be used for sharing an interrupt line between multiple devices; level-triggered interrupts are the intended trigger for such usage. When using edge-triggered interrupts, it is common for device drivers using that interrupt line to stop receiving interrupts. This would appear to the end user as those devices no longer working, and potentially as a full system hang. These problems tend to be more common when multiple devices are sharing that interrupt line.

Q. X starts for me, but OpenGL applications terminate immediately.

A. If X starts but you have trouble with OpenGL, you most likely have a problem with other libraries in the way, or there are stale symlinks. See Chapter 4 for details.

You should also check that the correct extensions are present;

```
% xdpyinfo
```

should show the "GLX" and "NV-GLX" extensions present. If these two extensions are not present, then there is most likely a problem loading the glx module, or it is unable to implicitly load GLcore. Check your X config file and make sure that you are loading glx (see Chapter 6). If your X config file is correct, then check the X log file for warnings/errors pertaining to GLX. Also check that all of the necessary symlinks are in place (refer to Chapter 4).

Q. When Xinerama is enabled, my stereo glasses are shuttering only when the stereo application is displayed on one specific X screen. When the application is displayed on the other X screens, the stereo glasses stop shuttering.

A. This problem occurs with DDC and "blue line" stereo glasses, that get the stereo signal from one video port of the graphics card. When a X screen does not display any stereo drawable the stereo signal is disabled on the associated video port.

Forcing stereo flipping allows the stereo glasses to shutter continuously. This can be done by enabling the OpenGL control "Force Stereo Flipping" in nvidia-settings, or by setting the X configuration option "ForceStereoFlipping" to "1".

Q. Stereo is not in sync across multiple displays.

A. There are two cases where this may occur. If the displays are attached to the same GPU, and one of them is out of sync with the stereo glasses, you will need to reconfigure your monitors to drive identical mode timings; see Chapter 18 for details.

If the displays are attached to different GPUs, the only way to synchronize stereo across the displays is with a G-Sync device, which is only supported by certain Quadro cards. See Chapter 25 for details. This applies to seperate GPUs on seperate cards as well as seperate GPUs on the same card, such as Quadro FX 4500 X2. Note that the Quadro FX 4500 X2 only provides a single DIN connector for stereo, tied to the bottommost GPU. In order to synchronize onboard stereo on the other GPU you must use a G-Sync device.

Q. X fails to start, and during boot up time I get error messages

```
nvidia0: NVRM: NVIDIA REG resource alloc failed.
```

or

```
nvidia0: NVRM: NVIDIA IRQ resource alloc failed.
```

A. The system BIOS has not properly set up your graphics card; FreeBSD can't currently set up PCI devices that the BIOS leaves unconfigured. Uncheck "PNP-OS" in your system BIOS.

Q. X fails to start, and during boot up time I get the following error message:

```
nvidia0: NVRM: NVIDIA MEM resource alloc failed.
```

A. On certain FreeBSD kernels, it may be necessary to add the following line to '/boot/loader.conf':

```
hw.pci.allow_unsupported_io_range="1"
```

This should allow the NVIDIA kernel module to attach.

Q. My X server fails to start, and my X log file contains the error:

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

A. Nothing will work if the NVIDIA kernel module does not function properly. If you see anything in the X log file like

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

then there is most likely a problem with the NVIDIA kernel module.

The NVIDIA kernel module may print error messages indicating a problem -- to view these messages check the output of `dmesg`, '/var/log/messages', or wherever syslog is directed to place kernel messages. These messages are prepended with "NVRM".

Q. When I attempt to start `nvidia-settings`, I get an error message of the form:

```
 Shared object "libgtk-x11-2.0.so.400" not found, required by
nvidia-settings
```

A. Due to differences between the gtk+-2.x ports packages included with different FreeBSD releases, the prebuilt nvidia-settings binary shipped with the NVIDIA driver may not work with FreeBSD releases more recent than FreeBSD 7.3.

If you have a recent ports package of gtk+-2.x and gmake installed on your system, you can build the nvidia-installer utility from source to solve this problem.

Download nvidia-settings-304.43.tar.bz2 from ftp://download.nvidia.com/XFree86/nvidia-settings You can then extract, build and install it (to '/usr/local/bin') with:

```
    % gmake install
```

Q. When I attempt to run `nvidia-xconfig` after the NVIDIA FreeBSD graphics driver installation, I get an error message of the form:

```
nvidia-xconfig: Command not found.
```

A. Depending on the shell you are using, you may need to force it to recompute its internal table of executable files present in the directories listed in the '$PATH' variable. Assuming you are using the FreeBSD default shell you can do so by issuing the command:

```
% rehash
```

Q. My system runs, but seems unstable.

A. If you're using an AGP card, your stability problems may be AGP-related. See Chapter 11 for details.

Q. OpenGL applications are running slowly

A. The application is probably using a different library that still remains on your system, rather than the NVIDIA supplied OpenGL library. See Chapter 4 for details.

Q. There are problems running Quake2.

A. Quake2 requires some minor setup to get it going. First, in the Quake2 directory, the install creates a symlink called 'libGL.so' that points at 'libMesaGL.so'. This symlink should be removed or renamed. Second, in order to run Quake2 in OpenGL mode, you must type

```
% 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 same resolution as Quake2 to emulate full-screen mode.

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

A. Most of the X startup delay problems we have found are caused by incorrect data in video BIOSes about what display devices are possibly connected or what i2c port should be used for detection. You can work around these problems with the X config option IgnoreDisplayDevices (see the description in Appendix B).

Q. Fonts are incorrectly sized after installing the NVIDIA driver.

A. Incorrectly sized fonts are generally caused by incorrect DPI (Dots Per Inch) information. You can check what X thinks the physical size of your monitor is, by running:

```
 % xdpyinfo | grep dimensions
```

This will report the size in pixels, and in millimeters.

If these numbers are wrong, you can correct them by modifying the X server's DPI setting. See Appendix E for details.

Q. General problems with ALi chipsets

A. There are some known timing and signal integrity issues on ALi chipsets. The following tips may help stabilize problematic ALI systems:

- o Disable TURBO AGP MODE in the BIOS.

- o When using a P5A upgrade to BIOS Revision 1002 BETA 2.

- o When using 1007, 1007A or 1009 adjust the IO Recovery Time to 4 cycles.

- o AGP is disabled by default on some ALi chipsets (ALi1541, ALi1647) to work around severe system stability problems with these chipsets. See the comments for EnableALiAGP in 'nv-reg.h' to force AGP on anyway.

Q. Using GNOME configuration utilities, I am unable to get a resolution above

800x600.

A. The installation of GNOME provided in operating systems such as FreeBSD 7
contain several competing interfaces for specifying resolution:

```
    'System Settings' -> 'Display'
```

which will update the X configuration file, and

```
    'Applications' -> 'Preferences' -> 'Screen Resolution'
```

which will update the per-user screen resolution using the XRandR
extension. Your desktop resolution will be limited to the smaller of the
two settings. Be sure to check the setting of each.

Q. OpenGL applications don't work, and my X log file contains the error:

```
(EE) NVIDIA(0): Unable to map device node /dev/zero with read and write
(EE) NVIDIA(0):     privileges.  The GLX extension will be disabled on this

(EE) NVIDIA(0):     X screen.  Please see the COMMON PROBLEMS section in
the
(EE) NVIDIA(0):     README for more information.
```

A. The NVIDIA OpenGL driver must be able to map anonymous memory with read and
write execute privileges in order to function correctly. The driver needs
this ability to allocate aligned memory, which is used for certain
optimizations. Currently, GLX cannot run without these optimizations.

Q. X doesn't start, and my log file contains a message like the following:

```
(EE) NVIDIA(0): Failed to allocate primary buffer: failed to set CPU access
(EE) NVIDIA(0):     for surface.  Please see Chapter 8: Common Problems in
(EE) NVIDIA(0):     the README for troubleshooting suggestions.
```

A. The NVIDIA X driver needs to be able to access the buffers it allocates
from the CPU, but wasn't able to set up this access. This commonly fails if
you're using a large virtual desktop size. Although your GPU may have
enough onboard video memory for the buffer, the amount of usable memory may
be limited if the "IndirectMemoryAccess" option is disabled, or if not
enough address space was reserved for indirect memory access (this commonly
occurs on 32-bit systems). If you're seeing this problem and are using a
32-bit operating system, it may be resolved by switching to a 64-bit
operating system.

Q. My log file contains a message like the following:

```
    (WW) NVIDIA(GPU-0): Unable to enter interactive mode, because
    non-interactive
    (WW) NVIDIA(GPU-0): mode has been previously requested.  The most common
    (WW) NVIDIA(GPU-0): cause is that a GPU compute application is currently
    (WW) NVIDIA(GPU-0): running. Please see the README for details.
```

A. This indicates that the X driver was not able to put the GPU in interactive
mode, because another program has requested non-interactive mode. The GPU
watchdog will not run, and long-running GPU compute programs may cause the
X server and OpenGL programs to hang. If you intend to run long-running GPU
compute programs, set the "Interactive" option to "off" to disable
interactive mode.

Chapter 9. Known Issues

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

Known Issues

Notebooks

If you are using a notebook see the "Known Notebook Issues" in Chapter 17.

Texture seams in Quake 3 engine

Many games based on the Quake 3 engine set their textures to use the "GL_CLAMP" clamping mode when they should be using "GL_CLAMP_TO_EDGE". This was an oversight made by the developers because some legacy NVIDIA GPUs treat the two modes as equivalent. The result is seams at the edges of textures in these games. To mitigate this, older versions of the NVIDIA display driver remap "GL_CLAMP" to "GL_CLAMP_TO_EDGE" internally to emulate the behavior of the older GPUs, but this workaround has been disabled by default. To re-enable it, uncheck the "Use Conformant Texture Clamping" checkbox in nvidia-settings before starting any affected applications.

FSAA

When FSAA is enabled (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.

libGL DSO finalizer and pthreads

When a multithreaded OpenGL application exits, it is possible for libGL's DSO finalizer (also known as the destructor, or "_fini") to be called while other threads are executing OpenGL code. The finalizer needs to free resources allocated by libGL. This can cause problems for threads that are still using these resources. Setting the environment variable "__GL_NO_DSO_FINALIZER" to "1" will work around this problem by forcing libGL's finalizer to leave its resources in place. These resources will still be reclaimed by the operating system when the process exits. Note that the finalizer is also executed as part of dlclose(3), so if you have an application that dlopens(3) and dlcloses(3) libGL repeatedly, "__GL_NO_DSO_FINALIZER" will cause libGL to leak resources until the process exits. Using this option can improve stability in some multithreaded applications, including Java3D applications.

Thread cancellation

Canceling a thread (see pthread_cancel(3)) while it is executing in the OpenGL driver causes undefined behavior. For applications that wish to use thread cancellation, it is recommended that threads disable cancellation using pthread_setcancelstate(3) while executing OpenGL or GLX commands.

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

Problems that Will Not Be Fixed

Gigabyte GA-6BX Motherboard

This motherboard uses a LinFinity regulator on the 3.3 V rail that is only rated to 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 GPU 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 GPU 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 Chip sets with AGP 2x

On 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.

Irongate Chip sets with AGP 1x

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

ALi chipsets, ALi1541 and ALi1647

On ALi1541 and ALi1647 chipsets, NVIDIA drivers disable AGP to work around timing issues and signal integrity issues. See Chapter 8 for more information on ALi chipsets.

NV-CONTROL versions 1.8 and 1.9

Version 1.8 of the NV-CONTROL X Extension introduced target types for setting and querying attributes as well as receiving event notification on targets. Targets are objects like X Screens, GPUs and G-Sync devices. Previously, all attributes were described relative to an X Screen. These new bits of information (target type and target id) were packed in a non-compatible way in the protocol stream such that addressing X Screen 1 or higher would generate an X protocol error when mixing NV-CONTROL client and server versions.

This packing problem has been fixed in the NV-CONTROL 1.10 protocol, making it possible for the older (1.7 and prior) clients to communicate with NV-CONTROL 1.10 servers. Furthermore, the NV-CONTROL 1.10 client library has been updated to accommodate the target protocol packing bug when communicating with a 1.8 or 1.9 NV-CONTROL server. This means that the NV-CONTROL 1.10 client library should be able to communicate with any version of the NV-CONTROL server.

NVIDIA recommends that NV-CONTROL client applications relink with version 1.10 or later of the NV-CONTROL client library (libXNVCtrl.a, in the nvidia-settings-304.43.tar.bz2 tarball). The version of the client library can be determined by checking the NV_CONTROL_MAJOR and NV_CONTROL_MINOR definitions in the accompanying nv_control.h.

The only web released NVIDIA FreeBSD driver that is affected by this problem (i.e., the only driver to use either version 1.8 or 1.9 of the NV-CONTROL X extension) is 1.0-8756.

CPU throttling reducing memory bandwidth on IGP systems

For some models of CPU, the CPU throttling technology may affect not only CPU core frequency, but also memory frequency/bandwidth. On systems using integrated graphics, any reduction in memory bandwidth will affect the GPU as well as the CPU. This can negatively affect applications that use significant memory bandwidth, such as video decoding using VDPAU, or certain OpenGL operations. This may cause such applications to run with lower performance than desired.

To work around this problem, NVIDIA recommends configuring your CPU throttling implementation to avoid reducing memory bandwidth. This may be as simple as setting a certain minimum frequency for the CPU.

Depending on your operating system and/or distribution, this may be as simple as writing to a configuration file in the /sys or /proc filesystems, or other system configuration file. Please read, or search the Internet for, documentation regarding CPU throttling on your operating system.

VDPAU initialization failures on supported GPUs

If VDPAU gives the VDP_STATUS_NO_IMPLEMENTATION error message on a GPU which was labeled or specified as supporting PureVideo or PureVideo HD, one possible reason is a hardware defect. After ruling out any other software problems, NVIDIA recommends returning the GPU to the manufacturer for a replacement.

Some applications, such as Quake 3, crash after querying the OpenGL extension string

Some applications have bugs that are triggered when the extension string is longer than a certain size. As more features are added to the driver, the length of this string increases and can trigger these sorts of bugs.

You can limit the extensions listed in the OpenGL extension string to the ones that appeared in a particular version of the driver by setting the "__GL_ExtensionStringVersion" environment variable to a particular version number. For example,

```
__GL_ExtensionStringVersion=17700 quake3
```

will run Quake 3 with the extension string that appeared in the 177.* driver series. Limiting the size of the extension string can work around this sort of application bug.

XVideo and the Composite X extension

XVideo will not work correctly when Composite is enabled unless using X.Org 7.1 or later. See Chapter 22.

GLX visuals in Xinerama

X servers prior to version 1.5.0 have a limitation in the number of visuals that can be available when Xinerama is enabled. Specifically, visuals with ID values over 255 will cause the server to corrupt memory, leading to incorrect behavior or crashes. In some configurations where many GLX features are enabled at once, the number of GLX visuals will exceed this limit. To avoid a crash, the NVIDIA X driver will discard visuals above the limit. To see which visuals are being discarded, run the X server with the -logverbose 6 option and then check the X server log file.

Please see "Q. How do I interpret X server version numbers?" in Chapter 7 when determining whether your X server is new enough to contain this fix.

Some X servers have trouble with multiple GPUs

Some versions of the X.Org X server starting with 1.5.0 have a bug that causes X to fail with an error similar to the following when there is more than one GPU in the computer:

```
(!!) More than one possible primary device found
(II) Primary Device is:
(EE) No devices detected.

Fatal server error:
no screens found
```

This bug was fixed in the X.Org X Server 1.7 release.

You can work around this problem by specifying the bus ID of the device you wish to use. For more details, please search the xorg.conf manual page for "BusID". You can configure the X server with an X screen on each NVIDIA GPU by running:

```
nvidia-xconfig --enable-all-gpus
```

Please see http://bugs.freedesktop.org/show_bug.cgi?id=18321 for more details on this X server problem. In addition, please see "Q. How do I interpret X server version numbers?" in Chapter 7 when determining whether your X server is new enough to contain this fix.

Chapter 10. Specifying OpenGL Environment Variable Settings

10A. 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. By setting the appropriate environment variable, you can enable full-scene antialiasing in any OpenGL application on these GPUs.

Several antialiasing methods are available and you can select between them by setting the __GL_FSAA_MODE environment variable appropriately. Note that increasing the number of samples taken during FSAA rendering may decrease performance.

To see the available values for __GL_FSAA_MODE along with their descriptions, run:

```
    nvidia-settings --query=fsaa --verbose
```

The __GL_FSAA_MODE environment variable uses the same integer values that are used to configure FSAA through nvidia-settings and the NV-CONTROL X extension. In other words, these two commands are equivalent:

```
    export __GL_FSAA_MODE=5

    nvidia-settings --assign FSAA=5
```

Note that there are three FSAA related configuration attributes (FSAA, FSAAAppControlled and FSAAAppEnhanced) which together determine how a GL application will behave. If FSAAAppControlled is 1, the FSAA specified through nvidia-settings will be ignored, in favor of what the application requests through FBConfig selection. If FSAAAppControlled is 0 but FSAAAppEnhanced is 1, then the FSAA value specified through nvidia-settings will only be applied if the application selected a multisample FBConfig.

Therefore, to be completely correct, the nvidia-settings command line to unconditionally assign FSAA should be:

```
    nvidia-settings --assign FSAA=5 --assign FSAAAppControlled=0 --assign
FSAAAppEnhanced=0
```

The driver may not be able to support a particular FSAA mode for a given application due to video or system memory limitations. In that case, the driver will silently fall back to a less demanding FSAA mode.

10B. ANISOTROPIC TEXTURE FILTERING

Automatic anisotropic texture filtering can be enabled by setting the environment variable __GL_LOG_MAX_ANISO. The possible values are:

__GL_LOG_MAX_ANISO	Filtering Type
0	No anisotropic filtering
1	2x anisotropic filtering
2	4x anisotropic filtering
3	8x anisotropic filtering
4	16x anisotropic filtering

4x and greater are only available on GeForce3 or newer GPUs; 16x is only available on GeForce 6800 or newer GPUs.

10C. VBLANK SYNCING

The __GL_SYNC_TO_VBLANK (boolean) environment variable can be used to control whether swaps are synchronized to a display device's vertical refresh.

o Setting __GL_SYNC_TO_VBLANK=0 allows glXSwapBuffers to swap without waiting for vblank.

o Setting __GL_SYNC_TO_VBLANK=1 forces glXSwapBuffers to synchronize with the vertical blanking period. This is the default behavior.

When sync to vblank is enabled with TwinView, OpenGL can only sync to one of the display devices; this may cause tearing corruption on the display device to which OpenGL is not syncing. You can use the environment variable __GL_SYNC_DISPLAY_DEVICE to specify to which display device OpenGL should sync. You should set this environment variable to the name of a display device; for example "CRT-1". Look for the line "Connected display device(s):" in your X log file for a list of the display devices present and their names. You may also find it useful to review Chapter 12 "Configuring Twinview" and the section on Ensuring Identical Mode Timings in Chapter 18.

10D. CONTROLLING THE SORTING OF OPENGL FBCONFIGS

The NVIDIA GLX implementation sorts FBConfigs returned by glXChooseFBConfig()
as described in the GLX specification. To disable this behavior set
__GL_SORT_FBCONFIGS to 0 (zero), then FBConfigs will be returned in the order
they were received from the X server. To examine the order in which FBConfigs
are returned by the X server run:

```
nvidia-settings --glxinfo
```

This option may be be useful to work around problems in which applications
pick an unexpected FBConfig.

10E. OPENGL YIELD BEHAVIOR

There are several cases where the NVIDIA OpenGL driver needs to wait for
external state to change before continuing. To avoid consuming too much CPU
time in these cases, the driver will sometimes yield so the kernel can
schedule other processes to run while the driver waits. For example, when
waiting for free space in a command buffer, if the free space has not become
available after a certain number of iterations, the driver will yield before
it continues to loop.

By default, the driver calls sched_yield() to do this. However, this can cause
the calling process to be scheduled out for a relatively long period of time
if there are other, same-priority processes competing for time on the CPU. One
example of this is when an OpenGL-based composite manager is moving and
repainting a window and the X server is trying to update the window as it
moves, which are both CPU-intensive operations.

You can use the __GL_YIELD environment variable to work around these
scheduling problems. This variable allows the user to specify what the driver
should do when it wants to yield. The possible values are:

__GL_YIELD	Behavior
<unset>	By default, OpenGL will call sched_yield() to yield.
"NOTHING"	OpenGL will never yield.
"USLEEP"	OpenGL will call usleep(0) to yield.

10F. CONTROLLING WHICH OPENGL FBCONFIGS ARE AVAILABLE

The NVIDIA GLX implementation will hide FBConfigs that are associated with a 32-bit ARGB visual when the XLIB_SKIP_ARGB_VISUALS environment variable is defined. This matches the behavior of libX11, which will hide those visuals from XGetVisualInfo and XMatchVisualInfo. This environment variable is useful when applications are confused by the presence of these FBConfigs.

10G. USING UNOFFICIAL GLX PROTOCOL

By default, the NVIDIA GLX implementation will not expose GLX protocol for GL commands if the protocol is not considered complete. Protocol could be considered incomplete for a number of reasons. The implementation could still be under development and contain known bugs, or the protocol specification itself could be under development or going through review. If users would like to test the client-side portion of such protocol when using indirect rendering, they can set the __GL_ALLOW_UNOFFICIAL_PROTOCOL environment variable to a non-zero value before starting their GLX application. When an NVIDIA GLX server is used, the related X Config option "AllowUnofficialGLXProtocol" will need to be set as well to enable support in the server.

10H. LIMITING HEAP ALLOCATIONS IN THE OPENGL DRIVER

The NVIDIA OpenGL implementation normally does not enforce limits on dynamic system memory allocations (i.e., memory allocated by the driver from the C library via the malloc(3) memory allocation package). The __GL_HEAP_ALLOC_LIMIT environment variable enables the user to specify a per-process heap allocation limit for as long as libGL is loaded in the application.

__GL_HEAP_ALLOC_LIMIT is specified in the form BYTES SUFFIX, where BYTES is a nonnegative integer and SUFFIX is an optional multiplicative suffix: kB = 1000, k = 1024, MB = 1000*1000, M = 1024*1024, GB = 1000*1000*1000, and G = 1024*1024*1024. SUFFIX is not case-sensitive. For example, to specify a heap allocation limit of 20 megabytes:

```
__GL_HEAP_ALLOC_LIMIT="20 MB"
```

If SUFFIX is not specified, the limit is assumed to be given in bytes. The minimum heap allocation limit is 12 MB. If a lower limit is specified, the limit is clamped to the minimum.

WARNING: Enforcing a limit on heap allocations may cause unintended behavior
and lead to application crashes, data corruption, and system instability.
ENABLE AT YOUR OWN RISK.

10I. OPENGL SHADER DISK CACHE

The NVIDIA OpenGL driver utilizes a shader disk cache. This optimization benefits some applications, by reusing shader binaries instead of compiling them repeatedly. The related environment variables __GL_SHADER_DISK_CACHE and __GL_SHADER_DISK_CACHE_PATH, as well as the GLShaderDiskCache X configuration option, allow fine-grained configuration of the shader cache behavior. The shader disk cache:

1. is always disabled for indirect rendering

2. is always disabled for setuid and setgid binaries

3. by default, is disabled for direct rendering when the OpenGL application
   is run as the root user

4. by default, is enabled for direct rendering when the OpenGL application
   is run as a non-root user

The GLShaderDiskCache X configuration option forcibly enables or disables the shader disk cache, for direct rendering as a non-root user.

By default, caches are stored in $HOME/.nv/GLCache. Caches are persistent across runs of an application. Cached shader binaries are specific to each driver version; changing driver versions will cause binaries to be recompiled.

The following environment variables configure shader disk cache behavior, and override the GLShaderDiskCache configuration option:

```
    Environment Variable                 Description
    ---------------------------------    ---------------------------------
    __GL_SHADER_DISK_CACHE (boolean)      Enables or disables the shader
                                          cache for direct rendering.
    __GL_SHADER_DISK_CACHE_PATH           Enables configuration of where
    (string)                              shader caches are stored on disk.
```

Chapter 11. Configuring AGP

There are several choices for configuring the NVIDIA kernel module's use of
AGP: you can choose to either use the NVIDIA AGP module (NVAGP), or the AGP
module that comes with the FreeBSD kernel (AGPGART). This is controlled
through the "NvAGP" option in your X config file:

```
    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
```

Unlike other operating systems such as Linux, this option is not the only
controlling factor at this point; because of known problems, 'nvidia.ko' is
built without support for FreeBSD's AGP driver by default. This behavior can
be changed, see 'nv-freebsd.h' for details.

Note that if you built nvidia.ko with support for the FreeBSD driver it will
not load unless 'agp.ko' is loaded. 'agp.ko' is special in that you can not
load it after the system boot is complete, you need to append the following
line to '/boot/loader.conf' to make sure it is pre-loaded:

```
    # -- load FreeBSD AGP GART driver -- #
    agp_load="YES"
```

Also note that if 'agp.ko' is loaded, it could conflict with the NVIDIA AGP
GART driver (NvAGP), resulting in stability problems; for this reason, the
NVIDIA driver will abort NvAGP initialization when it detects 'agp.ko'.

Current FreeBSD releases are shipped with 'agp.ko' built into the kernel; in
order to allow NvAGP to work, the kernel can be rebuilt without 'device agp'
or the following entry added to '/boot/device.hints':

```
    hint.agp.0.disabled="1"
```

When built with support for the FreeBSD AGP driver, 'nvidia.ko' will fall back
to using NvAGP when it doesn't detect 'agp.ko' (this will be the case when
'agp.ko' does not support your AGP chipset or was explicitly disabled with

device hints).

It is highly recommended that you use the NVIDIA AGP driver.

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

```
    Supported AGP Chipsets
    ----------------------------------------------------------------------
    Intel 440LX
    Intel 440BX
    Intel 440GX
    Intel 815 ("Solano")
    Intel 820 ("Camino")
    Intel 830M
    Intel 840 ("Carmel")
    Intel 845 ("Brookdale")
    Intel 845G
    Intel 850 ("Tehama")
    Intel 855 ("Odem")
    Intel 860 ("Colusa")
    Intel 865G ("Springdale")
    Intel 875P ("Canterwood")
    Intel E7205 ("Granite Bay")
    Intel E7505 ("Placer")
    AMD 751 ("Irongate")
    AMD 761 ("IGD4")
    AMD 762 ("IGD4 MP")
    AMD 8151 ("Lokar")
    VIA 8371
    VIA 82C694X
    VIA KT133
    VIA KT266
    VIA KT400
    VIA P4M266
    VIA P4M266A
    VIA P4X400
    VIA K8T800
    VIA K8N800
    VIA PT880
    VIA KT880
    RCC CNB20LE
    RCC 6585HE
```

Micron SAMDDR ("Samurai")
Micron SCIDDR ("Scimitar")
NVIDIA nForce
NVIDIA nForce2
NVIDIA nForce3
ALi 1621
ALi 1631
ALi 1647
ALi 1651
ALi 1671
SiS 630
SiS 633
SiS 635
SiS 645
SiS 646
SiS 648
SiS 648FX
SiS 650
SiS 651
SiS 655
SiS 655FX
SiS 661
SiS 730
SiS 733
SiS 735
SiS 745
SiS 755
ATI RS200M

If you are experiencing AGP stability problems, you should be aware of the following:

Additional AGP Information

AGP drive strength BIOS setting (Via-based motherboards)

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

If you decide to experiment with this, you need to be aware of the fact that you are doing so at your own risk and that you may render your system unbootable with improper settings until you reset the setting to a working value (w/ 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 motherboard manufacturer.

On ALi1541 and ALi1647 chipsets, NVIDIA drivers disable AGP to work around timing and signal integrity problems. You can force AGP to be enabled on these chipsets by setting NVreg_EnableALiAGP to 1. Note that this may cause the system to become unstable.

Early system BIOS revisions for the ASUS A7V8X-X KT400 motherboard misconfigure the chipset when an AGP 2.x graphics card is installed; if X hangs on your ASUS KT400 system with NvAGP enabled and the installed graphics card is not an AGP 8x device, make sure that you have the latest system BIOS installed.

Chapter 12. Configuring Multiple Display Devices on One X Screen

---

Multiple display devices (digital flat panels, CRTs, and TVs) can display the contents of a single X screen in any arbitrary configuration. Configuring multiple display devices on a single X screen has several distinct advantages over other techniques (such as Xinerama):

o A single X screen is used. The NVIDIA driver conceals all information about multiple display devices from the X server; as far as X is concerned, there is only one screen.

o Both display devices share one frame buffer. Thus, all the functionality present on a single display (e.g., accelerated OpenGL) is available with multiple display devices.

o No additional overhead is needed to emulate having a single desktop.

If you are interested in using each display device as a separate X screen, see Chapter 14.

12A. RELEVANT X CONFIGURATION OPTIONS

When the NVIDIA X driver starts, by default it will enable as many display devices as are connected and as the GPU supports driving simultaneously. Most NVIDIA GPUs based on the Kepler architecture, or newer, support driving up to four display devices simultaneously. Most NVIDIA GPUs older than Kepler support driving up to two display devices simultaneously.

If multiple X screens are configured on the GPU, the NVIDIA X driver will attempt to reserve display devices and GPU resources for those other X screens (honoring the "UseDisplayDevice" and "MetaModes" X configuration options of each X screen) and then allocate all remaining resources to the first X screen configured on the GPU.

There are several X configuration options that influence how multiple display devices are used by an X screen:

```
    Option "MetaModes"               "<list of MetaModes>"

    Option "HorizSync"               "<hsync range(s)>"
    Option "VertRefresh"             "<vrefresh range(s)>"

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

See detailed descriptions of each option below.

12B. DETAILED DESCRIPTION OF OPTIONS

HorizSync
VertRefresh

With these options, you can specify a semicolon-separated list of frequency ranges, each optionally prepended with a display device name. In addition, if SLI Mosaic mode is enabled, a GPU specifier can be used. For example:

```
    Option "HorizSync"   "CRT-0: 50-110; DFP-0: 40-70"
    Option "VertRefresh" "CRT-0: 60-120; GPU-0.DFP-0: 60"
```

See Appendix C on Display Device Names for more information.

These options are normally not needed: by default, the NVIDIA X driver retrieves the valid frequency ranges from the display device's EDID (see Appendix B for a description of the "UseEdidFreqs" option). The "HorizSync" and "VertRefresh" options override any frequency ranges retrieved from the EDID.

MetaModes

MetaModes are "containers" that store information about what mode should be used on each display device.

Multiple MetaModes list the combinations of modes and the sequence in which they should be used. In MetaMode syntax, modes within a MetaMode are comma separated, 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. Here is an example MetaMode:

```
    Option "MetaModes" "1280x1024,1280x1024; 1024x768,1024x768"
```

If you want a display device to not be active for a certain MetaMode, you can use the mode name "NULL", or simply omit the mode name entirely:

```
    "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;

e.g.,

```
    "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 X config file.

When no offsets are given for a MetaMode, the offsets will be computed
following the value of the MetaModeOrientation option (see below). Note
that if offsets are given for any one of the modes in a single MetaMode,
then offsets will be expected for all modes within that single MetaMode;
in such a case offsets will be assumed to be +0+0 when not given.

When not explicitly given, the virtual screen size will be computed as the
the bounding box of all MetaMode bounding boxes. MetaModes with a bounding
box larger than an explicitly given virtual screen size will be discarded.

A MetaMode string can be further modified with a "Panning Domain"
specification; e.g.,

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

A panning domain is the area in which a display device's viewport will be
panned to follow the mouse. Panning actually happens on two levels with
MetaModes: first, an individual display device's viewport will be panned
within its panning domain, as long as the viewport is contained by the
bounding box of the MetaMode. Once the mouse leaves the bounding box of
the MetaMode, the entire MetaMode (i.e., all display devices) will be
panned to follow the mouse within the virtual screen, unless the
"PanAllDisplays" X configuration option is disabled. Note that individual
display devices' panning domains default to being clamped to the position
of the display devices' viewports, thus the default behavior is just that
viewports remain "locked" together and only perform the second type of
panning.

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

```
    "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 (note that 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 describes two modes, each with a panning domain width of 1900 pixels, and the second display is positioned below the first:

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

Because it is often unclear which mode within a MetaMode will be used on each display device, mode descriptions within a MetaMode can be prepended with a display device name. For example:

```
    "CRT-0: 1600x1200,  DFP-0: 1024x768"
```

If no MetaMode string is specified, then the X driver uses the modes listed in the relevant "Display" subsection, attempting to place matching modes on each display device.

Each mode of the MetaMode may also have extra attributes associated with it, specified as a comma-separated list of token=value pairs inside curly brackets. The value for each token can optionally be enclosed in parentheses, to prevent commas within the value from being interpreted as token=value pair separators. Currently, the only token that requires a parentheses-enclosed value is "Transform".

The possible tokens within the curly bracket list are:

o "Stereo": possible values are "PassiveLeft" or "PassiveRight". When used in conjunction with stereo mode "4", this allows each display to be configured independently to show any stereo eye. For example:

```
     "CRT-0: 1600x1200 +0+0 { Stereo = PassiveLeft }, CRT-1: 1600x1200
  +1600+0 { Stereo=PassiveRight }"
```

If the X screen is not configured for stereo mode "4", these options are ignored. See Appendix B for more details about stereo configurations.

o "Rotation": this rotates the content of an individual display device. Possible values are "0" (with synonyms "no", "off" and "normal"), "90" (with synonyms "left" and "CCW"), "180" (with synonyms "invert" and "inverted") and "270" (with synonyms "right" and "CW"). For example:

```
    "DFP-0: nvidia-auto-select { Rotation=left }, DFP-1:
nvidia-auto-select { Rotation=right }"
```

Independent rotation configurability of each display device is also possible through RandR. See Chapter 16 for details.

o "Reflection": this reflects the content of an individual display device about either the X axis, the Y axis, or both the X and Y axes. Possible values are "X", "Y" and "XY". For example:

```
    "DFP-0: nvidia-auto-select { Reflection=X }, DFP-1:
nvidia-auto-select"
```

Independent reflection configurability of each display device is also possible through RandR. See Chapter 16 for details.

o "Transform": this is a 3x3 matrix of floating point values that defines a transformation from the ViewPortOut for a display device to a region within the X screen. This is equivalent to the transformation matrix specified through the RandR 1.3 RRSetCrtcTransform request. As in RandR, the transform is applied before any specified rotation and reflection values to compute the complete transform.

The 3x3 matrix is represented in the MetaMode syntax as a comma-separated list of nine floating point values, stored in row-major order. This is the same as the value passed to the xrandr(1) '--transform' command line option.

Note that the transform value must be enclosed in parentheses, so that the commas separating the nine floating point values are interpreted correctly.

For example:

```
    "DFP-0: nvidia-auto-select { Transform=(43.864288330078125,
  21.333328247070312, -16384, 0, 43.864288330078125, 0, 0,
  0.0321197509765625, 19.190628051757812) }"
```

o "ViewPortOut": this specifies the region within the mode sent to the display device that will display pixels from the X screen. The region of the mode outside the ViewPortOut will contain black. The format is "WIDTH x HEIGHT +X +Y".

This is useful, for example, for configuring overscan compensation. E.g., if the mode sent to the display device is 1920x1080, to configure a 10 pixel border on all four sides:

```
    "DFP-0: 1920x1080 { ViewPortOut=1900x1060+10+10 }"
```

Or, to only display an image in the lower right quarter of the 1920x1080 mode:

```
    "DFP-0: 1920x1080 { ViewPortOut=960x540+960+540 }"
```

When not specified, the ViewPortOut defaults to the size of the mode.

ViewPortOut is only available on G80 and later GPUs.

o "ViewPortIn": this defines the size of the region of the X screen which will be displayed within the ViewPortOut. The format is "WIDTH x HEIGHT".

ViewPortIn is useful for configuring scaling between the X screen and the display device. For example, to display an 800x600 region from the X screen on a 1920x1200 mode:

```
    "DFP-0: 1920x1200 { ViewPortIn=800x600 }"
```

Or, to display a 2560x1600 region from the X screen on a 1920x1200 mode:

```
    "DFP-0: 1920x1200 { ViewPortIn=2560x1600 }"
```

Or, in conjunction with ViewPortOut, to scale an 800x600 region of the X screen within a 1920x1200 mode while preserving the aspect ratio:

```
    "DFP-0: 1920x1200 { ViewPortIn=800x600, ViewPortOut=1600x1200+160+0
    }"
```

Scaling from ViewPortIn to ViewPortOut is also expressible through the "Transform" attribute. In fact, ViewPortIn is just a shortcut for populating the transformation matrix. If both ViewPortIn and Transform are specified in the MetaMode for a display device, ViewPortIn is ignored.

Note that the current MetaMode can also be configured through the NV-CONTROL X extension and the nvidia-settings utility. For example:

```
    nvidia-settings --assign CurrentMetaMode="DFP-0: 1920x1200 {
ViewPortIn=800x600, ViewPortOut=1600x1200+160+0 }"
```

MetaModeOrientation

This option controls the positioning of the display devices within the virtual X screen, when offsets are not explicitly given in the MetaModes. The possible values are:

```
    "RightOf"  (the default)
    "LeftOf"
    "Above"
    "Below"
    "Clone"
```

When "Clone" is specified, all display devices will be assigned an offset of 0,0.

Because it is often unclear which display device relates to which, MetaModeOrientation can be confusing. You can further clarify the MetaModeOrientation with display device names to indicate which display device is positioned relative to which display device. For example:

```
    "CRT-0 LeftOf DFP-0"
```

ConnectedMonitor

With this option you can override what the NVIDIA kernel module detects is connected to your graphics card. This may be useful, for example, if any of your display devices do not support detection using Display Data Channel (DDC) protocols. Valid values are a comma-separated list of display device names; for example:

```
"CRT-0, CRT-1"
"CRT"
"CRT-1, DFP-0"
```

WARNING: this option overrides what display devices are detected by the NVIDIA kernel module, and is very seldom needed. You really only need this if a display device is not detected, either because it does not provide DDC information, or because it is on the other side of a KVM (Keyboard-Video-Mouse) switch. In most other cases, it is best not to specify this option.

Just as in all X config entries, spaces are ignored and all entries are case insensitive.

12C. DYNAMIC TWINVIEW

Using the NV-CONTROL X extension, the display devices in use by an X screen, the mode pool for each display device, and the MetaModes for each X screen can be dynamically manipulated. The "Display Configuration" page in nvidia-settings uses this functionality to modify the MetaMode list and then uses XRandR to switch between MetaModes. This gives the ability to dynamically configure TwinView.

The details of how this works are documented in the nv-control-dpy.c sample NV-CONTROL client in the nvidia-settings source tarball.

Because the NVIDIA X driver can now transition into and out of TwinView dynamically, MetaModes are always used internally by the NVIDIA X driver, regardless of how many display devices are currently in use by the X screen and regardless of whether the TwinView X configuration option was specified.

One implication of this implementation is that each MetaMode must be uniquely identifiable to the XRandR X extension. Unfortunately, two MetaModes with the same bounding box will look the same to XRandR. For example, two MetaModes with different orientations:

```
    "CRT: 1600x1200 +0+0, DFP: 1600x1200 +1600+0"
    "CRT: 1600x1200 +1600+0, DFP: 1600x1200 +0+0"
```

will look identical to the XRandR or XF86VidMode X extensions, because they have the same total size (3200x1200), and nvidia-settings would not be able to use XRandR to switch between these MetaModes. To work around this limitation, the NVIDIA X driver "lies" about the refresh rate of each MetaMode, using the refresh rate of the MetaMode as a unique identifier.

The XRandR extension is currently being redesigned by the X.Org community, so the refresh rate workaround may be removed at some point in the future. This workaround can also be disabled by setting the "DynamicTwinView" X configuration option to FALSE, which will disable NV-CONTROL support for manipulating MetaModes, but will cause the XRandR and XF86VidMode visible refresh rate to be accurate.

FREQUENTLY ASKED TWINVIEW QUESTIONS

Q. Nothing gets displayed on my second monitor; what is wrong?

A. Monitors that do not support monitor detection using Display Data Channel (DDC) protocols (this includes most older monitors) are not detectable by your NVIDIA card. You need to explicitly tell the NVIDIA X driver what you have connected using the "ConnectedMonitor" option; e.g.,

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

Q. Will window managers be able to appropriately place windows (e.g., avoiding placing windows across both display devices, or in inaccessible regions of the virtual desktop)?

Yes. Window managers can query the layout of display devices through either RandR 1.2 or Xinerama.

The NVIDIA X driver provides a Xinerama extension that X clients (such as window managers) can use to discover the current layout of display devices. Note that the Xinerama protocol provides no way to notify clients when a configuration change occurs, so if you modeswitch to a different MetaMode, your window manager may still think you have the previous configuration. Using RandR 1.2, or the Xinerama extension in conjunction with the

XF86VidMode extension to get modeswitch events, window managers should be able to determine the display device configuration at any given time.

Unfortunately, the data provided by XineramaQueryScreens() appears to confuse some window managers; to work around such broken window managers, you can disable communication of the display device layout with the "nvidiaXineramaInfo" X configuration option (see Appendix B for details).

The order that display devices are reported in via the NVIDIA Xinerama information can be configured with the nvidiaXineramaInfoOrder X configuration option.

Be aware that the NVIDIA driver cannot provide the Xinerama extension if the X server's own Xinerama extension is being used. Explicitly specifying Xinerama in the X config file or on the X server commandline will prohibit NVIDIA's Xinerama extension from installing, so make sure that the X server's log file does not contain:

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

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

Another solution is to use panning domains to eliminate inaccessible regions of the virtual screen (see the MetaMode description above).

A third solution is to use two separate X screens, rather than use TwinView. See Chapter 14.

Q. Why can I not get a resolution of 1600x1200 on the second display device when using a GeForce2 MX?

A. Because the second display device on the GeForce2 MX 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 (for a description of how Pixel Clock frequencies limit the programmable modes, see the XFree86 Video Timings HOWTO). This constraint is not present on GeForce4 or GeForce FX GPUs -- the maximum pixel clock is the same on both heads.

Q. Do video overlays work across both display devices?

A. With GPUs based on G80 and later, and some older GPUs, video overlays are available on all displays.

On some older GPUs, hardware video overlays only work on the first display device. The current solution is that blitted video is used instead on TwinView with these GPUs.

Q. How are virtual screen dimensions determined in TwinView?

A. After all requested modes have been validated, and the offsets for each MetaMode's viewports have been computed, the NVIDIA driver computes the bounding box of the panning domains for each MetaMode. The maximum bounding box width and height is then found.

Note that one side effect of this 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 from game to game, 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:

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

produce two modes: one whose resolution is 2048x768, and another whose resolution is 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 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 that, 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 head modes available as well, an appropriate MetaMode string might be something like:

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

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

Chapter 13. Configuring GLX in Xinerama

The NVIDIA FreeBSD Driver supports GLX when Xinerama is enabled on similar GPUs. The Xinerama extension takes multiple physical X screens (possibly spanning multiple GPUs), and binds them into one logical X screen. This allows windows to be dragged between GPUs and to span across multiple GPUs. The NVIDIA driver supports hardware accelerated OpenGL rendering across all NVIDIA GPUs when Xinerama is enabled.

To configure Xinerama

1. Configure multiple X screens (refer to the XF86Config(5x) or xorg.conf(5x) man pages for details).

2. Enable Xinerama by adding the line

   ```
   Option "Xinerama" "True"
   ```

   to the "ServerFlags" section of your X config file.

Requirements:

o Using identical GPUs is recommended. Some combinations of non-identical, but similar, GPUs are supported. If a GPU is incompatible with the rest of a Xinerama desktop then no OpenGL rendering will appear on the screens driven by that GPU. Rendering will still appear normally on screens connected to other supported GPUs. In this situation the X log file will include a message of the form:

```
(WW) NVIDIA(2): The GPU driving screen 2 is incompatible with the rest of
(WW) NVIDIA(2):      the GPUs composing the desktop.  OpenGL rendering will
(WW) NVIDIA(2):      be disabled on screen 2.
```

o NVIDIA's GLX implementation only supports Xinerama when physical X screen 0 is driven by the NVIDIA X driver. This is because the X.Org X server bases the visuals of the logical Xinerama X screen on the visuals of physical X screen 0.

When physical X screen 0 is not being driven by the NVIDIA X driver and Xinerama is enabled, then GLX will be disabled. If physical X screens other than screen 0 are not being driven by the NVIDIA X driver, OpenGL rendering will be disabled on them.

o Only the intersection of capabilities across all GPUs will be advertised.

The maximum OpenGL viewport size depends on the hardware used, and is described by the following table. If an OpenGL window is larger than the maximum viewport, regions beyond the viewport will be blank.

OpenGL Viewport Maximums in Xinerama

GeForce GPUs before GeForce 8:	4096 x 4096 pixels
GeForce 8 through GeForce GTX 400 GPUs:	8192 x 8192 pixels
GeForce GTX 400 series GPUs:	16384x16384 pixels
Quadro GPUs before G80:	as large as the Xinerama desktop
Quadro G80 through Quadro Fermi:	8192 x 8192 pixels
Quadro Fermi and later GPUs:	16384x16384 pixels

- o X configuration options that affect GLX operation (e.g.: stereo, overlays) should be set consistently across all X screens in the X server.

Known Issues:

- o Versions of XFree86 prior to 4.5 and versions of X.Org prior to 6.8.0 lack the required interfaces to properly implement overlays with the Xinerama extension. On earlier server versions mixing overlays and Xinerama will result in rendering corruption. If you are using the Xinerama extension with overlays, it is recommended that you upgrade to XFree86 4.5, X.Org 6.8.0, or newer.

Chapter 14. Configuring Multiple X Screens on One Card

GPUs that support TwinView (Chapter 12) 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, hardware accelerated OpenGL cannot span the two X screens), it does offer several advantages over TwinView:

- o If each display device is a separate X screen, then properties that may vary between X screens may vary between displays (e.g.: depth, root window size, etc).

- o Hardware that can only be used on one display at a time (e.g.: video overlays and hardware accelerated RGB overlays on certain older GPUs), and which consequently cannot be used at all when in TwinView, can be exposed on the first X screen when each display is a separate X screen.

- o TwinView is a fairly new feature. X has historically used one screen per display device.

To configure two separate X screens to share one graphics card, here is what you will need to do:

First, create two separate Device sections, each listing the BusID of the graphics card to be shared and listing the driver as "nvidia", and 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
```

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
```

(Note: You'll also need to create a second Monitor section) Finally, update
the ServerLayout section to use and position both Screen sections:

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

For further details, refer to the XF86Config(5x) or xorg.conf(5x) man pages.

Chapter 15. Configuring TV-Out

NVIDIA GPU-based graphics cards with a TV-Out connector can use a television
as another display device (the same way that it would use a CRT or digital
flat panel). The TV can be used by itself, or in conjunction with another
display device in a TwinView or multiple X screen configuration. If a TV is
the only display device connected to your graphics card, it will be used as
the primary display when you boot your system (i.e. the console will come up
on the TV just as if it were a CRT).

The NVIDIA X driver populates the mode pool for the TV with all the mode sizes
that the driver supports with the given TV standard and the TV encoder on the
graphics card. These modes are given names that correspond to their
resolution; e.g., "800x600".

Because these TV modes only depend on the TV encoder and the TV standard, TV
modes do not go through normal mode validation. The X configuration options
HorizSync and VertRefresh are not used for TV mode validation.

Additionally, the NVIDIA driver contains a hardcoded list of mode sizes that
it can drive for each combination of TV encoder and TV standard. Therefore,
custom modelines in your X configuration file are ignored for TVs.

To use your TV with X, there are several relevant X configuration options:

o The Modes in the screen section of your X configuration file; you can use
  these to request any of the modes in the mode pool which the X driver
  created for this combination of TV standard and TV encoder. Examples
  include "640x480" and "800x600". If in doubt, use "nvidia-auto-select".

o The "TVStandard" option should be added to your screen section; valid
  values are:

TVStandard	Description
"PAL-B"	used in Belgium, Denmark, Finland, Germany, Guinea, Hong Kong, India, Indonesia, Italy, Malaysia, The Netherlands, Norway, Portugal, Singapore, Spain, Sweden, and Switzerland
"PAL-D"	used in China and North Korea
"PAL-G"	used in Denmark, Finland, Germany, Italy, Malaysia, The Netherlands, Norway, Portugal, Spain, Sweden, and Switzerland
"PAL-H"	used in Belgium
"PAL-I"	used in Hong Kong and The United Kingdom
"PAL-K1"	used in Guinea
"PAL-M"	used in Brazil
"PAL-N"	used in France, Paraguay, and Uruguay
"PAL-NC"	used in Argentina
"NTSC-J"	used in Japan
"NTSC-M"	used in Canada, Chile, Colombia, Costa Rica, Ecuador, Haiti, Honduras, Mexico, Panama, Puerto Rico, South Korea, Taiwan, United States of America, and Venezuela
"HD480i"	480 line interlaced
"HD480p"	480 line progressive
"HD720p"	720 line progressive
"HD1080i"	1080 line interlaced
"HD1080p"	1080 line progressive
"HD576i"	576 line interlace
"HD576p"	576 line progressive

The line in your X config file should be something like:

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

If you do not specify a TVStandard, or you specify an invalid value, the default "NTSC-M" will be used. Note: if your country is not in the above list, select the country closest to your location.

o The "UseDisplayDevice" option can be used if there are multiple display devices connected, and you want the connected TV to be used instead of the connected CRTs and/or DFPs. E.g.,

```
Option "UseDisplayDevice" "TV"
```

Using the "UseDisplayDevice" option, rather than the "ConnectedMonitor" option, is recommended.

o The "TVOutFormat" option can be used to force the output format. Without this option, the driver autodetects the output format. Unfortunately, it does not always do this correctly. The output format can be forced with the "TVOutFormat" option; valid values are:

TVOutFormat	Description	Supported TV standards
"AUTOSELECT"	The driver autodetects the output format (default value).	PAL, NTSC, HD
"COMPOSITE"	Force Composite output format	PAL, NTSC
"SVIDEO"	Force S-Video output format	PAL, NTSC
"COMPONENT"	Force Component output format, also called YPbPr	HD
"SCART"	Force Scart output format, also called Peritel	PAL, NTSC

The line in your X config file should be something like:

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

o The "TVOverScan" option can be used to enable Overscan, when the TV encoder supports it. Valid values are decimal values in the range 1.0 (which means overscan as much as possible: make the image as large as possible) and 0.0 (which means disable overscanning: make the image as small as possible). Overscanning is disabled (0.0) by default.

Chapter 16. Support for the X Resize and Rotate Extension

This NVIDIA driver release contains support for the X Resize and Rotate (RandR) Extension versions 1.1, 1.2, and 1.3. The version of the RandR extension advertised to X clients is controlled by the X server: the RandR extension and protocol are provided by the X server, which routes protocol requests to the NVIDIA X driver. Run `xrandr --version` to check the version of RandR provided by the X server.

16A. RANDR SUPPORT

Specific supported features include:

o Modes can be set per-screen, and the X screen can be resized through the RRSetScreenConfig request (e.g., with xrandr(1)'s '--size' and '--rate' command line options).

o The X screen can be resized with the RandR 1.2 RRSetScreenSize request (e.g., with xrandr(1)'s '--fb' command line option).

o The state of the display hardware can be queried with the RandR 1.2 and 1.3 RRGetScreenResources, RRGetScreenResourcesCurrent, RRGetOutputInfo, and RRGetCrtcInfo requests (e.g., with xrandr(1)'s '--query' command line option).

o Modes can be set with RandR CRTC granularity with the RandR 1.2 RRSetCrtcConfig request. E.g., `xrandr --output DVI-I-2 --mode 1920x1200`.

o Rotation can be set with RandR CRTC granularity with the RandR 1.2 RRSetCrtcConfig request. E.g., `xrandr --output DVI-I-2 --mode 1920x1200 --rotation left`.

o Per-CRTC transformations can be manipulated with the RandR 1.3 RRSetCrtcTransform and RRGetCrtcTransform requests. E.g., `xrandr

--output DVI-I-3 --mode 1920x1200 --transform 43.864288330078125,21.33332
8247070312,-16384,0,43.864288330078125,0,0,0.0321197509765625,19.19062805
1757812`.

o The RandR 1.0/1.1 requests RRGetScreenInfo and RRSetScreenConfig manipulate MetaModes. The MetaModes that the X driver uses (either user-requested or implicitly generated) are reported through the RRGetScreenInfo request (e.g., `xrandr --query --q1`) and chosen through the RRSetScreenConfig request (e.g., `xrandr --size 1920x1200 --orientation left`).

The configurability exposed through RandR is also available through the MetaMode syntax, independent of X server version. See Chapter 12 for more details. As an example, these two commands are equivalent:

```
  xrandr --output DVI-I-2 --mode 1280x1024 --pos 0x0 --rotate left \
    --output DVI-I-3 --mode 1920x1200 --pos 0x0

  nvidia-settings --assign CurrentMetaMode="DVI-I-2: 1280x1024 +0+0 \
    { Rotation=left }, DVI-I-3: 1920x1200 +0+0"
```

16B. RANDR 1.1 ROTATION BEHAVIOR

On X servers that support RandR 1.2 or later, when an RandR 1.1 rotation request is received (e.g., `xrandr --orientation left`), the NVIDIA X driver will apply that request to an entire MetaMode. E.g., if you configure multiple monitors, either through a MetaMode or through RandR 1.2:

```
  xrandr --output DVI-I-2 --mode 1280x1024 --pos 0x0 \
    --output DVI-I-3 --mode 1920x1200 --pos 1280x0

  nvidia-settings --assign CurrentMetaMode="DVI-I-2: 1280x1024 +0+0, \
    DVI-I-3: 1920x1200 +1280+0"
```

Requesting RandR 1.1 rotation through `xrandr --orientation left`, will rotate the entire MetaMode, producing the equivalent of either:

```
  xrandr --output DVI-I-2 --mode 1280x1024 --pos 176x0 --rotate left \
    --output DVI-I-3 --mode 1920x1200 --rotate left --pos 0x1280

  nvidia-settings --assign CurrentMetaMode="DVI-I-2: 1280x1024 +176+0, \
    { Rotation=left }, DVI-I-3: 1920x1200 +0+1280 { Rotation=left }"
```

On X servers that do not support RandR 1.2 or later, the NVIDIA X driver does not advertise RandR rotation support. On such X servers, it is recommended to configure rotation through MetaModes, instead.

16C. OUTPUT PROPERTIES

The NVIDIA FreeBSD driver supports a number of output device properties.

OFFICIAL PROPERTIES

Properties that do not start with an underscore are officially documented in the file "randrproto.txt" in X.Org's randrproto package. See that file for a full description of these properties.

o "ConnectorNumber"

This property groups RandR outputs by their physical connectors. For example, DVI-I ports have both an analog and a digital output, which is represented in RandR by two different output objects. One DVI-I port may be represented by RandR outputs "DVI-I-0" with "SignalFormat" "TMDS" (transition-minimized differential signaling, a digital signal format) and "DVI-I-1" with "SignalFormat" "VGA", representing the analog part. In this case, both RandR outputs would have the same value of "ConnectorNumber".

o "ConnectorType"

This property lists the physical type of the connector. For example, in the DVI-I example above, both "DVI-I-0" and "DVI-I-1" would have a "ConnectorType" of "DVI-I".

o "EDID"

This property contains the raw bytes of the display's extended display identification data. This data is intended for applications to use to glean information about the monitor connected.

o "SignalFormat"

This property describes the type of signaling used to send image data to the display device. For example, an analog device connected to a DVI-I port might use VGA as its signaling format.

UNOFFICIAL PROPERTIES

Properties whose names begin with an underscore are not specified by X.Org. They may be removed or modified in future driver releases. The NVIDIA FreeBSD driver supports the following unofficial properties:

o "_ConnectorLocation"

This property describes the physical location of the connector. On add-in graphics boards, connector location 0 should generally be the position closest to the motherboard, with increasing location numbers indicating connectors progressively farther away.

Type	INTEGER
Format	32
# Items	1
Flags	Immutable, Static
Range	0-

o "_GUID"

DisplayPort 1.2 specifies that all devices must have a globally-unique identifier, referred to as a GUID. When a GUID is available, the "_GUID" property contains its raw bytes.

Type	INTEGER
Format	8
# Items	16
Flags	Immutable
Range	0-

16D. DISPLAYPORT 1.2

When display devices are connected via DisplayPort 1.2 branch devices, additional RandR outputs will be created, one for each connected display device. These dynamic outputs will remain as long as the display device is connected or used in a MetaMode, even if they are not named in the current MetaMode. They will be deleted automatically when the display is disconnected and no MetaModes use them.

See Appendix C for a description of how the names of these outputs are generated.

If you are developing applications that use the RandR extension, please be aware that outputs can be created and destroyed dynamically. You should be sure to use the XRRSelectInput function to watch for events that indicate when this happens.

16E. KNOWN ISSUES

o RandR per-CRTC panning configuration through the RandR 1.3 RRGetPanning and RRSetPanning requests is not yet implemented.

o Rotation and Transformations (configured either through RandR or MetaModes) are not yet supported with Workstation overlays or stereo.

o The RandR 1.2 X configuration options provided by the XFree86 DDX implementation and documented in xorg.conf(5) are not yet supported.

o Transformations (configured either through RandR or MetaModes) are not yet correctly clipped.

Chapter 17. Configuring a Notebook

17A. INSTALLATION AND CONFIGURATION

Installation and configuration of the NVIDIA FreeBSD Driver Set on a notebook is the same as for any desktop environment, with a few additions, as described below.

17B. POWER MANAGEMENT

All notebook NVIDIA GPUs support power management, both S3 (also known as "Standby" or "Suspend to RAM") and S4 (also known as "Hibernate", "Suspend to Disk" or "SWSUSP"). Power management is system-specific and is dependent upon all the components in the system; some systems may be more problematic than other systems.

Most recent notebook NVIDIA GPUs also support PowerMizer, which monitors application work load to adjust system parameters to deliver the optimal balance of performance and battery life. However, PowerMizer is only enabled by default on some notebooks. Please see the known issues below for more details.

17C. HOTKEY SWITCHING OF DISPLAY DEVICES

Mobile NVIDIA GPUs also have the capacity to react to a display change hotkey event, toggling between each of the connected display devices and each possible combination of the connected display devices (note that only 2 display devices may be active at a time).

Hotkey switching dynamically changes the TwinView configuration; a given hotkey event will indicate which display devices should be in use at that time, and all MetaModes currently configured on the X screen will be updated to use the new configuration of display devices.

Another important aspect of hotkey functionality is that you can dynamically connect and remove display devices to/from your notebook and use the hotkey to activate and deactivate them without restarting X.

Note that there are two approaches to implementing this hotkey support: ACPI events and polling.

Most recent notebooks use ACPI events to deliver hotkeys from the System BIOS to the graphics driver. This is the preferred method of delivering hotkey events, but is still a new feature under most UNIX platforms and may not always function correctly.

The polling mechanism requires checking during the vertical blanking interval for a hotkey status change. It is an older mechanism for handling hotkeys, and is therefore not supported on all notebooks and is not tested by notebook manufacturers. It also does not always report the same combinations of display devices that are reported by ACPI hotkey events.

The NVIDIA FreeBSD Driver will attempt to use ACPI hotkey events, if possible. In the case that ACPI hotkey event support is not available, the driver will revert back to trying hotkey polling. In the case that the notebook does not support hotkey polling, hotkeys will not work. Please see the known issues section below for more details.

When switching away from X to a virtual terminal, the VGA console will always be restored to the display device on which it was present when X was started. Similarly, when switching back into X, the same display device configuration will be used as when you switched away, regardless of what display change hotkey activity occurred while the virtual terminal was active.

17D. DOCKING EVENTS

All notebook NVIDIA GPUs support docking, however support may be limited by the OS or system. There are three types of notebook docking (hot, warm, and cold), which refer to the state of the system when the docking event occurs. hot refers to a powered on system with a live desktop, warm refers to a system that has entered a suspended power management state, and cold refers to a system that has been powered off. Only warm and cold docking are supported by the NVIDIA driver.

17E. TWINVIEW

All notebook NVIDIA GPUs support TwinView. TwinView on a notebook can be configured in the same way as on a desktop computer (refer to Chapter 12);

note that in a TwinView configuration using the notebook's internal flat panel and an external CRT, the CRT is the primary display device (specify its HorizSync and VertRefresh in the Monitor section of your X config file) and the flat panel is the secondary display device (specify its HorizSync and VertRefresh through the SecondMonitorHorizSync and SecondMonitorVertRefresh options).

The "UseEdidFreqs" X config option is enabled by default, so normally you should not need to specify the "SecondMonitorHorizSync" and "SecondMonitorVertRefresh" options. See the description of the UseEdidFreqs option in Appendix B for details).

17F. KNOWN NOTEBOOK ISSUES

There are a few known issues associated with notebooks:

- o Display change hotkey switching is not available on all notebooks. In some cases, the ACPI infrastructure is not fully supported by the NVIDIA FreeBSD Driver. Work is ongoing to increase the robustness of NVIDIA's support in this area. Toshiba and Lenovo notebooks are known to be problematic.

- o ACPI Display change hotkey switching is not supported by X.Org X servers earlier than 1.2.0; see EnableACPIHotkeys in Appendix B for details and "Q. How do I interpret X server version numbers?" in Chapter 7 for information on how to determine whether your X server is new enough.

- o In many cases, suspending and/or resuming will fail. As mentioned above, this functionality is very system-specific. There are still many cases that are problematic. Here are some tips that may help:

  - o In some cases, hibernation can have bad interactions with the PCI Express bus clocks, which can lead to system hangs when entering hibernation. This issue is still being investigated, but a known workaround is to leave an OpenGL application running when hibernating.

- o On some notebooks, PowerMizer is not enabled by default. This issue is being investigated, and there is no known workaround.

- o ACPI is not currently supported on FreeBSD As a result, ACPI hotkey events are not supported.

o On GeForce 6- and 7-series GPUs, the video overlay only works on the first display device on which you started X. For example, if you start X on the internal LCD, run a video application that uses the video overlay (uses the "Video Overlay" adapter advertised through the XV extension), and then hotkey switch to add a second display device, the video will not appear on the second display device. To work around this, you can either configure the video application to use the "Video Blitter" adapter advertised through the XV extension (this is always available), or hotkey switch to the display device on which you want to use the video overlay *before* starting X.

Chapter 18. Programming Modes

The NVIDIA Accelerated FreeBSD Graphics Driver supports all standard VGA and VESA modes, as well as most user-written custom mode lines; double-scan and interlaced modes are supported on all GPUs supported by the NVIDIA driver.

To request one or more standard modes for use in X, you can simply add a "Modes" line such as:

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

in the appropriate Display subsection of your X config file (see the XF86Config(5x) or xorg.conf(5x) man pages for details). Or, the nvidia-xconfig(1) utility can be used to request additional modes; for example:

```
    nvidia-xconfig --mode 1600x1200
```

See the nvidia-xconfig(1) man page for details.

18A. DEPTH, BITS PER PIXEL, AND PITCH

While not directly a concern when programming modes, the bits used per pixel is an issue when considering the maximum programmable resolution; for this reason, it is worthwhile to address the confusion surrounding the terms "depth" and "bits per pixel". Depth is how many bits of data are stored per pixel. Supported depths are 8, 15, 16, and 24. Depth 30 is also available on GeForce 8 series and higher GPUs. 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 your depth, X selects the bits per pixel (bpp) size in which to store the data. Below is a table of what bpp is used for each possible depth:

Depth	BPP
8	8
15	16
16	16
24	32
30	32

Lastly, the "pitch" is how many bytes in the linear frame buffer there are between one pixel's data, and the data of the pixel immediately below. You can think of this as the horizontal resolution multiplied by the bytes per pixel (bits per pixel divided by 8). In practice, the pitch may be more than this product due to alignment constraints.

18B. MAXIMUM RESOLUTIONS

The NVIDIA Accelerated FreeBSD Graphics Driver and NVIDIA GPU-based graphics cards support resolutions up to 16384x16384 pixels for the GeForce GTX 400 series and newer GPUs, 8192x8192 pixels for the GeForce 8 series and above, and up to 4096x4096 pixels for the GeForce 7 series and below, though the maximum resolution your system can support is also limited by the amount of video memory (see USEFUL FORMULAS for details) and the maximum supported resolution of your display device (monitor/flat panel/television). Also note that 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 overlay quality.

18C. USEFUL FORMULAS

The maximum resolution is a function both of the amount of video memory and the bits per pixel you elect to use:

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

In other words, the amount of video memory used is equal to the horizontal resolution (HR) multiplied by the vertical resolution (VR) multiplied by the bytes per pixel (bits per pixel divided by eight). Technically, the video memory used is actually the pitch times the vertical resolution, and the pitch may be slightly greater than (HR * (bpp/8)) to accommodate the hardware requirement that the pitch be a multiple of some value.

Note that this is just memory usage for the frame buffer; video memory is also used by other things, such as OpenGL and pixmap caching.

Another important relationship is that between the resolution, the pixel clock (also known as the dot clock) and the vertical refresh rate:

RR = PCLK / (HFL * VFL)

In other words, the refresh rate (RR) is equal to the pixel clock (PCLK) divided by the total number of pixels: the horizontal frame length (HFL) multiplied by the vertical frame length (VFL) (note that these are the frame lengths, and not just the visible resolutions). As described in the XFree86 Video Timings HOWTO, the above formula can be rewritten as:

PCLK = RR * HFL * VFL

Given a maximum pixel clock, you can adjust the RR, HFL and VFL as desired, as long as the product of the three is consistent. The pixel clock is reported in the log file. Your X log should contain a line like this:

```
    (--) NVIDIA(0): ViewSonic VPD150 (DFP-1): 165 MHz maximum pixel clock
```

which indicates the maximum pixel clock for that display device.

18D. HOW MODES ARE VALIDATED

In traditional XFree86/X.Org mode validation, the X server takes as a starting point the X server's internal list of VESA standard modes, plus any modes specified with special ModeLines in the X configuration file's Monitor section. These modes are validated against criteria such as the valid HorizSync/VertRefresh frequency ranges for the user's monitor (as specified in the Monitor section of the X configuration file), as well as the maximum pixel clock of the GPU.

Once the X server has determined the set of valid modes, it takes the list of user requested modes (i.e., the set of modes named in the "Modes" line in the Display subsection of the Screen section of X configuration file), and finds the "best" validated mode with the requested name.

The NVIDIA X driver uses a variation on the above approach to perform mode validation. During X server initialization, the NVIDIA X driver builds a pool of valid modes for each display device. It gathers all possible modes from several sources:

- o The display device's EDID
- o The X server's built-in list
- o Any user-specified ModeLines in the X configuration file
- o The VESA standard modes

For every possible mode, the mode is run through mode validation. The core of mode validation is still performed similarly to traditional XFree86/X.Org mode validation: the mode timings are checked against things such as the valid HorizSync and VertRefresh ranges and the maximum pixelclock. Note that each individual stage of mode validation can be independently controlled through the "ModeValidation" X configuration option.

Note that when validating interlaced mode timings, VertRefresh specifies the field rate, rather than the frame rate. For example, the following modeline has a vertical refresh rate of 87 Hz:

```
 # 1024x768i @ 87Hz (industry standard)
 ModeLine "1024x768"  44.9  1024 1032 1208 1264  768 768 776 817 +hsync +vsync
Interlace
```

Invalid modes are discarded; valid modes are inserted into the mode pool. See MODE VALIDATION REPORTING for how to get more details on mode validation results for each considered mode.

Valid modes are given a unique name that is guaranteed to be unique across the whole mode pool for this display device. This mode name is constructed approximately like this:

```
    <width>x<height>_<refreshrate>
```

(e.g., "1600x1200_85")

The name may also be prepended with another number to ensure the mode is unique; e.g., "1600x1200_85_0".

As validated modes are inserted into the mode pool, duplicate modes are removed, and the mode pool is sorted, such that the "best" modes are at the beginning of the mode pool. The sorting is based roughly on:

- o Resolution

- o Source (EDID-provided modes are prioritized higher than VESA-provided modes, which are prioritized higher than modes that were in the X server's built-in list)

- o Refresh rate

Once modes from all mode sources are validated and the mode pool is constructed, all modes with the same resolution are compared; the best mode with that resolution is added to the mode pool a second time, using just the resolution as its unique modename (e.g., "1600x1200"). In this way, when you request a mode using the traditional names (e.g., "1600x1200"), you still get what you got before (the 'best' 1600x1200 mode); the added benefit is that all modes in the mode pool can be addressed by a unique name.

When verbose logging is enabled (see "Q. How can I increase the amount of data printed in the X log file?" in Chapter 7), the mode pool for each display device is printed to the X log file.

After the mode pool is built for all display devices, the requested modes (as specified in the X configuration file), are looked up from the mode pool. Each requested mode that can be matched against a mode in the mode pool is then advertised to the X server and is available to the user through the X server's mode switching hotkeys (ctrl-alt-plus/minus) and the XRandR and XF86VidMode X extensions.

Additionally, all modes in the mode pool of the primary display device are implicitly made available to the X server. See the "IncludeImplicitMetaModes" X configuration option in Appendix B for details.

18E. THE NVIDIA-AUTO-SELECT MODE

You can request a special mode by name in the X config file, named "nvidia-auto-select". When the X driver builds the mode pool for a display device, it selects one of the modes as the "nvidia-auto-select" mode; a new entry is made in the mode pool, and "nvidia-auto-select" is used as the unique name for the mode.

The "nvidia-auto-select" mode is intended to be a reasonable mode for the display device in question. For example, the "nvidia-auto-select" mode is normally the native resolution for flat panels, as reported by the flat panel's EDID, or one of the detailed timings from the EDID. The "nvidia-auto-select" mode is guaranteed to always be present, and to always be defined as something considered valid by the X driver for this display device.

Note that the "nvidia-auto-select" mode is not necessarily the largest possible resolution, nor is it necessarily the mode with the highest refresh rate. Rather, the "nvidia-auto-select" mode is selected such that it is a reasonable default. The selection process is roughly:

o If the EDID for the display device reported a preferred mode timing, and that mode timing is considered a valid mode, then that mode is used as the "nvidia-auto-select" mode. You can check if the EDID reported a preferred timing by starting X with logverbosity greater than or equal to 5 (see "Q. How can I increase the amount of data printed in the X log file?" in Chapter 7), and looking at the EDID printout; if the EDID contains a line:

```
    Prefer first detailed timing : Yes
```

  Then the first mode listed under the "Detailed Timings" in the EDID will be used.

o If the EDID did not provide a preferred timing, the best detailed timing from the EDID is used as the "nvidia-auto-select" mode.

o If the EDID did not provide any detailed timings (or there was no EDID at all), the best valid mode not larger than 1024x768 is used as the "nvidia-auto-select" mode. The 1024x768 limit is imposed here to restrict use of modes that may have been validated, but may be too large to be considered a reasonable default, such as 2048x1536.

o If all else fails, the X driver will use a built-in 800 x 600 60Hz mode as the "nvidia-auto-select" mode.

If no modes are requested in the X configuration file, or none of the requested modes can be found in the mode pool, then the X driver falls back to the "nvidia-auto-select" mode, so that X can always start. Appropriate warning messages will be printed to the X log file in these fallback scenarios.

You can add the "nvidia-auto-select" mode to your X configuration file by running the command

```
    nvidia-xconfig --mode nvidia-auto-select
```

and restarting your X server.

The X driver can generally do a much better job of selecting the "nvidia-auto-select" mode if the display device's EDID is available. This is one reason why it is recommended to only use the "UseEDID" X configuration option sparingly. Note that, rather than globally disable all uses of the EDID with the "UseEDID" option, you can individually disable each particular use of the EDID using the "UseEDIDFreqs", "UseEDIDDpi", and/or the "NoEDIDModes" argument in the "ModeValidation" X configuration option.

18F. MODE VALIDATION REPORTING

When log verbosity is set to 6 or higher (see "Q. How can I increase the amount of data printed in the X log file?" in Chapter 7), the X log will record every mode that is considered for each display device's mode pool, and report whether the mode passed or failed. For modes that were considered invalid, the log will report why the mode was considered invalid.

18G. ENSURING IDENTICAL MODE TIMINGS

Some functionality, such as Active Stereo with TwinView, requires control over exactly which mode timings are used. For explicit control over which mode timings are used on each display device, you can specify the ModeLine you want to use (using one of the ModeLine generators available), and using a unique name. For example, if you wanted to use 1024x768 at 120 Hz on each monitor in TwinView with active stereo, you might add something like this to the monitor section of your X configuration file:

```
    # 1024x768 @ 120.00 Hz (GTF) hsync: 98.76 kHz; pclk: 139.05 MHz
    Modeline "1024x768_120"  139.05  1024 1104 1216 1408  768 769 772 823
-HSync +Vsync
```

Then, in the Screen section of your X config file, specify a MetaMode like
this:

```
    Option "MetaModes" "1024x768_120, 1024x768_120"
```

18H. ADDITIONAL INFORMATION

An XFree86 ModeLine generator, conforming to the GTF Standard is available at
http://gtf.sourceforge.net/. Additional generators can be found by searching
for "modeline" on freshmeat.net.

Chapter 19. Configuring Flipping and UBB

The NVIDIA Accelerated FreeBSD Graphics Driver supports Unified Back Buffer
(UBB) and OpenGL Flipping. These features can provide performance gains in
certain situations.

- o Unified Back Buffer (UBB): UBB is available only on Quadro GPUs (Quadro NVS GPUs excluded) and is enabled by default when there is sufficient video memory available. This can be disabled with the UBB X config option described in Appendix B. When UBB is enabled, all windows share the same back, stencil and depth buffers. When there are many windows, the back, stencil and depth usage will never exceed the size of that used by a full screen window. However, even for a single small window, the back, stencil, and depth video memory usage is that of a full screen window. In that case video memory may be used less efficiently than in the non-UBB case.

- o Flipping: When OpenGL flipping is enabled, OpenGL can perform buffer swaps by changing which buffer is scanned out rather than copying the back buffer contents to the front buffer; this is generally a higher performance mechanism and allows tearless swapping during the vertical retrace (when __GL_SYNC_TO_VBLANK is set).

Chapter 20. The Sysctl Interface

The sysctl interface allows you to obtain run-time information about the driver, any installed NVIDIA graphics cards and the AGP status. It also allows you to control low-level configuration options and/or overrides.

The various pieces of information are held in a hierarchy under hw.nvidia and are accessible with the sysctl(8) command.

NVIDIA sysctl Entries

hw.nvidia.version

Prints the installed driver revision

hw.nvidia.cards.n.*

These OIDs provide information about NVIDIA device 'n':

ID	Description
model	the device's product name
irq	the IRQ claimed by this device
vbios	the device's Video BIOS revision
type	the bus type of this device

hw.nvidia.agp.host-bridge.*
hw.nvidia.agp.card.*

These OIDs provide information about the AGP capabilities of the installed AGP graphics card and host-bridge respectively. These values are most likely to be correct after system boot and before the X server is started (and the AGP subsystem initialized).

ID	Description
rates	the AGP rates supported by this device

fw	if the device supports AGP fast-writes
sba	if the device supports AGP side-band-addressing
registers	the device's AGP registers, status:command

hw.nvidia.agp.status.*

Prints AGP status information based on the AGP command registers of the host-bridge and of the AGP card.

ID	Description
status	if AGP is enabled or disabled
driver	which driver is being used
rate	the programmed AGP rate
fw	if fast-writes are enabled or disabled
sba	if side-band-addressing is enabled or disabled

hw.nvidia.registry.*

Low-level kernel module configuration options. Changing these is typically not necessary and potentially dangerous. If you do need to change any of these options, you will need to do so BEFORE you start the X server.

ID	Description
status	if AGP is enabled or disabled
driver	which driver is being used
rate	the programmed AGP rate
fw	if fast-writes are enabled or disabled
sba	if side-band-addressing is enabled or disabled

Chapter 21. Configuring Low-level Parameters

The NVIDIA resource manager recognizes several low-level configuration parameters that can be set using the sysctl driver interface BEFORE the X server is started. Normally you should not need to modify any of these parameters, but it is sometimes necessary or desirable to do so.

To view the current settings of these parameters, you need to issue this sysctl command ('nvidia.ko' needs to be loaded):

```
    % sysctl -a hw.nvidia.registry
```

To change any of the parameters, you need to pass the complete name of the OID followed by '=' and the new value, e.g.:

```
    % sysctl hw.nvidia.registry.EnableVia4x=1
```

It is possible to automate setting these parameters by adding them to the '/etc/sysctl.conf' file. See `man 5 sysctl.conf` for details.

The following parameters are recognized by 'nvidia.ko':

Resource Manager Parameters

VideoMemoryTypeOverride

We normally detect memory type on TNT cards by scanning the embedded BIOS. Unfortunately, we've seen some cases where a TNT card has been flashed with the wrong BIOS. For example, an SDRAM based TNT has been flashed with an SGRAM BIOS, and therefore claims to be an SGRAM TNT. We've therefore provided an override here. Make sure to set the value toe the type of memory used on your card.

Value	Meaning
1	SDRAM
2	SGRAM

Note that we can only do so much here. There are border cases where even this fails. For example, if 2 TNT cards are in the same system, one SGRAM, one SDRAM.

This option is disabled by default, see below for information on how to enable it.

EnableVia4x

We've had problems with some Via chipsets in 4x mode, we need force them back down to 2x mode. If you'd like to experiment with retaining 4x mode, you may try setting this value to 1 If that hangs the system, you're stuck with 2x mode; there's nothing we can do about it.

Value	Meaning
0	disable AGP 4x on Via chipsets (default)
1	enable AGP 4x on Via chipsets

EnableALiAGP

Some ALi chipsets (ALi1541, ALi1647) are known to cause severe system stability problems with AGP enabled. To avoid lockups, we disable AGP on systems with these chipsets by default. It appears that updating the system BIOS and using recent versions of the kernel AGP Gart driver can make such systems much more stable. If you own a system with one of the aforementioned chipsets and had it working reasonably well previously, or if you want to experiment with BIOS and AGPGART revisions, you can re-enable AGP support by setting this option to 1.

Value	Meaning
0	disable AGP on Ali1541 and ALi1647 (default)
1	enable AGP on Ali1541 and ALi1647

NvAGP

This options controls which AGP GART driver is used when no explicit request is made to change the default (X server).

Value	Meaning
0	disable AGP support
1	use the NVIDIA builtin driver (if possible)
2	use the kernel's AGPGART driver (if possible)
3	use any available driver (try 2, then 1)

Note that the NVIDIA internal AGP GART driver will not be used if AGPGART was either statically linked into your kernel or built as a kernel module and loaded before the NVIDIA kernel module.

ReqAGPRate

Normally, the driver will compare speed modes of the chipset and the card, picking the highest common rate. This key forces a maximum limit, to limit the driver to lower speeds. The driver will not attempt a speed beyond what the chipset and card claim they are capable of.

Make sure you really know what you're doing before you enable this override. By default, AGP drivers will enable the fastest AGP rate your card and motherboard chipset are capable of. Then, in some cases, our driver will force this rate down to work around bugs in both our chipsets, and motherboard chipsets. Using this variable will override our bug fixes. This may be desirable in some cases, but not most. THIS IS COMPLETELY UNSUPPORTED!

This option expects a bitmask (7 = 1|2|3|4, 3=1|2, etc.)

This option is disabled by default, see below for information on how to enable it.

EnableAGPSBA

For stability reasons, the driver will not use Side Band Addressing even if both the host chipset and the AGP card support it. You may override this behavior with the following registry key. THIS IS COMPLETELY UNSUPPORTED!

Value	Meaning
0	disable Side Band Addressing (default on x86, see below)
1	enable Side Band Addressing (if supported)

EnableAGPFW

Similar to Side Band Addressing, Fast Writes are disabled by default. If you wish to enable them on systems that support them, you can do so with this registry key. Note that this may render your system unstable with many AGP chipsets. THIS IS COMPLETELY UNSUPPORTED!

Value	Meaning

0	disable Fast Writes (default)
1	enable Fast Writes

Chapter 22. Using the X Composite Extension

X.Org X servers, beginning with X11R6.8.0, contain experimental support for a new X protocol extension called Composite. This extension allows windows to be drawn into pixmaps instead of directly onto the screen. In conjunction with the Damage and Render extensions, this allows a program called a composite manager to blend windows together to draw the screen.

Performance will be degraded significantly if the "RenderAccel" option is disabled in xorg.conf. See Appendix B for more details.

When the NVIDIA X driver is used with an X.Org X server X11R6.9.0 or newer and the Composite extension is enabled, NVIDIA's OpenGL implementation interacts properly with the Damage and Composite X extensions. This means that OpenGL rendering is drawn into offscreen pixmaps and the X server is notified of the Damage event when OpenGL renders to the pixmap. This allows OpenGL applications to behave properly in a composited X desktop.

If the Composite extension is enabled on an X server older than X11R6.9.0, then GLX will be disabled. You can force GLX on while Composite is enabled on pre-X11R6.9.0 X servers with the "AllowGLXWithComposite" X configuration option. However, GLX will not render correctly in this environment. Upgrading your X server to X11R6.9.0 or newer is recommended.

You can enable the Composite X extension by running 'nvidia-xconfig --composite'. Composite can be disabled with 'nvidia-xconfig --no-composite'. See the nvidia-xconfig(1) man page for details.

If you are using Composite with GLX, it is recommended that you also enable the "DamageEvents" X option for enhanced performance. If you are using an OpenGL-based composite manager, you may also need the "DisableGLXRootClipping" option to obtain proper output.

The Composite extension also causes problems with other driver components:

o In X servers prior to X.Org 7.1, Xv cannot draw into pixmaps that have been redirected offscreen and will draw directly onto the screen instead. For some programs you can work around this issue by using an alternative video driver. For example, "mplayer -vo x11" will work correctly, as will "xine -V xshm". If you must use Xv with an older server, you can also disable the compositing manager and re-enable it when you are finished.

On X.Org 7.1 and higher, the driver will properly redirect video into offscreen pixmaps. Note that the Xv adaptors will ignore the sync-to-vblank option when drawing into a redirected window.

o Workstation overlays, stereo visuals, and the unified back buffer (UBB) are incompatible with Composite. These features will be automatically disabled when Composite is detected.

o The Composite extension is currently incompatible with Xinerama, due to limitations in the X.Org X server. Composite will be automatically disabled when Xinerama is enabled.

This NVIDIA FreeBSD driver supports OpenGL rendering to 32-bit ARGB windows on X.Org 7.2 and higher or when the "AddARGBGLXVisuals" X config file option is enabled. 32-bit visuals are only available on screens with depths 24 or 30. If you are an application developer, you can use these new visuals in conjunction with a composite manager to create translucent OpenGL applications:

```
  int attrib[] = {
     GLX_RENDER_TYPE, GLX_RGBA_BIT,
     GLX_DRAWABLE_TYPE, GLX_WINDOW_BIT,
     GLX_RED_SIZE, 1,
     GLX_GREEN_SIZE, 1,
     GLX_BLUE_SIZE, 1,
     GLX_ALPHA_SIZE, 1,
     GLX_DOUBLEBUFFER, True,
     GLX_DEPTH_SIZE, 1,
     None };
  GLXFBConfig *fbconfigs, fbconfig;
  int numfbconfigs, render_event_base, render_error_base;
  XVisualInfo *visinfo;
  XRenderPictFormat *pictFormat;

  /* Make sure we have the RENDER extension */
  if(!XRenderQueryExtension(dpy, &render_event_base, &render_error_base)) {
        fprintf(stderr, "No RENDER extension found\n");
        exit(EXIT_FAILURE);
    }

    /* Get the list of FBConfigs that match our criteria */
    fbconfigs = glXChooseFBConfig(dpy, scrnum, attrib, &numfbconfigs);
    if (!fbconfigs) {
        /* None matched */
        exit(EXIT_FAILURE);
    }

    /* Find an FBConfig with a visual that has a RENDER picture format that
     * has alpha */
    for (i = 0; i < numfbconfigs; i++) {
        visinfo = glXGetVisualFromFBConfig(dpy, fbconfigs[i]);
        if (!visinfo) continue;
        pictFormat = XRenderFindVisualFormat(dpy, visinfo->visual);
        if (!pictFormat) continue;

        if(pictFormat->direct.alphaMask > 0) {
            fbconfig = fbconfigs[i];
            break;
        }

        XFree(visinfo);
    }

    if (i == numfbconfigs) {
        /* None of the FBConfigs have alpha.  Use a normal (opaque)
         * FBConfig instead */
        fbconfig = fbconfigs[0];
        visinfo = glXGetVisualFromFBConfig(dpy, fbconfig);
        pictFormat = XRenderFindVisualFormat(dpy, visinfo->visual);
    }

    XFree(fbconfigs);
```

When rendering to a 32-bit window, keep in mind that the X RENDER extension, used by most composite managers, expects "premultiplied alpha" colors. This means that if your color has components (r,g,b) and alpha value a, then you must render (a*r, a*g, a*b, a) into the target window.

More information about Composite can be found at
http://freedesktop.org/Software/CompositeExt

Chapter 23. Using the nvidia-settings Utility

A graphical configuration utility, 'nvidia-settings', is included with the NVIDIA FreeBSD graphics driver. After installing the driver and starting X, you can run this configuration utility by running:

```
    % nvidia-settings
```

in a terminal window.

Detailed information about the configuration options available are documented in the help window in the utility.

For more information, see the nvidia-settings man page.

The source code to nvidia-settings is released as GPL and is available here:
ftp://download.nvidia.com/XFree86/nvidia-settings/

If you have trouble running the nvidia-settings binary shipped with the NVIDIA FreeBSD Graphics Driver, refer to the nvidia-settings entry in Chapter 8.

Chapter 24. Configuring SLI and Multi-GPU FrameRendering

The NVIDIA FreeBSD driver contains support for NVIDIA SLI FrameRendering and NVIDIA Multi-GPU FrameRendering. Both of these technologies allow an OpenGL application to take advantage of multiple GPUs to improve visual performance.

The distinction between SLI and Multi-GPU is straightforward. SLI is used to leverage the processing power of GPUs across two or more graphics cards, while Multi-GPU is used to leverage the processing power of two GPUs colocated on the same graphics card. If you want to link together separate graphics cards, you should use the "SLI" X config option. Likewise, if you want to link together GPUs on the same graphics card, you should use the "MultiGPU" X config option. If you have two cards, each with two GPUs, and you wish to link them all together, you should use the "SLI" option.

24A. RENDERING MODES

In FreeBSD, with two GPUs SLI and Multi-GPU can both operate in one of three modes: Alternate Frame Rendering (AFR), Split Frame Rendering (SFR), and Antialiasing (AA). When AFR mode is active, one GPU draws the next frame while the other one works on the frame after that. In SFR mode, each frame is split horizontally into two pieces, with one GPU rendering each piece. The split line is adjusted to balance the load between the two GPUs. AA mode splits antialiasing work between the two GPUs. Both GPUs work on the same scene and the result is blended together to produce the final frame. This mode is useful for applications that spend most of their time processing with the CPU and cannot benefit from AFR.

With four GPUs, the same options are applicable. AFR mode cycles through all four GPUs, each GPU rendering a frame in turn. SFR mode splits the frame horizontally into four pieces. AA mode splits the work between the four GPUs, allowing antialiasing up to 64x. With four GPUs SLI can also operate in an additional mode, Alternate Frame Rendering of Antialiasing. (AFR of AA). With AFR of AA, pairs of GPUs render alternate frames, each GPU in a pair doing half of the antialiasing work. Note that these scenarios apply whether you have four separate cards or you have two cards, each with two GPUs.

With some GPU configurations, there is in addition a special SLI Mosaic Mode to extend a single X screen transparently across all of the available display outputs on each GPU. See below for the exact set of configurations which can be used with SLI Mosaic Mode.

24B. ENABLING MULTI-GPU

Multi-GPU is enabled by setting the "MultiGPU" option in the X configuration file; see Appendix B for details about the "MultiGPU" option.

The nvidia-xconfig utility can be used to set the "MultiGPU" option, rather than modifying the X configuration file by hand. For example:

```
    % nvidia-xconfig --multigpu=on
```

24C. ENABLING SLI

SLI is enabled by setting the "SLI" option in the X configuration file; see Appendix B for details about the SLI option.

The nvidia-xconfig utility can be used to set the SLI option, rather than modifying the X configuration file by hand. For example:

```
    % nvidia-xconfig --sli=on
```

24D. ENABLING SLI MOSAIC MODE

The simplest way to configure SLI Mosaic Mode using a grid of monitors is to use 'nvidia-settings' (see Chapter 23). The steps to perform this configuration are as follows:

1. Connect each of the monitors you would like to use to any connector from any GPU used for SLI Mosaic Mode. If you are going to use fewer monitors than there are connectors, connect one monitor to each GPU before adding a second monitor to any GPUs.

2. Install the NVIDIA display driver set.

3. Configure an X screen to use the "nvidia" driver on at least one of the GPUs (see Chapter 6 for more information).

4. Start X.

5. Run 'nvidia-settings'. You should see a tab in the left pane of nvidia-settings labeled "SLI Mosaic Mode Settings". Note that you may need to expand the entry for the X screen you configured earlier.

6. Check the "Use SLI Mosaic Mode" check box.

7. Select the monitor grid configuration you'd like to use from the "display configuration" dropdown.

8. Choose the resolution and refresh rate at which you would like to drive each individual monitor.

9. Set any overlap you would like between the displays.

10. Click the "Save to X Configuration File" button. NOTE: If you don't have permissions to write to your system's X configuration file, you will be prompted to choose a location to save the file. After doing so, you MUST copy the X configuration file into a location the X server will consider upon startup (usually '/etc/X11/xorg.conf' for X.Org servers or '/etc/X11/XF86Config' for XFree86 servers).

11. Exit nvidia-settings and restart your X server.

Alternatively, nvidia-xconfig can be used to configure SLI Mosaic Mode via a command like 'nvidia-xconfig --sli=Mosaic --metamodes=METAMODES' where the METAMODES string specifies the desired grid configuration. For example:

```
    nvidia-xconfig --sli=Mosaic --metamodes="GPU-0.DFP-0: 1920x1024+0+0,
GPU-0.DFP-1: 1920x1024+1920+0, GPU-1.DFP-0: 1920x1024+0+1024, GPU-1.DFP-1:
1920x1024+1920+1024"
```

will configure four DFPs in a 2x2 configuration, each running at 1920x1024, with the two DFPs on GPU-0 driving the top two monitors of the 2x2 configuration, and the two DFPs on GPU-1 driving the bottom two monitors of the 2x2 configuration.

See the MetaModes X configuration description in details in Chapter 12. See Appendix C for further details on GPU and Display Device Names.

24E. HARDWARE REQUIREMENTS

SLI functionality requires:

o Identical PCI Express graphics cards

o A supported motherboard (with the exception of Quadro Plex)

o In most cases, a video bridge connecting the two graphics cards

o To use SLI Mosaic Mode, the GPUs must either be part of a Quadro Plex Visual Computing System (VCS) Model IV or newer, or each GPU must be Quadro FX 5800, or Quadro Fermi or newer.

For the latest in supported SLI and Multi-GPU configurations, including SLI- and Multi-GPU capable GPUs and SLI-capable motherboards, see http://www.slizone.com.

24F. OTHER NOTES AND REQUIREMENTS

The following other requirements apply to SLI and Multi-GPU:

- o Mobile GPUs are NOT supported

- o SLI on Quadro-based graphics cards always requires a video bridge

- o TwinView is also not supported with SLI or Multi-GPU. Only one display can be used when SLI or Multi-GPU is enabled, with the exception of Mosaic.

- o If X is configured to use multiple screens and screen 0 has SLI or Multi-GPU enabled, the other screens configured to use the nvidia driver will be disabled. Note that if SLI or Multi-GPU is enabled, the GPUs used by that configuration will be unavailable for single GPU rendering.

FREQUENTLY ASKED SLI AND MULTI-GPU QUESTIONS

Q. Why is glxgears slower when SLI or Multi-GPU is enabled?

A. When SLI or Multi-GPU is enabled, the NVIDIA driver must coordinate the operations of all GPUs when each new frame is swapped (made visible). For most applications, this GPU synchronization overhead is negligible. However, because glxgears renders so many frames per second, the GPU synchronization overhead consumes a significant portion of the total time, and the framerate is reduced.

Q. Why is Doom 3 slower when SLI or Multi-GPU is enabled?

A. The NVIDIA Accelerated FreeBSD Graphics Driver does not automatically detect the optimal SLI or Multi-GPU settings for games such as Doom 3 and Quake 4. To work around this issue, the environment variable __GL_DOOM3 can be set to tell OpenGL that Doom 3's optimal settings should be used. In

Bash, this can be done in the same command that launches Doom 3 so the environment variable does not remain set for other OpenGL applications started in the same session:

```
% __GL_DOOM3=1 doom3
```

Doom 3's startup script can also be modified to set this environment variable:

```
#!/bin/sh
# Needed to make symlinks/shortcuts work.
# the binaries must run with correct working directory
cd "/usr/local/games/doom3/"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
export __GL_DOOM3=1
exec ./doom.x86 "$@"
```

This environment variable is temporary and will be removed in the future.

Chapter 25. Configuring Frame Lock and Genlock

NOTE: Frame Lock and Genlock features are supported only on specific hardware, as noted below.

Visual computing applications that involve multiple displays, or even multiple windows within a display, can require special signal processing and application controls in order to function properly. For example, in order to produce quality video recording of animated graphics, the graphics display must be synchronized with the video camera. As another example, applications presented on multiple displays must be synchronized in order to complete the illusion of a larger, virtual canvas.

This synchronization is enabled through the frame lock and genlock capabilities of the NVIDIA driver. This section describes the setup and use of frame lock and genlock.

25A. DEFINITION OF TERMS

GENLOCK: Genlock refers to the process of synchronizing the pixel scanning of one or more displays to an external synchronization source. NVIDIA Genlock requires the external signal to be either TTL or composite, such as used for NTSC, PAL, or HDTV. It should be noted that the NVIDIA Genlock implementation is guaranteed only to be frame-synchronized, and not necessarily pixel-synchronized.

FRAME LOCK: Frame Lock involves the use of hardware to synchronize the frames on each display in a connected system. When graphics and video are displayed across multiple monitors, frame locked systems help maintain image continuity to create a virtual canvas. Frame lock is especially critical for stereo viewing, where the left and right fields must be in sync across all displays.

In short, to enable genlock means to sync to an external signal. To enable frame lock means to sync 2 or more display devices to a signal generated internally by the hardware, and to use both means to sync 2 or more display devices to an external signal.

SWAP SYNC: Swap sync refers to the synchronization of buffer swaps of multiple application windows. By means of swap sync, applications running on multiple systems can synchronize the application buffer swaps between all the systems. In order to work across multiple systems, swap sync requires that the systems are frame locked.

G-SYNC DEVICE: A G-Sync Device refers to a device capable of Frame lock/Genlock. See "Supported Hardware" below.

25B. SUPPORTED HARDWARE

Frame lock and genlock are supported for the following hardware:

o Quadro G-Sync, used in conjunction with a Quadro FX 4400, Quadro FX 4500, or Quadro FX 5500

o Quadro G-Sync II, used in conjunction with a Quadro FX 4600, Quadro FX 5600, Quadro FX 4800, Quadro FX 5800, Quadro 5000, or Quadro 6000

o Quadro Plex 1000 Models I, II and IV, Quadro Plex 2200 D2, or Quadro Plex 7000.

25C. HARDWARE SETUP

Before you begin, you should check that your hardware has been properly installed. The following steps must be performed while the system is off.

1. On the Quadro G-Sync card, locate the G-Sync connector labeled "primary". If the associated ribbon cable is not already joined to this connector, do so now. If you plan to use frame lock or genlock in conjunction with SLI FrameRendering or Multi-GPU FrameRendering (see Chapter 24) or other multi-GPU configurations, you should connect the G-Sync connector labeled "secondary" to the second GPU. A section at the end of this appendix describes restrictions on such setups.

2. Install the Quadro G-Sync card in any available slot. Note that the slot itself is only used for physical mounting, so even a known "bad" slot is acceptable. The slot must be close enough to the graphics card that the ribbon cable can reach.

3. Connect the other end of the ribbon cable to the G-Sync connector on the graphics card.

You may now boot the system and begin the software setup of genlock and/or frame lock. These instructions assume that you have already successfully installed the NVIDIA Accelerated FreeBSD Driver Set. If you have not done so, see Chapter 3.

25D. CONFIGURATION WITH NVIDIA-SETTINGS GUI

Frame lock and genlock are configured through the nvidia-settings utility. See the 'nvidia-settings(1)' man page, and the nvidia-settings online help (click the "Help" button in the lower right corner of the interface for per-page help information).

From the nvidia-settings frame lock panel, you may control the addition of G-Sync (and display) devices to the frame lock/genlock group, monitor the status of that group, and enable/disable frame lock and genlock.

After the system has booted and X Windows has been started, run nvidia-settings as

```
    % nvidia-settings
```

You may wish to start this utility before continuing, as we refer to it
frequently in the subsequent discussion.

The setup of genlock and frame lock are described separately. We then describe
the use of genlock and frame lock together.

25E. GENLOCK SETUP

After the system has been booted, connect the external signal to the house
sync connector (the BNC connector) on either the graphics card or the G-Sync
card. There is a status LED next to the connector. A solid red LED indicates
that the hardware cannot detect the timing signal. A green LED indicates that
the hardware is detecting a timing signal. An occasional red flash is okay.
The G-Sync device (graphics card or G-Sync card) will need to be configured
correctly for the signal to be detected.

In the frame lock panel of the nvidia-settings interface, add the X Server
that contains the display and G-Sync devices that you would like to sync to
this external source by clicking the "Add Devices..." button. An X Server is
typically specified in the format "system:m", e.g.:

```
    mycomputer.domain.com:0
```

or

```
    localhost:0
```

After adding an X Server, rows will appear in the "G-Sync Devices" section on
the frame lock panel that displays relevant status information about the
G-Sync devices, GPUs attached to those G-Sync devices and the display devices
driven by those GPUs. In particular, the G-Sync rows will display the server
name and G-Sync device number along with "Receiving" LED, "Rate", "House" LED,
"Port 0"/"Port 1" Images, and "Delay" information. The GPU rows will display
the GPU product name information along with the GPU ID for the server. The
Display Device rows will show the display device name and device type along
with server/client check boxes, refresh rate, "Timing" LED and "Stereo" LED.

Once the G-Sync and display devices have been added to the frame lock/genlock
group, a Server display device will need to be selected. This is done by
selecting the "Server" check box of the desired display device.

If you are using a G-Sync card, you must also click the "Use House Sync if

Present" check box. To enable synchronization of this G-Sync device to the external source, click the "Enable Frame Lock" button. The display device(s) may take a moment to stabilize. If it does not stabilize, you may have selected a synchronization signal that the system cannot support. You should disable synchronization by clicking the "Disable Frame Lock" button and check the external sync signal.

Modifications to genlock settings (e.g., "Use House Sync if Present", "Add Devices...") must be done while synchronization is disabled.

25F. FRAME LOCK SETUP

Frame Lock is supported across an arbitrary number of Quadro G-Sync systems, although mixing Quadro G-Sync and Quadro G-Sync II in the same frame lock group is not supported. Additionally, each system to be included in the frame lock group must be configured with identical mode timings. See Chapter 18 for information on mode timings.

Connect the systems through their RJ45 ports using standard CAT5 patch cables. These ports are located on the frame lock card. DO NOT CONNECT A FRAME LOCK PORT TO AN ETHERNET CARD OR HUB. DOING SO MAY PERMANENTLY DAMAGE THE HARDWARE.
The connections should be made in a daisy-chain fashion: each card has two RJ45 ports, call them 1 and 2. Connect port 1 of system A to port 2 of system B, connect port 1 of system B to port 2 of system C, etc. Note that you will always have two empty ports in your frame lock group.

The ports self-configure as inputs or outputs once frame lock is enabled. Each port has a yellow and a green LED that reflect this state. A flashing yellow LED indicates an output and a flashing green LED indicates an input. A solid green LED indicates that the port has not yet configured.

In the frame lock panel of the nvidia-settings interface, add the X server that contains the display devices that you would like to include in the frame lock group by clicking the "Add Devices..." button (see the description for adding display devices in the previous section on GENLOCK SETUP. Like the genlock status indicators, the "Port 0" and "Port 1" columns in the table on the frame lock panel contain indicators whose states mirror the states of the physical LEDs on the RJ45 ports. Thus, you may monitor the status of these ports from the software interface.

Any X Server can be added to the frame lock group, provided that

1. The system supporting the X Server is configured to support frame lock and is connected via RJ45 cable to the other systems in the frame lock group.

2. The system driving nvidia-settings can communicate with the X server that is to be included for frame lock. This means that either the server must be listening over TCP and the system's firewall is permissive enough to allow remote X11 display connections, or that you've configured an alternative mechanism such as ssh(1) forwarding between the machines.

   For the case of listening over TCP, verify that the "-nolisten tcp" commandline option was not used when starting the X server. You can find the X server commandline with a command such as

   ```
   % ps ax | grep X
   ```

   If "-nolisten tcp" is on the X server commandline, consult your FreeBSD distribution documentation for details on how to properly remove this option. For example, distributions configured to use the GDM login manager may need to set "DisallowTCP=false" in the GDM configuration file (e.g., /etc/gdm/custom.conf, /etc/X11/gdm/gdm.conf, or /etc/gdb/gdb.conf; the exact configuration file name and path varies by the distribution). Or, distributions configured to use the KDM login manager may have the line

   ```
   ServerArgsLocal=-nolisten tcp
   ```

   in their kdm file (e.g., /etc/kde3/kdm/kdmrc). This line can be commented out by prepending with "#".

3. The system driving nvidia-settings can locate and has display privileges on the X server that is to be included for frame lock.

   A system can gain display privileges on a remote system by executing

   ```
   % xhost +
   ```

   on the remote system. See the xhost(1) man page for details.

Typically, frame lock is controlled through one of the systems that will be included in the frame lock group. While this is not a requirement, note that nvidia-settings will only display the frame lock panel when running on an X server that supports frame lock.

To enable synchronization on these display devices, click the "Enable Frame Lock" button. The screens may take a moment to stabilize. If they do not stabilize, you may have selected mode timings that one or more of the systems cannot support. In this case you should disable synchronization by clicking the "Disable Frame Lock" button and refer to Chapter 18 for information on mode timings.

Modifications to frame lock settings (e.g. "Add/Remove Devices...") must be done while synchronization is disabled.

nvidia-settings will not automatically enable Frame Lock via the nvidia-settings.rc file. To enable Frame Lock when starting the X server, a line such as the following can be added to the '~/.xinitrc' file:

```
    # nvidia-settings -a [gpu:0]/FrameLockEnable=1
```

25G. FRAME LOCK + GENLOCK

The use of frame lock and genlock together is a simple extension of the above instructions for using them separately. You should first follow the instructions for Frame Lock Setup, and then to one of the systems that will be included in the frame lock group, attach an external sync source. In order to sync the frame lock group to this single external source, you must select a display device driven by the GPU connected to the G-Sync card (through the primary connector) that is connected to the external source to be the signal server for the group. This is done by selecting the check box labeled "Server" of the tree on the frame lock panel in nvidia-settings. If you are using a G-Sync based frame lock group, you must also select the "Use House Sync if Present" check box. Enable synchronization by clicking the "Enable Frame Lock" button. As with other frame lock/genlock controls, you must select the signal server while synchronization is disabled.

25H. CONFIGURATION WITH NVIDIA-SETTINGS COMMAND LINE

Frame Lock may also be configured through the nvidia-settings command line. This method of configuring Frame Lock may be useful in a scripted environment to automate the setup process. (Note that the examples listed below depend on the actual hardware configuration and as such may not work as-is.)

To properly configure Frame Lock, the following steps should be completed:

1. Make sure Frame Lock Sync is disabled on all GPUs.

2. Make sure all display devices that are to be frame locked have the same refresh rate.

3. Configure which (display/GPU) device should be the master.

4. Configure house sync (if applicable).

5. Configure the slave display devices.

6. Enable frame lock sync on the master GPU.

7. Enable frame lock sync on the slave GPUs.

8. Toggle the test signal on the master GPU (for testing the hardware connectivity.)

For a full list of the nvidia-settings Frame Lock attributes, please see the 'nvidia-settings(1)' man page. Examples:

1. 1 System, 1 Frame Lock board, 1 GPU, and 1 display device syncing to the house signal:

```
# - Make sure frame lock sync is disabled
nvidia-settings -a [gpu:0]/FrameLockEnable=0
nvidia-settings -q [gpu:0]/FrameLockEnable

# - Query the enabled displays on the gpu
nvidia-settings -q [gpu:0]/EnabledDisplays

# - Check that the refresh rate is the one we want
nvidia-settings -q [gpu:0]/RefreshRate

# - Set the master display device to CRT-0.  The desired display
#   device(s) to be set are passed in as a hexadecimal number
#   in which specific bits denote which display devices to set.
#   examples:
#
```

```
#   0x00000001 - CRT-0
#   0x00000002 - CRT-1
#   0x00000003 - CRT-0 and CRT-1
#
#   0x00000100 - TV-0
#   0x00000200 - TV-1
#
#   0x00020000 - DFP-1
#
#   0x00010101 - CRT-0, TV-0 and DFP-0
#
#   0x000000FF - All CRTs
#   0x0000FF00 - All TVs
#   0x00FF0000 - All DFPs
#
#   Note that the following command:
#
#     nvidia-settings -q [gpu:0]/EnabledDisplays
#
#   will list the available displays on the given GPU.

nvidia-settings -a [gpu:0]/FrameLockMaster=0x00000001
nvidia-settings -q [gpu:0]/FrameLockMaster

# - Enable use of house sync signal
nvidia-settings -a [framelock:0]/FrameLockUseHouseSync=1

# - Configure the house sync signal video mode
nvidia-settings -a [framelock:0]/FrameLockVideoMode=0

# - Set the slave display device to none (to avoid
#   having unwanted display devices locked to the
#   sync signal.)
nvidia-settings -a [gpu:0]/FrameLockSlaves=0x00000000
nvidia-settings -q [gpu:0]/FrameLockSlaves

# - Enable frame lock
nvidia-settings -a [gpu:0]/FrameLockEnable=1

# - Toggle the test signal
nvidia-settings -a [gpu:0]/FrameLockTestSignal=1
nvidia-settings -a [gpu:0]/FrameLockTestSignal=0
```

2. 2 Systems, each with 2 GPUs, 1 Frame Lock board and 1 display device per
   GPU syncing from the first system's first display device:

```
    # - Make sure frame lock sync is disabled
    nvidia-settings -a myserver:0[gpu:0]/FrameLockEnable=0
    nvidia-settings -a myserver:0[gpu:1]/FrameLockEnable=0
    nvidia-settings -a myslave1:0[gpu:0]/FrameLockEnable=0
    nvidia-settings -a myslave1:0[gpu:1]/FrameLockEnable=0

    # - Query the enabled displays on the GPUs
    nvidia-settings -q myserver:0[gpu:0]/EnabledDisplays
    nvidia-settings -q myserver:0[gpu:1]/EnabledDisplays
    nvidia-settings -q myslave1:0[gpu:0]/EnabledDisplays
    nvidia-settings -q myslave1:0[gpu:1]/EnabledDisplays

    # - Check the refresh rate is the same for all displays
    nvidia-settings -q myserver:0[gpu:0]/RefreshRate
    nvidia-settings -q myserver:0[gpu:1]/RefreshRate
    nvidia-settings -q myslave1:0[gpu:0]/RefreshRate
    nvidia-settings -q myslave1:0[gpu:1]/RefreshRate

    # - Make sure the display device we want as master is masterable
    nvidia-settings -q myserver:0[gpu:0]/FrameLockMasterable

    # - Set the master display device (CRT-0)
    nvidia-settings -a myserver:0[gpu:0]/FrameLockMaster=0x00000001

    # - Disable the house sync signal on the master device
    nvidia-settings -a myserver:0[framelock:0]/FrameLockUseHouseSync=0

    # - Set the slave display devices
    nvidia-settings -a myserver:0[gpu:1]/FrameLockSlaves=0x00000001
    nvidia-settings -a myslave1:0[gpu:0]/FrameLockSlaves=0x00000001
    nvidia-settings -a myslave1:0[gpu:1]/FrameLockSlaves=0x00000001

    # - Enable frame lock on server
    nvidia-settings -a myserver:0[gpu:0]/FrameLockEnable=1

    # - Enable frame lock on slave devices
    nvidia-settings -a myserver:0[gpu:1]/FrameLockEnable=1
    nvidia-settings -a myslave1:0[gpu:0]/FrameLockEnable=1
    nvidia-settings -a myslave1:0[gpu:1]/FrameLockEnable=1
    # - Toggle the test signal
    nvidia-settings -a myserver:0[gpu:0]/FrameLockTestSignal=1
    nvidia-settings -a myserver:0[gpu:0]/FrameLockTestSignal=0
```

3. 1 System, 4 GPUs, 2 Frame Lock boards and 2 display devices per GPU
   syncing from the first GPU's display device:

```
    # - Make sure frame lock sync is disabled
    nvidia-settings -a [gpu:0]/FrameLockEnable=0
    nvidia-settings -a [gpu:1]/FrameLockEnable=0
    nvidia-settings -a [gpu:2]/FrameLockEnable=0
    nvidia-settings -a [gpu:3]/FrameLockEnable=0

    # - Query the enabled displays on the GPUs
    nvidia-settings -q [gpu:0]/EnabledDisplays
    nvidia-settings -q [gpu:1]/EnabledDisplays
    nvidia-settings -q [gpu:2]/EnabledDisplays
    nvidia-settings -q [gpu:3]/EnabledDisplays

    # - Check the refresh rate is the same for all displays
    nvidia-settings -q [gpu:0]/RefreshRate
    nvidia-settings -q [gpu:1]/RefreshRate
    nvidia-settings -q [gpu:2]/RefreshRate
    nvidia-settings -q [gpu:3]/RefreshRate

    # - Make sure the display device we want as master is masterable
    nvidia-settings -q myserver:0[gpu:0]/FrameLockMasterable

    # - Set the master display device (CRT-0)
    nvidia-settings -a [gpu:0]/FrameLockMaster=0x00000001

    # - Disable the house sync signal on the master device
    nvidia-settings -a [framelock:0]/FrameLockUseHouseSync=0

    # - Set the slave display devices
    nvidia-settings -a [gpu:0]/FrameLockSlaves=0x00000002 # CRT-1
    nvidia-settings -a [gpu:1]/FrameLockSlaves=0x00000003 # CRT-0 and CRT-1
    nvidia-settings -a [gpu:2]/FrameLockSlaves=0x00000003 # CRT-0 and CRT-1
    nvidia-settings -a [gpu:3]/FrameLockSlaves=0x00000003 # CRT-0 and CRT-1

    # - Enable frame lock on master GPU
    nvidia-settings -a [gpu:0]/FrameLockEnable=1

    # - Enable frame lock on slave devices
    nvidia-settings -a [gpu:1]/FrameLockEnable=1
    nvidia-settings -a [gpu:2]/FrameLockEnable=1
    nvidia-settings -a [gpu:3]/FrameLockEnable=1

    # - Toggle the test signal
    nvidia-settings -a [gpu:0]/FrameLockTestSignal=1
    nvidia-settings -a [gpu:0]/FrameLockTestSignal=0
```

25I. LEVERAGING FRAME LOCK/GENLOCK IN OPENGL

With the GLX_NV_swap_group extension, OpenGL applications can be implemented to join a group of applications within a system for local swap sync, and bind the group to a barrier for swap sync across a frame lock group. A universal frame counter is also provided to promote synchronization across applications.

25J. FRAME LOCK RESTRICTIONS:

The following restrictions must be met for enabling frame lock:

1. All display devices set as client in a frame lock group must have the same mode timings as the server (master) display device. If a House Sync signal is used (instead of internal timings), all client display devices must be set to have the same refresh rate as the incoming house sync signal.

2. All X Screens (driving the selected client/server display devices) must have the same stereo setting. See Appendix B for instructions on how to set the stereo X option.

3. The frame lock server (master) display device must be on a GPU on the primary connector to a G-Sync device.

4. If connecting a single GPU to a G-Sync device, the primary connector must be used.

5. In configurations with more than one display device per GPU, we recommend

enabling frame lock on all display devices on those GPUs.

6. Virtual terminal switching or mode switching will disable frame lock on the display device. Note that the glXQueryFrameCountNV entry point (provided by the GLX_NV_swap_group extension) will only provide incrementing numbers while frame lock is enabled. Therefore, applications that use glXQueryFrameCountNV to control animation will appear to stop animating while frame lock is disabled.

25K. SUPPORTED FRAME LOCK CONFIGURATIONS:

The following configurations are currently supported:

1. Basic Frame Lock: Single GPU, Single X Screen, Single Display Device with or without OpenGL applications that make use of Quad-Buffered Stereo and/or the GLX_NV_swap_group extension.

2. Frame Lock + TwinView: Single GPU, Single X Screen, Multiple Display Devices with or without OpenGL applications that make use of Quad-Buffered Stereo and/or the GLX_NV_swap_group extension.

3. Frame Lock + Xinerama: 1 or more GPU(s), Multiple X Screens, Multiple Display Devices with or without OpenGL applications that make use of Quad-Buffered Stereo and/or the GLX_NV_swap_group extension.

4. Frame Lock + TwinView + Xinerama: 1 or more GPU(s), Multiple X Screens, Multiple Display Devices with or without OpenGL applications that make use of Quad-Buffered Stereo and/or the GLX_NV_swap_group extension.

5. Frame Lock + SLI SFR, AFR, or AA: 2 GPUs, Single X Screen, Single Display Device with either OpenGL applications that make use of Quad-Buffered Stereo or the GLX_NV_swap_group extension. Note that for Frame Lock + SLI Frame Rendering applications that make use of both Quad-Buffered Stereo and the GLX_NV_swap_group extension are not supported. Note that only 2-GPU SLI configurations are currently supported.

6. Frame Lock + Multi-GPU SFR, AFR, or AA: 2 GPUs, Single X Screen, Single Display Device with either OpenGL applications that make use of Quad-Buffered Stereo or the GLX_NV_swap_group extension. Note that for Frame Lock + Multi-GPU Frame Rendering applications that make use of both Quad-Buffered Stereo and the GLX_NV_swap_group extension are not

supported.

Chapter 26. Configuring SDI Video Output

Broadcast, film, and video post production and digital cinema applications can require Serial Digital (SDI) or High Definition Serial Digital (HD-SDI) video output. SDI/HD-SDI is a digital video interface used for the transmission of uncompressed video signals as well as packetized data. SDI is standardized in ITU-R BT.656 and SMPTE 259M while HD-SDI is standardized in SMPTE 292M. SMPTE 372M extends HD-SDI to define a dual-link configuration that uses a pair of SMPTE 292M links to provide a 2.970 Gbit/second interface. SMPTE 424M extends the interface further to define a single 2.97 Gbit/second serial data link.

SDI and HD-SDI video output is provided through the use of the NVIDIA driver along with an NVIDIA SDI output daughter board. In addition to single- and dual-link SDI/HD-SDI digital video output, frame lock and genlock synchronization are provided in order to synchronize the outgoing video with an external source signal (see Chapter 25 for details on these technologies). This section describes the setup and use of the SDI video output.

26A. HARDWARE SETUP

Before you begin, you should check that your hardware has been properly installed. If you are using the Quadro FX 4000 SDI, the SDI/HD-SDI hardware is located on the dual-slot card itself, and after installing the card, no additional setup is necessary. If you are using the Quadro FX 4500/5500 SDI or Quadro FX 4600/5600 SDI II, the following additional setup steps are required in order to connect the SDI daughter card to the graphics card. These steps must be performed when the system is off.

1. Insert the NVIDIA SDI Output card into any available expansion slot within six inches of the NVIDIA Quadro graphics card. Secure the card's bracket using the method provided by the chassis manufacturer (usually a thumb screw or an integrated latch).

2. Connect one end of the 14-pin ribbon cable to the G-Sync connector on the NVIDIA Quadro graphics card, and the other end to the NVIDIA SDI output card.

3. On Quadro FX 4500/5500 SDI, connect the SMA-to-BNC cables by screwing the male SMA connectors onto the female SMA connectors on the NVIDIA SDI output card. On Quadro FX 4600/5600 SDI II, this step is not necessary: the SDI II has BNC connectors rather than SMA connectors.

4. Connect the DVI-loopback connector by connecting one end of the DVI cable to the DVI connector on the NVIDIA SDI output card and the other end to the "north" DVI connector on the NVIDIA Quadro graphics card. The "north" DVI connector on the NVIDIA Quadro graphics card is the DVI connector that is the farthest from the graphics card PCIe connection to the motherboard. The SDI output card will NOT function properly if this cable is connected to the "south" DVI connector.

Once the above installation is complete, you may boot the system and configure the SDI video output using nvidia-settings. These instructions assume that you have already successfully installed the NVIDIA FreeBSD Accelerated Graphics Driver. If you have not done so, see Chapter 3 for details.

26B. CLONE MODE CONFIGURATION WITH 'nvidia-settings'

SDI video output is configured through the nvidia-settings utility. See the 'nvidia-settings(1)' man page, and the nvidia-settings online help (click the "Help" button in the lower right corner of the interface for per-page help information).

After the system has booted and X Windows has been started, run nvidia-settings as

```
    % nvidia-settings
```

When the NVIDIA X Server Settings page appears, follow the steps below to configure the SDI video output.

1. Click on the "Graphics to Video Out" tree item on the side menu. This will open the "Graphics to Video Out" page.

2. Go to the "Synchronization Options" subpage and choose a synchronization method. From the "Sync Options" drop down click the list arrow to the right and then click the method that you want to use to synchronize the SDI output.

```
    Sync Method      Description
    -------------    ------------------------------------------------
    Free Running     The SDI output will be synchronized with the
                     timing chosen from the SDI signal format list.
    Genlock          SDI output will be synchronized with the external
                     sync signal.
    Frame Lock       The SDI output will be synchronized with the
                     timing chosen from the SDI signal format list. In
                     this case, the list of available timings is
                     limited to those timings that can be synchronized
                     with the detected external sync signal.
```

Note that on Quadro FX 4600/5600 SDI II, you must first choose the correct Sync Format before an incoming sync signal will be detected.

3. From the top Graphics to Video Out page, choose the output video format that will control the video resolution, field rate, and SMPTE signaling standard for the outgoing video stream. From the "Clone Mode" drop down box, click the "Video Format" arrow and then click the signal format that you would like to use. Note that only those resolutions that are smaller or equal to the desktop resolution will be available. Also, this list is pruned according to the sync option selected. If genlock synchronization is chosen, the output video format is automatically set to match the incoming video sync format and this drop down list will be grayed out preventing you from choosing another format. If frame lock synchronization has been selected, then only those modes that are compatible with the detected sync signal will be available.

4. Choose the output data format from the "Output Data Format" drop down list.

5. Click the "Enable SDI Output" button to enable video output using the settings above. The status of the SDI output can be verified by examining the LED indicators in the "Graphics to SDI property" page banner.

6. To subsequently stop SDI output, simply click on the button that now says "Disable SDI Output".

7. In order to change any of the SDI output parameters such as the Output Video Format, Output Data Format as well as the Synchronization Delay, it is necessary to first disable the SDI output.

26C. CONFIGURATION FOR TWINVIEW OR AS A SEPARATE X SCREEN

SDI video output can be configured through the nvidia-settings X Server Display Configuration page, for use in TwinView or as a separate X screen. The SDI video output can be configured as if it were a digital flat panel, choosing the resolution, refresh rate, and position within the desktop.

Similarly, the SDI video output can be configured for use in TwinView or as a separate X screen through the X configuration file. The supported SDI video output modes can be requested by name anywhere a mode name can be used in the X configuration file (either in the "Modes" line, or in the "MetaModes" option). E.g.,

```
 Option "MetaModes" "CRT-0:nvidia-auto-select, DFP-1:1280x720_60.00_smpte296"
```

The mode names are reported in the nvidia-settings Display Configuration page when in advanced mode.

As well, the initial output data format, sync mode and sync source can be set via the Appendix B, Appendix B, and Appendix B. See Appendix B for instructions on how to set these X options.

Note that SDI "Clone Mode" as configured through the Graphics to Video Out page in nvidia-settings is mutually exclusive with using the SDI video output in TwinView or as a separate X screen.

Chapter 27. Configuring Depth 30 Displays

This driver release supports X screens with screen depths of 30 bits per pixel (10 bits per color component) on NVIDIA GPUs based on G80 and higher chip architectures. This provides about 1 billion possible colors, allowing for higher color precision and smoother gradients.

When displaying a depth 30 image, the color data may be dithered to lower bit depths, depending on the capabilities of the display device and how it is connected to the GPU. Some devices connected via analog VGA or DisplayPort can display the full 10 bit range of colors. Devices connected via DVI or HDMI, as well as laptop internal panels connected via LVDS, will be dithered to 8 or 6 bits per pixel.

To work reliably, depth 30 requires X.Org 7.3 or higher and pixman 0.11.6 or higher.

In addition to the above software requirements, many X applications and toolkits do not understand depth 30 visuals as of this writing. Some programs may work correctly, some may work but display incorrect colors, and some may simply fail to run. In particular, many OpenGL applications request 8 bits of alpha when searching for FBConfigs. Since depth 30 visuals have only 2 bits of alpha, no suitable FBConfigs will be found and such applications will fail to start.

Chapter 28. NVIDIA Contact Info and Additional Resources

If you believe that you have found a bug or have a problem that you need assistance with and cannot find the solution elsewhere, or if you have found inaccuracies in this document, send email to freebsd-gfx-bugs@nvidia.com

NVIDIA provides a technical contact for information about potential security issues. Anyone who has identified what they believe to be a security issue with an NVIDIA UNIX driver is encouraged to directly contact the NVIDIA UNIX Graphics Driver security email alias, unix-security@nvidia.com, to report and evaluate any potential issues prior to publishing a public security advisory.

Additional Resources

XFree86 Video Timings HOWTO

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

The X.Org Foundation

    http://www.x.org/

OpenGL

http://www.opengl.org/

Chapter 29. Credits

The port of the NVIDIA driver to FreeBSD is due in no small part to the many contributions of Christian Zander <zander 'at' mail.minion.de> and Matthew N. Dodd <mdodd 'at' freebsd.org>.

Chapter 30. Acknowledgements

PNG Library

The NVIDIA X driver splash screen is decoded using 'libpng': http://libpng.org/pub/png/libpng.html

X.Org

This NVIDIA FreeBSD driver contains code from the X.Org project.

NetBSD Compiler Intrinsics

The NetBSD implementations of the following compiler intrinsics are used for better portability: __udivdi3, __umoddi3, __divdi3, __moddi3, __ucmpdi2, __cmpdi2, __fixunssfdi, __fixunsdfdi, __ashldi3 and __lshrdi3.

These carry the following copyright notice:

Appendix A. Supported NVIDIA GPU Products

For the most complete and accurate listing of supported GPUs, please see the Supported Products List, available from the NVIDIA FreeBSD x86_64 Graphics Driver download page. Please go to http://www.nvidia.com/object/unix.html, follow the Archive link under the FreeBSD x86_64 heading, follow the link for the 304.43 driver, and then go to the Supported Products List.

For an explanation of the VDPAU features column, please refer to the section called "VdpDecoder" of Appendix G.

Note that the list of supported GPU products provided below and on the driver download page is provided to indicate which GPUs are supported by a particular driver version. Some designs incorporating supported GPUs may not be compatible with the NVIDIA FreeBSD driver: in particular, notebook and all-in-one desktop designs with switchable (hybrid) or Optimus graphics will not work if means to disable the integrated graphics in hardware are not available. Hardware designs will vary from manufacturer to manufacturer, so please consult with a system's manufacturer to determine whether that particular system is compatible.

A1. NVIDIA GEFORCE GPUS

NVIDIA GPU product	Device PCI ID*	VDPAU features
GeForce 6800 Ultra	0x0040	-
GeForce 6800	0x0041	-
GeForce 6800 LE	0x0042	-
GeForce 6800 XE	0x0043	-
GeForce 6800 XT	0x0044	-
GeForce 6800 GT	0x0045	-
GeForce 6800 GT	0x0046	-
GeForce 6800 GS	0x0047	-
GeForce 6800 XT	0x0048	-
GeForce 7800 GTX	0x0090	-
GeForce 7800 GTX	0x0091	-
GeForce 7800 GT	0x0092	-
GeForce 7800 GS	0x0093	-
GeForce 7800 SLI	0x0095	-
GeForce Go 7800	0x0098	-
GeForce Go 7800 GTX	0x0099	-
GeForce 6800 GS	0x00C0	-
GeForce 6800	0x00C1	-

GeForce 6800 LE	0x00C2	-
GeForce 6800 XT	0x00C3	-
GeForce Go 6800	0x00C8	-
GeForce Go 6800 Ultra	0x00C9	-
GeForce 6600 GT	0x00F1	-
GeForce 6600	0x00F2	-
GeForce 6200	0x00F3	-
GeForce 6600 LE	0x00F4	-
GeForce 7800 GS	0x00F5	-
GeForce 6800 GS	0x00F6	-
GeForce 6800 Ultra	0x00F9	-
GeForce 6600 GT	0x0140	-
GeForce 6600	0x0141	-
GeForce 6600 LE	0x0142	-
GeForce 6600 VE	0x0143	-
GeForce Go 6600	0x0144	-
GeForce 6610 XL	0x0145	-
GeForce Go 6600 TE/6200 TE	0x0146	-
GeForce 6700 XL	0x0147	-
GeForce Go 6600	0x0148	-
GeForce Go 6600 GT	0x0149	-
GeForce 6200	0x014F	-
GeForce 6500	0x0160	-
GeForce 6200 TurboCache(TM)	0x0161	-
GeForce 6200SE TurboCache(TM)	0x0162	-
GeForce 6200 LE	0x0163	-
GeForce Go 6200	0x0164	-
GeForce Go 6400	0x0166	-
GeForce Go 6200	0x0167	-
GeForce Go 6400	0x0168	-
GeForce 6250	0x0169	-
GeForce 7100 GS	0x016A	-
GeForce 8800 GTX	0x0191	-
GeForce 8800 GTS	0x0193	-
GeForce 8800 Ultra	0x0194	-
Tesla C870	0x0197	-
GeForce 7350 LE	0x01D0	-
GeForce 7300 LE	0x01D1	-
GeForce 7550 LE	0x01D2	-
GeForce 7300 SE/7200 GS	0x01D3	-
GeForce Go 7200	0x01D6	-
GeForce Go 7300	0x01D7	-
GeForce Go 7400	0x01D8	-

GeForce 7500 LE	0x01DD	-
GeForce 7300 GS	0x01DF	-
GeForce 6200	0x0221	-
GeForce 6200 A-LE	0x0222	-
GeForce 6150	0x0240	-
GeForce 6150 LE	0x0241	-
GeForce 6100	0x0242	-
GeForce Go 6150	0x0244	-
GeForce Go 6100	0x0247	-
GeForce 7900 GTX	0x0290	-
GeForce 7900 GT/GTO	0x0291	-
GeForce 7900 GS	0x0292	-
GeForce 7950 GX2	0x0293	-
GeForce 7950 GX2	0x0294	-
GeForce 7950 GT	0x0295	-
GeForce Go 7950 GTX	0x0297	-
GeForce Go 7900 GS	0x0298	-
GeForce 7600 GT	0x02E0	-
GeForce 7600 GS	0x02E1	-
GeForce 7300 GT	0x02E2	-
GeForce 7900 GS	0x02E3	-
GeForce 7950 GT	0x02E4	-
GeForce 7650 GS	0x038B	-
GeForce 7650 GS	0x0390	-
GeForce 7600 GT	0x0391	-
GeForce 7600 GS	0x0392	-
GeForce 7300 GT	0x0393	-
GeForce 7600 LE	0x0394	-
GeForce 7300 GT	0x0395	-
GeForce Go 7700	0x0397	-
GeForce Go 7600	0x0398	-
GeForce Go 7600 GT	0x0399	-
GeForce 6150SE nForce 430	0x03D0	-
GeForce 6100 nForce 405	0x03D1	-
GeForce 6100 nForce 400	0x03D2	-
GeForce 6100 nForce 420	0x03D5	-
GeForce 7025 / nForce 630a	0x03D6	-
GeForce 8600 GTS	0x0400	A
GeForce 8600 GT	0x0401	A
GeForce 8600 GT	0x0402	A
GeForce 8600 GS	0x0403	A
GeForce 8400 GS	0x0404	A
GeForce 9500M GS	0x0405	A

GeForce 8300 GS	0x0406	-
GeForce 8600M GT	0x0407	A
GeForce 9650M GS	0x0408	A
GeForce 8700M GT	0x0409	A
GeForce GT 330	0x0410	A
GeForce 8400 SE	0x0420	-
GeForce 8500 GT	0x0421	A
GeForce 8400 GS	0x0422	A
GeForce 8300 GS	0x0423	-
GeForce 8400 GS	0x0424	A
GeForce 8600M GS	0x0425	A
GeForce 8400M GT	0x0426	A
GeForce 8400M GS	0x0427	A
GeForce 8400M G	0x0428	A
GeForce 9400 GT	0x042C	A
GeForce 9300M G	0x042E	A
GeForce 7150M / nForce 630M	0x0531	-
GeForce 7000M / nForce 610M	0x0533	-
GeForce 7050 PV / nForce 630a	0x053A	-
GeForce 7050 PV / nForce 630a	0x053B	-
GeForce 7025 / nForce 630a	0x053E	-
GeForce GTX 295	0x05E0	A
GeForce GTX 280	0x05E1	A
GeForce GTX 260	0x05E2	A
GeForce GTX 285	0x05E3	A
GeForce GTX 275	0x05E6	A
Tesla C1060	0x05E7	A
Tesla T10 Processor	0x05E7 0x0595	A
Tesla T10 Processor	0x05E7 0x068F	A
Tesla M1060	0x05E7 0x0697	A
Tesla M1060	0x05E7 0x0743	A
GeForce GTX 260	0x05EA	A
GeForce GTX 295	0x05EB	A
GeForce 8800 GTS 512	0x0600	A
GeForce 9800 GT	0x0601	A
GeForce 8800 GT	0x0602	A
GeForce GT 230	0x0603	A
GeForce 9800 GX2	0x0604	A
GeForce 9800 GT	0x0605	A
GeForce 8800 GS	0x0606	A
GeForce GTS 240	0x0607	A
GeForce 9800M GTX	0x0608	A
GeForce 8800M GTS	0x0609	A

GeForce GTX 280M	0x060A	A
GeForce 9800M GT	0x060B	A
GeForce 8800M GTX	0x060C	A
GeForce 8800 GS	0x060D	A
GeForce GTX 285M	0x060F	A
GeForce 9600 GSO	0x0610	A
GeForce 8800 GT	0x0611	A
GeForce 9800 GTX/9800 GTX+	0x0612	A
GeForce 9800 GTX+	0x0613	A
GeForce 9800 GT	0x0614	A
GeForce GTS 250	0x0615	A
GeForce 9800M GTX	0x0617	A
GeForce GTX 260M	0x0618	A
GeForce GT 230	0x0621	A
GeForce 9600 GT	0x0622	A
GeForce 9600 GS	0x0623	A
GeForce 9600 GSO 512	0x0625	A
GeForce GT 130	0x0626	A
GeForce GT 140	0x0627	A
GeForce 9800M GTS	0x0628	A
GeForce 9700M GTS	0x062A	A
GeForce 9800M GS	0x062B	A
GeForce 9800M GTS	0x062C	A
GeForce 9600 GT	0x062D	A
GeForce 9600 GT	0x062E	A
GeForce 9700 S	0x0630	A
GeForce GTS 160M	0x0631	A
GeForce GTS 150M	0x0632	A
GeForce 9600 GSO	0x0635	A
GeForce 9600 GT	0x0637	A
GeForce 9500 GT	0x0640	A
GeForce 9400 GT	0x0641	A
GeForce 9500 GT	0x0643	A
GeForce 9500 GS	0x0644	A
GeForce 9500 GS	0x0645	A
GeForce GT 120	0x0646	A
GeForce 9600M GT	0x0647	A
GeForce 9600M GS	0x0648	A
GeForce 9600M GT	0x0649	A
GeForce 9700M GT	0x064A	A
GeForce 9500M G	0x064B	A
GeForce 9650M GT	0x064C	A
GeForce G 110M	0x0651	A

GeForce GT 130M	0x0652	A
GeForce GT 120M	0x0653	A
GeForce GT 220M	0x0654	A
GeForce GT 120	0x0655 0x0633	A
GeForce 9650 S	0x0656	A
GeForce 9400 GT	0x065B	A
GeForce GTX 480	0x06C0	C
GeForce GTX 465	0x06C4	C
GeForce GTX 480M	0x06CA	C
GeForce GTX 470	0x06CD	C
Tesla C2050 / C2070	0x06D1	C
Tesla C2070	0x06D1 0x0772	C
Tesla M2070	0x06D2	C
Tesla T20 Processor	0x06DE	C
Tesla M2050	0x06DE 0x082F	C
Tesla M2050	0x06DE 0x0846	C
Tesla M2070-Q	0x06DF	C
GeForce 9300 GE	0x06E0	B 1
GeForce 9300 GS	0x06E1	B 1
GeForce 8400	0x06E2	B 1
GeForce 8400 SE	0x06E3	-
GeForce 8400 GS	0x06E4	A 1
GeForce 9300M GS	0x06E5	B 1
GeForce G100	0x06E6	B 1
GeForce 9300 SE	0x06E7	-
GeForce 9200M GS	0x06E8	B 1
GeForce 9300M GS	0x06E9	B 1
GeForce G 105M	0x06EC	B 1
GeForce G 103M	0x06EF	B 1
GeForce G105M	0x06F1	B 1
GeForce 7150 / nForce 630i	0x07E0	-
GeForce 7100 / nForce 630i	0x07E1	-
GeForce 7050 / nForce 630i	0x07E2	-
GeForce 7050 / nForce 610i	0x07E3	-
GeForce 7050 / nForce 620i	0x07E5	-
GeForce 8200M	0x0840	B 1
GeForce 9100M G	0x0844	B 1
GeForce 8200M G	0x0845	B 1
GeForce 9200	0x0846	B 1
GeForce 9100	0x0847	B 1
GeForce 8300	0x0848	B 1
GeForce 8200	0x0849	B 1
nForce 730a	0x084A	B 1

GeForce 9200	0x084B	B 1
nForce 980a/780a SLI	0x084C	B 1
nForce 750a SLI	0x084D	B 1
GeForce 8100 / nForce 720a	0x084F	-
GeForce 9400	0x0860	B 1
GeForce 9400	0x0861	B 1
GeForce 9400M G	0x0862	B 1
GeForce 9400M	0x0863	B 1
GeForce 9300	0x0864	B 1
ION	0x0865	B 1
GeForce 9400M G	0x0866	B 1
GeForce 9400	0x0867	B 1
nForce 760i SLI	0x0868	B 1
GeForce 9400	0x0869	B 1
GeForce 9400	0x086A	B 1
GeForce 9300 / nForce 730i	0x086C	B 1
GeForce 9200	0x086D	B 1
GeForce 9100M G	0x086E	B 1
GeForce 8200M G	0x086F	B 1
GeForce 9400M	0x0870	B 1
GeForce 9200	0x0871	B 1
GeForce G102M	0x0872	B 1
GeForce G102M	0x0873	B 1
ION	0x0874	B 1
ION	0x0876	B 1
GeForce 9400	0x087A	B 1
ION	0x087D	B 1
ION LE	0x087E	B 1
ION LE	0x087F	B 1
GeForce 320M	0x08A0	C
GeForce 320M	0x08A2	C
GeForce 320M	0x08A3	C
GeForce 320M	0x08A4	C
GeForce 320M	0x08A5	C
GeForce GT 220	0x0A20	C
GeForce 315	0x0A22	-
GeForce 210	0x0A23	C
GeForce 405	0x0A26	C
GeForce 405	0x0A27	C
GeForce GT 230M	0x0A28	C
GeForce GT 330M	0x0A29	C
GeForce GT 230M	0x0A2A	C
GeForce GT 330M	0x0A2B	C

GeForce GT 320M	0x0A2D	C
GeForce GT 415	0x0A32	C
GeForce GT 240M	0x0A34	C
GeForce GT 325M	0x0A35	C
GeForce G210	0x0A60	C
GeForce 205	0x0A62	C
GeForce 310	0x0A63	C
Second Generation ION	0x0A64	C
GeForce 210	0x0A65	C
GeForce 310	0x0A66	C
GeForce 315	0x0A67	-
GeForce G105M	0x0A68	B
GeForce G105M	0x0A69	B
GeForce 305M	0x0A6E	C
Second Generation ION	0x0A6F	C
GeForce 310M	0x0A70	C
GeForce 305M	0x0A71	C
GeForce 310M	0x0A72	C
GeForce 305M	0x0A73	C
GeForce G210M	0x0A74	C
GeForce 310M	0x0A75	C
Second Generation ION	0x0A76	C
GeForce 315M	0x0A7A	C
GeForce GT 330	0x0CA0	A
GeForce GT 320	0x0CA2	C
GeForce GT 240	0x0CA3	C
GeForce GT 340	0x0CA4	C
GeForce GT 220	0x0CA5	C
GeForce GT 330	0x0CA7	A
GeForce GTS 260M	0x0CA8	C
GeForce GTS 250M	0x0CA9	C
GeForce GT 220	0x0CAC	-
GeForce GT 335M	0x0CAF	C
GeForce GTS 350M	0x0CB0	C
GeForce GTS 360M	0x0CB1	C
GeForce GT 440	0x0DC0	C
GeForce GTS 450	0x0DC4	C
GeForce GTS 450	0x0DC5	C
GeForce GTS 450	0x0DC6	C
GeForce GT 555M	0x0DCD	C
GeForce GT 555M	0x0DCE	C
GeForce GTX 460M	0x0DD1	C
GeForce GT 445M	0x0DD2	C

GeForce GT 435M	0x0DD3	C
GeForce GT 550M	0x0DD6	C
GeForce GT 440	0x0DE0	C
GeForce GT 430	0x0DE1	C
GeForce GT 420	0x0DE2	C
GeForce GT 520	0x0DE4	C
GeForce GT 530	0x0DE5	C
GeForce GT 620M	0x0DE8	C
GeForce GT 630M	0x0DE9	C
GeForce 610M	0x0DEA	C
GeForce GT 555M	0x0DEB	C
GeForce GT 525M	0x0DEC	C
GeForce GT 520M	0x0DED	C
GeForce GT 415M	0x0DEE	C
GeForce GT 425M	0x0DF0	C
GeForce GT 420M	0x0DF1	C
GeForce GT 435M	0x0DF2	C
GeForce GT 420M	0x0DF3	C
GeForce GT 540M	0x0DF4	C
GeForce GT 525M	0x0DF5	C
GeForce GT 550M	0x0DF6	C
GeForce GT 520M	0x0DF7	C
GeForce GTX 460	0x0E22	C
GeForce GTX 460 SE	0x0E23	C
GeForce GTX 460	0x0E24	C
GeForce GTX 470M	0x0E30	C
GeForce GTX 485M	0x0E31	C
GeForce GT 630	0x0F00	C
GeForce GT 620	0x0F01	C
GeForce GT 640	0x0FC0	C
GeForce GT 640	0x0FC1	C
GeForce GT 630	0x0FC2	C
GeForce GT 640M LE	0x0FCE	D
GeForce GT 650M	0x0FD1	D
GeForce GT 640M	0x0FD2	D
GeForce GT 640M LE	0x0FD3	D
GeForce GTX 660M	0x0FD4	D
GeForce GT 650M	0x0FD5	D
GeForce GT 640M	0x0FD8	D
GeForce GTX 660M	0x0FE0	D
GeForce GT 520	0x1040	C
GeForce 510	0x1042	D
GeForce 605	0x1048	D

```
GeForce GT 620                  0x1049          C
GeForce GT 610                  0x104A          D
GeForce GT 520M                 0x1050          C
GeForce GT 520MX                0x1051          D
GeForce GT 520M                 0x1052          C
GeForce 410M                    0x1054          D
GeForce 410M                    0x1055          D
GeForce 610M                    0x1058          C
GeForce 610M                    0x1059          C
GeForce 610M                    0x105A          C
GeForce GTX 580                 0x1080          C
GeForce GTX 570                 0x1081          C
GeForce GTX 560 Ti              0x1082          C
GeForce GTX 560                 0x1084          C
GeForce GTX 570                 0x1086          C
GeForce GTX 560 Ti              0x1087          C
GeForce GTX 590                 0x1088          C
GeForce GTX 580                 0x1089          C
GeForce GTX 580                 0x108B          C
Tesla M2090                     0x1091          C
Tesla X2090                     0x1091 0x0974   C
Tesla M2075                     0x1094          C
Tesla C2075                     0x1096          C
GeForce 9300 GS                 0x10C0          B
GeForce 8400GS                  0x10C3          A
GeForce 405                     0x10C5          C
GeForce GT 630M                 0x1140 0x0565   C
GeForce GT 630M                 0x1140 0x0568   C
GeForce GT 620M                 0x1140 0x067A   C
GeForce GT 620M                 0x1140 0x0680   C
GeForce GT 620M                 0x1140 0x20DD   C
GeForce GTX 680                 0x1180          D
GeForce GTX 660 Ti              0x1183          D
GeForce GTX 690                 0x1188          D
GeForce GTX 670                 0x1189          D
Tesla K10                       0x118F          D
GeForce GTX 680M                0x11A0          D
GeForce GTX 560 Ti              0x1200          C
GeForce GTX 560                 0x1201          C
GeForce GTX 460 SE v2           0x1203          C
GeForce GTX 460 v2              0x1205          C
GeForce GTX 555                 0x1206          C
GeForce GT 645                  0x1207          C
```

GeForce GTX 560 SE	0x1208	C
GeForce GTX 570M	0x1210	C
GeForce GTX 580M	0x1211	C
GeForce GTX 675M	0x1212	C
GeForce GTX 670M	0x1213	C
GeForce GT 545	0x1241	C
GeForce GT 545	0x1243	C
GeForce GTX 550 Ti	0x1244	C
GeForce GTS 450	0x1245	C
GeForce GT 550M	0x1246	C
GeForce GT 555M	0x1247	C
GeForce GT 635M	0x1247 0x212A	C
GeForce GT 635M	0x1247 0x212B	C
GeForce GT 635M	0x1247 0x212C	C
GeForce GT 555M	0x1248	C
GeForce GTS 450	0x1249	C
GeForce GT 640	0x124B	C
GeForce GT 555M	0x124D	C
GeForce GT 635M	0x124D 0x10CC	C
GeForce GTX 560M	0x1251	C

A2. NVIDIA QUADRO GPUS

NVIDIA GPU product	Device PCI ID*	VDPAU features
Quadro FX 4000	0x004E	-
Quadro FX 4500	0x009D	-
Quadro FX Go1400	0x00CC	-
Quadro FX 3450/4000 SDI	0x00CD	-
Quadro FX 1400	0x00CE	-
Quadro FX 3400/Quadro FX 4000	0x00F8	-
Quadro NVS 440	0x014A	-
Quadro FX 540M	0x014C	-
Quadro FX 550	0x014D	-
Quadro FX 540	0x014E	-
Quadro NVS 285	0x0165	-
Quadro FX 5600	0x019D	-
Quadro FX 4600	0x019E	-
Quadro NVS 110M	0x01DA	-
Quadro NVS 120M	0x01DB	-

Quadro FX 350M	0x01DC	-
Quadro FX 350	0x01DE	-
Quadro NVS 210S / GeForce 6150LE	0x0245	-
Quadro NVS 510M	0x0299	-
Quadro FX 2500M	0x029A	-
Quadro FX 1500M	0x029B	-
Quadro FX 5500	0x029C	-
Quadro FX 3500	0x029D	-
Quadro FX 1500	0x029E	-
Quadro FX 4500 X2	0x029F	-
Quadro FX 560M	0x039C	-
Quadro FX 560	0x039E	-
Quadro FX 370	0x040A	A
Quadro NVS 320M	0x040B	A
Quadro FX 570M	0x040C	A
Quadro FX 1600M	0x040D	A
Quadro FX 570	0x040E	A
Quadro FX 1700	0x040F	A
Quadro NVS 140M	0x0429	A
Quadro NVS 130M	0x042A	A
Quadro NVS 135M	0x042B	A
Quadro FX 360M	0x042D	A
Quadro NVS 290	0x042F	A
Quadroplex 2200 D2	0x05ED	A
Quadroplex 2200 S4	0x05F8	A
Quadro CX	0x05F9	A
Quadro FX 5800	0x05FD	A
Quadro FX 4800	0x05FE	A
Quadro FX 3800	0x05FF	A
Quadro FX 4700 X2	0x0619	A
Quadro FX 3700	0x061A	A
Quadro VX 200	0x061B	A
Quadro FX 3600M	0x061C	A
Quadro FX 2800M	0x061D	A
Quadro FX 3700M	0x061E	A
Quadro FX 3800M	0x061F	A
Quadro FX 1800	0x0638	A
Quadro FX 2700M	0x063A	A
Quadro FX 380	0x0658	A
Quadro FX 580	0x0659	A
Quadro FX 1700M	0x065A	A
Quadro FX 770M	0x065C	A
Quadro 6000	0x06D8	C

Quadro 5000	0x06D9	C
Quadro 5000M	0x06DA	C
Quadro 6000	0x06DC	C
Quadro 4000	0x06DD	C
Quadro NVS 150M	0x06EA	B 1
Quadro NVS 160M	0x06EB	B 1
Quadro NVS 420	0x06F8	B 1
Quadro FX 370 LP	0x06F9	B 1
Quadro NVS 450	0x06FA	B 1
Quadro FX 370M	0x06FB	B 1
Quadro NVS 295	0x06FD	B 1
HICx16 + Graphics	0x06FF	B 1
NVS 5100M	0x0A2C	C
Quadro 400	0x0A38	C
Quadro FX 880M	0x0A3C	C
NVS 2100M	0x0A6A	C
NVS 3100M	0x0A6C	C
Quadro FX 380 LP	0x0A78	C
Quadro FX 380M	0x0A7C	C
Quadro FX 1800M	0x0CBC	C
Quadro 2000	0x0DD8	C
Quadro 2000D	0x0DD8 0x0914	C
Quadro 2000M	0x0DDA	C
NVS 5400M	0x0DEF	C
Quadro 600	0x0DF8	C
Quadro 500M	0x0DF9	C
Quadro 1000M	0x0DFA	C
NVS 5200M	0x0DFC	C
Quadro 3000M	0x0E3A	C
Quadro 4000M	0x0E3B	C
Quadro K2000M	0x0FFB	D
Quadro K1000M	0x0FFC	D
NVS 510	0x0FFD	D
Quadro 410	0x0FFF	D
NVS 4200M	0x1056	D
NVS 4200M	0x1057	D
NVS 310	0x107D	D
Quadro 5010M	0x109A	C
Quadro 7000	0x109B	C
NVS 300	0x10D8	C
Quadro K5000	0x11BA	D
Quadro K5000M	0x11BC	D
Quadro K4000M	0x11BD	D

Quadro K3000M	0x11BE	D

Below are the legacy GPUs that are no longer supported in the unified driver. These GPUs will continue to be maintained through the special legacy NVIDIA GPU driver releases.

The 173.14.xx driver supports the following set of GPUs:

NVIDIA GPU product	Device PCI ID
GeForce PCX 5750	0x00FA
GeForce PCX 5900	0x00FB
Quadro FX 330/GeForce PCX 5300	0x00FC
Quadro FX 330/Quadro NVS 280 PCI-E	0x00FD
Quadro FX 1300	0x00FE
GeForce FX 5800 Ultra	0x0301
GeForce FX 5800	0x0302
Quadro FX 2000	0x0308
Quadro FX 1000	0x0309
GeForce FX 5600 Ultra	0x0311
GeForce FX 5600	0x0312
GeForce FX 5600XT	0x0314
GeForce FX Go5600	0x031A
GeForce FX Go5650	0x031B
Quadro FX Go700	0x031C
GeForce FX 5200	0x0320
GeForce FX 5200 Ultra	0x0321
GeForce FX 5200	0x0322
GeForce FX 5200LE	0x0323
GeForce FX Go5200	0x0324
GeForce FX Go5250	0x0325
GeForce FX 5500	0x0326
GeForce FX 5100	0x0327
GeForce FX Go5200 32M/64M	0x0328
Quadro NVS 55/280 PCI	0x032A
Quadro FX 500/FX 600	0x032B
GeForce FX Go53xx	0x032C
GeForce FX Go5100	0x032D
GeForce FX 5900 Ultra	0x0330
GeForce FX 5900	0x0331
GeForce FX 5900XT	0x0332

GeForce FX 5950 Ultra	0x0333
GeForce FX 5900ZT	0x0334
Quadro FX 3000	0x0338
Quadro FX 700	0x033F
GeForce FX 5700 Ultra	0x0341
GeForce FX 5700	0x0342
GeForce FX 5700LE	0x0343
GeForce FX 5700VE	0x0344
GeForce FX Go5700	0x0347
GeForce FX Go5700	0x0348
Quadro FX Go1000	0x034C
Quadro FX 1100	0x034E

The 96.43.xx driver supports the following set of GPUs:

NVIDIA GPU product	Device PCI ID
GeForce2 MX/MX 400	0x0110
GeForce2 MX 100/200	0x0111
GeForce2 Go	0x0112
Quadro2 MXR/EX/Go	0x0113
GeForce4 MX 460	0x0170
GeForce4 MX 440	0x0171
GeForce4 MX 420	0x0172
GeForce4 MX 440-SE	0x0173
GeForce4 440 Go	0x0174
GeForce4 420 Go	0x0175
GeForce4 420 Go 32M	0x0176
GeForce4 460 Go	0x0177
Quadro4 550 XGL	0x0178
GeForce4 440 Go 64M	0x0179
Quadro NVS 400	0x017A
Quadro4 500 GoGL	0x017C
GeForce4 410 Go 16M	0x017D
GeForce4 MX 440 with AGP8X	0x0181
GeForce4 MX 440SE with AGP8X	0x0182
GeForce4 MX 420 with AGP8X	0x0183
GeForce4 MX 4000	0x0185
Quadro4 580 XGL	0x0188
Quadro NVS 280 SD	0x018A
Quadro4 380 XGL	0x018B

Quadro NVS 50 PCI	0x018C
GeForce2 Integrated GPU	0x01A0
GeForce4 MX Integrated GPU	0x01F0
GeForce3	0x0200
GeForce3 Ti 200	0x0201
GeForce3 Ti 500	0x0202
Quadro DCC	0x0203
GeForce4 Ti 4600	0x0250
GeForce4 Ti 4400	0x0251
GeForce4 Ti 4200	0x0253
Quadro4 900 XGL	0x0258
Quadro4 750 XGL	0x0259
Quadro4 700 XGL	0x025B
GeForce4 Ti 4800	0x0280
GeForce4 Ti 4200 with AGP8X	0x0281
GeForce4 Ti 4800 SE	0x0282
GeForce4 4200 Go	0x0286
Quadro4 980 XGL	0x0288
Quadro4 780 XGL	0x0289
Quadro4 700 GoGL	0x028C

The 71.86.xx driver supports the following set of GPUs:

NVIDIA GPU product	Device PCI ID
RIVA TNT	0x0020
RIVA TNT2/TNT2 Pro	0x0028
RIVA TNT2 Ultra	0x0029
Vanta/Vanta LT	0x002C
RIVA TNT2 Model 64/Model 64 Pro	0x002D
Aladdin TNT2	0x00A0
GeForce 256	0x0100
GeForce DDR	0x0101
Quadro	0x0103
GeForce2 GTS/GeForce2 Pro	0x0150
GeForce2 Ti	0x0151
GeForce2 Ultra	0x0152
Quadro2 Pro	0x0153

* If two IDs are listed, the first ID is the PCI Device ID and the second ID

is the PCI Subsystem Device ID.

Appendix B. X Config Options

The following driver options are supported by the NVIDIA X driver. They may be specified either in the Screen or Device sections of the X config file.

X Config Options

Option "Accel" "boolean"

Controls whether the X driver uses the GPU for graphics processing. Disabling acceleration is useful when another component, such as CUDA, requires exclusive use of the GPU's processing cores. Performance of the X server will be greatly reduced when acceleration is disabled, and some features may not be available.

Acceleration can be disabled on Fermi and later GPUs only.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: acceleration is enabled.

Option "NvAGP" "integer"

Configure AGP support. Integer argument can be one of:

Value	Behavior
0	disable AGP
1	use NVIDIA internal AGP support, if possible
2	use AGPGART, if possible
3	use any AGP support (try AGPGART, then NVIDIA AGP)

Note that 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. See Chapter 11 for details.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: 3.

Option "NoLogo" "boolean"

Disable drawing of the NVIDIA logo splash screen at X startup. Default: the logo is drawn for screens with depth 24.

Option "LogoPath" "string"

Sets the path to the PNG file to be used as the logo splash screen at X startup. If the PNG file specified has a bKGD (background color) chunk, then the screen is cleared to the color it specifies. Otherwise, the screen is cleared to black. The logo file must be owned by root and must not be writable by a non-root group. Note that a logo is only displayed for screens with depth 24. Default: The built-in NVIDIA logo is used.

Option "RenderAccel" "boolean"

Enable or disable hardware acceleration of the RENDER extension. Default: hardware acceleration of the RENDER extension is enabled.

Option "NoRenderExtension" "boolean"

Disable the RENDER extension. Other than recompiling it, the X server does not seem to have another way of disabling this. Fortunately, we can control this from the driver so we export this option. This is useful in depth 8 where RENDER would normally steal most of the default colormap. Default: RENDER is offered when possible.

Option "UBB" "boolean"

Enable or disable the Unified Back Buffer on Quadro-based GPUs (Quadro NVS excluded); see Chapter 19 for a description of UBB. This option has no effect on non-Quadro GPU products. Default: UBB is on for Quadro GPUs.

Option "NoFlip" "boolean"

Disable OpenGL flipping; see Chapter 19 for a description. Default: OpenGL will swap by flipping when possible.

Option "GLShaderDiskCache" "boolean"

This option controls whether the OpenGL driver will utilize a disk cache to save and reuse compiled shaders. See the description of the __GL_SHADER_DISK_CACHE and __GL_SHADER_DISK_CACHE_PATH environment variables in Chapter 10 for more details.

Option "Dac8Bit" "boolean"

Most Quadro products by default use a color look-up table (LUT) with a higher resolution, which can be 10, 11 or 14 bits depending on the card. This provides more accurate color on analog and DisplayPort outputs. Setting this option to TRUE forces these GPUs to use an 8-bit (LUT). Default: a high precision LUT is used, when available.

Option "Overlay" "boolean"

Enables RGB workstation overlay visuals. This is only supported on Quadro GPUs (Quadro NVS GPUs excluded) in depth 24. This option causes the server to advertise the SERVER_OVERLAY_VISUALS root window property and GLX will report single- and double-buffered, Z-buffered 16-bit overlay visuals. The transparency key is pixel 0x0000 (hex). There is no gamma correction support in the overlay plane. This feature requires XFree86 version 4.2.0 or newer, or the X.Org X server. On GPUs before G80, when the X screen is either wider than 2046 pixels or taller than 2047, the overlay may be emulated with a substantial performance penalty. RGB workstation overlays are not supported when the Composite extension is enabled.

UBB must be enabled when overlays are enabled (this is the default behavior).

Option "CIOverlay" "boolean"

Enables Color Index workstation overlay visuals with identical restrictions to Option "Overlay" above. This option causes the server to advertise the SERVER_OVERLAY_VISUALS root window property. Some of the visuals advertised that way may be listed in the main plane (layer 0) for compatibility purposes. They however belong to the overlay (layer 1). The server will offer visuals both with and without a transparency key. These are depth 8 PseudoColor visuals. Enabling Color Index overlays on X servers older than XFree86 4.3 will force the RENDER extension to be disabled due to bugs in the RENDER extension in older X servers. Color Index workstation overlays are not supported when the Composite extension is enabled. Default: off.

UBB must be enabled when overlays are enabled (this is the default behavior).

Option "TransparentIndex" "integer"

When color index overlays are enabled, use this option to choose which pixel 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's Maya require this to be zero in order to work correctly). Default: 0.

Option "OverlayDefaultVisual" "boolean"

When overlays are used, this option sets the default visual to an overlay visual thereby putting the root window in the overlay. This option is not recommended for RGB overlays. Default: off.

Option "EmulatedOverlaysTimerMs" "integer"

Enables the use of a timer within the X server to perform the updates to the emulated overlay or CI overlay. This option can be used to improve the performance of the emulated or CI overlays by reducing the frequency of the updates. The value specified indicates the desired number of milliseconds between overlay updates. To disable the use of the timer either leave the option unset or set it to 0. Default: off.

Option "EmulatedOverlaysThreshold" "boolean"

Enables the use of a threshold within the X server to perform the updates to the emulated overlay or CI overlay. The emulated or CI overlay updates can be deferred but this threshold will limit the number of deferred OpenGL updates allowed before the overlay is updated. This option can be used to trade off performance and animation quality. Default: on.

Option "EmulatedOverlaysThresholdValue" "integer"

Controls the threshold used in updating the emulated or CI overlays. This is used in conjunction with the EmulatedOverlaysThreshold option to trade off performance and animation quality. Higher values for this option favor performance over quality. Setting low values of this option will not cause the overlay to be updated more often than the frequence specified by the EmulatedOverlaysTimerMs option. Default: 5.

Option "SWCursor" "boolean"

Enable or disable software rendering of the X cursor. Default: off.

Option "HWCursor" "boolean"

Enable or disable hardware rendering of the X cursor. Default: on.

Option "CursorShadow" "boolean"

Enable or disable 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. Default: off (no cursor shadow).

Option "CursorShadowAlpha" "integer"

The alpha value to use for the cursor shadow; only applicable if CursorShadow is enabled. This value must be in the range [0, 255] -- 0 is completely transparent; 255 is completely opaque. Default: 64.

Option "CursorShadowXOffset" "integer"

The offset, in pixels, that the shadow image will be shifted to the right from the real cursor image; only applicable if CursorShadow is enabled. This value must be in the range [0, 32]. Default: 4.

Option "CursorShadowYOffset" "integer"

The offset, in pixels, that the shadow image will be shifted down from the real cursor image; only applicable if CursorShadow is enabled. This value must be in the range [0, 32]. Default: 2.

Option "ConnectedMonitor" "string"

Allows you to override what the NVIDIA kernel module detects is connected to your graphics card. This may be useful, for example, 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 which display devices are connected, and the NVIDIA X driver assumes you have a single CRT.

Valid values for this option are "CRT" (cathode ray tube), "DFP" (digital flat panel), or "TV" (television); if using TwinView, this option may be a comma-separated list of display devices; e.g.: "CRT, CRT" or "CRT, DFP".

It is generally recommended to not use this option, but instead use the "UseDisplayDevice" option.

NOTE: anything attached to a 15 pin VGA connector is regarded by the driver as a CRT. "DFP" should only be used to refer to digital flat panels connected via a DVI port.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: string is NULL (the NVIDIA driver will detect the connected display devices).

Option "UseDisplayDevice" "string"

The "UseDisplayDevice" X configuration option is a list of one or more display devices, which limits the display devices the NVIDIA X driver will consider for an X screen. The display device names used in the option may be either specific (with a numeric suffix; e.g., "DFP-1") or general (without a numeric suffix; e.g., "DFP").

When assigning display devices to X screens, the NVIDIA X driver walks through the list of all (not already assigned) display devices detected as connected. When the "UseDisplayDevice" X configuration option is specified, the X driver will only consider connected display devices which are also included in the "UseDisplayDevice" list. This can be thought of as a "mask" against the connected (and not already assigned) display devices.

Note the subtle difference between this option and the "ConnectedMonitor" option: the "ConnectedMonitor" option overrides which display devices are actually detected, while the "UseDisplayDevice" option controls which of the detected display devices will be used on this X screen.

Of the list of display devices considered for this X screen (either all connected display devices, or a subset limited by the "UseDisplayDevice" option), the NVIDIA X driver first looks at CRTs, then at DFPs, and finally at TVs. For example, if both a CRT and a DFP are connected, by default the X driver would assign the CRT to this X screen. However, by specifying:

```
    Option "UseDisplayDevice" "DFP"
```

the X screen would use the DFP instead. Or, if CRT-0, DFP-0, and DFP-1 are connected and TwinView is enabled, the X driver would assign CRT-0 and DFP-0 to the X screen. However, by specifying:

```
    Option "UseDisplayDevice" "CRT-0, DFP-1"
```

the X screen would use CRT-0 and DFP-1 instead.

Additionally, the special value "none" can be specified for the "UseDisplayDevice" option. When this value is given, any programming of the display hardware is disabled. The NVIDIA driver will not perform any mode validation or mode setting for this X screen. This is intended for use in conjunction with CUDA or in remote graphics solutions such as VNC or Hewlett Packard's Remote Graphics Software (RGS).

"UseDisplayDevice" defaults to "none" on GPUs that have no display capabilities, such as some Tesla GPUs and some mobile GPUs used in Optimus notebook configurations.

Note the following restrictions for setting the "UseDisplayDevice" to "none":

o OpenGL SyncToVBlank will have no effect.

o None of Stereo, Overlay, CIOverlay, or SLI are allowed when "UseDisplayDevice" is set to "none".

Option "UseEdidFreqs" "boolean"

This option controls whether the NVIDIA X driver will use the HorizSync and VertRefresh ranges given in a display device's EDID, if any. When UseEdidFreqs is set to True, EDID-provided range information will override the HorizSync and VertRefresh ranges specified in the Monitor section. 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 your X config file. These frequency ranges are used when validating modes for your display device.

Default: True (EDID frequencies will be used)

Option "UseEDID" "boolean"

By default, the NVIDIA X driver makes use of a display device's EDID, when available, during construction of its mode pool. The EDID is used as a source for possible modes, for valid frequency ranges, and for collecting data on the physical dimensions of the display device for computing the DPI (see Appendix E). However, if you wish to disable the driver's use of the EDID, you can set this option to False:

```
    Option "UseEDID" "FALSE"
```

Note that, rather than globally disable all uses of the EDID, you can individually disable each particular use of the EDID; e.g.,

```
    Option "UseEDIDFreqs" "FALSE"
    Option "UseEDIDDpi" "FALSE"
    Option "ModeValidation" "NoEdidModes"
```

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: True (use EDID).

Option "UseInt10Module" "boolean"

Enable use of the X Int10 module to soft-boot all secondary cards, rather than POSTing the cards through the NVIDIA kernel module.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: off (POSTing is done through the NVIDIA kernel module).

Option "TwinView" "boolean"

Enable or disable TwinView. See Chapter 12 for details. Default: off (TwinView is disabled).

Option "MetaModeOrientation" "string"

Controls the default relationship between display devices when using multiple display devices on a single X screen. Takes one of the following values: "RightOf" "LeftOf" "Above" "Below" "Clone". For backwards compatibility, "TwinViewOrientation" is a synonym for "MetaModeOrientation". See Chapter 12 for details. Default: string is NULL.

Option "MetaModes" "string"

This option describes the combination of modes to use on each monitor when using TwinView or SLI Mosaic Mode. See Chapter 12 and Chapter 24 for details. Default: string is NULL.

Option "nvidiaXineramaInfo" "boolean"

The NVIDIA X driver normally provides a Xinerama extension that X clients (such as window managers) can use to discover the current layout of display devices within an X screen. Some window mangers get confused by this information, so this option is provided to disable this behavior. Default: true (NVIDIA Xinerama information is provided).

On X servers with RandR 1.2 support, the X server's RandR implementation may provide its own Xinerama implementation if NVIDIA Xinerama information is not provided. So, on X servers with RandR 1.2, disabling "nvidiaXineramaInfo" causes the NVIDIA X driver to still register its Xinerama implementation but report a single screen-sized region. On X servers without RandR 1.2 support, disabling "nvidiaXineramaInfo" causes the NVIDIA X driver to not register its Xinerama implementation.

Due to bugs in some older software, NVIDIA Xinerama information is not provided by default on X.Org 7.1 and older when the X server is started with only one display device enabled.

For backwards compatibility, "NoTwinViewXineramaInfo" is a synonym for disabling "nvidiaXineramaInfo".

Option "nvidiaXineramaInfoOrder" "string"

When the NVIDIA X driver provides nvidiaXineramaInfo (see the nvidiaXineramaInfo X config option), it by default reports the currently enabled display devices in the order "CRT, DFP, TV". The nvidiaXineramaInfoOrder X config option can be used to override this order.

The option string is a comma-separated list of display device names. The display device names can either be general (e.g, "CRT", which identifies all CRTs), or specific (e.g., "CRT-1", which identifies a particular CRT). Not all display devices need to be identified in the option string; display devices that are not listed will be implicitly appended to the end of the list, in their default order.

Note that nvidiaXineramaInfoOrder tracks all display devices that could possibly be connected to the GPU, not just the ones that are currently enabled. When reporting the Xinerama information, the NVIDIA X driver walks through the display devices in the order specified, only reporting enabled display devices.

Examples:

```
"DFP"
"TV, DFP"
"DFP-1, DFP-0, TV, CRT"
```

In the first example, any enabled DFPs would be reported first (any enabled CRTs or TVs would be reported afterwards). In the second example, any enabled TVs would be reported first, then any enabled DFPs (any enabled CRTs would be reported last). In the last example, if DFP-1 were enabled, it would be reported first, then DFP-0, then any enabled TVs, and then any enabled CRTs; finally, any other enabled DFPs would be reported.

For backwards compatibility, "TwinViewXineramaInfoOrder" is a synonym for "nvidiaXineramaInfoOrder".

Default: "CRT, DFP, TV"

Option "nvidiaXineramaInfoOverride" "string"

This option overrides the values reported by the NVIDIA X driver's nvidiaXineramaInfo implementation. This disregards the actual display devices used by the X screen and any order specified in nvidiaXineramaInfoOrder.

The option string is interpreted as a comma-separated list of regions, specified as '[width]x[height]+[x-offset]+[y-offset]'. The regions' sizes and offsets are not validated against the X screen size, but are directly reported to any Xinerama client.

Examples:

```
"1600x1200+0+0, 1600x1200+1600+0"
"1024x768+0+0, 1024x768+1024+0, 1024x768+0+768, 1024x768+1024+768"
```

For backwards compatibility, "TwinViewXineramaInfoOverride" is a synonym for "nvidiaXineramaInfoOverride".

Option "TVStandard" "string"

See Chapter 15 for details on configuring TV-out.

Option "TVOutFormat" "string"

See Chapter 15 for details on configuring TV-out.

Option "TVOverScan" "Decimal value in the range 0.0 to 1.0"

Valid values are in the range 0.0 through 1.0; See Chapter 15 for details on configuring TV-out.

Option "Stereo" "integer"

Enable offering of quad-buffered stereo visuals on Quadro. Integer indicates the type of stereo equipment being used:

Value	Equipment
1	DDC glasses. The sync signal is sent to the glasses via the DDC signal to the monitor. These usually involve a passthrough cable between the monitor and the graphics card. This mode is not available on G8xGL and higher GPUs.
2	"Blueline" glasses. These usually involve a passthrough cable between the monitor and graphics card. The glasses know which 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). This mode is not available on G8xGL and higher GPUs.
3	Onboard stereo support. This is usually only found on professional cards. The glasses connect via a DIN connector on the back of the graphics card.
4	One-eye-per-display passive stereo. This mode allows each display to be configured to statically display either left or right eye content. This can be especially useful with multi-display configurations (TwinView or SLI Mosaic). For example, this is commonly used in conjunction with special projectors to produce 2 polarized images which are then viewed with polarized glasses. To use this stereo mode, it is recommended that you configure TwinView (or pairs of displays in SLI Mosaic) in clone mode with the same resolution, panning offset, and panning domains on each display. See Chapter 12 for more information about configuring multiple displays.
5	Vertical interlaced stereo mode, for use with SeeReal Stereo Digital Flat Panels.
6	Color interleaved stereo mode, for use with Sharp3D Stereo Digital Flat Panels.
7	Horizontal interlaced stereo mode, for use with Arisawa, Hyundai, Zalman, Pavione, and Miracube Digital Flat Panels.
8	Checkerboard pattern stereo mode, for use with 3D DLP Display Devices.
9	Inverse checkerboard pattern stereo mode, for use with 3D DLP Display Devices.
10	NVIDIA 3D Vision mode for use with NVIDIA 3D Vision glasses. The NVIDIA 3D Vision infrared emitter must be connected to a USB port of your computer, and to the 3-pin DIN connector of a Quadro graphics board (based on G8xGL or higher GPU) before starting the X server. Hot-plugging the USB infrared stereo emitter is not yet supported. Also, 3D Vision Stereo Linux support requires a Linux kernel built with USB device filesystem (usbfs) and USB 2.0 support. Not presently supported on FreeBSD or Solaris.
11	NVIDIA 3D VisionPro mode for use with NVIDIA 3D VisionPro glasses. The NVIDIA 3D VisionPro RF hub

must be connected to a USB port of your computer, and to the 3-pin DIN connector of a Quadro graphics board (based on G8xGL or higher GPU) before starting the X server. Hot-plugging the USB RF hub is not yet supported. Also, 3D VisionPro Stereo Linux support requires a Linux kernel built with USB device filesystem (usbfs) and USB 2.0 support. When RF hub is connected and X is started in NVIDIA 3D VisionPro stereo mode, a new page will be available in nvidia-settings for various configuration settings. Some of these settings can also be done via nvidia-settings command line interface. Refer to the corresponding Help section in nvidia-settings for further details. Not presently supported on FreeBSD or Solaris.

Default: 0 (Stereo is not enabled).

Stereo options 1, 2, 3, 10 and 11 are known as "active" stereo. Other options are known as "passive" stereo.

When active stereo is used with multiple display devices, it is recommended that modes within each MetaMode have identical timing values (modelines). See Chapter 18 for suggestions on making sure the modes within your MetaModes are identical.

The following table summarizes the available stereo modes, their supported GPUs, and their intended display devices:

Stereo mode (value)	Graphics card supported [1]	Display supported
DDC glasses (1)	Quadro graphics cards with pre-G8xGL GPUs	CRTs with high refresh rate
Blueline glasses (2)	Quadro graphics cards with pre-G8xGL GPUs	CRTs with high refresh rate
Onboard DIN (3)	Quadro graphics cards	Displays with high refresh rate
One-eye-per-display (4)	Quadro graphics cards	Any
Vertical Interlaced	Quadro graphics	SeeReal Stereo DFP

(5)	cards	
Color Interleaved (6)	Quadro graphics cards	Sharp3D stereo DFP
Horizontal Interlaced (7)	Quadro graphics cards with G8xGL or higher GPU	Arisawa, Hyundai, Zalman, Pavione, and Miracube
Checkerboard Pattern (8)	Quadro graphics cards with G8xGL or higher GPU	3D DLP display devices
Inverse Checkerboard (9)	Quadro graphics cards with G8xGL or higher GPU	3D DLP display devices
NVIDIA 3D Vision (10)	Quadro graphics cards with G8xGL or higher GPUs [2]	Supported 3D Vision ready displays [3]
NVIDIA 3D VisionPro (11)	Quadro graphics cards with G8xGL or higher GPUs [2]	Supported 3D Vision ready displays [3]

[1] Quadro graphics cards excluding Quadro NVS cards.
[2]
http://www.nvidia.com/object/quadro_pro_graphics_boards_linux.html
[3] http://www.nvidia.com/object/3D_Vision_Requirements.html

UBB must be enabled when stereo is enabled (this is the default behavior).

Active stereo can be enabled on digital display devices (connected via DVI, HDMI, or DisplayPort). However, some digital display devices might not behave as desired with active stereo:

o Some digital display devices may not be able to toggle pixel colors quickly enough when flipping between eyes on every vblank.

o Some digital display devices may have an optical polarization that interferes with stereo goggles.

o Active stereo requires high refresh rates, because a vertical refresh is needed to display each eye. Some digital display devices have a low refresh rate, which will result in flickering when used for active stereo.

o Some digital display devices might internally convert from other refresh rates to their native refresh rate (e.g., 60Hz), resulting in incompatible rates between the stereo glasses and stereo displayed on screen.

These limitations do not apply to any display devices suitable for stereo option 10.

Stereo applies to an entire X screen, so it will apply to all display devices on that X screen, whether or not they all support the selected Stereo mode.

Multi-GPU cards (such as the Quadro FX 4500 X2) provide a single DIN connector for onboard stereo support (option 3), NVIDIA 3D Vision stereo (option 10) and NVIDIA 3D VisionPro stereo (option 11), which is tied to the bottommost GPU. In order to synchronize stereo with the other GPU, you must use a G-Sync device (see Chapter 25 for details).

Option "ForceStereoFlipping" "boolean"

Stereo flipping is the process by which left and right eyes are displayed on alternating vertical refreshes. Normally, stereo flipping is only performed when a stereo drawable is visible. This option forces stereo flipping even when no stereo drawables are visible.

This is to be used in conjunction with the "Stereo" option. If "Stereo" is 0, the "ForceStereoFlipping" option has no effect. If otherwise, the "ForceStereoFlipping" option will force the behavior indicated by the "Stereo" option, even if no stereo drawables are visible. This option is useful in a multiple-screen environment in which a stereo application is run on a different screen than the stereo master.

Possible values:

Value	Behavior
0	Stereo flipping is not forced. The default behavior as indicated by the "Stereo" option is used.
1	Stereo flipping is forced. Stereo is running even if no stereo drawables are visible. The stereo mode depends on the value of the "Stereo" option.

Default: 0 (Stereo flipping is not forced).

Option "XineramaStereoFlipping" "boolean"

By default, when using Stereo with Xinerama, all physical X screens having a visible stereo drawable will stereo flip. Use this option to allow only one physical X screen to stereo flip at a time.

This is to be used in conjunction with the "Stereo" and "Xinerama" options. If "Stereo" is 0 or "Xinerama" is 0, the "XineramaStereoFlipping" option has no effect.

If you wish to have all X screens stereo flip all the time, see the "ForceStereoFlipping" option.

Possible values:

Value	Behavior
0	Stereo flipping is enabled on one X screen at a time. Stereo is enabled on the first X screen having the stereo drawable.
1	Stereo flipping in enabled on all X screens.

Default: 1 (Stereo flipping is enabled on all X screens).

Option "IgnoreDisplayDevices" "string"

This option tells the NVIDIA kernel module to completely ignore the indicated classes of display devices when checking which display devices are connected. You may specify a comma-separated list containing any of "CRT", "DFP", and "TV". For example:

```
Option "IgnoreDisplayDevices" "DFP, TV"
```

will cause the NVIDIA driver to not attempt to detect if any digital flat panels or TVs are connected. This option is not normally necessary; however, some video BIOSes contain incorrect information about which display devices may be connected, or which i2c port 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 this by telling the NVIDIA driver to ignore display devices which you know are not connected. NOTE: anything attached to a 15 pin VGA connector is regarded by the driver as a CRT. "DFP" should only be used to refer to digital flat panels connected via a DVI port.

When this option is set for an X screen, it will be applied to all X
screens running on the same GPU.

Option "MultisampleCompatibility" "boolean"

Enable or disable the use of separate front and back multisample buffers.
Enabling this will consume more memory but is necessary for correct output
when rendering to both the front and back buffers of a multisample or FSAA
drawable. This option is necessary for correct operation of SoftImage XSI.
Default: false (a single multisample buffer is shared between the front
and back buffers).

Option "NoPowerConnectorCheck" "boolean"

The NVIDIA X driver will fail initialization on a GPU if it detects that
the GPU that requires an external power connector does not have an
external power connector plugged in. This option can be used to bypass
this test.

When this option is set for an X screen, it will be applied to all X
screens running on the same GPU.

Default: false (the power connector test is performed).

Option "ThermalConfigurationCheck" "boolean"

The NVIDIA X driver will fail initialization on a GPU if it detects that
the GPU has a bad thermal configuration. This may indicate a problem with
how your graphics board was built, or simply a driver bug. It is
recommended that you contact your graphics board vendor if you encounter
this problem.

When this option is set for an X screen, it will be applied to all X
screens running on the same GPU.

This option can be set to False to bypass this test. Default: true (the
thermal configuration test is performed).

Option "XvmcUsesTextures" "boolean"

Forces XvMC to use the 3D engine for XvMCPutSurface requests rather than the video overlay. Default: false (video overlay is used when available).

Option "AllowGLXWithComposite" "boolean"

Enables GLX even when the Composite X extension is loaded. ENABLE AT YOUR OWN RISK. OpenGL applications will not display correctly in many circumstances with this setting enabled.

This option is intended for use on versions of X.Org older than X11R6.9.0. On X11R6.9.0 or newer, the NVIDIA OpenGL implementation interacts properly by default with the Composite X extension and this option should not be needed. However, on X11R6.9.0 or newer, support for GLX with Composite can be disabled by setting this option to False.

Default: false (GLX is disabled when Composite is enabled on X releases older than X11R6.9.0).

Option "AddARGBGLXVisuals" "boolean"

Adds a 32-bit ARGB visual for each supported OpenGL configuration. This allows applications to use OpenGL to render with alpha transparency into 32-bit windows and pixmaps. This option requires the Composite extension. Default: ARGB GLX visuals are enabled on X servers new enough to support them when the Composite extension is also enabled and the screen depth is 24 or 30.

Option "DisableGLXRootClipping" "boolean"

If enabled, no clipping will be performed on rendering done by OpenGL in the root window. This option is deprecated. It is needed by older versions of OpenGL-based composite managers that draw the contents of redirected windows directly into the root window using OpenGL. Most OpenGL-based composite managers have been updated to support the Composite Overlay Window, a feature introduced in Xorg release 7.1. Using the Composite Overlay Window is the preferred method for performing OpenGL-based compositing.

Option "DamageEvents" "boolean"

Use OS-level events to efficiently notify X when a client has performed direct rendering to a window that needs to be composited. This will significantly improve performance and interactivity when using GLX applications with a composite manager running. It will also affect applications using GLX when rotation is enabled. This option is currently incompatible with SLI and Multi-GPU modes and will be disabled if either are used. Enabled by default.

Option "ExactModeTimingsDVI" "boolean"

Forces the initialization of the X server with the exact timings specified in the ModeLine. Default: false (for DVI devices, the X server initializes with the closest mode in the EDID list).

The "AllowNonEdidModes" token in the "ModeValidation" X configuration option has the same effect as "ExactModeTimingsDVI", but "AllowNonEdidModes" has per-display device granularity.

Option "Coolbits" "integer"

Enables various unsupported features, such as support for GPU clock manipulation in the NV-CONTROL X extension. This option accepts a bit mask of features to enable.

WARNING: this may cause system damage and void warranties. This utility can run your computer system out of the manufacturer's design specifications, including, but not limited to: higher system voltages, above normal temperatures, excessive frequencies, and changes to BIOS that may corrupt the BIOS. Your computer's operating system may hang and result in data loss or corrupted images. Depending on the manufacturer of your computer system, the computer system, hardware and software warranties may be voided, and you may not receive any further manufacturer support. NVIDIA does not provide customer service support for the Coolbits option. It is for these reasons that absolutely no warranty or guarantee is either express or implied. Before enabling and using, you should determine the suitability of the utility for your intended use, and you shall assume all responsibility in connection therewith.

When "1" (Bit 0) is set in the "Coolbits" option value, the nvidia-settings utility will contain a page labeled "Clock Frequencies" through which clock settings can be manipulated. On mobile GPUs, limited clock manipulation support is available when "1" is set in the "Coolbits" option value: clocks can be lowered relative to the default settings, but overclocking is not supported due to the thermal constraints of notebook designs.

When "2" (Bit 1) is set in the "Coolbits" option value, the NVIDIA driver will attempt to initialize SLI when using GPUs with different amounts of video memory.

When "4" (Bit 2) is set in the "Coolbits" option value, the nvidia-settings Thermal Monitor page will allow configuration of GPU fan speed, on graphics boards with programmable fan capability.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

The default for this option is 0 (unsupported features are disabled).

Option "MultiGPU" "string"

This option controls the configuration of Multi-GPU rendering in supported configurations.

Value	Behavior
0, no, off, false, Single	Use only a single GPU when rendering
1, yes, on, true, Auto	Enable Multi-GPU and allow the driver to automatically select the appropriate rendering mode.
AFR	Enable Multi-GPU and use the Alternate Frame Rendering mode.
SFR	Enable Multi-GPU and use the Split Frame Rendering mode.
AA	Enable Multi-GPU and use antialiasing. Use this in conjunction with full scene antialiasing to improve visual quality.

Option "SLI" "string"

This option controls the configuration of SLI rendering in supported configurations.

Value	Behavior
0, no, off, false, Single	Use only a single GPU when

	rendering
1, yes, on, true, Auto	Enable SLI and allow the driver to automatically select the appropriate rendering mode.
AFR	Enable SLI and use the Alternate Frame Rendering mode.
SFR	Enable SLI and use the Split Frame Rendering mode.
AA	Enable SLI and use SLI Antialiasing. Use this in conjunction with full scene antialiasing to improve visual quality.
AFRofAA	Enable SLI and use SLI Alternate Frame Rendering of Antialiasing mode. Use this in conjunction with full scene antialiasing to improve visual quality. This option is only valid for SLI configurations with 4 GPUs.
Mosaic	Enable SLI and use SLI Mosaic Mode. Use this in conjunction with the MetaModes X configuration option to specify the combination of mode(s) used on each display.

Option "TripleBuffer" "boolean"

Enable or disable the use of triple buffering. If this option is enabled, OpenGL windows that sync to vblank and are double-buffered will be given a third buffer. This decreases the time an application stalls while waiting for vblank events, but increases latency slightly (delay between user input and displayed result).

Option "DPI" "string"

This option specifies the Dots Per Inch for the X screen; for example:

```
    Option "DPI" "75 x 85"
```

will set the horizontal DPI to 75 and the vertical DPI to 85. By default, the X driver will compute the DPI of the X screen from the EDID of any connected display devices. See Appendix E for details. Default: string is NULL (disabled).

Option "UseEdidDpi" "string"

By default, the NVIDIA X driver computes the DPI of an X screen based on the physical size of the display device, as reported in the EDID, and the size in pixels of the first mode to be used on the display device. If multiple display devices are used by the X screen, then the NVIDIA X screen will choose which display device to use. This option can be used to specify which display device to use. The string argument can be a display device name, such as:

```
    Option "UseEdidDpi" "DFP-0"
```

or the argument can be "FALSE" to disable use of EDID-based DPI calculations:

```
    Option "UseEdidDpi" "FALSE"
```

See Appendix E for details. Default: string is NULL (the driver computes the DPI from the EDID of a display device and selects the display device).

Option "ConstantDPI" "boolean"

By default on X.Org 6.9 or newer, the NVIDIA X driver recomputes the size in millimeters of the X screen whenever the size in pixels of the X screen is changed using XRandR, such that the DPI remains constant.

This behavior can be disabled (which means that the size in millimeters will not change when the size in pixels of the X screen changes) by setting the "ConstantDPI" option to "FALSE"; e.g.,

```
    Option "ConstantDPI" "FALSE"
```

ConstantDPI defaults to True.

On X releases older than X.Org 6.9, the NVIDIA X driver cannot change the size in millimeters of the X screen. Therefore the DPI of the X screen will change when XRandR changes the size in pixels of the X screen. The driver will behave as if ConstantDPI was forced to FALSE.

Option "CustomEDID" "string"

This option forces the X driver to use the EDID specified in a file rather than the display's EDID. You may specify a semicolon separated list of display names and filename pairs. Valid display device names include "CRT-0", "CRT-1", "DFP-0", "DFP-1", "TV-0", "TV-1", or one of the generic names "CRT", "DFP", "TV", which apply the EDID to all devices of the specified type. Additionally, if SLI Mosaic is enabled, this name can be prefixed by a GPU name (e.g., "GPU-0.CRT-0"). The file contains a raw EDID (e.g., a file generated by nvidia-settings).

For example:

```
    Option "CustomEDID" "CRT-0:/tmp/edid1.bin; DFP-0:/tmp/edid2.bin"
```

will assign the EDID from the file /tmp/edid1.bin to the display device CRT-0, and the EDID from the file /tmp/edid2.bin to the display device DFP-0. Note that a display device name must always be specified even if only one EDID is specified.

Caution: Specifying an EDID that doesn't exactly match your display may damage your hardware, as it allows the driver to specify timings beyond the capabilities of your display. Use with care.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Option "IgnoreEDIDChecksum" "string"

This option forces the X driver to accept an EDID even if the checksum is invalid. You may specify a comma separated list of display names. Valid display device names include "CRT-0", "CRT-1", "DFP-0", "DFP-1", "TV-0", "TV-1", or one of the generic names "CRT", "DFP", "TV", which ignore the EDID checksum on all devices of the specified type. Additionally, if SLI Mosaic is enabled, this name can be prefixed by a GPU name (e.g., "GPU-0.CRT-0").

For example:

```
    Option "IgnoreEDIDChecksum" "CRT, DFP-0"
```

will cause the nvidia driver to ignore the EDID checksum for all CRT monitors and the displays DFP-0 and TV-0.

Caution: An invalid EDID checksum may indicate a corrupt EDID. A corrupt EDID may have mode timings beyond the capabilities of your display, and using it could damage your hardware. Use with care.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Option "ModeValidation" "string"

This option provides fine-grained control over each stage of the mode validation pipeline, disabling individual mode validation checks. This option should only very rarely be used.

The option string is a semicolon-separated list of comma-separated lists of mode validation arguments. Each list of mode validation arguments can optionally be prepended with a display device name and GPU specifier.

```
"<dpy-0>: <tok>, <tok>; <dpy-1>: <tok>, <tok>, <tok>; ..."
```

Possible arguments:

o "AllowNon60HzDFPModes": some lower quality TMDS encoders are only rated to drive DFPs at 60Hz; the driver will determine when only 60Hz DFP modes are allowed. This argument disables this stage of the mode validation pipeline.

o "NoMaxPClkCheck": each mode has a pixel clock; this pixel clock is validated against the maximum pixel clock of the hardware (for a DFP, this is the maximum pixel clock of the TMDS encoder, for a CRT, this is the maximum pixel clock of the DAC). This argument disables the maximum pixel clock checking stage of the mode validation pipeline.

o "NoEdidMaxPClkCheck": a display device's EDID can specify the maximum pixel clock that the display device supports; a mode's pixel clock is validated against this pixel clock maximum. This argument disables this stage of the mode validation pipeline.

o "AllowInterlacedModes": interlaced modes are not supported on all NVIDIA GPUs; the driver will discard interlaced modes on GPUs where interlaced modes are not supported; this argument disables this stage of the mode validation pipeline.

o "NoMaxSizeCheck": each NVIDIA GPU has a maximum resolution that it
  can drive; this argument disables this stage of the mode validation
  pipeline.

o "NoHorizSyncCheck": a mode's horizontal sync is validated against the
  range of valid horizontal sync values; this argument disables this
  stage of the mode validation pipeline.

o "NoVertRefreshCheck": a mode's vertical refresh rate is validated
  against the range of valid vertical refresh rate values; this
  argument disables this stage of the mode validation pipeline.

o "NoWidthAlignmentCheck": the alignment of a mode's visible width is
  validated against the capabilities of the GPU; normally, a mode's
  visible width must be a multiple of 8. This argument disables this
  stage of the mode validation pipeline.

o "NoDFPNativeResolutionCheck": when validating for a DFP, a mode's
  size is validated against the native resolution of the DFP; this
  argument disables this stage of the mode validation pipeline.

o "NoVirtualSizeCheck": if the X configuration file requests a specific
  virtual screen size, a mode cannot be larger than that virtual size;
  this argument disables this stage of the mode validation pipeline.

o "NoVesaModes": when constructing the mode pool for a display device,
  the X driver uses a built-in list of VESA modes as one of the mode
  sources; this argument disables use of these built-in VESA modes.

o "NoEdidModes": when constructing the mode pool for a display device,
  the X driver uses any modes listed in the display device's EDID as
  one of the mode sources; this argument disables use of EDID-specified
  modes.

o "NoXServerModes": when constructing the mode pool for a display
  device, the X driver uses the built-in modes provided by the core
  XFree86/Xorg X server as one of the mode sources; this argument
  disables use of these modes. Note that this argument does not disable
  custom ModeLines specified in the X config file; see the
  "NoCustomModes" argument for that.

o "NoCustomModes": when constructing the mode pool for a display device, the X driver uses custom ModeLines specified in the X config file (through the "Mode" or "ModeLine" entries in the Monitor Section) as one of the mode sources; this argument disables use of these modes.

o "NoPredefinedModes": when constructing the mode pool for a display device, the X driver uses additional modes predefined by the NVIDIA X driver; this argument disables use of these modes.

o "NoUserModes": additional modes can be added to the mode pool dynamically, using the NV-CONTROL X extension; this argument prohibits user-specified modes via the NV-CONTROL X extension.

o "NoExtendedGpuCapabilitiesCheck": allow mode timings that may exceed the GPU's extended capability checks.

o "ObeyEdidContradictions": an EDID may contradict itself by listing a mode as supported, but the mode may exceed an EDID-specified valid frequency range (HorizSync, VertRefresh, or maximum pixel clock). Normally, the NVIDIA X driver prints a warning in this scenario, but does not invalidate an EDID-specified mode just because it exceeds an EDID-specified valid frequency range. However, the "ObeyEdidContradictions" argument instructs the NVIDIA X driver to invalidate these modes.

o "NoTotalSizeCheck": allow modes in which the individual visible or sync pulse timings exceed the total raster size.

o "DoubleScanPriority": on GPUs older than G80, doublescan modes are sorted before non-doublescan modes of the same resolution for purposes of mode pool sorting; but on G80 and later GPUs, doublescan modes are sorted after non-doublescan modes of the same resolution. This token inverts that priority (i.e., doublescan modes will be sorted after on pre-G80 GPUs, and sorted before on G80 and later GPUs).

o "NoDualLinkDVICheck": for mode timings used on dual link DVI DFPs, the driver must perform additional checks to ensure that the correct pixels are sent on the correct link. For some of these checks, the driver will invalidate the mode timings; for other checks, the driver will implicitly modify the mode timings to meet the GPU's dual link DVI requirements. This token disables this dual link DVI checking.

o "NoDisplayPortBandwidthCheck": for mode timings used on DisplayPort devices, the driver must verify that the DisplayPort link can be configured to carry enough bandwidth to support a given mode's pixel clock. For example, some DisplayPort-to-VGA adapters only support 2 DisplayPort lanes, limiting the resolutions they can display. This token disables this DisplayPort bandwidth check.

o "AllowNon3DVisionModes": modes that are not optimized for NVIDIA 3D Vision are invalidated, by default, when 3D Vision (stereo mode 10) or 3D Vision Pro (stereo mode 11) is enabled. This token allows the use of non-3D Vision modes on a 3D Vision monitor. (Stereo behavior of non-3D Vision modes on 3D Vision monitors is undefined.)

o "AllowNonEdidModes": if a mode is not listed in a display device's EDID mode list, then the NVIDIA X driver will discard the mode if the EDID 1.3 "GTF Supported" flag is unset, if the EDID 1.4 "Continuous Frequency" flag is unset, or if the display device is connected to the GPU by a digital protocol (e.g., DVI, DP, etc). This token disables these checks for non-EDID modes.

Examples:

```
    Option "ModeValidation" "NoMaxPClkCheck"
```

disable the maximum pixel clock check when validating modes on all display devices.

```
    Option "ModeValidation" "CRT-0: NoEdidModes, NoMaxPClkCheck;
GPU-0.DFP-0: NoVesaModes"
```

do not use EDID modes and do not perform the maximum pixel clock check on CRT-0, and do not use VESA modes on DFP-0 of GPU-0.

Option "ColorSpace" "string"

This option sets the color space for all or a subset of the connected flat panels.

The option string is a semicolon-separated list of device specific options. Each option can optionally be prepended with a display device name and a GPU specifier.

```
"<dpy-0>: <tok>; <dpy-1>: <tok>; ..."
```

Possible arguments:

o "RGB": sets color space to RGB. RGB color space supports two valid color ranges; full and limited. By default, full color range is set when the color space is RGB.

o "YCbCr444": sets color space to YCbCr 4:4:4. YCbCr supports only limited color range. It is not possible to set this color space if the GPU or display is not capable of limited range.

If the ColorSpace option is not specified, or is incorrectly specified, then the color space is set to RGB by default.

Examples:

```
Option "ColorSpace" "YCbCr444"
```

set the color space to YCbCr 4:4:4 on all flat panels.

```
Option "ColorSpace" "GPU-0.DFP-0: YCbCr444"
```

set the color space to YCbCr 4:4:4 on DFP-0 of GPU-0.

Option "ColorRange" "string"

This option sets the color range for all or a subset of the connected flat panels.

The option string is a semicolon-separated list of device specific options. Each option can optionally be prepended with a display device name and a GPU specifier.

```
"<dpy-0>: <tok>; <dpy-1>: <tok>; ..."
```

The set of legal ColorRange values depends upon the selected color space.

Possible arguments:

- o "Full": sets color range to full range. By default, full color range is set when the color space is RGB.

- o "Limited": sets color range to limited range. YUV supports only limited color range. Consequently, limited range is selected by the driver when color space is set to YUV, and can not be changed.

If the ColorRange option is not specified, or is incorrectly specified, then an appropriate default value is selected based on the selected color space.

Examples:

```
Option "ColorRange" "Limited"
```

set the color range to limited on all flat panels.

```
Option "ColorRange" "GPU-0.DFP-0: Limited"
```

set the color range to limited on DFP-0 of GPU-0.

Option "ModeDebug" "boolean"

This option causes the X driver to print verbose details about mode validation to the X log file. Note that this option is applied globally: setting this option to TRUE will enable verbose mode validation logging for all NVIDIA X screens in the X server.

Option "UseEvents" "boolean"

Enables the use of system events in some cases when the X driver is waiting for the hardware. The X driver can briefly spin through a tight loop when waiting for the hardware. With this option the X driver instead sets an event handler and waits for the hardware through the 'poll()' system call.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: the use of the events is disabled.

Option "FlatPanelProperties" "string"

This option requests particular properties for all or a subset of the connected flat panels.

The option string is a semicolon-separated list of comma-separated property=value pairs. Each list of property=value pairs can optionally be prepended with a flat panel name and GPU specifier.

    "<DFP-0>: <property=value>, <property=value>; <DFP-1>: <property=value>; ..."

Recognized properties:

   o "Scaling": controls the flat panel scaling mode; possible values are: 'Default' (the driver will use whichever scaling state is current), 'Native' (the driver will use the flat panel's scaler, if possible), 'Scaled' (the driver will use the NVIDIA GPU's scaler, if possible), 'Centered' (the driver will center the image, if possible), and 'aspect-scaled' (the X driver will scale with the NVIDIA GPU's scaler, but keep the aspect ratio correct).

   o "Dithering": controls the flat panel dithering configuration; possible values are: 'Auto' (the driver will decide when to dither), 'Enabled' (the driver will always dither, if possible), and 'Disabled' (the driver will never dither).

   o "DitheringMode": controls the flat panel dithering mode; possible values are: 'Auto' (the driver will choose possible default mode), 'Dynamic-2x2' (a 2x2 dithering pattern is updated for every frame), 'Static-2x2' (a 2x2 dithering pattern remains constant throughout the frames), and 'Temporal' (a pseudo-random dithering algorithm is used).

Examples:

    Option "FlatPanelProperties" "Scaling = Centered"

set the flat panel scaling mode to centered on all flat panels.

    Option "FlatPanelProperties" "GPU-0.DFP-0: Scaling = Centered; DFP-1: Scaling = Scaled, Dithering = Enabled, DitheringMode = Static-2x2" set the scaling mode of DFP-0 on GPU-0 to centered, set DFP-1's scaling mode to scaled, its dithering to enabled and dithering mode to static 2x2.

Option "ProbeAllGpus" "boolean"

When the NVIDIA X driver initializes, it probes all GPUs in the system, even if no X screens are configured on them. This is done so that the X driver can report information about all the system's GPUs through the NV-CONTROL X extension. This option can be set to FALSE to disable this behavior, such that only GPUs with X screens configured on them will be probed.

Note that disabling this option may affect configurability through nvidia-settings, since the X driver will not know about GPUs that aren't currently being used or the display devices attached to them.

Default: all GPUs in the system are probed.

Option "DynamicTwinView" "boolean"

Enable or disable support for dynamically configuring TwinView on this X screen. When DynamicTwinView is enabled (the default), the refresh rate of a mode (reported through XF86VidMode or XRandR) does not correctly report the refresh rate, but instead is a unique number such that each MetaMode has a different value. This is to guarantee that MetaModes can be uniquely identified by XRandR.

When DynamicTwinView is disabled, the refresh rate reported through XRandR will be accurate, but NV-CONTROL clients such as nvidia-settings will not be able to dynamically manipulate the X screen's MetaModes. TwinView can still be configured from the X config file when DynamicTwinView is disabled.

Default: DynamicTwinView is enabled.

Option "IncludeImplicitMetaModes" "boolean"

When the X server starts, a mode pool is created per display device, containing all the mode timings that the NVIDIA X driver determined to be valid for the display device. However, the only MetaModes that are made available to the X server are the ones explicitly requested in the X configuration file.

It is convenient for fullscreen applications to be able to change between the modes in the mode pool, even if a given target mode was not explicitly requested in the X configuration file.

To facilitate this, the NVIDIA X driver will implicitly add MetaModes for all modes in the primary display device's mode pool. This makes all the modes in the mode pool available to full screen applications that use the XF86VidMode extension or RandR 1.0/1.1 requests.

Further, to make sure that fullscreen applications have a reasonable set of MetaModes available to them, the NVIDIA X driver will also add implicit MetaModes for common resolutions: 1920x1200, 1920x1080, 1600x1200, 1280x1024, 1280x720, 1024x768, 800x600, 640x480. For these common resolution implicit MetaModes, the common resolution will be the ViewPortIn, and nvidia-auto-select will be the mode. The ViewPortOut will be configured such that the ViewPortIn is aspect scaled within the mode. Each common resolution implicit MetaMode will be added if there is not already a MetaMode with that resolution, and if the resolution is not larger than the nvidia-auto-select mode of the display device. See Chapter 12 for details of the relationship between ViewPortIn, ViewPortOut, and the mode within a MetaMode.

The IncludeImplicitMetaModes X configuration option can be used to disable the addition of implicit MetaModes. Or, it can be used to alter how implicit MetaModes are added. The option can have either a boolean value or a comma-separated list of token=value pairs, where the possible tokens are:

o "DisplayDevice": specifies the display device for which the implicit MetaModes should be created. Any name that can be used to identify a display device can be used here; see Appendix C for details.

o "Mode": specifies the name of the mode to use with the common resolution-based implicit MetaModes. The default is "nvidia-auto-select". Any mode in the display device's mode pool can be used here.

o "Scaling": specifies how the ViewPortOut should be configured between the ViewPortIn and the mode for the common resolution-based implicit MetaModes. Possible values are "Scaled", "Aspect-Scaled", or "Centered". The default is "Aspect-Scaled".

- "UseModePool": specifies whether modes from the display device's mode pool should be used to create implicit MetaModes. The default is "true".

- "UseCommonResolutions": specifies whether the common resolution list should be used to create implicit MetaModes. The default is "true".

- "Derive16x9Mode": specifies whether to create an implicit MetaMode with a resolution whose aspect ratio is 16:9, using the width of nvidia-auto-select. E.g., using a 2560x1600 monitor, this would create an implicit MetaMode of 2560x1440. The default is "true".

- "ExtraResolutions": a comma-separated list of additional resolutions to use for creating implicit MetaModes. These will be created in the same way as the common resolution implicit MetaModes: the resolution will be used as the ViewPortIn, the nvidia-auto-select mode will be used as the mode, and the ViewPortOut will be computed to aspect scale the resolution within the mode. Note that the list of resolutions must be enclosed in parentheses, so that the commas are not interpreted as token=value pair separators.

Some examples:

```
Option "IncludeImplicitMetaModes" "off"
Option "IncludeImplicitMetaModes" "on" (the default)
Option "IncludeImplicitMetaModes" "DisplayDevice = DVI-I-2,
Scaling=Aspect-Scaled, UseModePool = false"
Option "IncludeImplicitMetaModes" "ExtraResolutions = ( 2560x1440, 320x200
), DisplayDevice = DVI-I-0"
```

Option "IndirectMemoryAccess" "boolean"

Some graphics cards have more video memory than can be mapped at once by the CPU (generally at most 256 MB of video memory can be CPU-mapped). On graphics cards based on G80 and higher, this option allows the driver to:

- place more pixmaps in video memory, which will improve hardware rendering performance but may slow down software rendering;

- allocate buffers larger than 256 MB, which is necessary to reach the maximum buffer size on newer GPUs.

On some systems, up to 3 gigabytes of virtual address space may be reserved in the X server for indirect memory access. This virtual memory does not consume any physical resources. Note that the amount of reserved memory may be limited on 32-bit platforms, so some problems with large buffer allocations can be resolved by switching to a 64-bit operating system.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: on (indirect memory access will be used, when available).

Option "OnDemandVBlankInterrupts" "boolean"

Normally, VBlank interrupts are generated on every vertical refresh of every display device connected to the GPU(s) installed in a given system. This experimental option enables on-demand VBlank control, allowing the driver to enable VBlank interrupt generation only when it is required. This can help conserve power.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: off (on-demand VBlank control is disabled).

Option "AllowSHMPixmaps" "boolean"

This option controls whether applications can use the MIT-SHM X extension to create pixmaps whose contents are shared between the X server and the client. These pixmaps prevent the NVIDIA driver from performing a number of optimizations and degrade performance in many circumstances.

Disabling this option disables only shared memory pixmaps. Applications can still use the MIT-SHM extension to transfer data to the X server through shared memory using XShmPutImage.

Default: off (shared memory pixmaps are not allowed).

Option "InitializeWindowBackingPixmaps" "boolean"

This option controls whether the NVIDIA X Driver initializes newly created redirected windows using the contents of their parent window if the X server doesn't do it. Leaving redirected windows uninitialized may cause new windows to flash with black or random colors when some compositing managers are running.

This option will have no effect on X servers that already initialize redirected window contents. In most distributions, the X server is patched to skip that initialization. In this case, it is recommended to leave this option on for a better user experience.

Default: on (redirected windows are initialized).

Option "AllowUnofficialGLXProtocol" "boolean"

By default, the NVIDIA GLX implementation will not expose GLX protocol for GL commands if the protocol is not considered complete. Protocol could be considered incomplete for a number of reasons. The implementation could still be under development and contain known bugs, or the protocol specification itself could be under development or going through review. If users would like to test the server-side portion of such protocol when using indirect rendering, they can enable this option. If any X screen enables this option, it will enable protocol on all screens in the server.

When an NVIDIA GLX client is used, the related environment variable "__GL_ALLOW_UNOFFICIAL_PROTOCOL" will need to be set as well to enable support in the client.

Option "PanAllDisplays" "boolean"

When this option is enabled, all displays in the current MetaMode will pan as the pointer is moved. If disabled, only the displays whose panning domain contains the pointer (at its new location) are panned.

Default: enabled (all displays are panned when the pointer is moved).

Option "GvoDataFormat" "string"

This option controls the initial configuration of SDI (GVO) device's output data format.

```
Valid Values
-------------------------------------------------------------------
R8G8B8_To_YCrCb444
R8G8B8_To_YCrCb422
```

```
    X8X8X8_To_PassThru444
```

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: R8G8B8_To_YCrCb444.

Option "GvoSyncMode" "string"

This option controls the initial synchronization mode of the SDI (GVO) device.

```
    Value           Behavior
    --------------  ---------------------------------------------------
    FreeRunning     The SDI output will be synchronized with the
                    timing chosen from the SDI signal format list.
    GenLock         SDI output will be synchronized with the external
                    sync signal (if present/detected) with pixel
                    accuracy.
    FrameLock       SDI output will be synchronized with the external
                    sync signal (if present/detected) with frame
                    accuracy.
```

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: FreeRunning (Will not lock to an input signal).

Option "GvoSyncSource" "string"

This option controls the initial synchronization source (type) of the SDI (GVO) device. Note that the GvoSyncMode should be set to either GenLock or FrameLock for this option to take effect.

```
    Value           Behavior
    --------------  ---------------------------------------------------
    Composite       Interpret sync source as composite.
    SDI             Interpret sync source as SDI.
```

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: SDI.

Option "Interactive" "boolean"

This option controls the behavior of the driver's watchdog, which attempts to detect and terminate GPU programs that get stuck, in order to ensure that the GPU remains available for other processes. GPU compute applications, however, often have long-running GPU programs, and killing them would be undesirable. If you are using GPU compute applications and they are getting prematurely terminated, try turning this option off.

When this option is set for an X screen, it will be applied to all X screens running on the same GPU.

Default: on. The driver will attempt to detect and terminate GPU programs that cause excessive delays for other processes using the GPU.

Option "BaseMosaic" "boolean"

This option can be used to extend a single X screen transparently across display outputs on each GPU. This is like SLI Mosaic mode except that it does not require a video bridge connected to the graphics cards. Due to this Base Mosaic does not guarantee there will be no tearing between the display boundaries. Base Mosaic is supported on all G80 and higher SLI configurations up to three display devices. It is also supported on Quadro FX 380, Quadro FX 580 and all G80 or higher non-mobile NVS cards on all available display devices.

Use this in conjunction with the MetaModes X configuration option to specify the combination of mode(s) used on each display. nvidia-xconfig can be used to configure Base Mosaic via a command like 'nvidia-xconfig --base-mosaic --metamodes=METAMODES' where the METAMODES string specifies the desired grid configuration. For example, to configure four DFPs in a 2x2 configuration, each running at 1920x1024, with two DFPs connected to two cards, the command would be:

```
    nvidia-xconfig --base-mosaic --metamodes="GPU-0.DFP-0: 1920x1024+0+0,
GPU-0.DFP-1: 1920x1024+1920+0, GPU-1.DFP-0: 1920x1024+0+1024, GPU-1.DFP-1:
1920x1024+1920+1024"
```

Option "ConstrainCursor" "boolean"

    When this option is enabled, the mouse cursor will be constrained to the
    region of the desktop that is visible within the union of all displays'
    panning domains in the current MetaMode. When it is disabled, it may be
    possible to move the cursor to regions of the X screen that are not
    visible on any display.

    Note that if this would make display's panning domain is inaccessible (in
    other words, if the union of all panning domains is disjoint), then the
    cursor will not be constrained, so that it is still possible to move the
    cursor to each display.

    This option has no effect if the X server doesn't support cursor
    constraint. This support was added in X.Org server version 1.10 (see "Q.
    How do I interpret X server version numbers?" in Chapter 7).

    Default: on, if the X server supports it. The cursor will be constrained
    to the panning domain of each monitor, when possible.

Appendix C. Display Device Names

A "display device" refers to a hardware device capable of displaying an image.
Most NVIDIA GPUs can drive multiple display devices simultaneously.

Many X configuration options can be used to separately configure each display
device in use by the X screen. To address an individual display device, you
can use one of several names that are assigned to it.

For example, the "ModeValidation" X configuration option by default applies to
all display devices on the X screen. E.g.,

```
    Option "ModeValidation" "NoMaxPClkCheck"
```

You can use a display device name qualifier to configure each display device's
ModeValidation separately. E.g.,

```
    Option "ModeValidation" "DFP-0: NoMaxPClkCheck; CRT-1: NoVesaModes"
```

The description of each X configuration option in Appendix B provides more detail on the available syntax for each option.

The available display device names vary by GPU. To find all available names for your configuration, start the X server with verbose logging enabled (e.g., `startx -- -logverbose 5`, or enable the "ModeDebug" X configuration option with `nvidia-xconfig --mode-debug` and restart the X server).

The X log (normally /var/log/Xorg.0.log) will contain a list of what display devices are valid for the GPU. E.g.,

```
(--) NVIDIA(0): Valid display device(s) on Quadro 6000 at PCI:10:0:0
(--) NVIDIA(0):     CRT-0
(--) NVIDIA(0):     CRT-1
(--) NVIDIA(0):     DELL U2410 (DFP-0) (connected)
(--) NVIDIA(0):     NEC LCD1980SXi (DFP-1) (connected)
```

The X log will also contain a list of which display devices are assigned to the X screen. E.g.,

```
(II) NVIDIA(0): Display device(s) assigned to X screen 0:
(II) NVIDIA(0):   CRT-0
(II) NVIDIA(0):   CRT-1
(II) NVIDIA(0):   DELL U2410 (DFP-0)
(II) NVIDIA(0):   NEC LCD1980SXi (DFP-1)
```

Note that when multiple X screens are configured on the same GPU, the NVIDIA X driver assigns different display devices to each X screen. On X servers that support RandR 1.2 or later, the NVIDIA X driver will create an RandR output for each display device assigned to an X screen.

The X log will also report a list of "Name Aliases" for each display device. E.g.,

```
(--) NVIDIA(0): Name Aliases for NEC LCD1980SXi (DFP-1):
(--) NVIDIA(0):   DFP
(--) NVIDIA(0):   DFP-1
(--) NVIDIA(0):   DPY-3
(--) NVIDIA(0):   DVI-I-3
```

```
(--) NVIDIA(0):   DPY-EDID-373091cb-5c07-6430-54d2-1112efd64b44
```

These aliases can be used interchangeably to refer to the same display device
in any X configuration option, or in NV-CONTROL protocol that uses similar
strings, such as NV_CTRL_STRING_CURRENT_METAMODE_VERSION_2 (available through
the nvidia-settings command line as `nvidia-settings --query
CurrentMetaMode`).

Each alias has different properties that may affect which alias is appropriate
to use. The possible alias names are:

o A "type"-based name (e.g., "DFP-1"). This name is a unique index plus a
  display device type name, though in actuality the "type name" is selected
  based on the protocol through which the X driver communicates to the
  display device. If the X driver communicates using VGA, then the name is
  "CRT"; if the driver communicates using TMDS, LVDS, or DP, then the name
  is "DFP"; if the driver communicates using S-Video, composite video, or
  component video, then the name is "TV".

  This may cause confusion in some cases (e.g., a digital flat panel
  connected via VGA will have the name "CRT"), but this name alias is
  provided for backwards compatibility with earlier NVIDIA driver releases.

  Also for backwards compatibility, an alias is provided that uses the
  "type name" without an index. This name alias will match any display
  device of that type: it is not unique across the X screen.

  Note that the index in this type-based name is based on which physical
  connector is used. If you reconnect a display device to a different
  connector on the GPU, the type-based name will be different.

o A connector-based name (e.g., "DVI-I-3"). This name is a unique index
  plus a name that is based on the physical connector through which the
  display device is connected to the GPU. E.g., "VGA-1", "DVI-I-0",
  "DVI-D-3", "LVDS-1", "DP-2", "HDMI-3", "eDP-6". On X servers that support
  RandR 1.2 or later, this name is also used as the RandR output name.

  Note that the index in this connector-based name is based on which
  physical connector is used. If you reconnect a display device to a
  different connector on the GPU, the connector-based name will be
  different.

When Mosaic is enabled, this name is prefixed with a GPU identifier to make it unique. For example, a Mosaic configuration with two DisplayPort devices might have two different outputs with names "GPU-0.DP-0" and "GPU-1.DP-0", respectively.

o An EDID-based name (e.g., "DPY-EDID-373091cb-5c07-6430-54d2-1112efd64b44"). This name is a SHA-1 hash, formatted in canonical UUID 8-4-4-4-12 format, of the display device's EDID. This name will be the same regardless of which physical connector on the GPU you use, but it will not be unique if you have multiple display devices with the same EDID.

o An NV-CONTROL target ID-based name (e.g., "DPY-3"). The NVIDIA X driver will assign a unique ID to each display device on the entire X server. These IDs are not guaranteed to be persistent from one run of the X server to the next, so is likely not convenient for X configuration file use. It is more frequently used in communication with NV-CONTROL clients such as nvidia-settings.

When DisplayPort 1.2 branch devices are present, display devices will be created with type- and connector-based names that are based on how they are connected to the branch device tree. For example, if a connector named DP-2 has a branch device attached and a DisplayPort device is connected to the branch device's first downstream port, a display device named "DP-2.1" might be created. If another branch device is connected between the first branch device and the display device, the name might be "DP-2.1.1".

Any display device name can have an optional GPU qualifier prefix. E.g., "GPU-0.DVI-I-3". This is useful in Mosaic configurations: type- and connector-based display device names are only unique within a GPU, so the GPU qualifier is used to distinguish between identically named display devices on different GPUs. For example:

```
    Option "MetaModes"   "GPU-0.CRT-0: 1600x1200, GPU-1.CRT-0: 1024x768"
```

If no GPU is specified for a particular display device name, the setting will apply to any devices with that name across all GPUs.

Appendix D. GLX Support

This release supports GLX 1.4.

Additionally, the following GLX extensions are supported on appropriate GPUs:

o GLX_EXT_visual_info

o GLX_EXT_visual_rating

o GLX_SGIX_fbconfig

o GLX_SGIX_pbuffer

o GLX_ARB_get_proc_address

o GLX_SGI_video_sync

o GLX_SGI_swap_control

o GLX_ARB_multisample

o GLX_NV_float_buffer

o GLX_ARB_fbconfig_float

o GLX_NV_swap_group

o GLX_NV_video_out

o GLX_EXT_texture_from_pixmap

o GLX_NV_copy_image

o GLX_ARB_create_context

o GLX_EXT_import_context

o GLX_EXT_fbconfig_packed_float

o GLX_EXT_framebuffer_sRGB

o GLX_NV_present_video

   o GLX_NV_multisample_coverage

   o GLX_EXT_swap_control

   o GLX_NV_video_capture

   o GLX_ARB_create_context_profile

For a description of these extensions, see the OpenGL extension registry at
http://www.opengl.org/registry/

Some of the above extensions exist as part of core GLX 1.4 functionality,
however, they are also exported as extensions for backwards compatibility.

Unofficial GLX protocol support exists in NVIDIA's GLX client and GLX server
implementations for the following OpenGL extensions:

   o GL_ARB_geometry_shader4

   o GL_ARB_shader_objects

   o GL_ARB_texture_buffer_object

   o GL_ARB_vertex_buffer_object

   o GL_ARB_vertex_shader

   o GL_EXT_bindable_uniform

   o GL_EXT_compiled_vertex_array

   o GL_EXT_geometry_shader4

   o GL_EXT_gpu_shader4

   o GL_EXT_texture_buffer_object

   o GL_NV_geometry_program4

   o GL_NV_vertex_program

   o GL_NV_parameter_buffer_object

   o GL_NV_vertex_program4

Until the GLX protocol for these OpenGL extensions is finalized, using these
extensions through GLX indirect rendering will require the
AllowUnofficialGLXProtocol X configuration option, and the
__GL_ALLOW_UNOFFICIAL_PROTOCOL environment variable in the environment of the
client application. Unofficial protocol requires the use of NVIDIA GLX
libraries on both the client and the server. Note: GLX protocol is used when
an OpenGL application indirect renders (i.e., runs on one computer, but
submits protocol requests such that the rendering is performed on another
computer). The above OpenGL extensions are fully supported when doing direct
rendering.

GLX visuals and FBConfigs are only available for X screens with depths 16, 24,
or 30.

Appendix E. Dots Per Inch

DPI (Dots Per Inch), also known as PPI (Pixels Per Inch), is a property of an
X screen that describes the physical size of pixels. Some X applications, such
as xterm, can use the DPI of an X screen to determine how large (in pixels) to
draw an object in order for that object to be displayed at the desired
physical size on the display device.

The DPI of an X screen is computed by dividing the size of the X screen in
pixels by the size of the X screen in inches:

```
    DPI = SizeInPixels / SizeInInches
```

Since the X screen stores its physical size in millimeters rather than inches
(1 inch = 25.4 millimeters):

```
    DPI = (SizeInPixels * 25.4) / SizeInMillimeters
```

The NVIDIA X driver reports the size of the X screen in pixels and in
millimeters. On X.Org 6.9 or newer, when the XRandR extension resizes the X
screen in pixels, the NVIDIA X driver computes a new size in millimeters for
the X screen, to maintain a constant DPI (see the "Physical Size" column of
the `xrandr -q` output as an example). This is done because a changing DPI can

cause interaction problems for some applications. To disable this behavior, and instead keep the same millimeter size for the X screen (and therefore have a changing DPI), set the ConstantDPI option to FALSE (see Appendix B for details).

You can query the DPI of your X screen by running:

```
    % xdpyinfo | grep -B1 dot
```

which should generate output like this:

```
    dimensions:    1280x1024 pixels (382x302 millimeters)
    resolution:    85x86 dots per inch
```

The NVIDIA X driver performs several steps during X screen initialization to determine the DPI of each X screen:

o If the display device provides an EDID, and the EDID contains information about the physical size of the display device, that is used to compute the DPI, along with the size in pixels of the first mode to be used on the display device.

  Note that in some cases, the physical size information stored in a display device's EDID may be unreliable. This could result in a display device's DPI being computed incorrectly, potentially leading to undesired consequences such as fonts that are scaled larger or smaller than expected. These issues can be worked around by manually setting a DPI using the "DPI" X configuration option, or by disabling the use of the EDID's physical size information for computing DPI by setting the "UseEdidDpi" X configuration option to "FALSE"'. See Appendix B for details.

  If multiple display devices are used by this X screen, then the NVIDIA X screen will choose which display device to use. You can override this with the "UseEdidDpi" X configuration option: you can specify a particular display device to use; e.g.:

```
        Option "UseEdidDpi" "DFP-1"
```

or disable EDID-computed DPI by setting this option to false:

```
        Option "UseEdidDpi" "FALSE"
```

EDID-based DPI computation is enabled by default when an EDID is available.

o If the "-dpi" commandline option to the X server is specified, that is used to set the DPI (see `X -h` for details). This will override the "UseEdidDpi" option.

o If the "DPI" X configuration option is specified (see Appendix B for details), that will be used to set the DPI. This will override the "UseEdidDpi" option.

o If none of the above are available, then the "DisplaySize" X config file Monitor section information will be used to determine the DPI, if provided; see the xorg.conf or XF86Config man pages for details.

o If none of the above are available, the DPI defaults to 75x75.

You can find how the NVIDIA X driver determined the DPI by looking in your X log file. There will be a line that looks something like the following:

```
    (--) NVIDIA(0): DPI set to (101, 101); computed from "UseEdidDpi" X config
option
```

Note that the physical size of the X screen, as reported through `xdpyinfo` is computed based on the DPI and the size of the X screen in pixels.

The DPI of an X screen can be poorly defined when multiple display devices are enabled on the X screen: those display devices might have different actual DPIs, yet DPI is advertised from the X server to the X application with X screen granularity. Solutions for this include:

o Use separate X screens, with one display device on each X screen; see Chapter 14 for details.

o The RandR X extension version 1.2 and later reports the physical size of each RandR Output, so applications could possibly choose to render content at different sizes, depending on which portion of the X screen is displayed on which display devices. Client applications can also configure the reported per-RandR Output physical size. See, e.g., the xrandr(1) '--fbmm' command line option.

o Experiment with different DPI settings to find a DPI that is suitable for all display devices on the X screen.

Appendix F. XvMC Support

This release includes support for the XVideo Motion Compensation (XvMC) version 1.0 API on GeForce 6 series and GeForce 7 series add-in cards, as well as motherboard chipsets with integrated graphics that have PureVideo support based on these GPUs. There is a static library, "libXvMCNVIDIA.a", and a dynamic one, "libXvMCNVIDIA_dynamic.so", which is suitable for dlopening. XvMC's "IDCT" and "motion-compensation" levels of acceleration, AI44 and IA44 subpictures, and 4:2:0 Surfaces up to 2032x2032 are supported.

libXvMCNVIDIA observes the XVMC_DEBUG environment variable and will provide some debug output to stderr when set to an appropriate integer value. '0' disables debug output. '1' enables debug output for failure conditions. '2' or higher enables output of warning messages.

Appendix G. VDPAU Support

This release includes support for the Video Decode and Presentation API for Unix-like systems (VDPAU) on most GeForce 8 series and newer add-in cards, as well as motherboard chipsets with integrated graphics that have PureVideo support based on these GPUs.

VDPAU is only available for X screens with depths 16, 24, or 30.

VDPAU supports Xinerama. The following restrictions apply:

- o Physical X screen 0 must be driven by the NVIDIA driver.

- o VDPAU will only display on physical X screens driven by the NVIDIA driver, and which are driven by a GPU both compatible with VDPAU, and compatible with the GPU driving physical X screen 0.

Under Xinerama, VDPAU performs all operations other than display on a single GPU. By default, the GPU associated with physical X screen 0 is used. The environment variable VDPAU_NVIDIA_XINERAMA_PHYSICAL_SCREEN may be used to specify a physical screen number, and then VDPAU will operate on the GPU associated with that physical screen. This variable should be set to the integer screen number as configured in the X configuration file. The selected physical X screen must be driven by the NVIDIA driver.

G1. IMPLEMENTATION LIMITS

VDPAU is specified as a generic API - the choice of which features to support, and performance levels of those features, is left up to individual implementations. The details of NVIDIA's implementation are provided below.

VDPVIDEOSURFACE

The maximum supported resolution is 4096x4096.

The following surface formats and get-/put-bits combinations are supported:

- o VDP_CHROMA_TYPE_420 (Supported get-/put-bits formats are VDP_YCBCR_FORMAT_NV12, VDP_YCBCR_FORMAT_YV12)

- o VDP_CHROMA_TYPE_422 (Supported get-/put-bits formats are VDP_YCBCR_FORMAT_UYVY, VDP_YCBCR_FORMAT_YUYV)

VDPBITMAPSURFACE

The maximum supported resolution is 16384x16384 pixels for the GeForce GTX 400 series and newer GPUs, and 8192x8192 pixels for older GPUs.

The following surface formats are supported:

- o VDP_RGBA_FORMAT_B8G8R8A8
- o VDP_RGBA_FORMAT_R8G8B8A8
- o VDP_RGBA_FORMAT_B10G10R10A2
- o VDP_RGBA_FORMAT_R10G10B10A2
- o VDP_RGBA_FORMAT_A8

Note that VdpBitmapSurfaceCreate's frequently_accessed parameter directly controls whether the bitmap data will be placed into video RAM (VDP_TRUE) or system memory (VDP_FALSE). Note that if the bitmap data cannot be placed into video RAM when requested due to resource constraints, the implementation will automatically fall back to placing the data into system RAM.

VDPOUTPUTSURFACE

The maximum supported resolution is 16384x16384 pixels for the GeForce GTX 400 series and newer GPUs, and 8192x8192 pixels for older GPUs.

The following surface formats are supported:

- o VDP_RGBA_FORMAT_B8G8R8A8
- o VDP_RGBA_FORMAT_R10G10B10A2

For all surface formats, the following get-/put-bits indexed formats are supported:

- o VDP_INDEXED_FORMAT_A4I4
- o VDP_INDEXED_FORMAT_I4A4
- o VDP_INDEXED_FORMAT_A8I8
- o VDP_INDEXED_FORMAT_I8A8

For all surface formats, the following get-/put-bits YCbCr formats are
supported:

- o VDP_YCBCR_FORMAT_Y8U8V8A8

- o VDP_YCBCR_FORMAT_V8U8Y8A8

VDPDECODER

In all cases, VdpDecoder objects solely support 8-bit 4:2:0 streams, and only
support writing to VDP_CHROMA_TYPE_420 surfaces.

The exact set of supported VdpDecoderProfile values depends on the GPU in use.
Appendix A lists which GPUs support which video feature set. An explanation of
each video feature set may be found below. When reading these lists, please
note that VC1_SIMPLE and VC1_MAIN may be referred to as WMV, WMV3, or WMV9 in
other contexts. Partial acceleration means that VLD (bitstream) decoding is
performed on the CPU, with the GPU performing IDCT and motion compensation.
Complete acceleration means that the GPU performs all of VLD, IDCT, and motion
compensation.

VDPAU FEATURE SET A

GPUs with VDPAU feature set A support at least the following VdpDecoderProfile
values, and associated limits:

- o VDP_DECODER_PROFILE_MPEG1, VDP_DECODER_PROFILE_MPEG2_SIMPLE,
  VDP_DECODER_PROFILE_MPEG2_MAIN:

  - o Partial acceleration.

  - o Minimum width or height: 3 macroblocks (48 pixels).

  - o Maximum width or height: 128 macroblocks (2048 pixels).

  - o Maximum macroblocks: 8192

- o VDP_DECODER_PROFILE_H264_MAIN, VDP_DECODER_PROFILE_H264_HIGH:

    o Complete acceleration.

    o Minimum width or height: 3 macroblocks (48 pixels).

    o Maximum width or height: 128 macroblocks (2048 pixels).

    o Maximum macroblocks: 8192

  o VDP_DECODER_PROFILE_VC1_SIMPLE, VDP_DECODER_PROFILE_VC1_MAIN,
    VDP_DECODER_PROFILE_VC1_ADVANCED:

    o Partial acceleration.

    o Minimum width or height: 3 macroblocks (48 pixels).

    o Maximum width or height: 128 macroblocks (2048 pixels).

    o Maximum macroblocks: 8190

VDPAU FEATURE SET B

GPUs with VDPAU feature set B support at least the following VdpDecoderProfile
values, and associated limits:

  o VDP_DECODER_PROFILE_MPEG1, VDP_DECODER_PROFILE_MPEG2_SIMPLE,
    VDP_DECODER_PROFILE_MPEG2_MAIN:

    o Complete acceleration.

    o Minimum width or height: 3 macroblocks (48 pixels).

    o Maximum width or height: 128 macroblocks (2048 pixels).

    o Maximum macroblocks: 8192

  o VDP_DECODER_PROFILE_H264_MAIN, VDP_DECODER_PROFILE_H264_HIGH:

    o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width: 127 macroblocks (2032 pixels).

   o Maximum height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8190

o VDP_DECODER_PROFILE_VC1_SIMPLE, VDP_DECODER_PROFILE_VC1_MAIN,
  VDP_DECODER_PROFILE_VC1_ADVANCED:

   o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width or height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8190

VDPAU FEATURE SETS C AND D

GPUs with VDPAU feature set C or D support at least the following
VdpDecoderProfile values, and associated limits:

o VDP_DECODER_PROFILE_MPEG1, VDP_DECODER_PROFILE_MPEG2_SIMPLE,
  VDP_DECODER_PROFILE_MPEG2_MAIN:

   o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width or height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8192

o VDP_DECODER_PROFILE_H264_MAIN, VDP_DECODER_PROFILE_H264_HIGH:

   o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width or height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8192

o VDP_DECODER_PROFILE_VC1_SIMPLE, VDP_DECODER_PROFILE_VC1_MAIN,
VDP_DECODER_PROFILE_VC1_ADVANCED:

   o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width or height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8190

o VDP_DECODER_PROFILE_MPEG4_PART2_SP,
VDP_DECODER_PROFILE_MPEG4_PART2_ASP,
VDP_DECODER_PROFILE_DIVX4_QMOBILE, VDP_DECODER_PROFILE_DIVX4_MOBILE,
VDP_DECODER_PROFILE_DIVX4_HOME_THEATER,
VDP_DECODER_PROFILE_DIVX4_HD_1080P,
VDP_DECODER_PROFILE_DIVX5_QMOBILE,
VDP_DECODER_PROFILE_DIVX5_MOBILE,
VDP_DECODER_PROFILE_DIVX5_HOME_THEATER,
VDP_DECODER_PROFILE_DIVX5_HD_1080P

   o Complete acceleration.

   o Minimum width or height: 3 macroblocks (48 pixels).

   o Maximum width or height: 128 macroblocks (2048 pixels).

   o Maximum macroblocks: 8192

The following features are currently not supported:

   o GMC (Global Motion Compensation)

o Data partitioning

o reversible VLC

These GPUs also support VDP_VIDEO_MIXER_FEATURE_HIGH_QUALITY_SCALING_L1.

VDPAU FEATURES NOTE 1

GPUs with this note may not support H.264 streams with the following widths:
49, 54, 59, 64, 113, 118, 123, 128 macroblocks (769-784, 849-864, 929-944,
1009-1024, 1793-1808, 1873-1888, 1953-1968, 2033-2048 pixels).

VDPVIDEOMIXER

The maximum supported resolution is 4096x4096.

The video mixer supports all video and output surface resolutions and formats
that the implementation supports.

The video mixer supports at most 4 auxiliary layers.

The following features are supported:

o VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL

o VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL_SPATIAL

o VDP_VIDEO_MIXER_FEATURE_INVERSE_TELECINE

o VDP_VIDEO_MIXER_FEATURE_NOISE_REDUCTION

o VDP_VIDEO_MIXER_FEATURE_SHARPNESS

o VDP_VIDEO_MIXER_FEATURE_LUMA_KEY

In order for either VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL or VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL_SPATIAL to operate correctly, the application must supply at least 2 past and 1 future fields to each

VdpMixerRender call. If those fields are not provided, the VdpMixer will fall
back to bob de-interlacing.

Both regular de-interlacing and half-rate de-interlacing are supported. Both
have the same requirements in terms of the number of past/future fields
required. Both modes should produce equivalent results.

In order for VDP_VIDEO_MIXER_FEATURE_INVERSE_TELECINE to have any effect, one
of VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL or
VDP_VIDEO_MIXER_FEATURE_DEINTERLACE_TEMPORAL_SPATIAL must be requested and
enabled. Inverse telecine has the same requirement on the minimum number of
past/future fields that must be provided. Inverse telecine will not operate
when "half-rate" de-interlacing is used.

While it is possible to apply de-interlacing algorithms to progressive streams
using the techniques outlined in the VDPAU documentation, NVIDIA does not
recommend doing so. One is likely to introduce more artifacts due to the
inverse telecine process than are removed by detection of bad edits etc.

VDPPRESENTATIONQUEUE

The resolution of VdpTime is approximately 10 nanoseconds. At some arbitrary
point during system startup, the initial value of this clock is synchronized
to the system's real-time clock, as represented by nanoseconds since since Jan
1, 1970. However, no attempt is made to keep the two time-bases synchronized
after this point. Divergence can and will occur.

NVIDIA's VdpPresentationQueue supports two methods for displaying surfaces;
overlay and blit. The overlay method will be used wherever possible, with the
blit method acting as a more general fallback.

Whenever a presentation queue is created, the driver determines whether the
overlay method may ever be used, based on system configuration, and whether
any other application already owns the overlay. If overlay usage is
potentially possible, the presentation queue is marked as owning the overlay.

Whenever a surface is displayed, the driver determines whether the overlay
method may be used for that frame, based on both whether the presentation
queue owns the overlay, and the set of overlay usage limitations below. In
other words, the driver may switch back and forth between overlay and blit
methods dynamically. The most likely cause for dynamic switching is when a
compositing manager is enabled or disabled, and the window becomes redirected

or unredirected.

The following conditions or system configurations will prevent usage of the overlay path:

o Overlay hardware already in use, e.g. by another VDPAU, GL, or X11 application, or by SDI output.

o Desktop rotation enabled on the given X screen.

o The presentation target window is redirected, due to a compositing manager actively running.

o The environment variable VDPAU_NVIDIA_NO_OVERLAY is set to a string representation of a non-zero integer.

o The driver determines that the performance requirements of overlay usage cannot be met by the current hardware configuration.

Both the overlay and blit methods sync to VBLANK. The overlay path is guaranteed never to tear, whereas the blit method is classed as "best effort".

When TwinView is enabled, the blit method can only sync to one of the display devices; this may cause tearing corruption on the display device to which VDPAU is not syncing. You can use the environment variable VDPAU_NVIDIA_SYNC_DISPLAY_DEVICE to specify the display device to which VDPAU should sync. You should set this environment variable to the name of a display device, for example "CRT-1". Look for the line "Connected display device(s):" in your X log file for a list of the display devices present and their names. You may also find it useful to review Chapter 12 "Configuring Twinview" and the section on Ensuring Identical Mode Timings in Chapter 18.

A VdpPresentationQueue allows a maximum of 8 surfaces to be QUEUED or VISIBLE at any one time. This limit is per presentation queue. If this limit is exceeded, VdpPresentationQueueDisplay blocks until an entry in the presentation queue becomes free.

G2. PERFORMANCE LEVELS

This documentation describes the capabilities of the NVIDIA VDPAU implementation. Hardware performance may vary significantly between cards. No guarantees are made, nor implied, that any particular combination of system configuration, GPU configuration, VDPAU feature set, VDPAU API usage, application, video stream, etc., will be able to decode streams at any particular frame rate.

G3. GETTING THE BEST PERFORMANCE FROM THE API

System performance (raw throughput, latency, and jitter tolerance) can be affected by a variety of factors. One of these factors is how the client application uses VDPAU; i.e. the number of surfaces allocated for buffering, order of operations, etc.

NVIDIA GPUs typically contain a number of separate hardware modules that are capable of performing different parts of the video decode, post-processing, and display operations in parallel. To obtain the best performance, the client application must attempt to keep all these modules busy with work at all times.

Consider the decoding process. At a bare minimum, the application must allocate one video surface for each reference frame that the stream can use (2 for MPEG or VC-1, a variable stream-dependent number for H.264) plus one surface for the picture currently being decoded. However, if this minimum number of surfaces is used, performance may be poor. This is because back-to-back decodes of non-reference frames will need to be written into the same video surface. This will require that decode of the second frame wait until decode of the first has completed; a pipeline stall.

Further, if the video surfaces are being read by the video mixer for post-processing, and eventual display, this will "lock" the surfaces for even longer, since the video mixer needs to read the data from the surface, which prevents any subsequent decode operations from writing to the surface. Recall that when advanced de-interlacing techniques are used, a history of video surfaces must be provided to the video mixer, thus necessitating that even more video surfaces be allocated.

For this reason, NVIDIA recommends the following number of video surfaces be allocated:

o (num_ref + 3) for progressive content, and no de-interlacing.

o (num_ref + 5) for interlaced content using advanced de-interlacing.

Next, consider the display path via the presentation queue. This portion of the pipeline requires at least 2 output surfaces; one that is being actively displayed by the presentation queue, and one being rendered to for subsequent display. As before, using this minimum number of surfaces may not be optimal. For some video streams, the hardware may only achieve real-time decoding on average, not for each individual frame. Using compositing APIs to render on-screen displays, graphical user interfaces, etc., may introduce extra jitter and latency into the pipeline. Similarly, system level issues such as scheduler algorithms and system load may prevent the CPU portion of the driver from operating for short periods of time. All of these potential issues may be solved by allocating more output surfaces, and queuing more than one outstanding output surface into the presentation queue.

The reason for using more than the minimum number of video surfaces is to ensure that the decoding and post-processing pipeline is not stalled, and hence is kept busy for the maximum amount of time possible. In contrast, the reason for using more than the minimum number of output surfaces is to hide jitter and latency in various GPU and CPU operations.

The choice of exactly how many surfaces to allocate is a resource usage v.s. performance trade-off; Allocating more than the minimum number of surfaces will increase performance, but use proportionally more video RAM. This may cause allocations to fail. This could be particularly problematic on systems with a small amount of video RAM. A stellar application would automatically adjust to this by initially allocating the bare minimum number of surfaces (failures being fatal), then attempting to allocate more and more surfaces, provided the allocations kept succeeding, up to the suggested limits above.

The video decoder's memory usage is also proportional to the maximum number of reference frames specified at creation time. Requesting a larger number of reference frames can significantly increase memory usage. Hence it is best for applications that decode H.264 to request only the actual number of reference frames specified in the stream, rather than e.g. hard-coding a limit of 16, or even the maximum number of surfaces allowable by some specific H.264 level at the stream's resolution.

Note that the NVIDIA implementation correctly implements all required interlocks between the various pipelined hardware modules. Applications never need worry about correctness (providing their API usage is legal and sensible), but simply have to worry about performance.

G4. ADDITIONAL NOTES

Note that output and bitmap surfaces are not cleared to any specific value upon allocation. It is the application's responsibility to initialize all surfaces prior to using them as input to any function. Video surfaces are cleared to black upon allocation.

G5. DEBUGGING AND TRACING

The VDPAU wrapper library supports tracing VDPAU function calls, and their parameters. This tracing is controlled by the following environment variables:

VDPAU_TRACE

Enables tracing. Set to 1 to trace function calls. Set to 2 to trace all arguments passed to the function.

VDPAU_TRACE_FILE

Filename to write traces to. By default, traces are sent to stderr. This variable may either contain a plain filename, or a reference to an existing open file-descriptor in the format "&N" where N is the file descriptor number.

The VDPAU wrapper library is responsible for determining which vendor-specific driver to load for a given X11 display/screen. At present, it hard-codes "nvidia" as the driver. The environment variable VDPAU_DRIVER may be set to override this default. The actual library loaded will be libvdpau_${VDPAU_DRIVER}.so. Setting VDPAU_DRIVER to "trace" is not advised.

The NVIDIA VDPAU driver can emit some diagnostic information when an error occurs. To enable this, set the environment variable VDPAU_NVIDIA_DEBUG. A value of 1 will request a small diagnostic that will enable NVIDIA engineers to locate the source of the problem. A value of 3 will request that a complete stack backtrace be printed, which provide NVIDIA engineers with more detailed information, which may be needed to diagnose some problems.

G6. MULTI-THREADING

VDPAU supports multiple threads actively executing within the driver, subject to certain limitations.

If any object is being created or destroyed, the VDPAU driver will become single-threaded. This includes object destruction during preemption cleanup.

Otherwise, up to one thread may actively execute VdpDecoderRender per VdpDecoder object, and up to one thread may actively execute any other rendering API per VdpDevice (or child) object. Note that the driver enforces these restrictions internally; applications are not required to implement the rules outlined above.

Finally, some of the "query" or "get" APIs may actively execute irrespective of the number of rendering threads currently executing.

Appendix H. Tips for New FreeBSD Users

This installation guide assumes that the user has at least a basic understanding of FreeBSD techniques and terminology. In this section we provide tips that the new user may find helpful. While the these tips are meant to clarify and assist users in installing and configuring the NVIDIA FreeBSD Driver, it is by no means a tutorial on the use or administration of the FreeBSD operating system. Unlike many desktop operating systems, it is relatively easy to cause irreparable damage to your FreeBSD system. If you are unfamiliar with the use of FreeBSD, we strongly recommend that you seek a tutorial through your distributor before proceeding.

H1. THE COMMAND PROMPT

While newer releases of FreeBSD bring new desktop interfaces to the user, much of the work in FreeBSD takes place at the command prompt. If you are familiar with the Windows operating system, the FreeBSD command prompt is analogous to the Windows command prompt, although the syntax and use varies somewhat. All of the commands in this section are performed at the command prompt. Some systems are configured to boot into console mode, in which case the user is presented with a prompt at login. Other systems are configured to start the X window system, in which case the user must open a terminal or console window in order to get a command prompt. This can usually be done by searching the desktop menus for a terminal or console program. While it is customizable, the basic prompt usually consists of a short string of information, one of the

characters '#', '$', or '%', and a cursor (possibly flashing) that indicates
where the user's input will be displayed.

H2. NAVIGATING THE DIRECTORY STRUCTURE

FreeBSD has a hierarchical directory structure. From anywhere in the directory
structure, the 'ls' command will list the contents of that directory. The
'file' command will print the type of files in a directory. For example,

```
    % file filename
```

will print the type of the file 'filename'. Changing directories is done with
the 'cd' command.

```
    % cd dirname
```

will change the current directory to 'dirname'. From anywhere in the directory
structure, the command 'pwd' will print the name of the current directory.
There are two special directories, '.' and '..', which refer to the current
directory and the next directory up the hierarchy, respectively. For any
commands that require a file name or directory name as an argument, you may
specify the absolute or the relative paths to those elements. An absolute path
begins with the "/" character, referring to the top or root of the directory
structure. A relative path begins with a directory in the current working
directory. The relative path may begin with '.' or '..'. Elements of a path
are separated with the "/" character. As an example, if the current directory
is '/home/jesse' and the user wants to change to the '/usr/local' directory,
he can use either of the following commands to do so:

```
    % cd /usr/local
```

or

```
    % cd ../../usr/local
```

H3. FILE PERMISSIONS AND OWNERSHIP

All files and directories have permissions and ownership associated with them.
This is useful for preventing non-administrative users from accidentally (or
maliciously) corrupting the system. The permissions and ownership for a file

or directory can be determined by passing the -l option to the 'ls' command.
For example:

```
% ls -l
drwxr-xr-x     2    jesse    users    4096    Feb     8 09:32 bin
drwxrwxrwx    10    jesse    users    4096    Feb    10 12:04 pub
-rw-r--r--     1    jesse    users      45    Feb     4 03:55 testfile
-rwx------     1    jesse    users      93    Feb     5 06:20 myprogram
-rw-rw-rw-     1    jesse    users     112    Feb     5 06:20 README
%
```

The first character column in the first output field states the file type,
where 'd' is a directory and '-' is a regular file. The next nine columns
specify the permissions (see paragraph below) of the element. The second field
indicates the number of files associated with the element, the third field
indicates the owner, the fourth field indicates the group that the file is
associated with, the fifth field indicates the size of the element in bytes,
the sixth, seventh and eighth fields indicate the time at which the file was
last modified and the ninth field is the name of the element.

As stated, the last nine columns in the first field indicate the permissions
of the element. These columns are grouped into threes, the first grouping
indicating the permissions for the owner of the element ('jesse' in this
case), the second grouping indicating the permissions for the group associated
with the element, and the third grouping indicating the permissions associated
with the rest of the world. The 'r', 'w', and 'x' indicate read, write and
execute permissions, respectively, for each of these associations. For
example, user 'jesse' has read and write permissions for 'testfile', users in
the group 'users' have read permission only, and the rest of the world also
has read permissions only. However, for the file 'myprogram', user 'jesse' has
read, write and execute permissions (suggesting that 'myprogram' is a program
that can be executed), while the group 'users' and the rest of the world have
no permissions (suggesting that the owner doesn't want anyone else to run his
program). The permissions, ownership and group associated with an element can
be changed with the commands 'chmod', 'chown' and 'chgrp', respectively. If a
user with the appropriate permissions wanted to change the user/group
ownership of 'README' from jesse/users to joe/admin, he would do the
following:

```
    # chown joe README
    # chgrp admin README
```

The syntax for chmod is slightly more complicated and has several variations.

The most concise way of setting the permissions for a single element uses a triplet of numbers, one for each of user, group and world. The value for each number in the triplet corresponds to a combination of read, write and execute permissions. Execute only is represented as 1, write only is represented as 2, and read only is represented as 4. Combinations of these permissions are represented as sums of the individual permissions. Read and execute is represented as 5, where as read, write and execute is represented as 7. No permissions is represented as 0. Thus, to give the owner read, write and execute permissions, the group read and execute permissions and the world no permissions, a user would do as follows:

```
    % chmod 750 myprogram
```

H4. THE SHELL

The shell provides an interface between the user and the operating system. It is the job of the shell to interpret the input that the user gives at the command prompt and call upon the system to do something in response. There are several different shells available, each with somewhat different syntax and capabilities. The two most common flavors of shells used on FreeBSD stem from the Bourne shell ('sh') and the C-shell ('csh') Different users have preferences and biases towards one shell or the other, and some certainly make it easier (or at least more intuitive) to do some things than others. You can determine your current shell by printing the value of the 'SHELL' environment variable from the command prompt with

```
    % echo $SHELL
```

You can start a new shell simply by entering the name of the shell from the command prompt:

```
    % csh
```

or

```
    % sh
```

and you can run a program from within a specific shell by preceding the name of the executable with the name of the shell in which it will be run:

```
    % sh myprogram
```

The user's default shell at login is determined by whoever set up his account. While there are many syntactic differences between shells, perhaps the one that is encountered most frequently is the way in which environment variables are set.

H5. SETTING ENVIRONMENT VARIABLES

Every session has associated with it environment variables, which consist of name/value pairs and control the way in which the shell and programs run from the shell behave. An example of an environment variable is the 'PATH' variable, which tells the shell which directories to search when trying to locate an executable file that the user has entered at the command line. If you are certain that a command exists, but the shell complains that it cannot be found when you try to execute it, there is likely a problem with the 'PATH' variable. Environment variables are set differently depending on the shell being used. For the Bourne shell ('sh'), it is done as:

```
    % export MYVARIABLE="avalue"
```

for the C-shell, it is done as:

```
    % setenv MYVARIABLE "avalue"
```

In both cases the quotation marks are only necessary if the value contains spaces. The 'echo' command can be used to examine the value of an environment variable:

```
    % echo $MYVARIABLE
```

Commands to set environment variables can also include references to other environment variables (prepended with the "$" character), including themselves. In order to add the path '/usr/local/bin' to the beginning of the search path, and the current directory '.' to the end of the search path, a user would enter

```
    % export PATH=/usr/local/bin:$PATH:.
```

in the Bourne shell, and

```
    % setenv PATH /usr/local/bin:${PATH}:.
```

in C-shell. Note the curly braces are required to protect the variable name in
C-shell.

H6. EDITING TEXT FILES

There are several text editors available for the FreeBSD operating system.
Some of these editors require the X window system, while others are designed
to operate in a console or terminal. It is generally a good thing to be
competent with a terminal-based text editor, as there are times when the files
necessary for X to run are the ones that must be edited. Three popular editors
are 'vi', 'pico' and 'emacs', each of which can be started from the command
line, optionally supplying the name of a file to be edited. 'vi' is arguably
the most ubiquitous as well as the least intuitive of the three. 'pico' is
relatively straightforward for a new user, though not as often installed on
systems. If you don't have 'pico', you may have a similar editor called
'nano'. 'emacs' is highly extensible and fairly widely available, but can be
somewhat unwieldy in a non-X environment. The newer versions each come with
online help, and offline help can be found in the manual and info pages for
each (see the section on FreeBSD Manual and Info pages). Many programs use the
'EDITOR' environment variable to determine which text editor to start when
editing is required.

H7. ROOT USER

Upon installation, almost all distributions set up the default administrative
user with the username 'root'. There are many things on the system that only
'root' (or a similarly privileged user) can do, one of which is installing the
NVIDIA FreeBSD Driver. WE MUST EMPHASIZE THAT ASSUMING THE IDENTITY OF 'root'
IS INHERENTLY RISKY AND AS 'root' IT IS RELATIVELY EASY TO CORRUPT YOUR SYSTEM
OR OTHERWISE RENDER IT UNUSABLE. There are three ways to become 'root'. You
may log in as 'root' as you would any other user, you may use the switch user
command ('su') at the command prompt, or, on some systems, use the 'sudo'
utility, which allows users to run programs as 'root' while keeping a log of
their actions. This last method is useful in case a user inadvertently causes
damage to the system and cannot remember what he has done (or prefers not to
admit what he has done). It is generally a good practice to remain 'root' only
as long as is necessary to accomplish the task requiring 'root' privileges
(another useful feature of the 'sudo' utility).

H8. FREEBSD MANUAL AND INFO PAGES

System manual or info pages are usually installed during installation. These pages are typically up-to-date and generally contain a comprehensive listing of the use of programs and utilities on the system. Also, many programs include the --help option, which usually prints a list of common options for that program. To view the manual page for a command, enter

```
    % man commandname
```

at the command prompt, where commandname refers to the command in which you are interested. Similarly, entering

```
    % info commandname
```

will bring up the info page for the command. Depending on the application, one or the other may be more up-to-date. The interface for the info system is interactive and navigable. If you are unable to locate the man page for the command you are interested in, you may need to add additional elements to your 'MANPATH' environment variable. See the section on environment variables.