NVIDIA

This article covers the official **NVIDIA (https://www.nvidia.com)** graphics card drivers. For the community open-source driver, see **Nouveau**. If you have a laptop with hybrid graphics, see also **NVIDIA Optimus**.

Related articles

- NVIDIA/Tips and tricks
- NVIDIA/Troubleshooting
- Nouveau
- NVIDIA Optimus
- PRIME
- Bumblebee
- nvidia-xrun
- Xorg
- Vulkan

1 Installation

Warning

Avoid installing the NVIDIA driver through the package provided from the NVIDIA website. Installation through **pacman** allows upgrading the driver together with the rest of the system.

Note

When dual booting on a system with **hybrid graphics**, enabling Windows or third-party apps *Eco mode* (like **ASUS Eco mode (https://www.asus.com/support/faq/1043747/#a14)**) may fully disable the NVIDIA discrete GPU, making it undetectable.

First, find the family of your card (e.g. NV110, NVC0, etc.) on **nouveau wiki's code names page (https://nouveau.freedesktop.org/CodeNames.html)** corresponding to its model/official name obtained with:

```
$ lspci -k -d ::03xx
```

Then, install the appropriate driver for your card:

GPU family	Driver	Status
Turing (NV160/TUXXX) (https://nouveau.freedesktop.org/CodeNames.html#NV160) and newer	nvidia-open (https://archlinux.org/packages/?name=nvidia-open) for linux (https://archlinux.org/packages/?name=linux) nvidia-open-lts (https://archlinux.org/packages/?name=nvidia-open-lts) for linux-lts (https://archlinux.org/packages/?name=linux-lts) nvidia-open-dkms (https://archlinux.org/packages/?name=nvidia-open-dkms) for any kernel(s)	Recommended by upstream (https://developer.nvidia.com/blog/nvidia-transitions-fully-towards-open-source-gpu-kernel-modules/) Current, supported[1] Possible power management issue on Turing[2]
Maxwell (NV110/GMXXX) (https://nouveau.freedesktop.org/CodeNames.html#NV110) through Ada Lovelace (NV190/ADXXX) (https://nouveau.freedesktop.org/CodeNames.html#NV190)	nvidia (https://archlinux.org/packages/?name=nvidia) for linux (https://archlinux.org/packages/?name=linux) nvidia-lts (https://archlinux.org/packages/?name=nvidia-lts) for linux-lts (https://archlinux.org/packages/?name=linux-lts) nvidia-dkms (https://archlinux.org/packages/?name=nvidia-dkms) for any kernel(s)	Current, supported[1]
Kepler (NVE0/GKXXX) (https://nouveau.freedesktop.org/CodeNames.html#NVE0)	nvidia-470xx-dkms (https://aur.archlinux.org/packages/nvidia-470xx-dkms/)[AUR]	Legacy, unsupported[3,4]
Fermi (NVC0/GF1XX) (https://nouveau.freedesktop.org/CodeNames.html#NVC0)	nvidia-390xx-dkms (https://aur.archlinux.org/packages/nvidia-390xx-dkms/)[AUR]	
Tesla (NV50/G80-90-GT2XX) (https://nouveau.freedesktop.org/CodeNames.html#NV50)	nvidia-340xx-dkms (https://aur.archlinux.org/packages/nvidia-340xx-dkms/)[AUR]	
Curie (NV40/G70) (https://nouveau.freedesktop.org/CodeNames.html#NV40) and older	No longer packaged	

1. If these packages do not work, it is usually due to new hardware releases. nvidia-open-beta (https://aur.archlinux.org/packages/nvidia-open-beta/)[AUR] may have a newer driver version that offers support.
2. NVIDIA's open kernel modules cannot enable D3 Power Management on Turing. This reduces battery life on notebooks with Turing in an NVIDIA Optimus configuration. Use the proprietary module (e.g. nvidia (https://archlinux.org/packages/?name=nvidia)) with module parameter `NVreg_EnableGpuFirmware=0` instead. More information about this issue (https://github.com/NVIDIA/open-gpu-kernel-modules/issues/640#issuecomment-2188114679).
3. May not function correctly on Linux 5.18 (or later) on systems with Intel CPUs 11th Gen and newer (https://web.archive.org/web/20250209200337/https://www.intel.com/content/www/us/en/newsroom/opinion/intel-cet-answers-call-protect-common-malware-threats.html) due an incompatibility with Indirect Branch Tracking (https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generati

on-intel-core-processors-datasheet-volume-1-of-2/007/indirect-branch-tracking/). You can disable it by setting the `ibt=off` kernel parameter from the boot loader. Be aware, this security feature is responsible for mitigating a class of exploit techniques (https://lwn.net/Articles/889475/).

4. NVIDIA no longer actively supports these cards and their drivers may not officially support the current Xorg version (https://nvidia.custhelp.com/app/answers/detail/a_id/3142/). It might be easier to use the nouveau driver; however, NVIDIA's legacy drivers are still available and might provide better 3D performance/stability.

> **Note**
>
> - When installing `dkms` (https://archlinux.org/packages/?name=dkms), read Dynamic Kernel Module Support#Installation.
> - The DKMS variants are not tied to a specific kernel, as they recompile the NVIDIA kernel module for each kernel for which header files are installed.

For 32-bit application support, also install the corresponding *lib32* package from the multilib repository (e.g. `lib32-nvidia-utils` (https://archlinux.org/packages/?name=lib32-nvidia-utils)).

The `nvidia-utils` (https://archlinux.org/packages/?name=nvidia-utils) package contains a file which blacklists the `nouveau` module once you reboot. Optionally, you can also remove `kms` from the `HOOKS` array in `/etc/mkinitcpio.conf` and regenerate the initramfs. This will prevent the initramfs from containing the `nouveau` module making sure the kernel cannot load it during early boot.

> **Note**
>
> If you are using Wayland you should not restart until after following #DRM kernel mode setting or you may end up with a black screen.

Once the driver has been installed, continue to #Xorg configuration or #Wayland configuration.

1.1 Custom kernel

Ensure your kernel has `CONFIG_DRM_SIMPLEDRM=y` , and if using `CONFIG_DEBUG_INFO_BTF` then this is needed in the PKGBUILD (since kernel 5.16):

```
install -Dt "$builddir/tools/bpf/resolve_btfids" tools/bpf/resolve_btfids/resolve_btfids
```

If your kernel is compiled with `CONFIG_NOVA_CORE` enabled, you may need to prevent the new NVIDIA GPU driver Nova (https://docs.kernel.org/gpu/nova/index.html) from loading. `nvidia-utils` (https://archlinux.org/packages/?name=nvidia-utils) adds it to the blacklist by default. You can check this by running systemd-analyze. If you have installed a different version of the driver, you may need to blacklist the `nova_core` and `nova_drm` modules manually.

1.2 DRM kernel mode setting

Since NVIDIA does not support **automatic KMS late loading**, enabling DRM (**Direct Rendering Manager**) **kernel mode setting** is required to make Wayland compositors function properly.

Starting from `nvidia-utils (https://archlinux.org/packages/?name=nvidia-utils)` 560.35.03-5, DRM defaults to enabled.**[1] (https://gitlab.archlinux.org/archlinux/packaging/packages/nvidia-utils/-/commit/1b02daa2ccca6a69fa4355fb5a369c2115ec3e22)** For older drivers, set the `modeset=1` **kernel module parameter** for the `nvidia_drm` module.

To verify that DRM is actually enabled, execute the following:

```
# cat /sys/module/nvidia_drm/parameters/modeset
```

Which should now return `Y`, and not `N`.

> **Note**
>
> Kernels **officially supported by Arch** enable `simpledrm`, while NVIDIA driver requires `efifb` or `vesafb` when `nvidia_drm.fbdev` is disabled/unavailable (version < 545).

1.2.1 Early loading

For basic functionality, just adding the kernel parameter should suffice. If you want to ensure it is loaded as early as possible, or you are noticing startup issues (such as the `nvidia` kernel module being loaded after the **display manager**), you can add `nvidia`, `nvidia_modeset`, `nvidia_uvm` and `nvidia_drm` to the initramfs. See **Kernel module#Early module loading** to learn how to configure your initramfs generator. **mkinitcpio** users may also need to **regenerate the initramfs** image every time there is a `nvidia (https://archlinux.org/packages/?name=nvidia)` driver update. See **#pacman hook** to automate these steps.

1.2.1.1 pacman hook

> **Note**
>
> A custom pacman hook is only needed for packages that ship built kernel modules. `*-dkms` packages do not need it as their upgrades will automatically trigger a *mkinitcpio* run.

To avoid the possibility of forgetting to update **initramfs** after an NVIDIA driver upgrade, you may want to use a **pacman hook**:

/etc/pacman.d/hooks/nvidia.hook

```
[Trigger]
Operation=Install
Operation=Upgrade
Operation=Remove
Type=Package
# You can remove package(s) that don't apply to your config, e.g. if you only use nvidia-open you can remove
nvidia-lts as a Target
```

```
Target=nvidia
Target=nvidia-open
Target=nvidia-lts
# If running a different kernel, modify below to match
Target=linux

[Action]
Description=Updating NVIDIA module in initcpio
Depends=mkinitcpio
When=PostTransaction
NeedsTargets
Exec=/bin/sh -c 'while read -r trg; do case $trg in linux*) exit 0; esac; done; /usr/bin/mkinitcpio -P'
```

Note

The complication in the `Exec` line above is in order to avoid running *mkinitcpio* multiple times if both `nvidia` and `linux` get updated. In case this does not bother you, the `Target=linux` and `NeedsTargets` lines may be dropped, and the `Exec` line may be reduced to simply `Exec=/usr/bin/mkinitcpio -P`.

1.3 Hardware accelerated video decoding

Accelerated video decoding with VDPAU is supported on GeForce 8 series cards and newer. Accelerated video decoding with NVDEC is supported on Fermi (~400 series) cards and newer. See **Hardware video acceleration** for details.

1.4 Hardware accelerated video encoding with NVENC

NVENC requires the `nvidia_uvm` module and the creation of related device nodes under `/dev`.

The latest driver package provides a **udev rule** which creates device nodes automatically, so no further action is required.

If you are using an old driver (e.g. **nvidia-340xx-dkms (https://aur.archlinux.org/packages/nvidia-340xx-dkms/)**[AUR]), you need to create device nodes. Invoking the `nvidia-modprobe` utility automatically creates them. You can create `/etc/udev/rules.d/70-nvidia.rules` to run it automatically:

/etc/udev/rules.d/70-nvidia.rules

```
ACTION=="add", DEVPATH=="/bus/pci/drivers/nvidia", RUN+="/usr/bin/nvidia-modprobe -c 0 -u"
```

2 Wayland configuration

Regarding Xwayland take a look at **Wayland#Xwayland**.

For further configuration options, take a look at the wiki pages or documentation of the respective **compositor**.

Note

> Prior to driver version 555.xx, or when using a Wayland compositor that does not support Explicit Sync via the `linux-drm-syncobj-v1` protocol, the NVIDIA driver can have major issues manifesting as flickering, out of order frames, and more, in both native Wayland and Xwayland applications.

2.1 Basic support

There are two kernel parameters for the `nvidia_drm` module to be considered: `modeset` and `fbdev`. Both are **enabled by default (https://gitlab.archlinux.org/archlinux/packaging/packages/nvidia-utils/-/blob/3b439109/PKGBUILD#L60)** when using the **`nvidia-utils` (https://archlinux.org/packages/?name=nvidia-utils)** package. NVIDIA also **plans to enable them by default in a future release (https://indico.freedesktop.org/event/6/contributions/287/attachments/210/288/NVIDIA%20Wayland%20Roadmap.pdf)**.

2.1.1 modeset

Enabling `modeset` is necessary for all Wayland configurations to function properly.

For unsupported drivers, where `modeset` needs to be enabled manually, see **#DRM kernel mode setting**, and **Wayland#Requirements** for more information.

2.1.2 fbdev

Enabling `fbdev` is necessary for some Wayland configurations.

It is specifically a hard requirement on Linux 6.11 and later, but it is currently unclear whether this is intended behavior or a bug, see **[2] (https://forums.developer.nvidia.com/t/drm-fbdev-wayland-presentation-support-with-linux-kernel-6-11-and-above/307920)** for more details.

It can be set the same way as the **modesetting parameter**, with the difference that executing:

```
# cat /sys/module/nvidia_drm/parameters/fbdev
```

Will return a missing file error if it is not set at all, instead of `N`.

2.2 Suspend support

Wayland suspend can suffer from the defaults more than X does, see **/Tips and tricks#Preserve video memory after suspend** for details.

If you use GDM, also see **GDM#Wayland and the proprietary NVIDIA driver**.

2.3 nvidia-application-profiles-rc.d

Some wayland compositors will consume a large quantity of VRAM by default if the **GLVidHeapReuseRatio (https://www.nvidia.com/en-us/drivers/details/237587/)** application profile key is not **applied against their process name (https://github.com/NVIDIA/egl-wayland/issues/126#issuecomment-2379945259)**. For example, **niri** users can free up to ~2.5GiB of idle vram consumption with the following:

/etc/nvidia/nvidia-application-profiles-rc.d/50-limit-free-buffer-pool-in-wayland-compositors.json

```
{
    "rules": [
        {
            "pattern": {
                "feature": "procname",
                "matches": "niri"
            },
            "profile": "Limit free buffer pool on Wayland compositors"
        }
    ],
    "profiles": [
        {
            "name": "Limit free buffer pool on Wayland compositors",
            "settings": [
                {
                    "key": "GLVidHeapReuseRatio",
                    "value": 0
                }
            ]
        }
    ]
}
```

3 Xorg configuration

The proprietary NVIDIA graphics card driver does not need any Xorg server configuration file. You can **start X** to see if the Xorg server will function correctly without a configuration file. However, it may be required to create a configuration file (prefer `/etc/X11/xorg.conf.d/20-nvidia.conf` over `/etc/X11/xorg.conf`) in order to adjust various settings. This configuration can be generated by the NVIDIA Xorg configuration tool, or it can be created manually. If created manually, it can be a minimal configuration (in the sense that it will only pass the basic options to the **Xorg** server), or it can include a number of settings that can bypass Xorg's auto-discovered or pre-configured options.

> **Tip**
>
> For more configuration options, see **NVIDIA/Troubleshooting**.

3.1 Automatic configuration

The NVIDIA package includes an automatic configuration tool to create an Xorg server configuration file (`xorg.conf`) and can be run by:

```
# nvidia-xconfig
```

This command will auto-detect and create (or edit, if already present) the `/etc/X11/xorg.conf` configuration according to present hardware.

Double-check your `/etc/X11/xorg.conf` to make sure your default depth, horizontal sync, vertical refresh, and resolutions are acceptable.

3.2 nvidia-settings

The `nvidia-settings` (https://archlinux.org/packages/?name=nvidia-settings) tool lets you configure many options using either CLI or GUI. Running `nvidia-settings` without any options launches the GUI, for CLI options see nvidia-settings(1) (https://man.archlinux.org/man/nvidia-settings.1).

You can run the CLI/GUI as a non-root user and save the settings to `~/.nvidia-settings-rc` by using the option *Save Current Configuration* under *nvidia-settings Configuration* tab.

To load the `~/.nvidia-settings-rc` for the current user:

```
$ nvidia-settings --load-config-only
```

See **Autostarting** to start this command on every boot.

> **Note**
>
> **Xorg** may not start or crash on startup after saving `nvidia-settings` changes. Adjusting or deleting the generated `~/.nvidia-settings-rc` and/or **Xorg** file(s) should recover normal startup.

3.3 Manual configuration

Several tweaks (which cannot be enabled **automatically** or with **nvidia-settings**) can be performed by editing your configuration file. The Xorg server will need to be restarted before any changes are applied.

See **NVIDIA Accelerated Linux Graphics Driver README and Installation Guide (https://download.nvidia.com/XFree86/Linux-x86_64/575.64/README/)** for additional details and options.

3.3.1 Minimal configuration

A basic configuration block in `20-nvidia.conf` (or deprecated in `xorg.conf`) would look like this:

/etc/X11/xorg.conf.d/20-nvidia.conf

```
Section "Device"
        Identifier "NVIDIA Card"
        Driver "nvidia"
        VendorName "NVIDIA Corporation"
        BoardName "GeForce GTX 1050 Ti"
EndSection
```

3.3.2 Disabling the logo on startup

If you are using an old driver (`nvidia-340xx-dkms` (https://aur.archlinux.org/packages/nvidia-340xx-dkms/)[AUR]), you may want to disable the NVIDIA logo splash screen that is displayed at X startup. Add the `"NoLogo"` option under section `Device`:

```
Option "NoLogo" "1"
```

3.3.3 Overriding monitor detection

The `"ConnectedMonitor"` option under section `Device` allows overriding monitor detection when X server starts, which may save a significant amount of time at start up. The available options are: `"CRT"` for analog connections, `"DFP"` for digital monitors and `"TV"` for televisions.

The following statement forces the NVIDIA driver to bypass startup checks and recognize the monitor as DFP:

```
Option "ConnectedMonitor" "DFP"
```

> **Note**
>
> Use "CRT" for all analog 15 pin VGA connections, even if the display is a flat panel. "DFP" is intended for DVI, HDMI, or DisplayPort digital connections only.

3.3.4 Enabling brightness control

Add to kernel parameters:

```
nvidia.NVreg_RegistryDwords=EnableBrightnessControl=1
```

Alternatively, add the following under section `Device` :

```
Option "RegistryDwords" "EnableBrightnessControl=1"
```

If brightness control still does not work with this option, try installing **nvidia-bl-dkms (https://aur.archlinux.org/packages/nvidia-bl-dkms/)**[AUR].

> **Note**
>
> Installing **nvidia-bl-dkms (https://aur.archlinux.org/packages/nvidia-bl-dkms/)**[AUR] will provide a `/sys/class/backlight/nvidia_backlight/` interface to backlight brightness control, but your system may continue to issue backlight control changes on `/sys/class/backlight/acpi_video0/` . One solution in this case is to watch for changes on, e.g. `acpi_video0/brightness` with *inotifywait* and to translate and write to `nvidia_backlight/brightness` accordingly. See **Backlight#sysfs modified but no brightness change**.

3.3.5 Enabling SLI

> **Warning**
>
> Since the GTX 10xx Series (1080, 1070, 1060, etc) only 2-way SLI is supported. 3-way and 4-way SLI may work for CUDA/OpenCL applications, but will most likely break all OpenGL applications.

Taken from the NVIDIA driver's **README (https://download.nvidia.com/XFree86/Linux-x86_64/575.64/README/xconfigoptions.html#SLI)** Appendix B: *This option controls the configuration of SLI rendering in supported configurations.* A "supported configuration" is a computer equipped with an SLI-Certified Motherboard and 2 or 3 SLI-Certified GeForce GPUs.

Find the first GPU's PCI Bus ID using `lspci`:

```
# lspci -d ::03xx
```

```
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor Graphics Control
ler (rev 09)
03:00.0 VGA compatible controller: NVIDIA Corporation GK107 [GeForce GTX 650] (rev a1)
04:00.0 VGA compatible controller: NVIDIA Corporation GK107 [GeForce GTX 650] (rev a1)
08:00.0 3D controller: NVIDIA Corporation GM108GLM [Quadro K620M / Quadro M500M] (rev a2)
```

Add the BusID (3 in the previous example) under section `Device`:

```
BusID "PCI:3:0:0"
```

Note

The format is important. The BusID value must be specified as `"PCI:<BusID>:0:0"`

Add the desired SLI rendering mode value under section `Screen`:

```
Option "SLI" "AA"
```

The following table presents the available rendering modes.

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.

Alternatively, you can use the *nvidia-xconfig* utility to insert these changes into `xorg.conf` with a single command:

```
# nvidia-xconfig --busid=PCI:3:0:0 --sli=AA
```

To verify that SLI mode is enabled from a shell:

```
$ nvidia-settings -q all | grep SLIMode
```

```
  Attribute 'SLIMode' (arch:0.0): AA
    'SLIMode' is a string attribute.
    'SLIMode' is a read-only attribute.
    'SLIMode' can use the following target types: X Screen.
```

Warning

After enabling SLI, your system may become frozen/non-responsive upon starting xorg. It is advisable that you disable your display manager before restarting.

If this configuration does not work, you may need to use the PCI Bus ID provided by `nvidia-settings`,

```
$ nvidia-settings -q all | grep -i pcibus
```

```
Attribute 'PCIBus' (host:0[gpu:0]): 101.
  'PCIBus' is an integer attribute.
  'PCIBus' is a read-only attribute.
  'PCIBus' can use the following target types: GPU, SDI Input Device.
Attribute 'PCIBus' (host:0[gpu:1]): 23.
  'PCIBus' is an integer attribute.
  'PCIBus' is a read-only attribute.
  'PCIBus' can use the following target types: GPU, SDI Input Device.
```

and comment out the PrimaryGPU option in your xorg.d configuration,

```
/usr/share/X11/xorg.conf.d/10-nvidia-drm-outputclass.conf
```

```
...

Section "OutputClass"
...
    # Option "PrimaryGPU" "yes"
...
```

Using this configuration may also solve any graphical boot issues.

3.4 Multiple monitors

See **Multihead** for more general information.

3.4.1 Using nvidia-settings

The **nvidia-settings** tool can configure multiple monitors.

For CLI configuration, first get the `CurrentMetaMode` by running:

```
$ nvidia-settings -q CurrentMetaMode
```

```
Attribute 'CurrentMetaMode' (hostnmae:0.0): id=50, switchable=no, source=nv-control :: DPY-1: 2880x1620 @288
0x1620 +0+0 {ViewPortIn=2880x1620, ViewPortOut=2880x1620+0+0}
```

Save everything after the `::` to the end of the attribute (in this case: `DPY-1: 2880x1620 @2880x1620 +0+0 {ViewPortIn=2880x1620, ViewPortOut=2880x1620+0+0}`) and use to reconfigure your displays with `nvidia-settings --assign "CurrentMetaMode=*your_meta_mode*"`.

> **Tip**
>
> You can create shell aliases for the different monitor and resolution configurations you use.

3.4.2 ConnectedMonitor

If the driver does not properly detect a second monitor, you can force it to do so with ConnectedMonitor.

/etc/X11/xorg.conf

```
Section "Monitor"
    Identifier     "Monitor1"
    VendorName     "Panasonic"
    ModelName      "Panasonic MICRON 2100Ex"
    HorizSync       30.0 - 121.0 # this monitor has incorrect EDID, hence Option "UseEDIDFreqs" "false"
    VertRefresh     50.0 - 160.0
    Option         "DPMS"
EndSection

Section "Monitor"
    Identifier     "Monitor2"
    VendorName     "Gateway"
    ModelName      "GatewayVX1120"
    HorizSync       30.0 - 121.0
    VertRefresh     50.0 - 160.0
    Option         "DPMS"
EndSection

Section "Device"
    Identifier     "Device1"
    Driver         "nvidia"
    Option         "NoLogo"
    Option         "UseEDIDFreqs" "false"
    Option         "ConnectedMonitor" "CRT,CRT"
    VendorName     "NVIDIA Corporation"
    BoardName      "GeForce 6200 LE"
    BusID          "PCI:3:0:0"
    Screen          0
EndSection

Section "Device"
    Identifier     "Device2"
    Driver         "nvidia"
    Option         "NoLogo"
    Option         "UseEDIDFreqs" "false"
    Option         "ConnectedMonitor" "CRT,CRT"
    VendorName     "NVIDIA Corporation"
    BoardName      "GeForce 6200 LE"
    BusID          "PCI:3:0:0"
    Screen          1
EndSection
```

The duplicated device with `Screen` is how you get X to use two monitors on one card without `TwinView`. Note that `nvidia-settings` will strip out any `ConnectedMonitor` options you have added.

3.4.3 TwinView

You want only one big screen instead of two. Set the `TwinView` argument to `1`. This option should be used if you desire compositing. TwinView only works on a per-card basis, when all participating monitors are connected to the same card.

```
Option "TwinView" "1"
```

Example configuration:

/etc/X11/xorg.conf.d/10-monitor.conf

```
Section "ServerLayout"
    Identifier     "TwinLayout"
    Screen         0 "metaScreen" 0 0
EndSection

Section "Monitor"
    Identifier     "Monitor0"
    Option         "Enable" "true"
EndSection

Section "Monitor"
    Identifier     "Monitor1"
    Option         "Enable" "true"
EndSection

Section "Device"
    Identifier     "Card0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"

    #refer to the link below for more information on each of the following options.
    Option         "HorizSync"          "DFP-0: 28-33; DFP-1: 28-33"
    Option         "VertRefresh"        "DFP-0: 43-73; DFP-1: 43-73"
    Option         "MetaModes"          "1920x1080, 1920x1080"
    Option         "ConnectedMonitor"   "DFP-0, DFP-1"
    Option         "MetaModeOrientation" "DFP-1 LeftOf DFP-0"
EndSection

Section "Screen"
    Identifier     "metaScreen"
    Device         "Card0"
    Monitor        "Monitor0"
    DefaultDepth    24
    Option         "TwinView" "True"
    SubSection "Display"
        Modes          "1920x1080"
    EndSubSection
EndSection
```

Device option information (https://download.nvidia.com/XFree86/Linux-x86_64/575.64/README/configtwinview.html).

If you have multiple cards that are SLI capable, it is possible to run more than one monitor attached to separate cards (for example: two cards in SLI with one monitor attached to each). The "MetaModes" option in conjunction with SLI Mosaic mode enables this. Below is a configuration which works for the aforementioned example and runs **GNOME** flawlessly.

/etc/X11/xorg.conf.d/10-monitor.conf

```
Section "Device"
        Identifier      "Card A"
```

```
        Driver          "nvidia"
        BusID           "PCI:1:00:0"
EndSection

Section "Device"
        Identifier      "Card B"
        Driver          "nvidia"
        BusID           "PCI:2:00:0"
EndSection

Section "Monitor"
        Identifier      "Right Monitor"
EndSection

Section "Monitor"
        Identifier      "Left Monitor"
EndSection

Section "Screen"
        Identifier      "Right Screen"
        Device          "Card A"
        Monitor         "Right Monitor"
        DefaultDepth    24
        Option          "SLI" "Mosaic"
        Option          "Stereo" "0"
        Option          "BaseMosaic" "True"
        Option          "MetaModes" "GPU-0.DFP-0: 1920x1200+4480+0, GPU-1.DFP-0:1920x1200+0+0"
        SubSection      "Display"
                        Depth           24
        EndSubSection
EndSection

Section "Screen"
        Identifier      "Left Screen"
        Device          "Card B"
        Monitor         "Left Monitor"
        DefaultDepth    24
        Option          "SLI" "Mosaic"
        Option          "Stereo" "0"
        Option          "BaseMosaic" "True"
        Option          "MetaModes" "GPU-0.DFP-0: 1920x1200+4480+0, GPU-1.DFP-0:1920x1200+0+0"
        SubSection      "Display"
                        Depth           24
        EndSubSection
EndSection

Section "ServerLayout"
        Identifier      "Default"
        Screen 0        "Right Screen" 0 0
        Option          "Xinerama" "0"
EndSection
```

3.4.3.1 Vertical sync using TwinView

If you are using TwinView and vertical sync (the *Sync to VBlank* option in `nvidia-settings`), you will notice that only one screen is being properly synced, unless you have two identical monitors. Although `nvidia-settings` does offer an option to change which screen is being synced (the *Sync to this display device* option), this does not always work. A solution is to add the following environment variables at startup, for example append in `/etc/profile`:

```
export __GL_SYNC_TO_VBLANK=1
export __GL_SYNC_DISPLAY_DEVICE=DFP-0
export VDPAU_NVIDIA_SYNC_DISPLAY_DEVICE=DFP-0
```

You can change `DFP-0` with your preferred screen (`DFP-0` is the DVI port and `CRT-0` is the VGA port). You can find the identifier for your display from `nvidia-settings` in the *X Server XVideoSettings* section.

3.4.3.2 Gaming using TwinView

In case you want to play full-screen games when using TwinView, you will notice that games recognize the two screens as being one big screen. While this is technically correct (the virtual X screen really is the size of your screens combined), you probably do not want to play on both screens at the same time.

To correct this behavior for SDL 1.2, try:

```
export SDL_VIDEO_FULLSCREEN_HEAD=1
```

For OpenGL, add the appropriate Metamodes to your xorg.conf in section `Device` and restart X:

```
Option "Metamodes" "1680x1050,1680x1050; 1280x1024,1280x1024; 1680x1050,NULL; 1280x1024,NULL;"
```

Another method that may either work alone or in conjunction with those mentioned above is **starting games in a separate X server**.

3.4.4 Mosaic mode

Mosaic mode is the only way to use more than 2 monitors across multiple graphics cards with compositing. Your window manager may or may not recognize the distinction between each monitor. Mosaic mode requires a valid SLI configuration. Even if using Base mode without SLI, the GPUs must still be SLI capable/compatible.

3.4.4.1 Base Mosaic

Base Mosaic mode works on any set of Geforce 8000 series or higher GPUs. It cannot be enabled from within the nvidia-setting GUI. You must either use the *nvidia-xconfig* command line program or edit `xorg.conf` by hand. Metamodes must be specified. The following is an example for four DFPs in a 2x2 configuration, each running at 1920x1024, with two DFPs connected to two cards:

```
# 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"
```

> **Note**
>
> While the documentation lists a 2x2 configuration of monitors, **GeForce cards are artificially limited to 3 monitors (https://forums.developer.nvidia.com/t/basemosaic-v295-vs-v310-vs-v325-only-up-to-three-screens/30583#3954733)** in Base Mosaic mode. Quadro cards support more than 3 monitors. As of September 2014, the Windows driver has dropped this artificial restriction, but it remains in the Linux driver.

3.4.4.2 SLI Mosaic

If you have an SLI configuration and each GPU is a Quadro FX 5800, Quadro Fermi or newer, then you can use SLI Mosaic mode. It can be enabled from within the nvidia-settings GUI or from the command line with:

```
# 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"
```

4 NVswitch

For systems with NVswitch, like H100x8 on AWS, the following is need.

- install nvidia-fabricmanager
- install matching kernel module needed by the fabric manager

With the fabricmanager, pytorch would report no GPU is found.

To install the fabric manager:

1. download the tarball from nvidia. **here (https://developer.download.nvidia.com/compute/cuda/redist/fabricmanager/linux-x86_64/)**
2. version 555.42.02 works well
3. modify the install script in sbin/fm_run_package_installer.sh to fix the installed file path

To get the matching kernel driver:

1. git clone the AUR for nvidia-beta-dkms and nvidia-utils-beta
2. change the PKGBUILD to use version 555.42.02
3. build and install them
4. reboot

finally, `systemctl enable nvidia-fabricmanager` and `systemctl start nvidia-fabricmanager` , then pytorch should work.

5 Tips and tricks

See **NVIDIA/Tips and tricks**.

6 Troubleshooting

See **NVIDIA/Troubleshooting**.

7 See also

- **Current graphics driver releases in official NVIDIA Forum (https://forums.developer.nvidia.com/t/current-graphics-driver-releases/28500)**

- **NVIDIA Developers Forum - Linux Subforum (https://forums.developer.nvidia.com/c/gpu-graphics/linux/148)**

Retrieved from "https://wiki.archlinux.org/index.php?title=NVIDIA&oldid=846358"