Configuring accelerated graphics cards in Linux

CARD TRICKS

Some graphics card vendors offer limited support for open source systems like Ubuntu. If you're looking for high-end effects, you might need to get involved. **BY KLAUS KNOPPER**

Linux probably has the best overall hardware support because of its wide application on a variety of devices. However, some hardware vendors make a strategic choice to avoid revealing their specifications – including example code and technical documentation for writing drivers – to any third parties except those bound by non-disclosure agreements. This secrecy makes it very difficult to write open source drivers for such hardware. These vendors argue that this policy ensures a market advantage; yet, other vendors who provide open specifications and good support for Linux compete well, often gaining a reputation for well-supported hardware that is easy to use and widely deployed on a range of electronic devices. Philosophical and political discussions about proprietary vs. open source business models are endless, so I'll just say I'm strongly biased in favor of open technical specifications and open source, even as I write this article about installing proprietary drivers for graphics cards through the use of general Linux techniques.

If you're troubleshooting a graphics card in Ubuntu, you might also be interested in this month's "Answer Man" column (pg. 46), which includes an answer on installing an nVidia graphics card.

Graphics Under GNU/Linux

Most GNU/Linux distributions for desktop and notebook computers use the Xorg graphics subsystem, which is an open source implementation of the X11 API used by almost all programs that have graphical elements. Xorg handles communications with graphical programs, and it also interfaces with hardware drivers to make access to one or more graphics cards stable in a multiuser/multitasking environment, providing a wide range of configuration options: transparency, hardware-based shading and rendering,

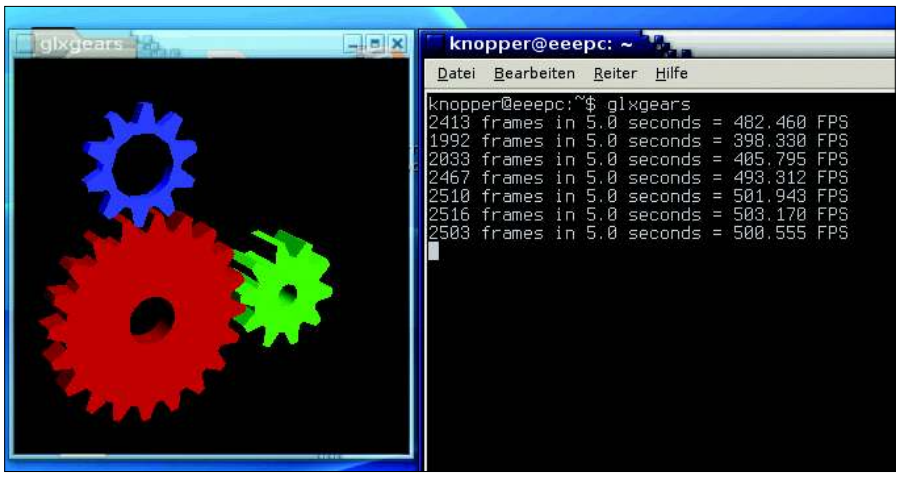

Figure 1: Glxgears measures the drawing speed of the graphics system.

support for multiple monitors, and direct communication between the program and the hardware using a kernel-provided direct hardware access feature.

This direct and unfiltered access to hardware through Xorg explains why careful programming of the hardware-dependent components, especially the driver modules for various chipsets, is crucial for stability of the entire system. Locking up the graphics card can result in an unusable system or even a total freeze of the operating system. Open Source chipset drivers that come with Xorg are therefore carefully reviewed for potential problems by developers, and every now and then, updates are released that provide new features or improve performance.

Acceleration

Many modern graphics cards provide onboard processing capabilities for graphics-intensive applications such as 3D computer games. Instead of letting the CPU perform the calculations and transfer pixels between memory locations, the graphics card does this independent of the main system with the use of special command sets. This feature is often called "accelerated graphics."

Each chipset has its own command set and access method for acceleration. For cards for which vendors provide sufficient information, accelerated open source drivers are available at Xorg, such as some of the older ATI Radeon cards [1], as well as many Intel-based chipsets (especially those that are present in many notebooks and netbooks).

The degree of available acceleration varies because not every feature of every chipset is fully documented. But a good hint for Compiz Fusion, the well-known 3D window manager that features transparent windows, magnification, and fancy desktop effects, is the output of the *glxinfo* utility (Listing 1), which you can find by installing the mesa-utils package.

In Listing 1, the line with *Direct Rendering: Yes* is the most significant hint that the card is supported in accelerated mode.

Some *GLX_ARB...* extensions listed in the full output are needed by Compiz, so direct rendering (DRI) [2] is not sufficient in all cases. However, with DRI, chances are good that games that use OpenGL/Mesa will run fine on this card (e.g., tuxracer, neverball, and egoboo).

For measuring the drawing speed of your system, *glxgears* is a nice tool that's also present in mesa-utils (Figure 1). Once you start *glxgears*, it shows an animation sequence of gears while it tries to refresh the animation and count the frame rate reached in each five-second interval.

An ATI Radeon 9800 card on a Pentium II running at a 16-bit 1280x1024 resolution reaches about 1,100 frames per second in *glxgears* with the open source *radeon* Xorg driver module [3], whereas on an EeePC 701 (900MHz, Intel 915GM graphics chipset), the *intel* Xorg driver reaches at least 500 frames per second.

Composite and 3D Support

Composite is the "transparency" extension that lets the graphics system overlap multiple layers, creating an "invisible" appearance (Figure 2).

Both the ATI Radeon 9800 and Intel cards I tested ran fine with Compiz, once the necessary AIGLX and Composite extensions were configured in the Xorg configuration file */etc/X11/xorg.conf*.

Listing 2 shows some of the most common options for different graphics cards needed to get Compiz (and other programs that use composite effects) to work properly. Not all of them are necessary for each and every card; some of these options are now even the defaults on newer Xorg versions and don't have to be present anymore (although their presence usually doesn't hurt).

After adding these changes, you have to restart the X server, which will terminate any currently running local graphical desktop sessions and most likely show the login screen again. Before killing your X server with Ctrl + Alt + Backspace (without knowing whether your new configuration will work), you can test the changes by starting a second server on a different console.

As root, type

```
X :1
```

in a terminal window, which will start a new Xorg session on the next free console – you should see a gray grid and the X-shaped mouse cursor. If that works,

Listing 1: *glxinfo* Output

```
01 name of display: :0.0
02 display: :0  screen: 0
03 direct rendering: Yes
04 server glx vendor string: SGI
05 server glx version string: 1.2
06 server glx extensions:
07     GLX_ARB_multisample, GLX_EXT_import_context, GLX_EXT_texture_from_pixmap,
08     GLX_EXT_visual_info, GLX_EXT_visual_rating, GLX_MESA_copy_sub_buffer,
09     GLX_OML_swap_method, GLX_SGI_make_current_read, GLX_SGI_swap_control,
10     GLX_SGIS_multisample, GLX_SGIX_fbconfig, GLX_SGIX_visual_select_group
```

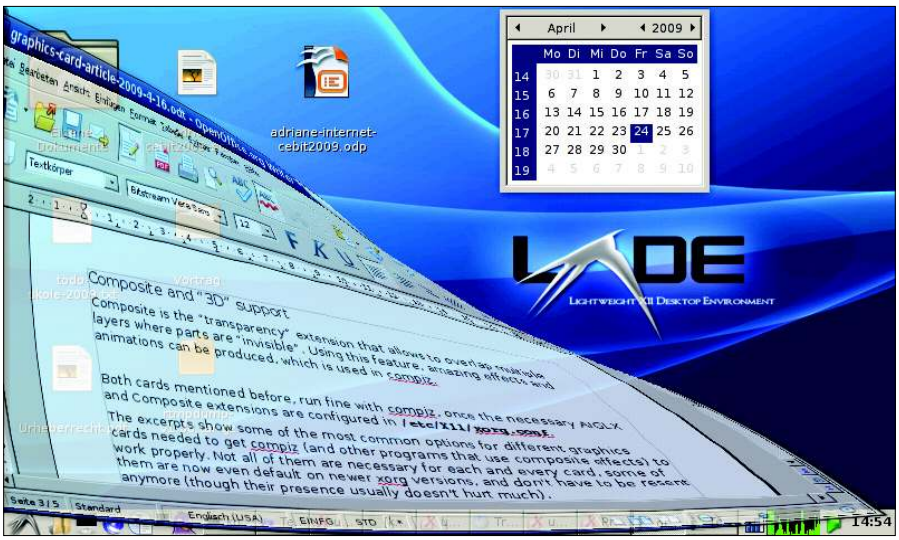

Figure 2: The transparency extension produces some interesting effects.

chances are good that your configuration is OK. With Ctrl + Alt + Backspace, you can kill the new X server and do the same (for restarting the X login process) on your first X session; better yet, logout and kill X in the login screen.

A more Unix-like way to achieve the same effect is to go to runlevel 2 or 3 by typing

```
init 2
```

(which is a text console–only runlevel in most distros) and restart the graphical session with:

```
init 5
```

Also, you need to do this when configuring a different chipset driver.

Choosing a Different Chipset Driver

If *glxinfo* reports *Direct Rendering: No*, the wrong chipset driver might be configured, so check */etc/X11/xorg.conf* in the *Driver* subsection:

```
Section "Device"
    ...
       Driver "fbdev"
EndSection
```

The framebuffer driver for Xorg, *fbdev*, is the one that always works if everything else fails, as long as framebuffer mode is enabled in your Linux kernel and possibly activated at startup.

To enable framebuffer mode, add *option vga = 891* to the kernel boot options, which will give you a 1024x768, 16-bit framebuffer resolution that will be used by the *fbdev* Xorg driver. On startup, you will see the Linux penguin logo with a glowing border if *highcolor* framebuffer mode is on. Although this mode works well with most graphics cards, it is never accelerated, and the previously mentioned options for Compiz just won't work.

Try changing the *Driver* option in the *Device* section of *xorg.conf* to the best matching chipset driver, like *ati*, *intel*, or *mga*, corresponding to what the *lspci* command shows as the vendor for the graphics chipset.

For nVidia cards, *nv* would be the free driver, but currently that one will not work in accelerated mode, and it could cause awful effects like stripes or artifacts on the screen when the *Composite* option is enabled. Sometimes these effects seem to stay forever, unless you completely disconnect the computer from the power supply and wait for a while, which seems to reset the internal memory state of nVidia cards.

If you leave out the *Driver* option, Xorg tries to autodetect the correct chipset on startup, and you get a pretty verbose protocol of Xorg initialization in */var/log/Xorg.0.log*, which also tells you about supported compared with found chipsets.

If you have a relatively new ATI or nVidia card, the free drivers that come with Xorg might not give you the best performance or support for all resolutions. Some cards run only in generic *vesa* mode (driver *vesa* in the *Device* section), which is very slow but still better than nothing.

If you want to run these cards with support for all their features, your only option might be to install proprietary drivers for Linux, downloaded directly from the hardware vendor's homepage. Policies and licenses for these drivers make it hard if not impossible to directly integrate them into Linux, and the work and risk of properly installing them falls on the user.

Installing nVidia Proprietary Drivers

Before starting the installer, make sure no X server is running, which you can do by switching to runlevel 2 or 3 in most distributions. Just switching to the text console with Ctrl + Alt + F1 is not sufficient because Xorg running on a different console is still accessing the graphics hardware. First you need to stop all login managers (xdm, gdm, kdm) that might keep Xorg running, then you can log in on the console. Installing the nVidia driver with its supplied script requires root permissions, even though not all parts of the configuration process have to run as root.

The nVidia installer is a self-unpacking and self-installing archive that you can get from the nVidia support homepage [4]. If you don't know the exact name of the card, the *lspci* command might help. If the card is very new, and it just appears as an *unknown/unlisted graphics card*, you might look at the Xorg log file in */var/log/Xorg.log* to find out what it is.

After download, you should have a file called *NVIDIA-Linux-x86- … .run*, which you can make executable and run (as root) with:

```
chmod a+x NVIDIA*.run
./NVIDIA*.run
```

The installer is self-explanatory. nVidia's driver installation consists of a kernel module, which has to exactly match the installed kernel version and binary objects for Xorg and library functions.

The kernel module is the most breakable part of the installation. For some kernels, pre-compiled modules are present in the installer, but chances are that they won't match your installation, so the installer will try to compile a module

for your installed kernel during the installation. For this procedure to succeed, a working C compiler and development tools like the *binutils* package must be present on your system, and the kernel sources matching your installed kernel version must also be installed. If this is not the case, the installation will fail. If the necessary tools are available, a new nVidia kernel module is compiled and loaded. The installer might also add a configuration option to */etc/modules* to make sure the module is also loaded at the next startup.

A new *nvidia* Xorg driver is added to Xorg's driver module directory and configured in */etc/X11/xorg.conf*. Check for the *Driver* option in the *Device* section to see if the installer has actually activated the new driver. A backup of *xorg.conf* is created (it cannot hurt if you have made a backup on your own before running the installer). Although you cannot do much here, wait until installation is complete. nVidia provides a small utility called *nvidia-xconfig* for later reconfiguration, but the *xorg.conf* file entries for Compiz should also work with the proprietary driver (just not with the free *nv* driver).

After going back to runlevel 5 (init 5), or restarting the graphical login/session manager, the new driver module should be active. (To verify this, look for the nVidia logo shown shortly before your desktop session starts.) *glxinfo* should now show *Direct Rendering: Yes*.

Installing ATI/AMD Proprietary Drivers

Like the nVidia driver, the ATI/AMD driver requires a proprietary kernel module that matches your own kernel's version; however, the ATI installer has a graphical GUI for installation, so you won't need to shut down a running X session.

Start by downloading an installer matching your card from the AMD homepage; make it executable and run it from the command line as root:

```
chmod a+x ati-driver-➥
installer-*-x86.x86_64.run
./ati-driver-installer-*-➥
x86.x86_64.run
```

The graphical installer will perfom all the necessary steps. Kernel source code matching your running kernel and a working C compiler must be installed to build the kernel module.

(The ATI installer is also capable of building a Debian-based package such as those used by Ubuntu, which might be an easier way of installing the driver than the file-based method.) After installation, you can run the command *aticonfig* to reconfigure *xorg.conf*.

More Tuning

Depending on the kind of graphical applications you use, different options in */etc/X11/xorg.conf* might provide better performance, more colors, or sharper pictures. Some of the options affect all drivers, and others only work with a specific chipset driver. Options that are not known to a driver are ignored with a warning in newer Xorg versions, which is convenient because you don't have to be extra careful not to break your X server by mistyping an option.

Most options are described in detail in the online manual pages of Xorg or in Xorg's driver-specific pages. However, pay attention to the recommendations on configuring options and parameters because some graphics cards can break or overheat when "overclocked" beyond their specifications. It has been a long time since I tried a monitor with a badly chosen resolution. Most LCD displays should just cut off signals they cannot handle, but playing too much with Xorg acceleration settings might not gain enough in performance to justify the risk. Naturally, some good hints about performance tuning come from the game community, so look for useful tips and links at websites supported by Linux gamers [5]. ■

Listing 2: *xorg.conf* Options

```
01 Section "Device"
02   ...
03   Option "DamageEvents" "true"
04   Option "DisableGLXRootClipping" "true"
05   Option "XaaNoOffscreenPixmaps" "true"
06   Option "AllowGLXWithComposite" "true"
07   Option "TripleBuffer"   "true"
08 EndSection
09
10 Section "Extensions"
11      ?
12      # Please do NOT enable this with the "nv" driver, it may
13      # put your NVIDIA card in a confused state.
14      Option   "Composite"   "Enable"
15 EndSection
16
17 Section "Screen"
18     ...
19        Option "AddARGBGLXVisuals" "true"
20 EndSection
21
22 Section "ServerLayout"
23     ...
24        Option "AIGLX" "true"
25 EndSection
```

INFO

[1] Proprietary drivers for ATI/AMD cards for GNU/Linux: *http://ati.amd.com/support/drivers/linux/linux-radeon.html*

[2] Direct Rendering Infrastructure (DRI): *http://dri.freedesktop.org/wiki/*

[3] Xorg homepage: *http://www.x.org/*

[4] Proprietary drivers for nVidia cards for GNU/Linux: *http://www.nvidia.com/object/unix.html*

[5] linuX-gamers.net (with tips for Wine/Xorg configuration for various games): *http://www.linux-gamers.net/*