Nvidia Optimus

Some laptops come with what is known as “nVidia Optimus” technology. This is an nVidia and Intel hybrid graphical processing unit (GPU); it is *not* two separate GPUs in one laptop. It uses nVidia graphics for performance and Intel graphics for power saving during basic usage. Unfortunately, until the release of version 435 in 2019, the closed-source proprietary drivers have not offered a means for adjusting or switching between the two.

Historically, there has been an open-source project called Bumblebee that has aimed to fix this problem. Bumblebee has not been maintained for some time; however, it may still be of value for users with older hardware that is no longer supported by version 435 or greater of the proprietary driver. Moreover, as the official Optimus support in the proprietary driver requires xorg-server 1.20.7 or greater, Slackware Current is required as of the time of writing. Thus, we will be providing instructions for both the Bumblebee method and the newer, official method provided by the nVidia proprietary driver.

To avoid confusion, from here on we will be using nVidia to refer to the company and `nvidia` to refer to the closed source proprietary driver. New sentences that start with nVidia also refer to the company and `Nvidia` to the driver.

You should also note the terminal commands used on this page. The code lines that begin with `$` refers to commands to be run by a regular user, and `#` refers to commands to be run as root or a user with sufficient privileges.

Also please note the differences between Bumblebee and bumblebee. bumblebee is the daemon for bumblebee.

Note: The Intel Driver is Often Sufficient

You do *not* need to use Optimus via Bumblebee or the nvidia proprietary driver in order to use modern applications and desktop environments under X11. The Intel graphics often provide sufficient 2D and 3D acceleration for those programs to work smoothly without having to use the nVidia GPU. If you still wish to use Bumblebee to switch between the nVidia GPU and the Intel GPU, read on. We just wanted to clarify that Intel's 3-D acceleration has progressed quite far and that not all programs need to use the nVidia GPU in order to work.

Installing Bumblebee

The Automated Way

There is a new, automated way of installing Bumblebee thanks to Slacker: Ryan McQuen.

USE AT YOUR OWN RISK! If you have modified the SlackBuilds in any way, this script may not be for you.

You can run this as root:

```
curl
https://raw.githubusercontent.com/ryanpcmcquen/linuxTweaks/master/slackware/
crazybee.sh | sh
```

Alternatively you can download the script and run it as root on your system directly.

This script detects multilib, creates the necessary group, adds users and applies the necessary /etc/rc.d/rc.local entries and also comes with a stable option which can be used by passing this instead:

```
curl
https://raw.githubusercontent.com/ryanpcmcquen/linuxTweaks/master/slackware/
crazybee.sh | STABLE=yes sh
```

It also uses upgradepkg's --reinstall --install-new features for use after kernel upgrades.

The Manual Way

Getting the SlackBuilds

A fellow Slacker, jgeboski, originally provided SlackBuilds that mostly follow the SlackBuilds.org requirements for Bumblebee. Now, another Slacker has taken over where he has left off on his own github repo. Remember to check which directory you're in first, as they will be downloaded to that directory. Assuming you're ready, the SlackBuilds can simply be git cloned as such. Then we're going to change into the new directory and begin the build process.

```
$ git clone https://github.com/whitewolf1776/Bumblebee-SlackBuilds.git
$ cd Bumblebee-SlackBuilds
```

Or you can download the full tarball containing the SlackBuilds instead:

```
$ wget --content-disposition
https://github.com/whitewolf1776/Bumblebee-SlackBuilds/tarball/master
$ tar -xf whitewolf1776-Bumblebee-SlackBuilds-<changing hexadecimal>    #
Tab completion is helpful.
$ cd whitewolf1776-Bumblebee-SlackBuilds
```

There is a README file, but you probably won't be reading it as you're reading this instead, so we're basically going to post it here.

Getting the Source Files

The source files, as with all SlackBuilds, must be downloaded manually and placed into their respective directories. Amongst the repos are a SLACKBUILDS.TXT file that does give links for the source files as well as their MD5SUMs to make sure the files were not corrupt. You can use the provided script to download them and check them against their MD5SUM for corruption, like this:

```
./download.sh
```

Preparing the Environment

There are a few things you need to do before we begin. First off, we need to have a specific group of users who are going to be allowed access to Bumblebee technology. For the sake of simplicity, we'll call this group “bumblebee”. In addition to the new group, we need to add the users that are allowed to access bumblebee technology:

```
# groupadd bumblebee
# usermod -G bumblebee -a <USERNAME>
```

where <USERNAME> is the name of the user or users you would like to add.

Building and Installing Bumblebee's Core

1. Build and install: bbswitch (optional but highly recommended).

```
# cd bbswitch # You don't actually need to ''cd'' as root, but who wants to
''su'' all the time?
# ./bbswitch.SlackBuild
# upgradepkg --install-new bbswitch-*.t?z
```

2. Build and install: libbsd (required).

```
# cd ../libbsd
# ./libbsd.SlackBuild
# upgradepkg --install-new libbsd-*.t?z
```

3. Build and install: bumblebee (obviously required).

```
# cd ../bumblebee
# ./bumblebee.SlackBuild
# upgradepkg --install-new bumblebee-*.t?z
```

Building and Installing Other Dependencies

Some SlackBuilds also offer a COMPAT32 option if you're running a multilib system. In the following examples, we build it with and without 32-bit compatibility. Use the one you desire. Keep these in

mind when building the specified package.

Bumblebee can use either primusrun (requires a mesa rebuild in Slackware 14.0 and below) or bumblebee's own optirun. Take your pick which one you would like to use (or you can install both). Each one has their own advantages and disadvantages.

Primus

Primus, like optirun (instructions below) can be used to run your desired program via bumblebee. Primus also comes with a COMPAT32 option as well.

If you are using Slackware 14.0 or older you need to rebuild mesa. The new github repository does not offer a SlackBuild for mesa to support Slackware 14.0 or older. Therefore you must use Slackware 14.1 or newer, or rebuild mesa manually with a different script. One can choose to use the old repository and simply update the SlackBuild script appropriately, or add `--enable-shared-glapi` into a mesa.SlackBuild from the source of a newer Slackware version.

```
# # For the old repository, one would have done:
# cd ../mesa
# ./mesa.SlackBuild
```

mesa does not have a COMPAT32 option because it is an official Slackware package, but you can easily create the compat32 package after creating the original package by using:

```
# ./mesa-compat32.SlackBuild # Only in the old repository
```

1. Build and install: primus

```
# cd ../primus
# ./primus.SlackBuild
# upgradepkg --install-new primus-*.t?z
```

```
# COMPAT32=yes ./primus.SlackBuild
```

VirtualGL

VirtualGL provides support for optirun that also allows one to run programs via bumblebee. However, it is recommended that one uses primusrun for a few reasons.

Although VirtualGL (and its dependency libjpeg-turbo) are still supported by the Bumblebee project and maintained by their developers, the new github repo does not

include SlackBuild scripts for these packages. This section is for the old github repository that was once maintained by jgeboski. If you would like to use VirtualGL (e.g. you're in one of the rare cases where VirtualGL provides better performance than primus) then you can use your own personal SlackBuilds or the old ones (although the old SlackBuilds will be out-of-date).

1. Build and install: libjpeg-turbo (32-bit compatibility available)

```
# cd ../libjpeg-turbo
# ./libjpeg-turbo.SlackBuild                          # This does not build
with 32-bit compatibility.
# upgradepkg --install-new libjpeg-turbo-*.t?z
```

If you do wish to build with 32-bit compatibility, add `COMPAT32=yes` before executing the script as so:

```
# COMPAT32=yes ./libjpeg-turbo.SlackBuild  # Only for 32-bit compatible
binaries and libraries on an x86_64 based system.
```

libjpeg-turbo installs in /opt because otherwise it can overwrite some things from the original libjpeg.

2. Build and install: VirtualGL (32-bit compatibility available)

```
# cd ../VirtualGL
# ./VirtualGL.SlackBuild
# upgradepkg --install-new VirtualGL-*.t?z
```

I think you're starting to get the idea now...

```
# COMPAT32=yes ./VirtualGL.SlackBuild
```

The nvidia Proprietary Driver

If you want to use the nVidia proprietary drivers, you must not use nouveau, as the drivers interfere with each other. This can be prevented by removing nouveau, installing xf86-video-nouveau-blacklist from /extra, or blacklisting nouveau manually.

This part is entirely optional. Slackware 13.37 and above comes with xf86-video-nouveau, the open source nVidia graphics card drivers. If you are using this, you do not need the closed source proprietary nVidia drivers. However, if you wish to use the closed source drivers, then install the packages listed in this section before going on.

Some nvidia driver versions are incompatible with certain kernel versions and vice versa (some kernel versions are incompatible with some nvidia driver versions) for various reasons. A specific kernel configuration, a specific nvidia driver against a specific kernel version amongst other possibilities. Generally it is best to use the nvidia

driver package provided along with the kernel packages provided by your Slackware version. You can also follow the SlackBuilds.org guideline for manually updating a version of some software yourself.

If you are using Slackware 14.1 or older, you will need to install libvdpau. Newer Slackware versions now ship libvdpau.

```
# cd ../libvdpau
# ./libvdpau.Slackbuild
# upgradepkg --install-new libvdpau-*.t?z
```

1. Build and install: nvidia-bumblebee

```
# cd ../nvidia-bumblebee
# ./nvidia-bumblebee.SlackBuild
# upgradepkg --install-new nvidia-bumblebee-*.t?z
```

```
# COMPAT32=yes ./nvidia-bumblebee.SlackBuild # Only for 32-bit compatible
binaries and libraries on an x86_64 based system.
```

2. Build and install: nvidia-kernel

```
# cd ../nvidia-kernel
# ./nvidia-kernel.SlackBuild
# upgradepkg --install-new nvidia-kernel-*.t?z
```

Post-Installation

Excellent. Now we're ready to do some post-installation setup. The bumblebee package provided us with an rc.bumblebee script in /etc/rc.d where the other startup scripts are also located. Remember to make this script executable and, if you so desire, start it!

```
# chmod +x /etc/rc.d/rc.bumblebeed
# /etc/rc.d/rc.bumblebeed start
```

If you would like bumblebee to autostart with the system, you can add the following to /etc/rc.d/rc.local:

```
[ -x /etc/rc.d/rc.bumblebeed ] && /etc/rc.d/rc.bumblebeed start
```

You can even go a step further by having bumblebee stop with your system by adding the following lines to /etc/rc.d/rc.local_shutdown. You can create this file if it does not exist.

```
[ -x /etc/rc.d/rc.bumblebeed ] && /etc/rc.d/rc.bumblebeed stop
```

Done! Any program we want using the nVidia graphics card can be run with “optirun” or “primusrun”

like this:

```
$ optirun glxspheres
$ primusrun glxspheres
```

Pretty easy, right? Well not for all I suppose. If you chose to use VirtualGL (optirun), some programs will not work directly like this. Some require optirun to run bash first like this:

```
$ optirun bash
```

Now we can execute the program directly without adding “optirun” at the beginning:

```
$ wine ~/.wine/drive_c/Program_Files/Starcraft\ 2/Starcraft\ 2.exe
```

Editing the Configuration Files

A few general /etc/bumblee/bumblebee.conf configuration edits (likely suited for all):

```
# If you added your user account to a different user group than "bumblebee"
at the start, make sure to change the "ServerGroup=" line accordingly.
# Here we set it to "ImAllergicToBees":
ServerGroup=ImAllergicToBees

# Perhaps you don't like using Display :8 for X11. We can change that too,
in this example we set it to 3:
VirtualDisplay=:3

# I want to specify my driver that bumblebee will use by default. Here we
show nouveau, but you can easily replace it with "nvidia".
# For the rest of this small section, we'll be using nouveau for
simplification.
Driver=nouveau

# Don't forget to add their respective lines as well.
KernelDriver=nouveau
Module=nouveau
```

Official Optimus Support with the nVidia Proprietary Driver

Note

This will *not* work with Slackware 14.2 or any stable version of Slackware, as at the time of writing. Slackware Current is required.

Dependencies/Requirements

From main -current tree:

- `xorg-server >= 1.20.7`

Do not try this with older versions - it will *not* work. Certain specific git commits are required to have been added to xorg-server; see the official nVidia documentation linked to in the Sources section. These commits are only present in versions >= 1.20.7.

- `xf86-video-nouveau-blacklist >= 1.0 or xf86-video-nouveau >= “blacklist”`

Patrick Volkerding recently changed 'blacklist' from being the version number which would trample over the actual package containing the module, to 'blacklist' being part of the package-name with '1.0' being the version number. Either one is fine; just be aware of this.

The one with the package-name 'xf86-video-nouveau-blacklist' will coexist with xf86-video-nouveau, which is fine. The important thing is that `/etc/modprobe.d/BLACKLIST-nouveau.conf` be present on your system, which will ensure that the nouveau module is properly blacklisted.

From SlackBuilds.org:

- `nvidia-kernel >= 435.17`

Build normally. I got a gcc mismatch error as I am using a kernel that is quite old. I was able to override it successfully by adding the environment variable spouted out by the nvidia installer during the build process when it failed the first time. YMMV. For best results, build nvidia-kernel using the same version of gcc that was used to build the kernel itself.

- `nvidia-driver >= 435.17`

Be sure to build with `CURRENT=“yes”` and optionally with `COMPAT32=“yes”` in the custom build options of the SlackBuild. The nvidia-switch scripts are *not* required with -current as libglvnd is now included in mainline -current.

Next Steps

Once you have the packages listed above installed, I would recommend rebooting. Is this strictly necessary? Probably not, but in my experience when you start playing around with kernel modules involving graphics drivers, it is a best practice to reboot, because it is easy to *think* you have a working setup only to find that it didn't survive a reboot (and chances are, by then you will have forgotten what you did to make it work).

I would also recommend changing your runlevel to 3 if it is currently 4.

Once you have rebooted into runlevel 3, ensure that the `nvidia_drm` module has been successfully loaded:

```
$ lsmod |grep nvidia_drm
```

As root, add the following to `/etc/X11/xorg.conf.d/21-LAR-nvidia-screens.conf` (or any

filename you would like):

```
# cat << EOF > /etc/X11/xorg.conf.d/21-LAR-nvidia-screens.conf
Section "ServerLayout"
  Identifier "layout"
  Option "AllowNVIDIAGPUScreens"
  Screen 0 "iGPU"
EndSection

Section "Device"
  Identifier "iGPU"
  Driver "modesetting"
EndSection

Section "Screen"
  Identifier "iGPU"
  Device "iGPU"
EndSection

Section "Device"
  Identifier "nvidia"
  Driver "nvidia"
EndSection
EOF
```

Note that the official nVidia documentation seems to indicate you only need the following:

```
Section "ServerLayout"
  Identifier "layout"
  Option "AllowNVIDIAGPUScreens"
EndSection
```

However, for me this has *not* worked. YMMV. It is possible that, in the future, a more minimal configuration will in fact work. It is suggested that this be monitored at this time. As is often the case with Xorg configuration, the less that is manually configured, and the more that is left to be auto-configured, the better.

Next, launch your xserver.

If GPU screen creation was successful, the log file `/var/log/Xorg.0.log` should contain lines with `NVIDIA(G0)`, and querying the RandR providers with `xrandr –listproviders` should display a provider named `NVIDIA-G0` (for “NVIDIA GPU screen 0”).

If that is the case, the following command:

```
$ __NV_PRIME_RENDER_OFFLOAD=1 __GLX_VENDOR_LIBRARY_NAME=nvidia glxinfo |grep
'OpenGL vendor'
```

should display:

```
OpenGL vendor string: NVIDIA Corporation
```

and the following command:

```
$ glxinfo |grep 'OpenGL vendor'
```

should display your integrated GPU's vendor as opposed to NVIDIA (e.g., Intel).

You can then test using something like:

```
$ __NV_PRIME_RENDER_OFFLOAD=1 __GLX_VENDOR_LIBRARY_NAME=nvidia glxgears
```

You can now launch any program using your Nvidia dGPU by utilizing those two environment variables and prepending them to the program command you are running. I would refrain from using the GUI tools that come with nvidia-driver. I have no idea to what extent they will trample all over your /etc/ directory, and I have no idea if they actually work to get the above working, as this setup is still fairly bleeding-edge at this point.

As per usual, YMMV.

Known Issues

BrokenCog on IRC has reported that this setup may not work with external monitors.

Again, as indicated above, we are working as a community to find out what the abolute minimum manual xorg.conf settings are in order to just get Optimus working. As per usual with xorg settings nowadays, the less you can get away with manually specifying, the better. The xorg server works best when it can just do its own autoconfig to the largest extent possible, in my experience.

Do *not* try to run applications by setting `DRI_PRIME=1` by utilizing this setup. It may cause your applications to crash or (as was the case for me) to dead-lock your X server.

FAQ That Google Might not Answer

---

1. Q: I upgraded or downgraded my kernel and now I get “Fatal: module bbswitch not found” and bumblebee doesn't work! What do I do?

A: bbswitch and nvidia-kernel are kernel modules that place themselves in /lib/modules/<SOME KERNEL VERSION> so if you upgraded your kernel, just rebuild your nvidia-kernel and bbswitch packages so that the new package will place itself into the newer /lib/modules/<KERNEL VERSION>.

2. Q: If I upgrade nvidia-kernel, will I have to upgrade nvidia-bumblebee (or vice versa) too?

A: No. Although they both use the same source files, they install very different things. The nvidia-kernel is just the kernel-module, and nvidia-bumblebee is the rest of the application.

3. Q: (Extension to #2) What if it's a new version of the nvidia proprietary driver?

A: Then in that case, then you should probably upgrade both nvidia-bumblebee and nvidia-kernel.

4. Q: What is this multilib/compat32 stuff and do I need it?

A: This isn't bumblebee-specific and depends on what you want. Read the multilib page for more details and if you will need/want it or not.

Specific Laptop Models and nVidia GPU's

Bumblebee has mixed results depending on the nVidia graphics card and driver. Some are better than others. I hope that in this section, if you have a configuration suggestion, post it in the discussion section to help those with the same graphics cards on their nVidia optimus setups (and the configuration will later be added to the article). Some of these settings may work for all laptops, some may not. We're taking precautions just in case. Because we don't know what little differences there might be between video cards, don't forget to mention your laptop model and the driver xorg.conf you're using.

This one change is suitable for those running a multilib system.

- /etc/bumblebee/bumblebee.conf

```
# This is on a multilib system, so that is why /usr/lib64 AND /usr/lib are
being used in LibraryPath:
LibraryPath=/usr/lib64/nvidia-bumblebee:/usr/lib/nvidia-
bumblebee:/usr/lib64:/usr/lib
XorgModulePath=/usr/lib64/nvidia-bumblebee/xorg,/usr/lib64/xorg/modules
```

Sources

- Bumblebee Project: http://bumblebee-project.org/
- Bumblebee Project Wiki: https://github.com/Bumblebee-Project/Bumblebee/wiki
- For making this page possible: jgeboski
- Original SlackBuilds: https://github.com/jgeboski/Bumblebee-SlackBuilds
- New SlackBuilds: https://github.com/whitewolf1776/Bumblebee-SlackBuilds
- Originally written by TommyC
- Section on official Optimus support by the nvidia proprietary driver contributed by Poprocks, based partly on official documentation from nVidia: https://download.nvidia.com/XFree86/Linux-x86_64/450.80.02/README/primerenderoffload.html

howtos, software, hardware, nvidia, author tommyc