**vGPU\_Unlock Wiki**

Made possible by the community

Project created by DualCoder (Jonathan Johansson) [[GitHub](https://github.com/DualCoder/vgpu_unlock)]

Documentation written by Krutav Shah [[www.krutav.com](http://krutav.com)]

Join our Discord server [here](https://discord.gg/qh6dvPtxvb).

Edited by:

Julius Arkenberg,

Luclu7,

And more from the community.

Last updated: December 8, 2021

**Table of contents:**

1. Preface
2. Supported hardware and software
   1. Supported vGPU types
   2. Supported hardware
   3. Supported operating systems
3. How to get started with vGPU\_Unlock
   1. Setting up the hardware
   2. Obtaining drivers and licenses
   3. Preparing a linux system
      1. Downloading the vGPU\_Unlock scripts
      2. Red Hat Enterprise Linux (Recommended)
      3. Debian/Ubuntu
      4. Proxmox VE
   4. Creating a vGPU
      1. Using mdevctl normally with libvirt
      2. (Proxmox only) Using the Proxmox web interface
4. Setting up a licensing server
5. Setting up the guest VM
   1. Microsoft Windows
   2. Linux
6. Using the host graphics alongside vGPU (Single GPU systems)
   1. Installing the merged driver
   2. Using Looking Glass
7. Troubleshooting and FAQ
8. Limitations, known issues, and fixes
   1. High OpenGL instability
   2. Ampere support isn’t here yet
9. Alternatives to Nvidia vGPU

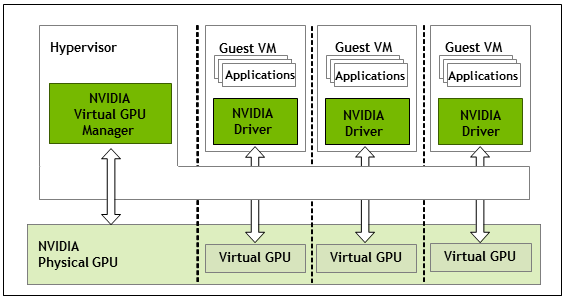
This project is provided to the end user with no warranty or support whatsoever. Nvidia vGPU and GRID are trademarks of NVIDIA Corporation. They reserve all rights to their products and trademarks. All other trademarks and products mentioned here are the property of their respective owners. This project does allow the bypassing of any expenses or costs associated with using the mentioned technologies in this document and is to be used at the risk of the end user, not the maintainers of the project. The project maintainers are also not liable for any damages to the end user, should a user ever incur one. Though extremely unlikely for any physical hardware damage to occur, the project maintainers and/or creators are not responsible in any way. This project has been made open source in an effort to keep all code and resources viewable by any users in order to help build a more informed decision as to whether one should use the tool or not. We are not providing this project for use in any malicious or harmful context. In addition, this tool shall NOT be used in any production environment.

1. Preface

Nvidia vGPU is a proprietary technology developed by Nvidia for use in the data center with certain Tesla and Quadro cards. The technology is aimed at the VDI (Virtual Desktop Infrastructure) market, with the intention of remote users using a remote desktop solution to carry out work. Different remote applications include CAD/CAM software, gaming (think: GeForce Now), architecture, and many other graphics accelerated programs.

The technology works by splitting a supported graphics card’s resources amongst multiple VMs (virtual machines). The advantage of this over traditional virtual machines is full graphics acceleration, which is not as feasible with CPU rasterization. Virtual Machines are popular in the datacenter and in VDI scenarios as they are easy to migrate and provision to employees and users. Remote users can connect to a central VDI server to access their own workspace on a “lighter” computer such as a laptop or thin client. Graphical workloads that require an elevated amount of 3D and/or compute power can’t always be run on ultrabooks and other “light” computers.

That would make sense for office and other enterprise environments where graphical power is needed remotely. But what does that mean for you? The goal of vGPU\_unlock is to permit a user to run Nvidia vGPU technology on inferior graphics cards or consumer variants to the professional datacenter graphics cards designed for vGPU. With this, users can virtualize their own graphics cards for a couple virtual machines. This could allow a user to run a virtual machine capable of gaming for friends, or use windows on linux with graphical acceleration (using Looking Glass) for example. There are certainly more uses out there, but these are just some of the ideal use cases.



How vGPU works, credit: NVIDIA

Do you need vGPU? Depends. First of all, it’s best to have a good understanding of using Linux and how this technology works before going forth and attempting the setup. If you have a need for graphically accelerated Windows or Linux virtual machines, you may consider vGPU for your environment. There are alternatives that exist for specific cases which we will cover later.

2. Supported hardware and Software

**A. Supported vGPU types:**

| **vGPU Type** | **OS** | **Use Case** | **License** | **Display** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| A-series | Windows, Linux | Virtual Applications | vApps | 1280x1024, 1 display | Good for RDSH |
| B-series | Windows, Linux | Basic PC work | vPC | Up to 5K, 2 displays | 45 FPS max |
| C-series | Linux | Compute server | vCS | Up to 4K, 1 display | CUDA Only |
| Q-series | Windows, Linux | ProfessionalWorkstation | vDWS | Up to 8K, 4 displays | CUDA, OpenGL |

Note: gaming series vGPU types exist with the appropriate drivers

**B. Supported hardware**

CPU and Motherboard:

Virtualization extensions required. For intel, this means VT-X. AMD-V is necessary for virtualization on AMD systems. To achieve adequate VM performance, virtualization extensions should be enabled. Please consult the vendor provided documentation to confirm that your CPU and motherboard have support for virtualization and for the steps to enable it in the BIOS.

Note that IOMMU may be necessary for vGPU to work properly on some systems. Ampere GPUs will require IOMMU to be enabled.

Graphics cards:

| **Nvidia vGPU cards** | **GPU Chip** | **vGPU unlock supported:** |
| --- | --- | --- |
| Tesla M10 | GM107 x4 | Most Maxwell 1.0 cards |
| Tesla M60 | GM204 x2 | Most Maxwell 2.0 cards |
| Tesla P40 | GP102 | Most Pascal cards |
| Tesla V100 16GB | GV100 | Titan V, Quadro GV100 |
| Quadro RTX 6000 | TU102 | Most Turing cards |
| RTX A6000 | GA102 | Ampere is not supported |

**C. Supported Operating Systems**

Host:

* Red Hat Enterprise Linux (certified by Nvidia for vGPU, tested kernel 4.18)
* Proxmox VE (Tested kernel 5.4)
* There may be more that we don’t explicitly list here.

For any host Linux kernels higher than 5.10, please use patches.

Linux kernel 5.13 does not appear to work and is not recommended.

Guest VM:

Enterprise Linux distributions (RHEL, CentOS, Fedora)

Debian/Ubuntu (20.04 LTS)

Windows 10, 8.1, Server 2019 and 2016

3. Getting started with using vGPU\_Unlock

**Disclaimer:**

Using vGPU on unsupported/uncertified hardware is not recommended, but this script will still enable certain cards to run the technology. We provide this to you at your own risk, so keep in mind that it may not work for you. The project is provided under the MIT license, with no warranty whatsoever. Community support for the project through online channels may be available, but is not guaranteed.

**A. Setting up the hardware**

PC hardware:

Start by installing a compatible graphics card into an available PCI Express slot on the system mainboard while the computer is off. This step will likely already be completed beforehand. You’ll also need to verify that your hardware is capable of running vGPU. Refer to section 2 for details on this. A system upgrade may be necessary if you do not have compatible hardware.

BIOS:

If your system hardware is ready for vGPU, turn on the system and head into the BIOS. You’ll need to look for two options, virtualization extensions (Intel VT-x/AMD-V, virtualization support) and IOMMU\* (Intel VT-D/AMD-Vi/SR-IOV). You’ll want to enable both of those features if possible, to use vGPU.

\*Note: IOMMU is not strictly required. It is only required for Ampere systems and/or something goes wrong without it.

For both of these processes, you should refer to documentation from your system vendor to get specific details on how to achieve the desired configuration and results for your system.

**B. Obtaining drivers and licensing**

You can apply for a 90 day evaluation of Nvidia vGPU on their application page for vGPU/GRID. You will be asked to fill out various information before submitting. From there, it can take anywhere from 2 minutes to 48 hours for the application to be processed. You will be sent an email to set a password and log in to the Nvidia Licensing Portal. Here is where you can find drivers for your system, license server installers, and your actual licenses.

Once you are in your licensing portal, download the latest release of the Linux KVM vGPU drivers. It will come in the form of a ZIP file with PDF guides and drivers for both the host and virtual machine inside it. Installation of drivers and licensing will be covered later.

**C. Setting up a Linux host**

Note: this section of the guide shows the setup of normal vGPU. The host graphics card cannot be used in this mode. Please refer to section 6 for this. Instructions for setting up a Linux based system for vGPU varies by the distribution. Nvidia certifies Red Hat Enterprise Linux KVM for use with their vGPU technology, but they also provide a Linux KVM package which we will be covering here. We will also show the steps right up from installation to the end in order to include as much as possible.

1. Downloading vGPU\_Unlock

This project and its files are available for download at <https://github.com/DualCoder/vgpu_unlock>. You can use git at the command line to clone it directly, or you can download the zip file from GitHub and use SFTP to transfer the files to your host. For the purpose of this guide, we will show you how to set up vGPU as root, with git CLI.

1. Become root if you aren’t already

sudo -i

1. Enter the /opt directory

cd /opt

1. Use `git` to clone the repository (Install git if it’s not installed)

git clone https://github.com/DualCoder/vgpu\_unlock

1. Make the newly downloaded folder executable

chmod -R +x vgpu\_unlock

1. Setting up an Enterprise Linux distribution (RHEL, CentOS)
2. Installing the operating system:

Red Hat Enterprise Linux is available for download with the free developer subscription. Fedora is not recommended.

1. Burn the ISO to a USB and install the operating system normally with the settings you would like. We highly recommend enabling the root user and using the “Server” installation along with the following packages: Virtualization Hypervisor, Development Tools, and Headless Management.
2. After installation, run the following commands.

sudo -i

dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm && update

dnf update

dnf --enablerepo="epel" install dkms kernel-devel python3 python3-pip mdevctl

pip3 install frida

1. Move over to the /opt/ directory where we will work.

cd /opt

1. Download the vGPU\_Unlock script

See section I of 3. C. which covers downloading the scripts.

1. Enable IOMMU and blacklist nouveau for vGPU, see:

[Red Hat 8 - Enabling IOMMU in GRUB](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/managing-virtual-devices_configuring-and-managing-virtualization#attaching-sr-iov-networking-devices-to-virtual-machines_managing-sr-iov-devices)

[Blacklisting Nouveau driver](https://access.redhat.com/solutions/1155663)

For Intel: intel\_iommu=on iommu=pt

For AMD: amd\_iommu=on iommu=pt

1. Install the vGPU driver with DKMS (Replace <version> with your driver version)

chmod +x NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run

./NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run --dkms

1. Edit the 4 nvidia files as outlined in the [GitHub](https://github.com/DualCoder/vgpu_unlock).

nano /lib/systemd/system/nvidia-vgpud.service

nano /lib/systemd/system/nvidia-vgpu-mgr.service

nano /usr/src/nvidia-<version>/nvidia/os-interface.c

nano /usr/src/nvidia-<version>/nvidia/nvidia.Kbuild

1. Remove and rebuild DKMS for the Nvidia driver (Replace <version> with your driver version)

dkms remove -m nvidia -v <version> --all

dkms install -m nvidia -v <version>

1. Reboot
2. Setting up a debian based system (e.g Ubuntu)
   * 1. Installing the operating system:

Get an ISO with kernel less than 5.10. Any newer versions like Debian Bullseye with 5.10 or newer will need patches for the vGPU drivers. [Here is one](https://github.com/rupansh/vgpu_unlock_5.12).

* + 1. Burn the ISO to a USB and install the operating system normally.
    2. After installation, run the following commands.

sudo -i

apt-get update && upgrade

apt-get install dkms python3 python3-pip

* + 1. Move over to the /opt/ directory where we will work.

cd /opt

* + 1. Download the vGPU\_Unlock script

See section I of 3. C. which covers downloading the scripts.

* + 1. Enable IOMMU and blacklist nouveau for vGPU, see:

[Red Hat 8 - Enabling IOMMU in GRUB](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/managing-virtual-devices_configuring-and-managing-virtualization#attaching-sr-iov-networking-devices-to-virtual-machines_managing-sr-iov-devices)

[Blacklisting Nouveau driver](https://access.redhat.com/solutions/1155663)

For Intel: intel\_iommu=on iommu=pt

For AMD: amd\_iommu=on iommu=pt

* + 1. Install the vGPU driver with DKMS (Replace <version> with your driver version)

chmod +x NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run

./NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run --dkms

* + 1. Edit these 4 nvidia files as outlined in the [GitHub](https://github.com/DualCoder/vgpu_unlock).

nano /lib/systemd/system/nvidia-vgpud.service

nano /lib/systemd/system/nvidia-vgpu-mgr.service

nano /usr/src/nvidia-<version>/nvidia/os-interface.c

nano /usr/src/nvidia-<version>/nvidia/nvidia.Kbuild

* + 1. Remove and rebuild DKMS for the Nvidia driver (Replace <version> with your driver version)

dkms remove -m nvidia -v <version> --all

dkms install -m nvidia -v <version>

* + 1. Reboot

1. Setting up a Proxmox VE system

Proxmox 5.4 (kernel 4) is recommended over PVE 6.x

1. See debian installation instructions

**D. Creating a vGPU**

Creating a vGPU instance varies by the system. Most distributions will come with libvirt as the de-facto virtualization library and tools. For these systems, mdevctl is the preferred way of managing mediated devices, which vGPU is classified as. However, distributions like Proxmox handle the management of mediated devices differently with their own interface.

I. Using mdevctl normally with libvirt

1. Start by finding out which vGPU you want to use by listing the available types and their features.

mdevctl types

[Red Hat Documentation - Obtaining vGPU Information](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/assembly_managing-gpu-devices-in-virtual-machines_configuring-and-managing-virtualization#proc_obtaining-nvidia-vgpu-information-about-your-system_assembly_managing-nvidia-vgpu-devices)

1. Create a vGPU instance. The guide for Red Hat systems applies to other distributions as well.

[Red Hat documentation - Creating vGPU instances](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/assembly_managing-gpu-devices-in-virtual-machines_configuring-and-managing-virtualization#proc_setting-up-nvidia-vgpu-devices_assembly_managing-nvidia-vgpu-devices)

1. Attach your vGPU to a VM by editing the VM’s XML configuration file. (You will need to generate a UUID first)

[Red Hat documentation - Attaching vGPU instances](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/assembly_managing-gpu-devices-in-virtual-machines_configuring-and-managing-virtualization#proc_assigning-a-gpu-to-a-virtual-machine_assembly_managing-gpu-devices-in-virtual-machines)

Here is a sample vGPU device. This block would be added to the <devices> section of the XML configuration of a VM.

<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>

<source>

<address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>

</source>

</hostdev>

II. Using the web interface on Proxmox

1. Install mdevctl if you want to easily list out the kind of vGPU types you want to use. List all of them with this command:

mdevctl types

[Red Hat Documentation - Obtaining vGPU Information](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/assembly_managing-gpu-devices-in-virtual-machines_configuring-and-managing-virtualization#proc_obtaining-nvidia-vgpu-information-about-your-system_assembly_managing-nvidia-vgpu-devices)

1. Give your virtual machines a UUID. This can be done by either generating a new UUID with uuidgen or using your VM ID with zeroes before it in UUID format.
2. Attach the newly created UUID to your VM config. Replace <VM-ID> with the numerical ID that your VM has.

nano /etc/pve/qemu-server/<VM-ID>.conf

Next, add the UUID (Replace <Your-UUID> with the UUID that you generated) to the top of the file:

args: -uuid <Your-UUID>

1. Add the vGPU device:

Go to the ‘Hardware’ tab of your VM, press ‘Add’ and select ‘PCI Device’. Click on ‘Device:’ and it will give you a list of PCI devices, but you will want to select the Nvidia graphics card. A new menu should appear next to it called ‘MDev Type’ where you can select the vGPU profile that you want to use.

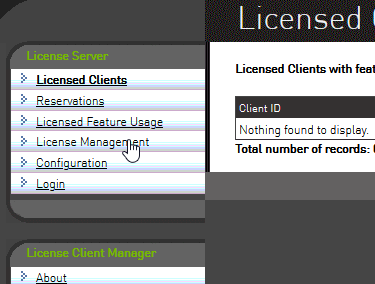
4. Setting up a licensing server

Licensing is essential to the proper function of vGPU technology. It is a licensed product and requires a valid license provided by either Nvidia or their partners to be used. To set up the server that hands out licenses to each of the vGPU instances, you’ll want to start by downloading the server itself. Available on the Nvidia Licensing Portal, you can go to the downloads tab and select the latest license server available to download. There are downloads for both Windows and Linux, so you can run the license server on either one.

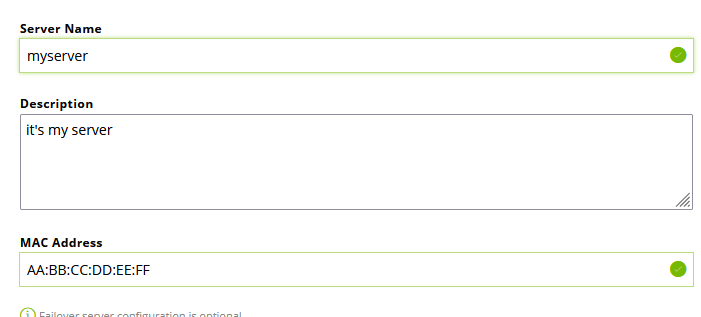
**Note: Nvidia provides official documentation and instruction for this that we recommend referring to.**

On Windows, setup is very straight-forward with the Nvidia installer UI. You will need Oracle Java software installed in order for the license server to install and function.

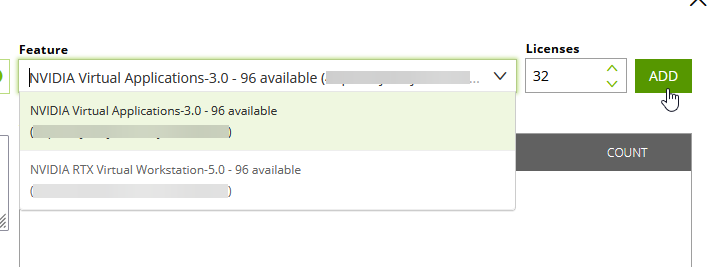
On Linux, you’ll need to install both Java and Tomcat web server first.

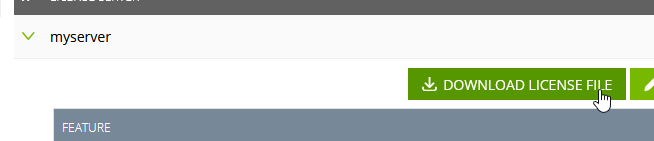
Make sure to install the license server with web access so that you can access it from a separate computer. Once the license server is installed and running, go to http://<serverIP>:8080/licserver and click on *License Management*.

You’ll need a license file (.bin). To get it, go back to the Nvidia Licensing Portal. Click on the *Create server* button and fill in details such as the name of your server, a description, and most importantly, the MAC address of the server it is running on. Finding the address of the network adapter in the system can be done using ipconfig /all on Windows and ip a on Linux.

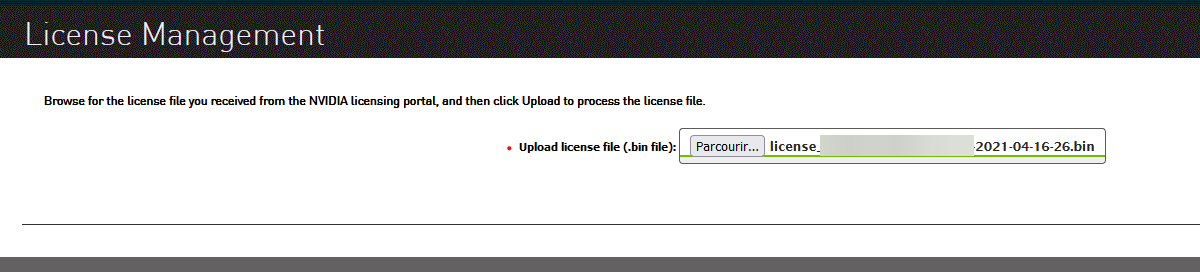


The name and description can be whatever you like. However, the MAC address is important for functionality and must be set to the same MAC address as your main network adapter.



You now need to add licenses themselves (*features*) to your server. There are different types of licenses, and you can add multiple types to one license server file. Different vGPU profiles require a specific license type to use, for example the A-series profiles need a vApps license.

After you create the license server on the portal, you can download the associated license (.bin) file for use on your server.



Head on to your licensing server web interface and go to the “License Management” section and upload the downloaded license file.

5. Setting up the guest VM

This section will be about installing drivers for Nvidia vGPU mediated devices in the virtual machines that they are assigned to. Assuming you have already completed installation of a supported operating system on your virtual machine, you can proceed with the installation with the device drivers for a vGPU. Keep in mind that drivers are only available for Microsoft Windows and Linux guests. BSD based operating systems and MacOS are not supported and aren’t likely to ever receive support either.

**A. Microsoft Windows**

Inside of the .ZIP file that the Nvidia vGPU driver and documentation PDF is included in, the drivers for both Windows 10/Server 2019/2016 and Windows 8/Server 2012 are included.

To install the drivers, simply run the corresponding executable on your virtual machine and follow the steps in the graphical installer to install the driver. Once it completes, you may or may not be asked to reboot.

On rare occasions, a prompt for reboot can be presented to a user which may mean that the driver might not work for some reason. A code 43 error is an example of such an error that can be caused by a configuration error or using the wrong drivers.

The next thing you must do is bring up the Nvidia control panel application by right clicking on the desktop and selecting it. You can then get it connected to your license server that you created. Find the licensing tab in the control panel menu and enter the IP address or hostname of the server, and enter the port number. 7070 is the default port number, unless you changed it, in which case you must specify the new port number.

**B. Linux**

There are a multitude of Linux distributions available, but Nvidia has only certified a few of them to be natively compatible with the vGPU drivers. Enterprise Linux and Debian based systems with particular kernel versions are most likely to work. Other distributions or newer versions with different and newer kernels may cause troubles during installation, and require a patch to fix as a result.

Before installing, you will need to blacklist nouveau drivers and rebuild the initramfs image. After that, you need to reboot.

[Red Hat Documentation - Blacklisting Nouveau drivers](https://access.redhat.com/solutions/1155663)

[Debian/Ubuntu - Blacklisting Nouveau drivers](https://linuxconfig.org/how-to-disable-blacklist-nouveau-nvidia-driver-on-ubuntu-20-04-focal-fossa-linux)

To install the driver, make the .run package included executable, and then install it normally. The installer will bring up a graphical interface in your command line that will be relatively easy to follow. Answer “yes” to the questions it presents you unless you have your own requirements.

chmod +x NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run

./NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run

From here, you can reboot the virtual machine and you should be able to use a graphics accelerated desktop environment. The next thing you must do is bring up the Nvidia Settings application and get it connected to your license server that you created. Find the licensing tab in the application and enter the IP address/hostname of the server, and enter the port number. 7070 is the default port number, unless you changed it, in which case you must specify the new port number.

6. Using the host graphics alongside vGPU

Many people using the vGPU\_Unlock script may only have one graphics card in their system, and would like to be able to use that system for more than a server to provide graphics accelerated virtual machines like a server. Turns out it is in fact possible to merge together the regular GeForce / Quadro driver with the vGPU binaries, and unlock vGPU functionality while still using your Linux host’s graphical desktop normally.

There are still limitations to this approach, such as uncertainty as to whether the host can use CUDA and OpenCL or not. The guest VM’s will still run normally, but may experience a loss in performance if there are tasks being actively carried out by the user on the host.

Creating this “merged driver” is a bit of an involved process, and it’s best to download a pre-made one rather than going through the trouble of doing it from scratch if you are a beginner.

[~~Google Drive link (Merged-driver 460.32.04)~~](https://drive.google.com/file/d/1y04jXsyWaObP5ed-EdEQ8_CHmgbcUUVD/view?usp=sharing)

[~~Google Drive link (Merged-driver 460.73.01) \*For manual unlock\*~~](https://drive.google.com/file/d/19HMkYV_Uu3bI-hxY1IxLy5ABXj0feLPg/view)

[Google Drive link (Merged-driver 460.73.01) \*Pre-unlocked and Recommended!\*](https://drive.google.com/file/d/1dCyUteA2MqJaemRKqqTu5oed5mINu9Bw/view?usp=sharing)

Note: The following steps can be skipped if you have the pre-unlocked version of the merged driver. You can install it normally, just like the original vGPU drivers.

**A. Installing the merged driver**

This driver may be missing files that are not necessary and may be showing errors at installation, but you can just disregard the errors and skip past them. This driver has only been tested working for kernel 4.18 up to 5.9 without patches. For newer kernels such as 5.12, see [this patch](https://github.com/rupansh/vgpu_unlock_5.12).

To use this merged driver, run the .RUN file just like in the instructions for the regular vGPU drivers outlined earlier. You may get some errors during installation, though they shouldn’t stop you from installing.

Add the environment variable Environment=”\_\_RM\_NO\_VERSION\_CHECK=1” to the files nvidia-vgpud.service and nvidia-vgpu-mgr.service on the line beneath ExecStop. These files are found in /lib/systemd/system/.

Without this environment variable, you will get an API mismatch error because the merged driver contains the geforce driver which has a slightly different driver version number than the vgpu-kvm driver.

**B. Using Looking Glass**

Looking Glass allows the use of a shared memory buffer between the virtual machine and the host to also carry the virtual machine’s frame buffer and display it on the host or another VM. This can be especially useful if you want to mirror the display of a vGPU VM on your host in a window. You may consider this to give your vGPUs a “physical display output” or to get a real-time view without external software and hardware encoding.

Setting up Looking Glass is the same with vGPU on Windows and Linux as it would be with a regular graphics card passed through to a VM. Therefore, we are linking the [official instructions here](https://looking-glass.io/wiki/Installation_on_other_distributions).

Note: Looking Glass with the NVFBC API and vGPU requires a 512MB IVSHMEM size. If you are using Looking Glass with DXGI, you can use a size according to their formula to calculate size based on monitor resolution.

7. Troubleshooting and FAQ

The occurrence of issues can be quite common, especially when using an external script that requires some setup to interface with something originally designed to work out of the box. The goal of this section will be to answer some common questions and how you can fix certain issues. More scenarios will be added here as they come up.

To find out what issue you are having, take the logs of the Nvidia vGPU service with journalctl -u nvidia-vgpu-mgr and scroll down to the end for the newest error. Other sources of error information include journalctl -u nvidia-vgpud and dmesg.

* **Q: What driver does this need?**

A: You need to use the proprietary Nvidia vGPU and GRID drivers, and the Linux KVM version to be specific. We have tested versions R440 to R460 working with this unlock tool.

* **Q: My system has multiple GPUs, can this tool unlock all of them?**

A: No, unless all of those GPUs are the same exact ones with the same device IDs.

* **Q: Can I use the host GPU display and vGPU at the same time?**

Nvidia vGPU was designed to run on a GPU with no display output, so only the vGPU technology could be used. However those with only one GPU may want to still use the host GPU, so in section 6 we cover the community made “merged-driver” that enables this.

* **Error: GPU not supported by vGPU at PCI Id:**

Make sure /path\_to\_vgpu\_unlock/ has been added to /lib/systemd/system/nvidia-vgpud.service and restart the service with systemctl restart nvidia-vgpud

* **Error: vGPU is supported only on VGX capable boards:**

Make sure /path\_to\_vgpu\_unlock/ has been added to /lib/systemd/system/nvidia-vgpu-mgr.service and restart the service: systemctl stop nvidia-vgpu-mgr; killall nvidia-vgpu-mgr; systemctl start nvidia-vgpu-mgr

* **Error: NVRM: API mismatch: the client has the version \_\_\_:**

Add Environment="\_\_RM\_NO\_VERSION\_CHECK=1" underneath the line ExecStopPost in both of the Nvidia vGPU service files under /lib/systemd/system/ and restart both Nvidia vGPU services.

8. Limitations, known issues, and fixes

There are certain issues and limitations that you may run into while using vGPU\_Unlock.

**A. High OpenGL instability**

OpenGL is a graphics API that many programs use to render their application on your screen. It is most commonly found in professional software.

The issue is the fact that OpenGL doesn’t seem to work with the Q-series vGPU profiles in versions of vGPU GRID newer than R440. A and B series vGPU profiles still have functioning OpenGL, but there are some instabilities here and there. There are ways to get OpenGL working on Q-series cards, though, and one is to install the R440 vGPU host driver and R440 GRID guest driver.

The other solution is to “spoof” the vGPU to a normal Quadro, such as a Quadro P4000 for example with the x-pci arguments for the mediated vGPU device in the virtual machine configuration. By doing this and using a driver below 450, we were able to achieve really good OpenGL performance, but lost access to CUDA and OpenCL compute technologies.

**B. Ampere graphics cards don’t work**

At the time of writing this article, Nvidia’s Ampere generation of GPUs is their newest and most advanced. Ampere-based Teslas and Quadros that support vGPU now use a function known as “SR-IOV” to provide a more hardware based approach to traditional vGPU that has better performance. The issue is that this is a hardware feature and is most likely disabled in the firmware VBIOS on non vGPU certified cards. This makes it increasingly harder to figure out how to get vGPU working on these Ampere consumer cards without some higher level modifications.

Currently, we are able to pass through a vGPU instance to a VM with an RTX 3090, but the VM will immediately crash and blue screen upon driver initialization of the vGPU. It’s unlikely that Ampere GPUs will be supported by vGPU unlock.

9. Alternatives to Nvidia vGPU

It’s worth noting that Nvidia vGPU may not work for everyone, especially if you have an older or unsupported graphics card. There are alternatives though, and here are some.

**A. Intel GVT-G**

This technology designed for Intel graphics processors allows them to be virtualized much like Nvidia vGPU. At the time of writing, graphics processors from the Intel Broadwell generation of CPUs to the Comet Lake generation are supported. Driver support is good, but the performance of these integrated GPUs is not that amusing.

**B. AMD MxGPU**

This technology designed for certain AMD professional graphics cards allows them to be virtualized much like Nvidia vGPU. At the time of writing, only a few cards are supported, such as the Radeon Instinct lineup, the GCN3.0 FirePro S7150s, and some datacenter Radeon Pros. This technology also makes use of SR-IOV, which can be beneficial in some areas. Obtaining these cards however, is difficult, with only the FirePro S7150s being easily attainable. This card does however suffer limitations such as a lack of hardware encoding and poor performance.

**C. Microsoft GPU-P**

Microsoft has developed a much better successor to the old RemoteFX graphics acceleration that it provided to users to give Hyper-V virtual machines graphics acceleration. This technology, known as GPU-P or GPU Partitioning / Para-virtualization allows for a virtual machine running on the Hyper-V hypervisor to access the host graphics card’s resources and gain high performance graphics acceleration. It can be used with many graphics cards and can even be overprovisioned. It is available for use with the Hyper-V hypervisor, which can be installed for free on Windows 10 and Windows Server. This is a solution we recommend, especially in Windows environments as the potential for resource and cost savings is huge.

**D. The “free” Nvidia vGPU**

Back when Nvidia was still producing Kepler GPUs, they released several Kepler based graphics cards including the GRID K1 and GRID K2 that are capable of running vGPU up to GRID 4. From there, support has been dropped and Nvidia no longer releases any drivers for this card. The EOL for this generation was back in 2020. This first generation graphics virtualization technology did not come with any licensing costs outside of the cost for the hypervisor, which is either Citrix XenServer or VMware ESXi in this case. The cards themselves are rather cheap now, but we can’t exactly recommend you go buy these and use them in this day and age after their EOL.