

```
---
title: 'Modern Graphics from Boot to Shutdown and Retiring fbdev'
author:
- Thomas Zimmermann <tzimmermann@suse.com>
abstract:
  In this paper we describe the introduction of simplifiedrm, a
  hardware-agnostic DRM graphics driver for the Linux kernel.
  We outline the booting process from a graphics perspective
  and illustrate key points and problems for a fully DRM-based
  graphics stack. We further report on the current and future
  state of simplifiedrm.
keywords:
- Linux
- kernel
- graphics
- driver
- DRM
- fbdev
- simplifiedrm
...
```

Introduction

Linux has a long and convoluted history of graphics systems. The two prevailing ones are the Framebuffer Device Interface, more commonly known as fbdev, and the Direct Rendering Manager, more commonly known as DRM.

The fbdev subsystem was added to the Linux kernel during the 1.3 release cycle to display output on systems without text-mode display. It grew to support a wide range of graphics hardware on various platforms. In contrast, DRM started during the kernel's 2.2 release cycle as a handful of small drivers that gave the X server access to 3D hardware's DMA capabilities. Only during the late 2.6 release cycle, DRM introduced the ability to perform full-featured modesetting, rendering, and memory management on any graphics hardware. Nevertheless DRM has slowly replaced fbdev as kernel's leading graphics subsystem over the last decade.

This change has mainly been motivated by fbdev's deficiencies and the desire for reducing technical debt throughout the graphics stack. Only corner cases, such as early-boot graphics output, remain a niche for fbdev. For DRM to take over the remaining uses, Linux needs a DRM graphics driver that is small enough to be linked into the kernel binary and hardware-agnostic enough to cover a wide range of graphics devices. In addition to the stated motivation, such a driver enables the use of modern graphical user environments on a lot of old graphics hardware with SVGA-like¹ capabilities; thereby reducing the user's acquisition

¹https://en.wikipedia.org/wiki/Super_VGA

cost and electrical waste.

Over the course of the last 12 to 18 months, we added *simplifiedrm*, a DRM driver that covers the use cases that were previously served by fbdev. We enabled *simplifiedrm* on a number of Linux distributions, including openSUSE Tumbleweed.

Going from fbdev to DRM

Implemented entirely within the kernel, the *Framebuffer Device Interface* (fbdev) provides graphics output on a variety of graphics devices and architectures. It is an abstraction of a simple graphics card's video memory and modesetting functionality. Features include

- good, but aging, hardware support,
- userspace interfaces for modesetting and page flipping,
- support for mmap'ed video memory,
- possibly accelerated bit-blitting operations, and
- a text console for systems without text mode.

To the user, the fbdev subsystem acts like a traditional text-mode terminal with additional graphics capabilities. But fbdev's conceptual simplicity also results in a number of shortcomings, which include

- the inability to directly configure more than one CRTC with a single connector per device,
- no 3D or video acceleration, and therefore
- no appropriate support for current hardware.

The design of the userspace interfaces and the use of mmap on raw video memory prevents shared access to graphics devices among concurrent programs. The interface's semantics also vary slightly among fbdev drivers. These shortcomings, among other reasons, have led to the development of the Direct Rendering Manager.

The *Direct Rendering Manager* (DRM) is a kernel-based multiplexer for the system's graphics resources. It provides userspace programs with interfaces for rendering and modesetting on the available graphics devices. To render graphics with DRM, the program creates one or more buffer objects and instructs the DRM graphics driver to draw into each buffer object's memory. In contrast to fbdev, DRM multiplexes access to the necessary hardware resources, such as graphics memory or GPU shaders. Any userspace program can perform these operations. Additional software libraries, such as Mesa, provide portable abstractions on top. It can be implemented in software, but is intended to be accelerated by hardware.

Separate from rendering, a privileged *DRM master*, such as a Wayland compositor or the X server, performs modesetting and page flipping. The DRM master is also typically the program that takes each program's rendered buffer objects and composes a final image to be displayed.

Over the past decade, the Linux graphics stack has migrated away from fbdev and towards DRM interfaces and drivers. There are no new fbdev drivers being accepted into the kernel and userspace programs, such as the X server, Weston or Plymouth, are slowly losing the ability to use the fbdev subsystem for graphics output. For those components that still do require fbdev, such as the kernel's graphics console, DRM provides an fbdev emulation layer.

The two remaining use cases for fbdev are the graphics output during the initial stages of booting the system and on systems without native DRM drivers. Until recently, DRM provided no out-of-the-box solution for these scenarios.

Bootting with DRM drivers requires them to be available on the boot partition. But given the rich feature set of most DRM drivers, adding them to the kernel or initramfs incurs a significant overhead on disk-storage consumption. The size of the DRM kernel drivers on openSUSE Tumbleweed's with kernel 5.19.2 is about 8.5 MiB on disk, while the overall initramfs and kernel binary are about 31 MiB and 13 MiB respectively on disk. Even if storage consumption is not a concern, most DRM drivers are specific to certain hardware. Systems without a compatible DRM driver would not offer graphics output at all.

Fbdev resolves these problems by providing small, hardware-agnostic drivers that can be linked into the kernel binary and therefore be loaded early in the boot process. But as userspace support for fbdev has been degrading, only the kernel's graphics console is reliably available.² If no hardware-native DRM driver is available or fails to load later on, the user is stuck on the command-line prompt without productivity applications or a modern web browser.

We set out to resolve this problem in a similar way as fbdev does: by adding hardware-agnostic graphics drivers to DRM. Being built on DRM interfaces, such drivers can serve modern desktop environments and enable basic productivity applications until a native driver becomes available.

Graphics Output from POST to Native Drivers

As part of the bootstrap process, the system's firmware initializes the graphics output to get at least some status messages displayed. The required information include the display resolution, color format and location of the scanout buffer.³ The exact details depend on the system, but a number of common patterns can be identified.

- On 32-bit x86, the system BIOS executes the VGA graphics card's BIOS code as part of its POST process. Because the VGA BIOS hides the

²The kernel's graphics console is problematic by itself. It is known to be slow, prone to locking bugs and buffer overflows.

³The *scanout buffer* is the memory area that contains the displayed data. The graphics output devices fetches the data and sends it to the display. We avoid the term *framebuffer* here, as it already refers to a type of data structure in the context of DRM.

hardware-specific details of the VGA graphics card,⁴ the system BIOS can set up a text-mode display independently. Later during the boot process, the GRUB boot loader provides the option to set the graphics output to a higher display resolution and color depth. It utilizes the VESA interface, which is provided by most graphics cards. As with VGA, the VESA interface's hardware details are implemented by the graphics card's BIOS, such that the bootloader does not need to know or care.

- On 64-bit x86-64, the BIOS has been replaced by UEFI firmware, but the same mechanisms are in place. The system's graphics firmware implements a UEFI graphics driver for its hardware, so that UEFI services can initialize and access the graphics output via hardware-agnostic interfaces. The interface provides similar functionality as its VESA counterpart on 32-bit x86. During the boot-loader stage, GRUB can immediately reuse the initialized graphics output.
- On most embedded systems, the firmware provides a list of available hardware components via DeviceTree. After initializing the system's graphics output, the firmware adds a node to the DeviceTree that describes the created output's properties. Boot loaders and operating systems can use the node's information to access the display.

With these mechanisms, the system firmware and boot loader provide hardware-agnostic graphics output and make it available to the Linux kernel. During the early stages of loading itself, the kernel reads the graphics output's configuration from the available sources, such as VESA, UEFI, or DeviceTree. While the configuration's exact source and format again depend on the system, the provided parameters always include the display resolution, the color format and the scanout buffer's location in physical memory. From these parameters the Linux kernel creates a platform device that represents (an abstraction of) the system's graphics card. The firmware-provided graphics output is now part of the kernel's device hierarchy and can be claimed by the available device drivers.

Linux has supported this for at least two decades and provided output to the (pretended) graphics card via generic fbdev drivers. Programs render to the graphics device by memcpy-ing into the provided scanout buffer. The graphics hardware displays whatever has been stored there. The key limitation is that there's no change to the physical characteristics of the output possible. The scanout buffer cannot be relocated in memory, nor can the display resolution or color format be changed.

At some point, the user expects to be able to use accelerated rendering or switch resolutions. This requires a native DRM driver for the graphics hardware. It is stored as a kernel module on the root partition, which becomes available at a late stage of the boot process. Once the kernel has loaded the driver module, it compares the driver's table of hardware IDs with the system's devices and let's

⁴While VGA provides a standardized set of hardware registers and features, the VGA graphics card still needs to be powered up and initialized.

the driver probe matching devices. This poses a problem: the native driver's ID table matches against the actual graphics device, such as a PCI graphics card, but not against the platform device that is being served by the hardware-agnostic driver. Both drivers and devices refer to the same underlying hardware, although by different interfaces. Results are undefined.

The hardware-native drivers resolve this problem by first hot-unplugging the platform device and hence any related driver and current users of the driver. This is a two-step process. As a first step, the hardware-agnostic driver claims ownership of the physical location of its scanout buffer. In the second step, the native driver hot-unplugs all devices owning parts of its video memory before probing the hardware. For a discrete graphics card on the PCI bus, the scanout buffer is located in video memory. This limits the possible candidates for removal to platform devices referring to this memory region. For SoCs or integrated graphics chips, the scanout buffer can potentially be located anywhere in physical memory. Hence the native driver removes all such platform devices. After removing the devices, the native driver continues by probing the hardware.

The presented mechanisms provide consistent display output from starting the firmware to the point where the desktop or graphics console accepts user input. As all graphics output is being done on the same hardware, handover from one driver to the next can be flicker free as long as the initial display resolution and color format is reasonable for modern environments. On the userspace side, boot-splash programs or graphics user environments, such as Plymouth or Gnome, release opened graphics device as they are being hot-unplugged and open graphics devices as they are being announced. Display output can continue seamlessly.

Implementation Details

The hardware-agnostic DRM driver, called *simplifiedrm*, is a regular Linux kernel driver for platform devices. Several details of the implementation are noteworthy though.

- The hardware-agnostic fbdev drivers registered their platform devices by themselves as part of initializing the module. We moved this code to a central location under drivers/firmware. The devices will therefore show up in the device hierarchy even if no driver is available.
- The forced removal of drivers during the handover step used to be implemented within the fbdev subsystem. We reimplemented this code as *aperture helpers* under drivers/video. It is now available to any subsystem for maintaining ownership of the graphics devices. In fact, VFIO already uses these helpers to clear ownership before giving a graphics device to a virtual machine.
- The aperture helpers need to prevent hardware-agnostic drivers from probing if a native driver has acquired the device. This has been a long-

standing problem ever since fbdev introduced the handover mechanism. It is not an issue in practice though and we have taken steps to resolve it.

- DRM has full support for hot-unplugging devices. Userspace compositors can continue to operate on a hot-unplugged device even if the scanout buffer does not exist any longer. Fbdev drivers instead provide mmap'ed video memory to userspace and are prone to generate SIGBUS errors on hot-unplugging. Alternatively, delaying the hot-unplug operation until userspace has closed the final file descriptor on the device blocks progress in the kernel for an unbounded amount of time.⁵
- For `simplifiedrm`, all DRM buffer objects are located in system memory. When the DRM master pageflips to the framebuffer⁶ that is to be displayed, `simplifiedrm` copies the buffer's content to the firmware-provided scanout buffer. If the device has been hot-unplugged, the driver omits the copying step. This prevents the unbounded blocking in the kernel that plagues fbdev.
- DRM offers explicit damage handling to userspace. Compositors call it to update the parts of the framebuffer that changed during an update. This is an improvement over fbdev, where damage handling can at best be done at a course-grained level by tracking memory pages, or has to be implemented by each compositor individually in userspace.
- DRM offers conversion among different color formats. Most of today's userspace programs expect hardware to support 32-bit XRGB. If the scanout buffer is of any other format, such as 16-bit RGB, `simplifiedrm` automatically converts the output format as part of the damage handling. The native format is the preferred one, but alternatives exist for compatibility.
- Almost all of the code for memory management and color-format conversion is located in DRM helpers and shared with other drivers. The `simplifiedrm` driver is only ~900 LoC in size.
- We split DRM's core and helpers into smaller libraries in different kernel modules to reduce the size of the kernel binary. For example, the modesetting helper library has to be linked into the kernel binary for using `simplifiedrm` early in the booting process. Moving unnecessary helpers for DisplayPort into a separate library reduced its size from 447 KiB to 240 KiB. As the DisplayPort helpers are not required by `simplifiedrm`, they can remain in a separate kernel module on the system's root partition.
- The X server parses the bus by itself and tries to match the parsed devices to DRM device files. This fails for platform devices. We had to add a (one-line) patch to make it work with `simplifiedrm`. Modern Wayland

⁵https://bugzilla.opensuse.org/show_bug.cgi?id=1181913.

⁶In the context of DRM, a *framebuffer* is a software data structure that refers to a number of buffer objects with a specific resolution and color format.

compositors simply open the device files and use whatever hardware they find. No fixes are needed.

We are quite happy with the current state of `simplifiedrm`. Occasionally we receive patches with improvements or support for new platforms. Most notably, `simplifiedrm` drives graphics output on Apple's M1 systems until the native driver becomes available.

We switched openSUSE Tumbleweed over to the fully DRM-based graphics stack during the kernel 5.18 timeframe on all x86 and ARM platforms. In the wider Linux ecosystem, Red Hat switched to `simplifiedrm` with Fedora 36 and Alpine Linux switched with release 3.15.

Future Work

With `simplifiedrm` in place and running on a meaningful number of systems, we turn our attention to future projects. A number of ideas are either in the works or being planned.

- We are working on a hardware-agnostic DRM driver for PowerPC systems. The driver, named *ofdrm*, replaces the existing *offb* driver that is currently being used on those systems. It uses the same principles as `simplifiedrm`, but the platform is different enough to justify a separate driver. We expect to merge *ofdrm* and enable it on Tumbleweed in the coming weeks.
- On upstream mailing lists, there has recently been an interest in supporting very old and low-end devices with DRM. One candidate is the VGA graphics standard. The major challenge lies in the assumptions set by userspace compositors. Emulating 32-bit XRGB output on top of VGA's 16-color palette requires a significant amount of image dithering to produce pleasant-looking results.
- The proprietary NVIDIA does not use DRM's console emulation, but relies on *fbdev* to provide a console. In its current form it cannot be used with `simplifiedrm` without losing console functionality. The closed-source nature of NVIDIA's code makes it hard to fix this. So for the time being, we mitigate the problem: when the user installs one of the NVIDIA driver packages, we set a kernel parameter that brings back the *fbdev* stack. This needs to be resolved at some point.
- With a fully DRM-based graphics stack, we can phase out the kernel's graphics console and implement the terminal entirely in userspace. This opens up the possibility for additional features, such as TrueType font rendering, right-to-left scripts, multi-screen output, night mode or Wayland compositing within the terminal. Preliminary code is available for some of these ideas.
- The graphics stack currently breaks when *kdump* is being invoked during a kernel panic. At best, the display remains static until a native driver

has reinitialized the graphics hardware, which only happens late in the booting process. By recording the state of the graphics output on each modeset operation, the kdump kernel's `simplifiedrm` driver can pick up the settings and provide a minimal output during the earlier stages of booting.

Conclusion

In this paper we outlined the introduction of `simplifiedrm` as a hardware-agnostic DRM driver and the switch to a fully DRM-based graphics stack. We also went through the booting process and highlighted key points and problems and how DRM helps solving them. Since its addition to the Linux kernel, `simplifiedrm` has been successfully introduced in a number of Linux distributions where it provides graphics output when no native driver is available.

Acknowledgements

Thanks to the various members of the Linux graphics development community who helped with `simplifiedrm`. Special thanks go to Javier Martinez Canillas of Red Hat. Javier provided many bugfixes and features for `simplifiedrm` and the related infrastructure. He also updated Fedora 36 to use the driver.