

The Linux Graphics Stack in a Nutshell

Thomas Zimmermann <tzimmermann@suse.com>

Keep this pixel in mind! :) We're tracing its way through the graphics stack.

The screenshot shows a web browser window displaying the openSUSE Wiki page for Portal:Tumbleweed. The browser's address bar shows the URL `https://en.opensuse.org/Portal:Tumbleweed`. The page title is "Portal:Tumbleweed". A large blue arrow points from a Go game board overlay on the left towards the Tumbleweed logo. The Go board is a green grid with several white and black stones. The Tumbleweed logo features a circular arrow icon above the word "Tumbleweed" in a teal font. Below the logo, the text describes Tumbleweed as a pure rolling release version of openSUSE. To the right of the main content, there are two sidebars: "Distribution" and "Navigation". The "Distribution" sidebar lists the rolling release (Tumbleweed), the current regular release (openSUSE Leap 15.4), and the previous regular release (openSUSE Leap 15.3). The "Navigation" sidebar lists various links related to Tumbleweed, including installation, JeOS, MicroOS, and upgrade guides.

Portal:Tumbleweed

Welcome to the **openSUSE Tumbleweed** portal!



Tumbleweed distribution is a pure **rolling release** version of openSUSE containing the latest "bleeding edge" versions of all software instead of relying on rigid periodic release cycles. The project does this to provide users that want the newest stable software.

Tumbleweed is based on **Factory**, openSUSE's main development codebase. Tumbleweed is updated whenever the Factory's bleeding edge software has been integrated, stabilized and tested. Tumbleweed contains the latest stable applications and is ready and reliable for daily use.

The idea has been discussed in mailing lists for a long time and was conceived into action by Greg Kohn-Hartman, originally as an 'add-on' set of rolling updates which could be layered on top of a regular openSUSE release. On November 4th 2014 the Tumbleweed rolling release and Factory rolling release merged, leaving the single openSUSE Tumbleweed rolling release we have today.

Please take note that many of the pages you will see on these wikis offer contents that are either partially outdated, or partially in tension with best-practices. As we work toward migrating an

Distribution

- Rolling release**
Tumbleweed **rolling**
- Current regular release**
openSUSE Leap 15.4 **stable**
- Previous regular release**
openSUSE Leap 15.3 **old stable**
- [More about derivatives](#)

Navigation

- [openSUSE:Tumbleweed installation](#)
- [Portal:JeOS](#)
- [Portal:MicroOS](#)
- [SDB:Nomodeset: Work Around Graphic Upgrade & Installation Obstacles](#)
- [Portal:Tumbleweed](#)
- [Tumbleweed zypper dist-upgrade as a systemd service](#)
- [openSUSE:Tumbleweed upgrade](#)
- [SDB:Upgrade Tumbleweed](#)



Graphics used to be done with X11.

- Linux graphics stack was originally built around X11
- X server was a single monolithic userspace program that handled all output. X clients (i.e., applications) sent drawing commands to the X server
- Network transparent, but includes a certain lag
- Even worse: 3d graphics hardware requires locality to keep pipeline saturated



Buffer sharing improves performance.

- With the Direct Rendering Infrastructure (DRI), buffers in video memory can be shared between X client and X server
- DRI consists of a number of components; including the kernel's Direct Rendering Manager (DRM)
- X11 had to drop support for network transparency (at least if performance was a concern)
- X client renders into shared buffer; X server displays the buffer
- Multiple X clients can shared the same hardware



Video memory is the central resource.

- Clients render into video memory
- Discrete video memory is located on the graphics device, available via PCI BAR
- Graphics Address Remapping Table (GART) extends discrete memory
- DMA memory is usually used by SoC designs
- System memory



Graphics drivers manage video memory.

- Kernel's Direct Rendering Manager (DRM) provides helpers for common use cases: Translation Table Manager (TTM) for VRAM and GART, DMA helpers, SHMEM helpers for system memory
- Kernel graphics drivers implement memory management
- The Graphics Execution Manager (GEM) provides the interface to memory buffers
- GEM buffer object is the central data structure



Buffer creation depends on the graphics driver.

- GEM provides common buffer operations, such as pinning, mmap-ing, releasing
- Except for buffer creation
- Driver ioctls for hardware accelerated rendering
- Buffer objects tied to hardware features
- Dumb-buffer interface for plain software rendering via `mmap()` and `memcpy()`
- Dumb buffers good enough for basic userspace or the kernel console



Userspace libraries provide rendering.

- Rendering implemented in userspace
- Mesa library provides 3d graphics driver for each supported chipset
- Mesa also implements common software rendering interfaces such as OpenGL, OpenGLES, Vulkan
- Mesa translates interfaces into internal state trackers (Gallium) and shader IR (NIR)
- Mesa drivers translate and forward to hardware via DRM kernel drivers
- Data resides in GEM buffer objects



The Wayland protocol enables compositing.

- Modern compositors and their clients use the Wayland protocol
- The compositor also uses software rendering interfaces to render screen output
- Application window is a rectangle textured with the application's output buffer
- The application's output buffer became an input buffer for the compositor
- Wayland is only for local clients; no more network transparency



Plain shared memory can be enough.

- Client creates a file descriptor for shared memory and transfers it to the compositors
- Both mmap the memory pages
- The client renders into the buffer, then sends a damage message to the compositor
- The compositor updates the on-screen window on each damage message
- Weak synchronization makes it only feasible with software rendering



Linux' dma-buf enables high-performance rendering.

- Client creates a GEM object buffer that support hardware-accelerated rendering
- Driver-specific buffer-object creation is required to do this
- Client and compositor export/import GEM object buffers via kernel's dma-buf framework
- Both share the same pages within video memory



Fencing synchronizes client and compositor.

- Each dma-buf object comes with a dma-buf reservation lock for synchronization within kernel
- Clients send rendering commands to the hardware (via Mesa) and a damage message to the compositor (via Wayland)
- Compositor renders screen
- DRM kernel drivers implement fencing on top of dma-buf reservation lock, so that the client's job has been completed when the compositor's job starts
- DRM scheduler tracks dependencies



Video decoding works the same.

- Client decodes video with available software or hardware
- Various interfaces available, such as VA-API, VDPAU, maybe Vulkan Video, plenty of higher-level decoders
- Transfer resulting video frame to compositor as with rendered output
- Linux kernel's media subsystem also uses dma-buf; can synchronize with DRM rendering
- Video is typically in YUV format



DRM kernel drivers implement the modesetting pipeline.

- Compositor attaches its output buffer object to DRM framebuffer, which gives it a color format and a resolution
- Compositor implements modesetting and page flips
- Instructs DRM driver to set a display mode and display the framebuffer's content
- DRM provides individual stages of the modesetting pipeline, such as planes, CRTC, encoders, connectors



DRM kernel drivers implement the modesetting pipeline.

- Plane represents a (partial) framebuffer that is shown somewhere on the screen
- CRTC represents a screen with display mode with one or more independent planes shown
- A full-size primary plane contains the background, user interface, windows, etc
- A cursor planes floats above the screen and shows the mouse pointer
- Overlay planes show additional output, such as video frames in YUV format



Encoder and connector represent the output.

- Encoder transforms the display signal
- Connector represents the physical connector and attached screen
- Connector also reads EDID information from attached monitor and returns display modes



All pipeline changes are atomic.

- Complex pipelines might push hardware beyond limits
- Limited video memory or number of planes can be a problem
- Modesetting fail in an intermediate state
- The modesetting pipeline and each of its stage has a state attached
- Atomic modesetting implements check/commit procedure for every change
- Atomic commit is all-or-nothing



There's more.

- Kernel's framebuffer console is an in-kernel client that is being shown while no userspace has opened the device file
- DRM leases allow temporary handover of modesetting capabilities to other programs; used for 3d headsets
- Compositors implement network support independently from Wayland, via RDP
- Xwayland implements X11 on top of Wayland
- The Linux kernel's new accelerator subsystem is build on top of DRM; think of "rendering without graphics output"



That's it; except for details.

- We quickly rushed through the life of an on-screen pixel from storage allocation to rendering to displaying
- Q&A, if there's time left

