What is udlfb?

This is a driver for DisplayLink USB 2.0 era graphics chips.

DisplayLink chips provide simple hline/blit operations with some compression, pairing that with a hardware framebuffer (16MB) on the other end of the USB wire. That hardware framebuffer is able to drive the VGA, DVI, or HDMI monitor with no CPU involvement until a pixel has to change.

The CPU or other local resource does all the rendering; optionally compares the result with a local shadow of the remote hardware framebuffer to identify the minimal set of pixels that have changed; and compresses and sends those pixels line-by-line via USB bulk transfers.

Because of the efficiency of bulk transfers and a protocol on top that does not require any acks - the effect is very low latency that can support surprisingly high resolutions with good performance for non-gaming and non-video applications.

Mode setting, EDID read, etc are other bulk or control transfers. Mode setting is very flexible - able to set nearly arbitrary modes from any timing.

Advantages of USB graphics in general:

- Ability to add a nearly arbitrary number of displays to any USB 2.0 capable system. On Linux, number of displays is limited by fbdev interface (FB_MAX is currently 32). Of course, all USB devices on the same host controller share the same 480Mbs USB 2.0 interface.

Advantages of supporting DisplayLink chips with kernel framebuffer interface:

- The actual hardware functionality of DisplayLink chips matches nearly one-to-one with the fbdev interface, making the driver quite small and tight relative to the functionality it provides.
- X servers and other applications can use the standard fbdev interface from user mode to talk to the device, without needing to know anything about USB or DisplayLink's protocol at all. A "displaylink" X driver and a slightly modified "fbdev" X driver are among those that already do.

Disadvantages:

- Fbdev's mmap interface assumes a real hardware framebuffer is mapped. In the case of USB graphics, it is just an allocated (virtual) buffer. Writes need to be detected and encoded into USB bulk transfers by the CPU. Accurate damage/changed area notifications work around this problem. In the future, hopefully fbdev will be enhanced with an small standard interface to allow mmap clients to report damage, for the benefit of virtual or remote framebuffers.
- Fbdev does not arbitrate client ownership of the framebuffer well.
- Fbcon assumes the first framebuffer it finds should be consumed for console.
- It's not clear what the future of fbdev is, given the rise of KMS/DRM.

How to use it?

Udlfb, when loaded as a module, will match against all USB 2.0 generation DisplayLink chips (Alex and Ollie family). It will then attempt to read the EDID of the monitor, and set the best common mode between the DisplayLink device and the monitor's capabilities.

If the DisplayLink device is successful, it will paint a "green screen" which means that from a hardware and fbdev software perspective, everything is good.

At that point, a /dev/fb? interface will be present for user-mode applications to open and begin writing to the framebuffer of the DisplayLink device using standard fbdev calls. Note that if mmap() is used, by default the user mode application must send down damage notifications to trigger repaints of the changed regions. Alternatively, udlfb can be recompiled with experimental defio support enabled, to support a page-fault based detection mechanism that can work without explicit notification.

The most common client of udlfb is xf86-video-displaylink or a modified xf86-video-fbdev X server. These servers have no real DisplayLink specific code. They write to the standard framebuffer interface and rely on udlfb to do its thing. The one extra feature they have is the ability to report rectangles from the X DAMAGE protocol extension down to udlfb via udlfb's damage interface (which will hopefully be standardized for all virtual framebuffers that need damage info). These damage notifications allow udlfb to efficiently process the changed pixels.

Module Options

Special configuration for udlfb is usually unnecessary. There are a few options, however.

From the command line, pass options to modprobe modprobe udlfb fb_defio=0 console=1 shadow=1

Or modify options on the fly at /sys/module/udlfb/parameters directory via sudo nano fb_defio change the parameter in place, and save the file.

Unplug/replug USB device to apply with new settings

Or for permanent option, create file like /etc/modprobe.d/udlfb.conf with text options udlfb fb_defio=0 console=1 shadow=1

Accepted boolean options:

fb_defio	Make use of the fb_defio (CONFIG_FB_DEFERRED_IO) kernel module to track changed areas of the framebuffer by page faults. Standard fbdev applications that use mmap but that do not report damage, should be able to work with this enabled. Disable when running with X server that supports reporting changed regions via ioctl, as this method is simpler, more stable, and higher performance. default: fb_defio=1
console	Allow fbcon to attach to udlfb provided framebuffers. Can be disabled if fbcon and other clients (e.g. X with --shared-vt) are in conflict. default: console=1
shadow	Allocate a 2nd framebuffer to shadow what's currently across the USB bus in device memory. If any pixels are unchanged, do not transmit. Spends host memory to save USB transfers. Enabled by default. Only disable on very low memory systems. default: shadow=1

Sysfs Attributes

Udlfb creates several files in /sys/class/graphics/fb? Where ? is the sequential framebuffer id of the particular DisplayLink device

edid	If a valid EDID blob is written to this file (typically by a udev rule), then udlfb will use this EDID as a backup in case reading the actual EDID of the monitor attached to the DisplayLink device fails. This is especially useful for fixed panels, etc. that cannot communicate their capabilities via EDID. Reading this file returns the current EDID of the attached monitor (or last backup value written). This is useful to get the EDID of the attached monitor, which can be passed to utilities like parse-edid.
metrics_bytes_rendered	32-bit count of pixel bytes rendered
metrics_bytes_identical	32-bit count of how many of those bytes were found to be unchanged, based on a shadow framebuffer check
metrics_bytes_sent	32-bit count of how many bytes were transferred over USB to communicate the resulting changed pixels to the hardware. Includes compression and protocol overhead
metrics_cpu_kcycles_used	32-bit count of CPU cycles used in processing the above pixels (in thousands of cycles).
metrics_reset	Write-only. Any write to this file resets all metrics above to zero. Note that the 32-bit counters above roll over very quickly. To get reliable results, design performance tests to start and finish in a very short period of time (one minute or less is safe).

Bernie Thompson <bernie@plugable.com>