A deeper look into GPUs and the Linux Graphics Stack

Martin Peres
CC By-SA 3.0

Nouveau developer
Ph.D. student at LaBRI

November 26, 2012
General overview

Outline

1 I - GPU & Hardware
   - General overview
     - Driving screens
     - Host < − > GPU communication

2 II - Host: summary
   - General overview

3 DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4 Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5 X11 and the XServer
   - Overview
   - X11
   - X-Server
General overview of a modern GPU’s functions

- Display content on a screen
- Accelerate 2D operations
- Accelerate 3D operations
- Decode videos
- Accelerate scientific calculations
General overview

- CPU
- Flash ROM (BIOS)
- Super I/O
- Serial Port
- Parallel Port
- Floppy Disk
- Keyboard
- Mouse
- Northbridge (memory controller hub)
- Southbridge (I/O controller hub)
- Chipset
- Front-side bus
- Memory bus
- Memory Slots
- High-speed graphics bus (AGP or PCI Express)
- Graphics card slot
- PCI Bus
- PCI Slots
- LPC Bus
- Onboard graphics controller
- Cables and ports leading off-board

- IDE
- SATA
- USB
- Ethernet
- Audio Codec
- CMOS Memory
- Serial Port
- Parallel Port
- Floppy Disk
- Keyboard
- Mouse
- Clock Generator
## General overview

### Hardware architecture

- **GPU**: Where all the calculations are made
- **VRAM**: Stores the textures or general purpose data
- **Video Outputs**: Connects to the screen(s)
- **Power stage**: Lower the voltage, regulate current
- **Host communication bus**: Communication with the CPU

Source: [http://www.flickr.com/photos/stefan_ledwina/557505323](http://www.flickr.com/photos/stefan_ledwina/557505323)
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
     - Host <-> GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Driving screens: the big picture

- Framebuffer: The image to be displayed on the screen (VRAM)
- CRTC: Streams the framebuffer following the screen’s timings
- Encoder: Convert the CRTC’s output to the right PHY signal
- Connector: The actual connector where the screen is plugged
Screen connectors

- VGA: Video, introduced in 1987 by IBM
- DVI: Video, introduced in 1999 by DDWG
- DP: Video & Audio, introduced in 2006 by VESA
- HDMI: Video & Audio, introduced in 1999 by HDMI Founders
Driving screens: the CRT Controller

- Streams the framebuffer following the screen’s timings
- After each line, the CRTC must wait for the CRT to go back to the beginning of the next line (Horizontal Blank)
- After each frame, the CRTC must wait for the CRT to go back to the first line (Vertical Blank)
- Timings are met by programming the CRTC clock using PLLs
Configuring the CRTC: Extended display identification data

- Stored in each connector of the screen (small EEPROM)
- Is usually accessed via a dedicated I2C line in the connector
- Holds the modes supported by the screen connector
- Processed by the host driver and exposed with the tool `xrandr`
  (see `xrandr --verbose`
Example: Some display standards

- **1981: Monochrome Display Adapter (MDA)**
  - text-only
  - monochrome
  - 720 * 350 px or 80*25 characters (50Hz)

- **1981: Color Graphics Adapter (CGA)**
  - text & graphics
  - 4 bits (16 colours)
  - 320 * 200 px (60 Hz)

- **1987: Video Graphics Array (VGA)**
  - text & graphics
  - 4 bits (16 colours) or 8 bits (256 colours)
  - 320*200px or 640*480px (<= 70 Hz)
Driving screens

**Initial video mode**
- Very low resolution (640*480px, 4 bits);
- Provide a simple “accelerated” terminal;
- Allow per-pixel access;
- Accessible from real mode, 10h BIOS call.

**VESA BIOS Extensions (VBE)**
- Bios call to change the mode;
- High-resolution video mode (\(\leq 1600*1200\));
- 16 or 24 bits colour resolution;
- Page flipping, access from the protected mode;
- etc...
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - **Host < – > GPU communication**

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Modern host communication busses

- **1993**: Peripheral Component Interconnect (PCI)
  - 32 bit & 33.33 MHz
  - Maximum transfer rate: 133 MB/s
- **1996**: Accelerated Graphics Port (AGP)
  - 32 bit & 66.66 MHz
  - Maximum transfer rate: 266 to 2133 MB/s (1x to 8x)
- **2004**: PCI Express (PCIe)
  - 1 lane: 0.25 — > 2 GB/s (PCIe v1.x — > 4.0)
  - up to 32 lanes (up to 64 GB/s)
  - Improve device-to-device communication (no arbitration)

Features

- Several generic configuration address spaces (BAR)
- Interruption RQuest (IRQ)
Programming the GPU: Register access via MMIO

- A GPU’s configuration is mostly stored in registers;
- A register is usually identified by an address in a BAR;
- Device’s BARs are all accessible in the physical address space;
- They are thus mappable in the CPU’s virtual memory;
- Registers are then accessed like a “uint32_t array”;  
- This is called Memory-Mapped Input/Output (MMIO).

Example of a CPU process's virtual memory space:

- Disk
- RAM
- PCI-01:00 BAR0

Logical address
Physical address

Unused
Another process's memory
GPU 0, BAR 0 Register Space

(swap)
GPU-accessible memory areas

- Video RAM (VRAM) : Blazing fast
- Host RAM via Direct Memory Access (DMA) : Fast
  - Graphics Translation Table (GTT/GART);
    - Exposes a linear buffer from multiple RAM pages;
- VGA window (physical address range: 0xa0000-0xbffff).
GTT/GART

Providing the GPU with easy access to the Host RAM

Process virtual address space (VM)

Physical address

GPU virtual address (VRAM + GART)

Location of the address/memory:

- CPU
- GPU
- RAM
- GTT/GART (references RAM)
- Device

GTT/GART as a CPU-GPU shared-buffer for communication

- GPU feature to gather some RAM pages in the physical space;
- Can be seen as a host-managed MMU on the GPU;
- The host maps a RAM buffer into GART and then maps this new address into a GPU virtual address space. Shared Mem!
GTT/GART usage

- Upload textures or scientific data;
- Store the pushbuffer (GPU command submission).

Event reporting: Interruption RQuest(IRQ)

- GPUs often report events such as screen (un)plugged, processing error, etc... They should be processed ASAP;
- A device can send an IRQ to wake/interrupt the CPU;
- The CPU jumps to some code to handle the IRQ;
- Once the event is acknowledged, the CPU can continue what it was doing before the event occurred.
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < - > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
The GPU needs the host for:

- Setting the screen mode/resolution (mode setting);
- Configuring the engines and communication busses;
- Handling power management;
  - Thermal management (fan, react to overheating/power);
  - Change the GPU’s frequencies/voltage to save power;
- Processing data:
  - Allocate processing contexts (GPU VM + context ID);
  - Upload textures or scientific data;
  - Send commands to be executed in a context.
Overview of the components of a graphics stack

- A GPU with its screen;
- One or several input devices (mouse, keyboard);
- A windowing system (such as the X-Server and Wayland);
- Accelerated-rendering protocols (such as OpenGL);
- Graphical applications (such as Firefox or a 3D game).

Components of the Linux Graphics stack

- Direct Rendering Manager (DRM) : exports GPU primitives;
- X-Server/Wayland : provide a windowing system;
- Mesa : provides advanced acceleration APIs;
General overview

- Kernel space:
  - drm
  - nouveau
  - radeon
  - intel

- User space:
  - Applications
    - Qt
    - gtk
    - nexuiz
  - Xorg
    - xlib
    - network
    - x-server
    - ddx
    - mesa
    - libdrm

- Hardware:
  - GPU
  - CPU

- Rasterizer
  - If UCS*
Outline

1 I - GPU & Hardware
   • General overview
   • Driving screens
   • Host < − > GPU communication

2 II - Host: summary
   • General overview

3 DRM
   • Overview
   • Kernel Mode Setting
   • Graphics buffer management
   • How to contribute code

4 Mesa
   • Mesa
   • OpenGL
   • Video Acceleration

5 X11 and the XServer
   • Overview
   • X11
   • X-Server
Direct Rendering Manager

- Inits and configures the GPU;
- Performs Kernel Mode Setting (KMS);
- Exports privileged GPU primitives:
  - Create context + VM allocation;
  - Command submission;
  - VRAM memory management: GEM & TTM;
  - Buffer-sharing: GEM & DMA-Buf;
- Implementation is driver-dependent.

libDRM

- Wraps the DRM interface into a usable API;
- Factors-out some code;
- Is meant to be only used by Mesa & the DDX;
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < — > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
**Kernel Mode Setting (KMS)**

- Opposed to User Mode Setting (UMS);
- The kernel manages modesetting;
- Enables:
  - root-less graphic server;
  - glitch-free boot;
  - fast VT-Switching;
  - better power management;
  - kernel crash logs.
Graphics buffer management

Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < − > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - **Graphics buffer management**
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Overview

- Graphics memory allocator;
- Manages VRAM and GART;

Buffer management: Constraints

- Contrarily to a CPU MMU, a page fault is “fatal”;
- We don’t know when the GPU will actually need the buffers;
- This means all pages should be available and addresses shouldn’t change during processing;
- What actually defines the processing time frame?
Detecting when a buffer is needed

- Buffers should be managed by the driver;
- A user should reference which buffers are needed in what command batch;
- When the kernel emits a command batch, it should pin the buffers that are used by this command batch.

When should we unpin a buffer?: Fencing

- Detecting the end of the execution of a command batch is done by fencing;
- Fencing can be implemented by adding an instruction at the end of the command batch to increment a counter (fence);
- When the counter’s value is higher or equal to the fence, we know the batch has been executed correctly.
- Alternatively, an IRQ could be sent instead of incrementing a counter.
Auto-deallocation

- Buffers are reference-counted to allow buffer deallocation;
- When a command batch references a buffer, the buffer’s reference counter is incremented;
- When a command batch has been processed, the reference counter is decremented;
- When the reference counter drops to 0, free the BO.
Translation Table Maps (TTM)

- Open-sourced by Tungsten Graphics (now VMware);
- Does what was described before (fencing, validation (forcing pinning));
- Used by Radeon and Nouveau.

Graphics Execution Manager (GEM)

- The standard interface for creating, sharing, mapping and modifying buffers;
- Intel only implements GEM, radeon and nouveau use a GEMified-TTM;
- Allow buffer sharing: flink() to share (returns an ID), open() to open a shared buf;
- Doesn’t specify fences/validation mechanisms.
DMA-Buf

- GEM flink is unsecure (once flinked, a buffer can be accessed with few controls’);
- GEM flink doesn’t work across drivers;
- DMA-Buf solves the latter and uses file descriptors to identify shared buffers;
- This fd can then be passed on to another process using unix sockets;
- The requesting process is responsible for transmitting the buffer.

More information on security

- https://lwn.net/Articles/517375/
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host $<$ -- $>$ GPU communication
2. II - Host: summary
   - General overview
3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code
4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration
5. X11 and the XServer
   - Overview
   - X11
   - X-Server
6. Attribution
How to contribute code

1: Get in touch with the X.org/DRM community;
2: Get to know how they work;
3: Pick something of interest to you and study it;
4: Can you improve current support? If not, goto 3
5: Write and propose a patch
6: Accepted? If not, listen their feedback and goto 5
7: Well done! Goto 3
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < − > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
     - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Mesa

- Provides advanced acceleration APIs:
  - 3D acceleration: OpenGL / Direct3D
  - Video acceleration: XVMC, VAAPI, VDPAU
- Mostly device-dependent (requires many drivers);
- Divided between mesa classics and gallium 3D;

Mesa classics

- Old code-base, mostly used by drivers for old cards;
- No code sharing between drivers, provide only OpenGL;

Gallium 3D

- Built for code-sharing between drivers (State Trackers);
- Pipe drivers follow the instructions from the Gallium interface;
- Pipe drivers are the device-dependent part of Gallium3D;
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < – > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Overview: Summary of OpenGL’s history

- OpenGL is edited and managed by the Khronos Group;
- First version accelerated transforming 3D coordinates to 2D;
- Later version added Transform, Clipping and lightning (TCL);
- These instructions were too fixed and limited creativity;
- This pipeline got replaced by a programmable one: Shaders!

Shaders

Shaders are roughly separated in 2 stages:

- Vertex Shaders: To implement coordinate transforming;
- Fragment Shaders: To implement post-processing effects.
Outline

1 I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < – > GPU communication

2 II - Host: summary
   - General overview

3 DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4 Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5 X11 and the XServer
   - Overview
   - X11
   - X-Server
Video Acceleration : Overview

The video pipeline is composed of the following stages:

- Decoding the video
- Convert the colourspace from YUV to RGB;
- Scale each frame to the wanted size;
- Composite the subtitles and the OSDs.
Decoding a VP8 video stream

Complex operations that can be implemented in hardware.
**XVideo Motion Compensation (XVMC)**

- Entire video-decoding offloading of:
  - MPEG and MPEG2 formats (DVD).
- Limits:
  - non-accelerated OSD/subtitles compositing.

**Video Decode and Presentation API (VDPAU)**

- Entire video-decoding offloading for most common formats;
- The presentation allows compositing the subtitles/OSD;
- OpenGL/CL-compat: use the output in GL/CL;
- Limits:
  - The hardware must support every format (not really a problem on firmware-based hw like NVidia).
Outline

1 I - GPU & Hardware
   • General overview
   • Driving screens
   • Host < — > GPU communication

2 II - Host: summary
   • General overview

3 DRM
   • Overview
   • Kernel Mode Setting
   • Graphics buffer management
   • How to contribute code

4 Mesa
   • Mesa
   • OpenGL
   • Video Acceleration

5 X11 and the XServer
   • Overview
   • X11
   • X-Server
X11: The X protocol version 11

- Rendering protocol over a socket;
- Provides a very simple rendering API;
- Was meant for mainframe environment (programs run on the mainframe and rendering on thin clients);
- Applications used to be rendered on a CPU;
- Is over 25 years old but supports extensions;
- Motif is a X11-based toolkit (see above).
XLib: Drawing applications with X11

- The X-Server is oftenly accessed through the XLib;
- It provides an interface to generate X11 commands;
- The XLib shouldn’t be used anymore, use XCB!

XCB: The X protocol C-language Binding

- New attempt to create a wrapper for X11;
- Designed to be lightweight, asynchronous;
- Provide a clean thread-safe interface;
- Allow direct access to the protocol as needed;
- Designed for toolkits and X-specific applications;
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < − > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Objectives

- Receive drawing commands from X Clients;
- Render them as efficiently as possible (acceleration!);
- Handle input events and client attributes;
- Work with the window manager.

Basic Acceleration

Implemented by the DDX:

- 2D acceleration (XAA, EXA, etc...);
- Simple video acceleration (video overlay XV).
The X-Server's internals
2D Acceleration

- XAA: Mostly accelerates lines and solid fills;
- EXA: Meant to accelerate XRender;
- SNA: Intel’s Sandy bridge New Acceleration;
- Glamor: Basing acceleration over OpenGL;
- Towards a Gallium3D-based common acceleration;
Video Acceleration: XVideo extension (XV)

- **Offloads:**
  - video scaling (with cubic filtering): Up scaling;
  - colour conversion: Converting YUV to RGB;
  - display.

- **Limits:**
  - Doesn’t offloads video decoding and OSD compositing.
The X Resize, Rotate and Reflect Extension (XRandR)

- Common X API to configure screens and multi head;
- Implemented by the open and proprietary drivers;

Composite extension

- Keep a copy of each window in memory;
- Re-calculate the framebuffer when a window moves;
- Moving a window doesn’t force the redraw of the windows under it;
- Also allow window-closing animation.
OpenGL X Extension (GLX)

- Allow the use of OpenGL on X;
- 2 modes:
  - Direct: The preferred rendering method (local rendering);
  - Indirect: The application asks the X-Server to be a GL proxy.

GLX direct rendering

The behaviour of this mode depends on the DRI protocol version:

- DRI1: X tells the app the size of its window and where to draw in the framebuffer;
- DRI2: The app renders to a buffer and passes it to the window manager via GEM;
- DRI-next: like DRI2 but uses DMA-Buf instead of GEM flink.

On DRI1, moving the window of a direct-rendered application resulted in strange behaviours.
Direct rendering with DRI1

- An application willing to use direct rendering (OpenGL, video dec, other);
- Asks the XServer for its position on the framebuffer and clipping rectangles (SAREA);
- Asks the XServer to put a lock on this SAREA;
- Render the frame.

Limits to direct rendering with DRI1

- Rendering not synchronized with applications (tearing!);
- Requires synchronisation between clients & the server;
- Doesn’t integrate well with compositing environments;
- Saving video memory is not needed anymore.
Some of the DRI1 problems
Direct rendering with DRI2

- An application willing to use direct rendering (OpenGL, video decoding) renders to a buffer and GEM flink it to send it to the XServer/compositor;
- The compositor keeps this buffer until it needs to render a new frame;
- → works on compositing environments, synchronized rendering, no locks :).

Limits to direct rendering with DRI2

- GEM Flink is unsafe (a flinked buffer is accessible with few controls);
- Increases the number of context switches, slower performance;
- Takes more memory than DRI1 (who cares?).
Direct rendering with DRI-next

- Same as DRI2 but use DMA-Buf instead of GEM_flink;

Limits to direct rendering with DRI-next

- Same as DRI2 but no security problem.
GLX Indirect rendering

- The application asks the X-Server to be a GL proxy;
- Supports up to OpenGL 1.4 on opensource drivers;
- At first, it meant no acceleration (swrast);
- With AIGLX (Accelerated Indirect GLX), commands are redirected to the right mesa driver.
Reaction to an input event

1. The kernel driver evdev sends an event to the X-Server;
2. The X-Server forwards it to the window with the focus;
3. The client updates its window and tells the X-Server;
4 & 5: The X-Server lets the compositor update its view;
6. The X-Server updates sends the new buffer to the GPU.
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < − > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server

Attributions
Overview

- Protocol started in 2008 by Kristian Høgsberg;
- Aims to address some of X11 shortcomings;
- Wayland manages:
  - Input events: Send input events to the right application;
  - Copy/Paste & Drag’n’Drop;
  - Window buffer sharing (the image representing the window);

Wayland Compositor

- Implements the server side of the Wayland protocol;
- Talks to Wayland clients and to the driver for compositing;
- The reference implementation is called Weston.
Current implementation of buffer-sharing

- Wayland applications share buffers using GEM_flink;
- They then send the share_ID to wayland which opens it;
Reaction to an input event

1: The kernel driver evdev sends an input event to “Weston”;
2: “Weston” forwards the event to the right Wayland client;
3: The client updates its window and send it to “Weston”;
4: Weston updates its view and send it to the GPU.
I - GPU & Hardware
- General overview
- Driving screens
- Host $\leftarrow \rightarrow$ GPU communication

II - Host: summary
- General overview

III - DRM
- Overview
- Kernel Mode Setting
- Graphics buffer management
- How to contribute code

IV - Mesa
- Mesa
- OpenGL
- Video Acceleration

V - X11 and the XServer
- Overview
- X11
- X-Server
X11 vs Wayland

- Rendering protocol vs compositing API:
  - X11 provides old primitives to get 2D acceleration (such as plain circle, rectangle, ...);
  - Wayland lets applications render their buffers the way they want;

- Complex & heavy-weight vs minimal & efficient:
  - X11 is full of old and useless functions that are hard to implement;
  - Wayland is minimal and only cares about efficient buffer sharing;

- Cannot realistically be made secure vs secureable protocol.
X11 : Security

- X doesn’t care about security and cannot be fixed:
  - Confidentiality: X applications can spy other applications;
  - Integrity: X applications can modify other apps’ buffers;
  - Availability: X applications can grab input and be fullscreen.
- An X app can get hold of your credentials or bank accounts!
- An X app can make you believe you are using SSL in Firefox!

Wayland : Security

- Wayland is secure if using a secure buffer-sharing mechanism;
- See https://lwn.net/Articles/517375/.
Outline

1. I - GPU & Hardware
   - General overview
   - Driving screens
   - Host < – > GPU communication

2. II - Host: summary
   - General overview

3. DRM
   - Overview
   - Kernel Mode Setting
   - Graphics buffer management
   - How to contribute code

4. Mesa
   - Mesa
   - OpenGL
   - Video Acceleration

5. X11 and the XServer
   - Overview
   - X11
   - X-Server
Thanks to the fellows on #Nouveau for answering my questions

The following X.org devs helped me getting this presentation into shape:

- lynxeye
- mwk
- mlankhorst
- ahuillet
- ymanton
- airlied
- RSpliet
Attributions : Hardware

- Own work: https://en.wikipedia.org/wiki/File:Virtual_memory.svg
Attributions: Host: The Linux graphics stack

- X.org community: X.org schematic
- Kristian Høgsberg: http://wayland.freedesktop.org/
- Emeric Grange:
  http://blog.mecheye.net/2012/06/the-linux-graphics-stack/
  http://fosswire.com/post/2009/05/xorg-dri2-uxa/