X and Modesetting: Atrophy illustrated.

Luc Verhaegen, March 2nd 2006.

For the last decade or so, every new year gets hailed as “the year of the linux desktop”. At the end of each year, all the analysts who made that claim and who were proven wrong (again), claim several causes. They always list Microsofts desktop monopoly as reason number one, with a lack of applications as a close second, and they invariably look at either the linux kernel or the desktop environment for possible improvements, always ignoring the big blob that is X that sits in between.

Modesetting is the core functionality of all graphics devices and the most important part of any video driver. Without it, everything else becomes irrelevant, as you need to be able to produce an image first before you can enjoy the eye-candy that is deemed so crucial for modern desktops.

Sadly, the same modesetting has been getting little to no attention in the past few years, and it is now the great source of frustration in all desktop users. So if linux wants to ever make it on the desktop, X modesetting should be handled properly too.

The appeal of Modesetting.

There are various reasons for modesetting receiving as little attention as it does.

- Modesetting is not sexy.

  Modesetting is as far away from eye-candy as possible. It is bland, boring and basic and therefor not very attractive. It is human nature to always gravitate towards the flashy stuff.

  It is also taken for granted and people expect it to work out of the box. Most people aren't prepared to spend a lot of time trying various drivers and configuration options to get a useful result. Yet they invariably get forced to do so.

- Modesetting requires more resources.

  Any form of hardware acceleration, be it 2d, 3d, or a video overlay, requires a single instance of the hardware. For a single version of the device, you only need to implement or fix things once. The chance of running into further issues are usually down to initialisation differences (different manufacturers BIOSes for instance).

  With modesetting, you require all hardware combinations: you can not make do with only a single CRT.

- Modesetting is hard.

  In modesetting, a single bit off might not show up on most of the cases, but somewhere, somehow, might ruin a given mode or bitdepth or a given output function, or a combination thereof. You have to always tread carefully, make small incremental changes and spend ages testing. This makes all of modesetting slow, hard and sometimes very frustrating.

The result of the above is that there is very little developer interest in only modesetting. It is seen as a necessary hurdle to be taken in order to get eye-candy to show up, it is never seen as a goal on its own. Companies also see modesetting as a waste of money, as it does take a lot of time and resources to do properly. This is why nasty little shortcuts are taken and hard bits are glossed over or purposefully ignored.

Modesetting in X.

X.org has 44 different video drivers, but sadly only a handful see active development. Some of them haven't changed since XFree86 4.0.0 was released, except for some maintenance fixes to keep them building.

Looking back, Xfree86 4.0.0 was a massive task. All the different, until then separate, servers, each supporting a single family of graphics cards, were moulded together, in a unified driver model with drivers as loadable modules. And over the course of just a few years, from 1996 to 2000, the code from the old servers was poured into the new modules.

This sort of effort would not be possible today, the knowledge, manpower and motivation just isn't there any more. Luckily we don't need to either, as the basic driver model is sound, it just hasn't been kept up to date.

The DDX - Driver gridlock.

Apart from economic and political reasons, and the lack of appeal detailed earlier, modesetting suffers from a self-inflicted burden as well.

The DDX or device dependant X is closely tied to all X drivers. Any change there means changing all drivers. The lack of driver maintainers means that people are required to delve deep in drivers themselves and this makes them very weary of any DDX side changes.

In fact, this is a vicious circle: The DDX doesn't support recent hardware well. But altering the DDX is tricky, time-consuming and very painful. So driver side workarounds are used. These make altering that driver trickier, more time-consuming and more painful. And so on.

One end of the spectrum: Tseng.

At one end of the X modesetting spectrum, there are the tseng devices; ET4000W32p, ET6000 and ET6100. This hardware is at least a decade old now, as the manufacturer was bought out by ATI in 1996, but it is the sort of hardware the current driver model was developed for.

One PCI device, one CRTC, one RAMDAC, one VGA connector. No DDC.

A CRTC or CRT controller, is the subdevice responsible for scanning out the framebuffer (or the various possible overlays) and producing the pixel data and the sync signals.
RAMDAC is the device that takes in the (digital) pixel data and the sync signals from the CRTC, and produces the analogue RGB signals a CRT can handle. With the ET4000 the RAMDAC was still an external device, with ET6000 the RAMDAC became an internal device.

Tseng is, or rather, was, also interesting because it was the sole reason for a bit of unnecessary modesetting complexity: multiple clockRanges. More on that later.

Other end of the spectrum: Unichrome.

VIAs Unichrome represents cheap, buggy and powerless integrated graphics, yet it is the perfect basis for developing a clear view of where the X driver modesetting model needs to go.

Like all devices produced in this millennium, it has two CRTCs. On top of that, it is capable of mixing up to 3 different outputs between those two CRTCs (4 for the CN700).

There is of course one internal RAMDAC for driving a CRT, but there are also 2 pixelbusses, which can drive several different devices. There's a special scaler in front of the bus meant for the panel, but this bus can also be used to connect an external device like an LVDS or TMDS encoder. The other bus is usually used for an external TV or TMDS encoder.

With the advent of the Matrox G400, single device dual CRTC implementations became the norm, yet the X driver model is and was not devised with that in mind. The short term solution was to use 2 distinct instances of the same driver to each drive one CRTC, and it is up to the driver to coordinate this within itself. This is one area where the X driver model urgently needs adjusting.

The use of external devices for outputs is common, although in the discrete market they are often included on the main silicon itself. These devices implement the various possible outputs in small chips, and all take in a pixel bus, sync and I2C, making them compatible with many video chips. So output devices are modular, reasonably similar, and not too hard to abstract in software. Yet currently, no two drivers share any of the output device code, so another long term goal for Modesetting is providing a nice abstraction in a dynamically loadable fashion (modules).

The above 2 paragraphs actually talk about common features in all recent devices, and for which there is less than ideal support. Discrete graphics rely less and less on external output devices, although this is often still up to the board maker, and therefore many people argue that it's no use separating out this code. At the other end Intel IGP graphics has by far the largest selection of external output devices, but it does depend solely on those external output devices.

The unichrome is special because it sits somewhere in between. In comparison with the intel hardware, the unichrome is more complex as it has a scaler for the panel. Also, while the unichromes can be found with 11 different external output devices from at least 6 different vendors, the latest unichrome (VT3157 on CN700) has an internal TV encoder. So an abstraction created on the unichrome alone should be well suited for all drivers.

The biggest advantage of the unichrome though, is the state of the current driver (the one at http://unichrome.sf.net/). Even though support isn't very wide and still limited due to a lack of resources, this driver has fully free and completely algorithmic modesetting for the primary CRTC and already has a reasonably complete abstraction for its TV encoders. The panel code, which, apart from the output bus and the scaler, also still messes with both CRTCs, has been forced into this abstraction too. So there is a very clear path towards creating a fully functional driver, one which is also nicely structured and which has modular output devices. Most of the information necessary to complete this work is available, all it takes is time and hardware.

All in all, this very basic device is quite different from devices like the tsengs, yet it is still being forced into a driver model meant for hardware like the tseng. This will have to change.

Modesetting enemy: VBE

VBE or VESA BIOS Extension is, as the name suggests, an extension to the old IBM VGA BIOS. And however one wishes to look at it, a VGA BIOS remains a self-modifying binary blob. The BIOS from the primary VGA device gets loaded at 0xC0000 and hooks itself into the interrupt vector table. Binaries don't come nastier than this.

VBE was created somewhere in 1989 or 1990, and it was meant to help application developers cope with the complexity of the new generation of video devices. One might remember that, in those days, every application and every game programmed the graphics subsystem directly. Modesetting was a serious burden that had to be reproduced for all supported devices each and every time, but which was usually limited to VGA.

So VBE is as binary and as “legacy” as it gets, yet X drivers quite often depend on it. This dependence is often only in part, but sometimes there is no way of getting any output without it.

The most stunning and painful example is of course the i830 driver for the Intel IGPs. Here the issues generally range from bad modes to the native resolution of a laptops panel not being supported. Both cases result in a lot of user frustration, and there is no direct solution for any of thist. Sometimes, it is possible to insert resolutions into a table in the bios, but this hack is only possible due to the specific construction of intel bioses.

The standard i830 excuse is the vast range of external output devices possible, but the datasheets for those are, in most cases, freely available, and much of the code can be shared between drivers. A possible excuse could be intellectual property, which is hard to defend in case of modesetting. The reality is laziness though, i810 was split up in September 2002 by David Dawes, and the i830 driver was moved to VBE.

What to expect for 7.1

- CVT:

  CVT or Coordinated Video Timing, is a newish VESA standard designed to replace GTF (generalized timing formula). It offers a simplified calculation for GTF like timing and it can also generate reduced blanking modes. The latter was created to reduce the bandwidth used for panels based screens, where there no longer is an electron beam that needs to be retraced. The win is in reducing the time needed for retraces and by, of course, reducing the blanking.

  CVS xserver now contains the util cvt. Its function and usage is similar to the already well-known gtf utility, although the refresh rate now defaults to 60Hz. The 'r' flag enables reduced blanking.

- Removal of multiple clockRanges and non-programmable dotclocks:

  Currently X allows for multiple dotclock ranges to be defined by the driver, and then passes a flag to modes to designate which range was found acceptable for the given mode. This was only used in the tseng driver to distinguish between an 8 and 16bit pixelbus to the RAMDAC. All ET4000W32p RAMDACs support a 16bit pixelbus, so this is now used by default.

  All currently supported PCI era (and later) devices are able to generate a freely chosen dotclock, the use of a fixed clocks was limited to older devices, which should no longer be considered in active use. After removing ISA support from atimisc, chips, tseng and trident, the vga driver is the only one which is tied to 2 fixed dotclocks. The (future) lack of higher level support for fixed clocks will be handled by forcing the VGA driver to only use the modes for which those dotclocks were designed (640x480 and 720x480).

  Both of the above driver side changes result in a nice clean-up in the DDX mode handling code (xf86Mode.c), making it more accessible and more hackable. They do break the X driver ABI.

- Active use of basic EDID information:

  X has had the ability to parse EDID (enhanced display identification) blocks since 1998, but apart from generating horizontal and vertical frequency ranges for the monitor, this information was never used.

  In bug #5386 there is currently code which handles the standard EDID information as part of xf86DCCGetProperties. This is an existing function which formerly did nothing more than tie a given EDID block to the monitor structure (MonRec). This function now handles all the basic EDID information, and creates a modeslist. If a ranges section is available then it used to set up the MonRac ranges, otherwise the generated modelist is used.

  By moving the range code out of the DDX and into the ddc module, we give the driver full control of how the EDID block is used. The driver is now free to ignore the standard handling and parse the EDID block itself, or alter information set by xf86DDCGetProperties before going into mode validation.

This code does create a lot of duplicate modes which aren't properly filtered by mode validation yet. A type based priority system is therefore being discussed. The difficulty with it is that it needs to strike the right balance between new functionality and automatic configuration, and keeping the use of well known and well understood configuration options. We don't want to trample the Modes directive or the ability to provide modelines in the config.

Long term goals

- Output probing and mode validation outside PreInit.

  We no longer live in the world of single device, single monitor, and we most certainly don't use the same monitor for the entire duration of an X servers life. So we need some infrastructure to reprobe all outputs and regenerate or dynamically alter the modelist, outside of the drivers PreInit function. A new callback will probably have to be added to ScrnInfoPtr.

- General X attribute/configuration system.

  Output devices, especially TV encoders, provide a lot of configuration options, and we currently have no standard means of bringing this configurability to the user. A general attribute backend will offer driver developers a means of exporting all sorts of options, and it will mean that applications can be created to let the users set those options.

  Some examples of functionality that the whole should provide:
  - Adding, removing, and setting modes
  - Probing outputs, enabling, disabling and assigning them to CRTCs
  - Deciding the layout of the CRTCs with respect to the framebuffer
  - Setting tv standard
  - Alter filter and gamma settings
  - Adjust viewable frame position (crucial on a TV).
  - Etc.

  The example for this is of course Thomas Winischhofers sisctrl, which currently operates on an extension of its own. Dave Airlie recently hacked the sisctrl extension to be able to reprobe the CRT on his laptop. sisctrl like functionality is a direct goal for me for unichrome as well.

  A shared back-end for this is needed, it will not only relieve a bit of the burden on the driver people, it will also provide, for instance, GNOME and KDE developers with a single means of accessing the most common functionality. More esoteric options will probably exist in driver specific clients to this back-end.

- Output device modules.

  As described earlier, modesetting must make use of the modularity that exists with respect to output devices. Doing this separation will result in better structured, better maintained and more functional drivers, and it is the one area where lowlevel code can be shared directly.

Sadly, this is a long term goal, as it will need to happen on an output device per output device and driver per driver basis. My personal aim is to implement as much as possible inside the unichrome driver first, i believe the abstraction in there is already quite good, and can only get better when more devices are added there. Moving to others will likely involve little changes, but I'm not going to impose something like this, it has to grow as needed when expanding.

- Multiple CRTC per device or driver instance support.
  As outlined several times already, the driver model assumes one CRTC per device while most modern devices have 2 CRTCs.

  This will require a massive amount of rework, so this will not be for tomorrow.

- Separation of modesetting from Xserver.
  The seperation of modesetting to a userspace library is where Egbert Eich wants X modesetting to go, and i follow him in this completely. Linus will never accept all the code necessary at kernelspace, and this way our driver code will not be limited to a single operating system only. Another long term goal which will require a lot of driver rework.

Conclusion

All in all, X modesetting is richer, more diverse and better understood than other driver models currently in existence. But it does need a lot of work and we can't afford to ignore it. The biggest problem is the lack of appeal and the fact that most drivers are actually unmaintained and underdeveloped. When sound and long term solutions are chosen consistently, we will be able to get out of the current impasse. But it will not be an easy task.