Accelerated Graphic Cards

While it is nice to have some graphics on the screen, it would be even nicer to have the video card do the dirty work. Beware though, things are not always as easy as they seem.

Contents

Cards with documentation

3Dfx

These are a sort of good news for 3dfx cards (those with a Voodoo chipset). These cards could be the only ones with relatively fast 3D support (Glide/OpenGL) and open specifications (http://darwin-3dfx.sourceforge.net/). Voodoo cards are still available from sources such as eBay.

Intel Integrated Graphics

The Intel video chipsets also have open standards (see www.intel.com (http://www.intel.com)). Some useful links:

- Intel 740 Graphics Accelerator (ftp://download.intel.com/support/graphics/intel740/29061902.pdf)
- Intel Graphics Documentations on X.org (http://www.x.org/docs/intel/)
- Intel Graphics Documentation on the 01.org (https://01.org/linuxgraphics/documentation/)

The intel-gpu-tools package mentioned on the 01.org page is recommended as it can be used to dump current register values for examination.

ATI/AMD

Since September 2007, AMD started opening up specifications for their recent GPUs. The ones regarding a R630 or M56 chip can be downloaded from X.org (https://www.x.org/docs/AMD/old/).

Since early 2014, AMD maintains up to date open programming guides for all their GPUs, including a specific email address for support.

X.org provides a documentation for AMD GPUs in their website (https://www.x.org/wiki/RadeonFeature/#index13h2). An official AMD Open GPU documentation is also available on AMD's website (https://developer.amd.com/resources/developer-guides-manuals/#open_gpu) and an implementation for the Linux kernel is available on GitHub (https://github.com/GPUOpen-Drivers/AMDVLK).

NVIDIA

NVIDIA now provides official documentations for most of their GPUs on GitHub (https://nvidia.github.io/open-gpu-doc/) and open sourced their Linux kernel module for their GPUs also on GitHub (https://github.com/NVIDIA/open-gpu-kernel-modules). Before NVIDIA officially open sourced their GPU stuff, there are many NVIDIA GPU reverse engineering projects like Nouveau (http://nouveau.freedesktop.org) and EnvyTools (https://github.com/envytools/envytools) and a group of hackers even leaked most of NVIDIA's GPU stuff like documentation and even full-on reference FPGA codes, which led to NVIDIA officially open sourcing of their GPU documentation and Linux kernel module. Most of these reverse engineering project aren't complete unlike NVIDIA's official documentation but you can still use it as an another source of NVIDIA GPU documentation in addition with NVIDIA's official open source documentation.

VMware SVGA-II

Not exactly an actual GPU, but the VMWare SVGA-II device makes for a nice virtual machine device for some 3D acceleration. It is available on both VMware and QEMU. A well-documented and easy-to-port reference driver has also been made available by VMware under the MIT license. Although it is deprecated, the device works perfectly fine under QEMU and is relatively simple to work with, thus it makes for a very good starting point for graphics acceleration. Reference driver available at the old vmware-svga page on SourceForge (http://vmware-svga.sourceforge.net/index.old.html).

Where can I find low-level information about NVIDIA, AMD, and other graphic cards?

Good question. Unless you find something else, there is virtually no information publicly available (for free or otherwise) about the internal workings of current GPUs. There are only small bits that are relevant to game programmers but nothing an OS developer could use.

Now, if you have a VIA integrated GPU, things may be better since they released an open source driver (http://linux.via.com.tw/) for both 2D and 3D operations.

There are virtually no tutorials or datasheets for the 2D acceleration features either, but at least we have open-source code for them. Among other sources, the X.org/XFree86 drivers, Haiku (was Open BeOS) accelerants (https://github.com/haiku/haiku/tree/master/src/add-ons/accelerants) and FreeBE/AF (http://www.shawnhargreaves.com/freebe/). They may provide enough information to reverse-engineer and figure out a model that could be used to program/port for your environment. Some older GPUs with only 2D acceleration might be documented by the VGADoc (https://pdos.csail.mit.edu/6.828/2018/readings/hardware/vgadoc/) (vgadoc4b.zip (https://pdos.csail.mit.edu/6.828/2018/readings/hardware/vgadoc4b.zip), also on GitHub (https://github.com/achernya/iap-6.828-website/tree/master/readings/hardware/vgadoc)). Anyone who wishes to put time into that kind of research is welcome to post their results here.

What can 2D acceleration do for me?

- Hardware mouse cursor, drawn and managed (e.g. you provide coordinates, the card does the rest)
- Bitblt (for "**BIT BL**ock **T**ransfer") can be used for screen-to-screen memory copy like window moving, scrolling, etc. You provide from and to boxes and the card does the rest. Some might know this as the "rasterop" or remember the hardware "blitter" in Amiga computers.
- Tiles. You enter a small NxN dataset (usually a bitmap of between 8x8 and 32x32), a foreground color and a mixing style plus some coordinates and the card "paints" the area with the given pattern. That can be handy to render Win95-like backgrounds (tiled ones) or even to draw fonts quickly.

How do 3D-accelerated programs talk to 3D-accelerating hardware?

Let's assume that you have a strong knowledge of OpenGL and that you don't need it covered here. If you take the example of NVIDIA's GPU driver for Linux (if you have a correctly configured NVIDIA GPU, you can see almost all this by a simple "strace" on a 3D program in linux), things are organized this way:

- Two libraries libGL.so and libGLcore.so will be loaded by any program that wishes to do accelerated 3D operations. Upon startup, those libraries open "/dev/nvidiactl" and "/dev/nvidia0".
- The kernel module is made of an "obscure" file nv-kernel.o which contains only "anonymized" symbols and the open source part that mainly glues the nv-kernel (which is actually almost system transparent) to the Linux kernel.
- The actual "conversation" between the library and the driver cannot be traced by conventional means: /dev/nvidia* only allows "ioctl" operations and mmap. The values you can observe in "/proc/XXXXX/maps" while the 3D program is running, which let us believe the driver actually exposes the hardware resources (e.g. texture space, vertex space, etc) directly to the library. Current Linux kernels can however be compiled with the mmiotrace option which can then be used to log all individual accesses to mmapped space.

```
l /proc/pci
Bus  0, device   0, function  0:
    Host bridge: VIA Technologies, Inc. VT82C693A/694x [Apollo PRO133x] (rev 196).
      Prefetchable 32 bit memory at 0xfc000000 [0xfdffffff].

Bus  1, device   0, function  0:
    VGA compatible controller: nVidia Corporation RIVA TNT2 Model 64 (rev 21).
      IRQ 11.
      Master Capable.  Latency=248.  Min Gnt=5.Max Lat=1.
      Non-prefetchable 32 bit memory at 0xf7000000 [0xf7ffffff].
      Prefetchable 32 bit memory at 0xfa000000 [0xfbffffff].
```

```
cat /proc/XXXXX/maps
...
40019000-40029000 rw-s f7810000 03:06 54934      /dev/nvidia0
40029000-4002a000 rw-s 0ba98000 03:06 54934      /dev/nvidia0
```

```
...
40a9b000-42a9b000 rw-s fa000000 03:06 54934      /dev/nvidia0
42a9b000-42b79000 rw-p 00000000 00:00 0
42b79000-42c7a000 rw-s fc010000 03:06 54934      /dev/nvidia0
42c7a000-42d7c000 rw-p 00000000 00:00 0
42d7c000-42dfc000 rw-s fc111000 03:06 54934      /dev/nvidia0
```

External links

- A topic on duplicating/reverse engineering existing driver code