

Flexible and Secure Hardware 3D Rendering on Xen

Christopher Smowton (chris.smowton@cl.cam.ac.uk), Cambridge University

Abstract

I describe the design and implementation of a highly general system for hardware accelerated 3D graphics rendering by unprivileged guest VMs. Tungsten Graphics' Gallium3D driver architecture is used to provide a hardware-independent interface for remoting.

1 Introduction

Recently, a number of commercial products and academic projects have endeavoured to provide full 3D acceleration for guest virtual machines; a number of different pieces of software provide support for various combinations of host operating system, guest operating system, and graphics API. In most cases these have used API remoting techniques [4, 3, 5], in which calls to a graphics API such as OpenGL are remoted RPC-style.

This use of API remoting suffers from inflexibility: in order to support a different graphics API to that which is remoted, the remoting solution must be substantially redeveloped. It also involves running a complete 3D graphics driver in domain 0, inflating the trusted computing base.

My solution differs from previous API-remoting software in that it makes use of Tungsten Graphics' Gallium driver architecture. Gallium graphics drivers are independent of the graphics API used by programmers; I utilise this neutrality in order to provide a single virtualisation solution which is itself API-independent. They further split the graphics driver into modules with well-defined interfaces; I take advantage of this to move large proportions of the driver out of domain 0.

This solution also maintains desirable properties of previous solutions, including high performance, fault isolation between VMs, and the ability to run unmodified guest applications.

2 Gallium3D and its Benefits

Gallium3D is a driver architecture developed by Tungsten Graphics which is ordinarily utilised as a labour-saving device for developers of ordinary, non-virtualised graphics drivers. It saves work by dividing graphics drivers into three modular layers: a State Tracker, which implements a graphics API and is hardware-independent, a Pipe Driver which is hardware dependent, API-independent and operating system independent, and a Window System Driver which is API-independent, hardware dependent and operating system dependent. This modular architecture permits multiple drivers to share state tracker modules, with each driver author contributing only a single pipe driver and a window system module per OS to be supported.

The use of a single state tracker by multiple driver authors means that the hardware-independent aspects of driver development effort can be shared. A Gallium-based architectural model uses these state-tracker modules unmodified, permitting it to ben-

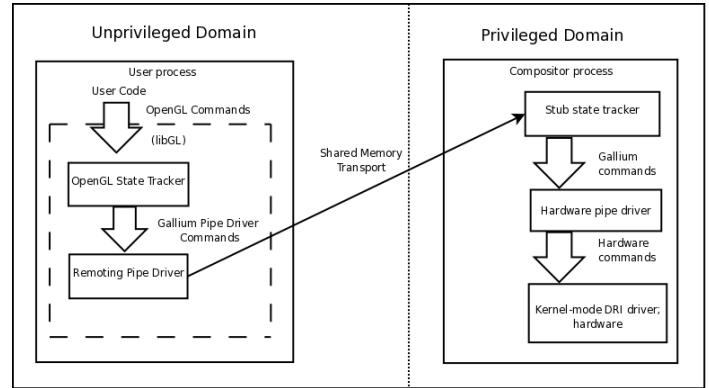


Figure 1: Simplified overview of Xen3D

efit directly from that shared effort and so rapidly gain support for new API revisions.

The Gallium pipe driver interface is also much thinner in terms of number of methods than both OpenGL and Direct3D. This should lead to reliable and maintainable remoting code. The use of the Gallium driver model also serves to export large quantities of code out of Domain 0 compared to previous solutions which needed to run a complete graphics driver as trusted code [1, 4].

3 Implementation

I implemented Xen3D, a set of virtual graphics drivers to run on guest VMs coupled with a compositor application run on the host system.

The system as currently implemented supports Linux guests running Xorg and a Linux host running any X server. It depends on the Xen hypervisor, because it uses Xen-specific shared memory mechanisms for interdomain communication. It currently only supports the OpenGL graphics API.

3.1 Architectural Overview

The system as implemented performs RPC-style API remoting, but does so at the level of the state tracker / pipe driver interface, rather than at the level of the graphics API used by the programmer. It comprises a modified libGL, which is linked by unmodified 3D applications running on the guest and which includes the OpenGL state tracker coupled with remoting code, and a compositor application run on the host, which executes remoted commands as though on behalf of a local process.

Drawing commands received by the Pipe and Window System layers of the client GL drivers are relayed to the compositor for drawing using a pair of simple shared memory ring-buffers. These buffers are directly accessible to both client application and compositor, and so, similarly to the VMWare Virtual GPU's command FIFOs [1], permit zero-copy command forwarding and batching.

The guest driver replaces local pointers with opaque handles

for safety, and optimises copying of vertex buffers and textures; otherwise, it simply marshals and remotes pipe driver calls. The guest also runs an Xorg extension which reports the coordinates, dimensions and clipping information relating to 3D client windows. 2D content drawn by this guest must then be integrated with the output of 3D clients to produce a complete display; this is achieved using Xen's existing paravirtualised framebuffer device.

The compositor, which runs on the host, is then responsible for accepting connections from guests' 3D clients, Xorg extensions, and instances of the Qemu device model and synthesising these different inputs to create a coherent display corresponding to each guest, as well as allowing the user to navigate amongst the displays of different VMs.

In response to 3D client commands, the compositor acts as though each is a 3D application running on the local system. It harnesses the Pipe Driver and Window System Driver components of a Gallium driver for the local graphics card, and creates a rendering context corresponding to each remote rendering context. It then executes drawing commands in the appropriate context as it services each remote client in a round robin fashion. By retaining control of these context switches, the compositor is able to prevent guest applications from modifying the state of other contexts and therefore breaking isolation. All rendering is directed to offscreen surfaces and then copied to the compositor's window or screen, again in order to avoid overruns which may break isolation.

Clipping information and 2D content from the guest is then used to reconstruct the guest's desktop as it would have been drawn locally.

4 Discussion and Lessons Learned

4.1 Generality and Portability

The implemented solution is very flexible. Remoting a different graphics API would be trivial, given the existence of a state tracker. Linking the GL state tracker with my client graphics drivers required no modification to the state tracker at all. Outside of the tracker module, the current prototype required only 21 lines of code which are graphics API specific. Because state trackers are used in all Gallium graphics drivers supporting a given API, the prospects of a tracker for Direct3D or future APIs is strong.

Remoting the 3D drawing of a guest running a different operating system would mean reimplementing the shared memory interface, substituting the GLX integration of the current Window System driver for similar WGL integration, and replacing the Xorg extension with a similar mechanism for extracting window coordinates and clipping information from Windows' graphics subsystem.

Running the compositor on a different operating system would be easy: the compositor as implemented interacts with its graphics driver using only the Pipe Driver interface, which is invariant across operating systems, and the EGL API for creating and managing rendering contexts, which is both cross-platform and graphics API independent. It could therefore be easily integrated with a different graphics driver supporting EGL running on any operating system.

Running on a VMM other than Xen would naturally require replacement of the shared memory transport; however, conversion from the TCP-based transport used during early development to the current shared memory transport took only around two weeks for a single person with no prior knowledge of Xen's mechanisms for inter-domain communication, so again the prospects for an easy port look promising.

4.2 Security Benefits

The Mesa3D/OpenGL state tracker used in the prototype is implemented by over 150,000 lines of code [2]; by comparison, the sample pipe driver for Intel's 915 graphics accelerator composes only 80,000 lines of code, of which 60,000 are shared libraries intended for all driver developers. Therefore a large proportion of driver code is moved out of the trusted computing base, and most of that which remains trusted is for broad use and so is less likely to retain bugs in the long term.

4.3 Improving on Gallium

Gallium's pipe driver interface broadly works well for remoting: it is thin, and so easy to remote, it lies close enough to the hardware that large amounts of code can be run in the guest VM, yet far enough from the hardware that it isn't tied to a particular vendor.

Unfortunately however, the Gallium Pipe Driver interface was not designed for RPC. Gallium has no support for *display lists* (a restricted form of downloadable code which permits the storage and invocation of long sequences of OpenGL commands), and Gallium features some pipe commands which return values affecting state tracker control flow. The only current state tracker mercifully ignores these return values, and so I am able to avoid the need to wait for the renderer to return on every draw command; however this could become a problem in the future. In the long term, a similar interface which eliminated these problems and could harness a Gallium state tracker with minimal modification would work around these issues. Similarly, in order to improve performance the causes of synchronisation between guest and host need to be minimised. This may be achieved using a thicker remoting layer which delays calls requiring a wait for the compositor, hoping to merge multiple waits into one.

In order to provide better isolation of guests, a quota system preventing excessive use of, for example, texture memory will be implemented. This will involve interposing on texture and buffer allocation methods in order to store these in main memory and load when appropriate, perhaps penalising clients which use most memory by allocating less time on the GPU.

5 Conclusion

The Gallium pipe driver interface itself is not perfect for use in remoting 3D rendering commands between VMs. However, I believe that its ability to make use of a single piece of remoting code and general purpose state trackers will mean that Xen3D is likely to see real use with new graphics APIs sooner than other open-source projects bound to specific APIs.

References

- [1] Micah Dowty and Jeremy Sugerman. GPU virtualization on VMware's hosted I/O architecture. In *First Workshop on I/O Virtualisation*, 2008.
- [2] Tungsten Graphics. Mesa3d source code. Available from [git://anongit.freedesktop.org/git/mesa/mesa](https://anongit.freedesktop.org/git/mesa/mesa).
- [3] Jacob Gorm Hansen. Blink: Advanced display multiplexing for virtualized applications. In *Proceedings of the 17th International workshop on Network and Operating Systems Support for Digital Audio and Video*, 2007.
- [4] H. Andres Lagar-Cavilla, Niraj Tolia, Mahadev Satyanarayanan, and Eyal de Lara. VMM-independent graphics acceleration. In *Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, 2007.
- [5] Parallels. Parallels desktop for mac: Video and 3d. Available from <http://www.parallels.com/products/desktop/features/3d/>.