VMGL: VMM-Independent Graphics Acceleration

H. Andrés Lagar-Cavilla, U of Toronto
andreslc@cs.toronto.edu
Niraj Tolia (CMU), Eyal de Lara (Toronto),
M. Satyanarayanan (CMU)
Why Virtualize 3D Acceleration?

Two simultaneous trends

- VMs out of the server room
- Client apps going 3D
Virtualization of Client Apps

- Soulpads
- The Collective
- Internet Suspend/Resume
- Virtual Appliances
- Moka5
The World Is Going 3D
Why Is 3D Virtualization Hard?

3D vendors compete through HW diversity
- Lack of unifying hardware abstraction
- Closed specs

Open HW abstractions simplify virtualization:
- Network -> Ethernet Frame
- Block Devices -> BIO request
- SCSI drives -> SCSI command packet
- ....

How could we ever write 3D applications?
3D Rendering APIs

De facto unifying software abstraction
Developer gets vendor independence

Two main APIs
- OpenGL
- Direct3D

OpenGL
- Cross-platform
VMGL: Virtualizing OpenGL

Provides 3D HW acceleration to applications running inside virtual machines

- GPU independent
- VMM independent
- Guest OS independent

- 87% or better of native HW acceleration
- Two orders of magnitude better than Mesa
VMGL Design

API virtualization
  • GPU vendor independence

OpenGL: cross-platform API
  • Guest OS independence

Network Communication
  • VMM independence
OpenGL Apps In X11 Systems

Application

Vendor Open GL

3D

X Server

2D

GPU
VMGL Apps in X11 Guest VMs

- X Server
- VMGL

Application

- VM Viewer: VNC, SDL
- VMGL Stub
- Vendor Open GL

- GL Commands
- Host: Dom 0

GPU

Guest

Host:

Dom 0
Implementation Aspects

- Efficient GL network transport
- 3D and 2D output composing in VM viewer
- Suspend/Resume implementation
- Dom0 drivers
Efficient GL Transport

Transport over network
- VMM Independence

WireGL / Chromium

Only send updates that "matter"
- `glTextureXY` only when texture visible

Combine, reorder and buffer commands
- `glRotate + glTranslate` -> Single matrix transformation
Window Manager Extension

Compose 3D and 2D output on VM viewer

Extension in VM's X server tells viewer 3D output
- Position
- Size
- Clipping
Suspend / Resume

Think each GL app as a GL device
• Runtime: keep track of OpenGL state
• Suspend: “freeze” GL device (trivial)
• Resume: flush state to new GL stub

OpenGL state is GPU independent
• Suspend/resume across different GPUs

OpenGL state is bounded
• Upper bound: GPU mem size
VMGL Suspend / Resume State

Windows
- Visual bits
- Binding to window manager extension

GL Contexts
- Context data: fog, transformations...
- Textures: pixmap, clamp mode
- Display Lists: verbatim unrolling
Domain 0 GPU Drivers

ATI & Nvidia:
- GPU Mem mapping in user-space GL lib

Oblivious to Xen additional indirection
- Virtual -> Physical (VM) -> Machine
- Even for domain 0

Fix open source portion of driver
Use Xen-paravirt mem mapping functions
VMGL Evaluation

VMGL: OpenGL Virtualization
• API v1.5
• Shaders through extensions

Frames per second
CPU, bandwidth consumption
Resume latency, state size

Workloads
• Games & entertainment fuel 3D industry
Workloads

- Quake 3
- Enemy Territory
- Unreal 2004
- Mplayer
Performance (FPS)
Performance (FPS)

- Quake 3
- Enemy
- Unreal
- Mplayer

Native
Mesa
Performance (FPS)

- 87% or better of native performance
VMM Portability (FPS)

- VMM and VM type independent
Guest OS Portability (FPS)

- Easily ported to other X11-based OSs
Suspend Resume Performance

- State size bounded
- Also across GPUs from different vendors
Wrapping Up

VMGL: OpenGL virtualization -> 1K downloads

Enable intersection of two growing trends
- Virtualization
- 3D Graphics

GPU/vendor independence
VMM independence
Guest OS independence

To appear @ VEE 2007
- More eval & details there
TODO

Xen-specific improvements
- Shared memory transport (XenSocket?)

Windows
- Code porting
- Window Manager hooks
- Direct3D support via translation layers
THANKS

Demo

Q&A

andreslc@cs.toronto.edu
www.cs.toronto.edu/~andreslc/vmgl/