Blink: 3D Display Multiplexing for Virtualized Applications

Jacob Gorm Hansen, University of Copenhagen

January 20, 2006

Introduction
- Motivation

History: The Tahoma Project
- Sprites and Tiles
- Lessons Learned

Blink
- GL in, GL out
- Communication Protocol
- JIT Compiler for OpenGL
- Stored Procedures
- Safety Checks

Evaluation
- Compiler
- Graphics Throughput

Demo

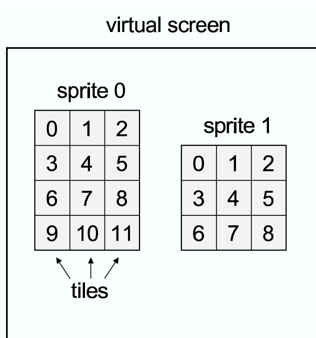
Why VM's need Graphic Acceleration

Virtual Machines are not only for data centers:

- VMWare now ships a free Firefox VM
- The Stanford Collective "Virtual Appliance" Application Bundles
- The CMU Internet Suspend Resume project
- Some distro installers need graphics
- We should be able to do better than VNC

Project Background

- Blink started as part of the “Tahoma” Browser OS
- First developed at the University of Washington, early 2005
- For *Tiled* 2D graphics
- Idea: Treat 4kB pages as 32x32 tiles, track updates with MMU
- Backend draws tile-grids (*Sprites*) to the screen with OpenGL



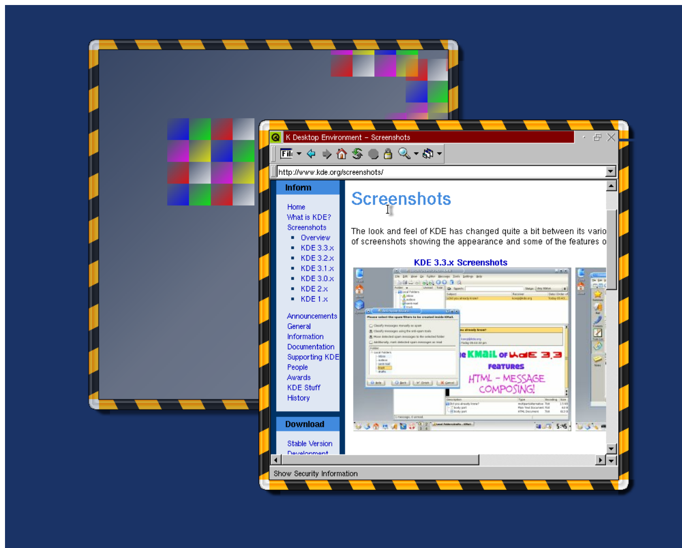
(a)

```
struct {
   int  tile_MFN;  // tile frame #
} tile;

struct {
   int   x, y;               // sprite posn.
   int   width, height  // # tiles
   tile  tile_array[ ];
} sprite;

struct {
   int       num_sprites;
   sprite  sprite_array[ ];
} virt_screen;
```

(b)


K Desktop Environment - Screenshots
http://www.kde.org/screenshots/
Inform
Home
What is KDE?
Screenshots
Overview
KDE 3.3.x
KDE 3.2.x
KDE 3.1.x
KDE 3.0.x
KDE 2.x
KDE 1.x
Announcements
General
Information
Documentation
Supporting KDE
People
Awards
KDE Stuff
History
Download
Stable Version
Screenshots
The look and feel of KDE has changed quite a bit between its vario
of screenshots showing the appearance and some of the features o
KDE 3.3.x Screenshots
Show Security Information

Lessons Learned

- Modern GPU's are extremely fast
- Tiles make update tracking easy, thus reduces bus traffic
- But most software expects a linear framebuffer
- And porting QT embedded to tiles was quite painful

A Natural Thought

So...

- If we are going to *output* OpenGL anyway...
- Why not just take OpenGL as input as well?
- Read serialized from Client
  - Interpret it
  - Check that it's safe
  - Execute it
- (Then we can always do the tiling in the client if we want)

Serialize OpenGL in Client

Turn:

```
glBegin(n);
```

into:

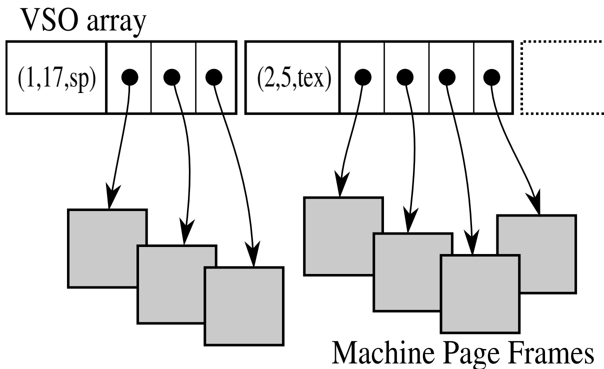
```
op->code=GL_Begin;
op->args[0]=n;
```

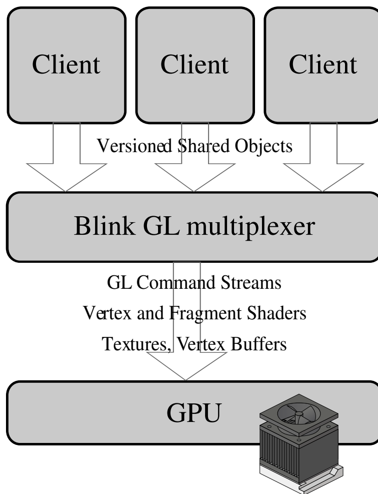
And Interpret it in the Server

```
switch(op->code)
{
    case GL_Begin:
        glBegin(op->args.integers[0]);
        break;
        ...
}
```

(turns in to a rather large switch() statement)

Versioned Shared Objects




Client
Client
Client
Versioned Shared Objects
Blink GL multiplexer
GL Command Streams
Vertex and Fragment Shaders
Textures, Vertex Buffers
GPU

A Couple of Improvements

- Turn the Interpreter into a JIT Compiler
- Add virtual registers, arithmetic, conditionals
- Call client code as *Stored Procedure* callbacks
- Results: Native speed asynchronous execution

List of Callbacks

Callback Name	Executed
`init()`	At first display
`update()`	On client VM request
`reshape()`	On window move or resize
`redraw()`	For each display
`input()`	On user input

Static Verification

Safety:

- We need to check certain properties during compilation...
- Not all callbacks are allowed to draw to screen
- Client should not exceed its *scissor* rectangle, or disable scissor

Performance:

- Advance knowledge of client actions can also be used for optimizing display
- E.g. if client does not enable Z-buffer, there is no need to clear it first
- If client does not use transparency, we do not have to draw windows behind it

Evaluation

Test Machine:

- 2GHz single-threaded Intel Pentium4 CPU
- 1024MB SDRAM
- 2+ years old
- ATI Radeon 9600SE 4xAGP graphics card with 128MB DDR RAM ($70).

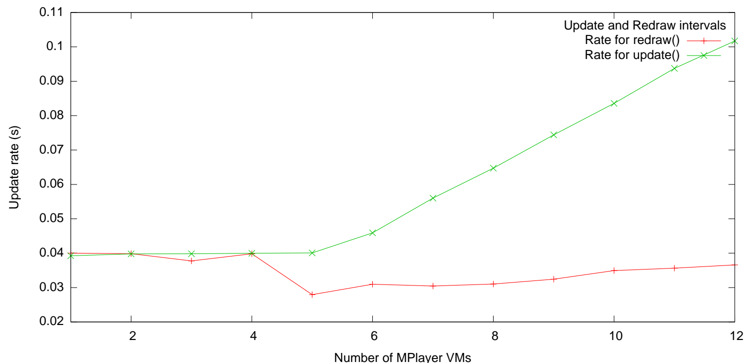
JIT Performance

Type of input	#Instr.	Compile	Execute
OpenGL-mix	8,034	102 (41) cpi	41 cpi
Arith-mix	8,192	99 (55) cpi	50 cpi

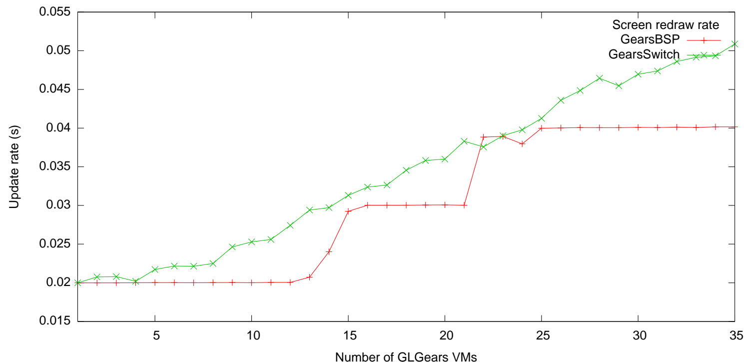
Quality of JIT'ed Code

Scenario	Execute
OpenGL-mix Native	552 cpi
OpenGL-mix Blink	554 cpi
OpenGL-mix Blink + JIT	656 cpi

MPlayer VM's



Gears VM's



Demo