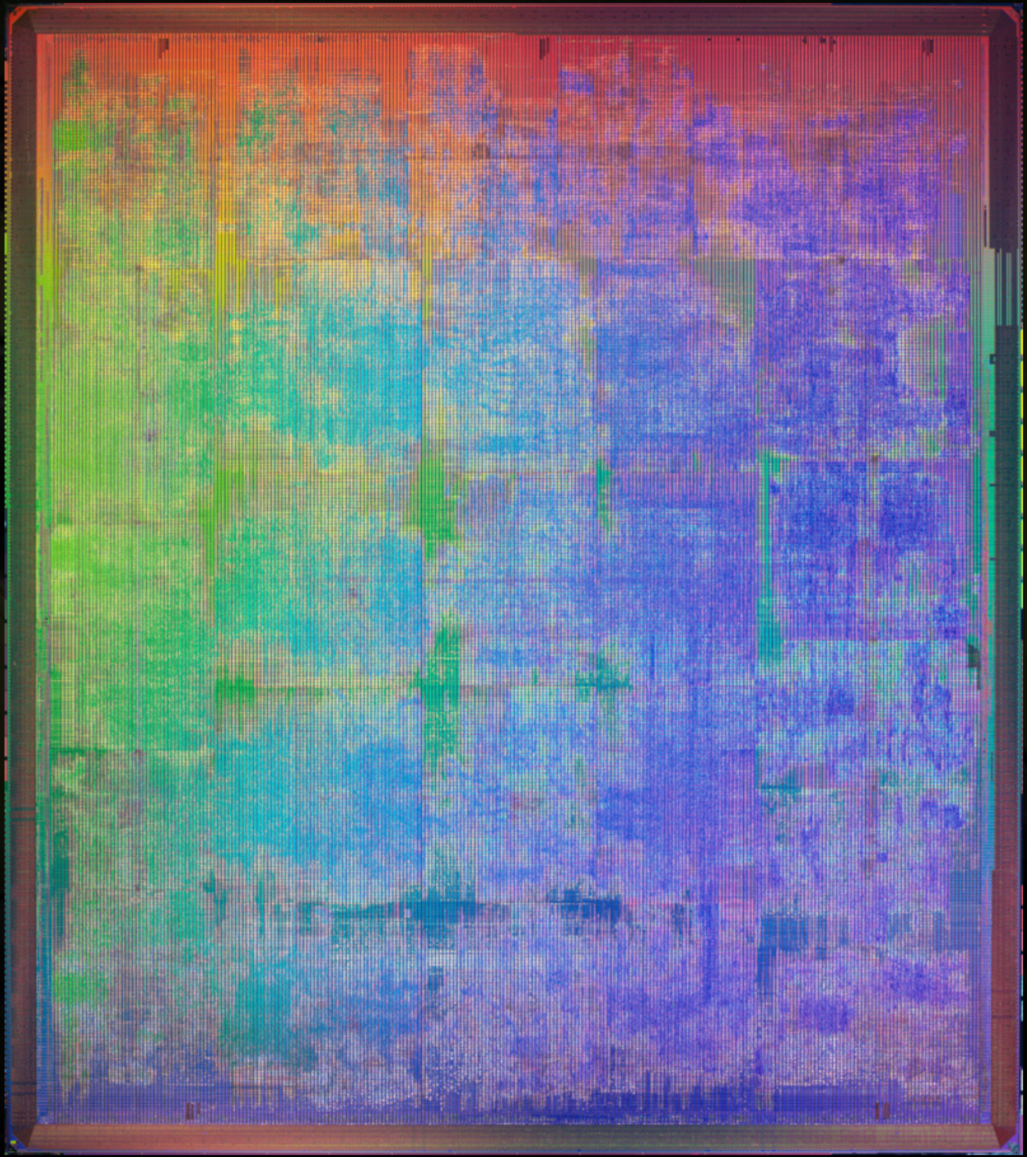
Windowing System on a 3D Pipeline

February 2005

Agenda

1. Overview of the 3D pipeline
2. NVIDIA software overview
3. Strengths and challenges with using the 3D pipeline

GeForce 6800 – 220M Transistors



April 2004

16 pix/clock, 256-bit mem interface

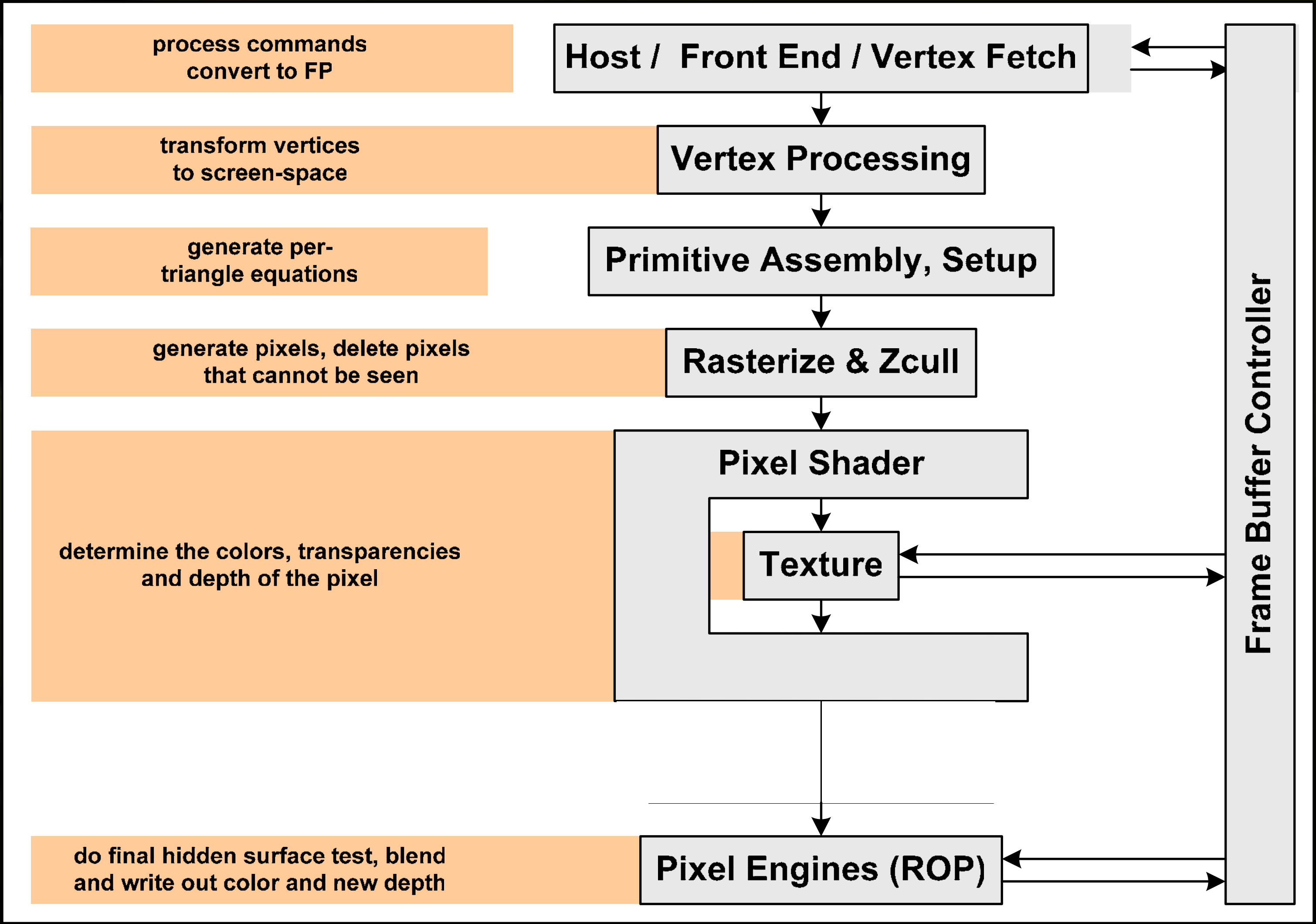
Scalable architecture

NVIDIA.

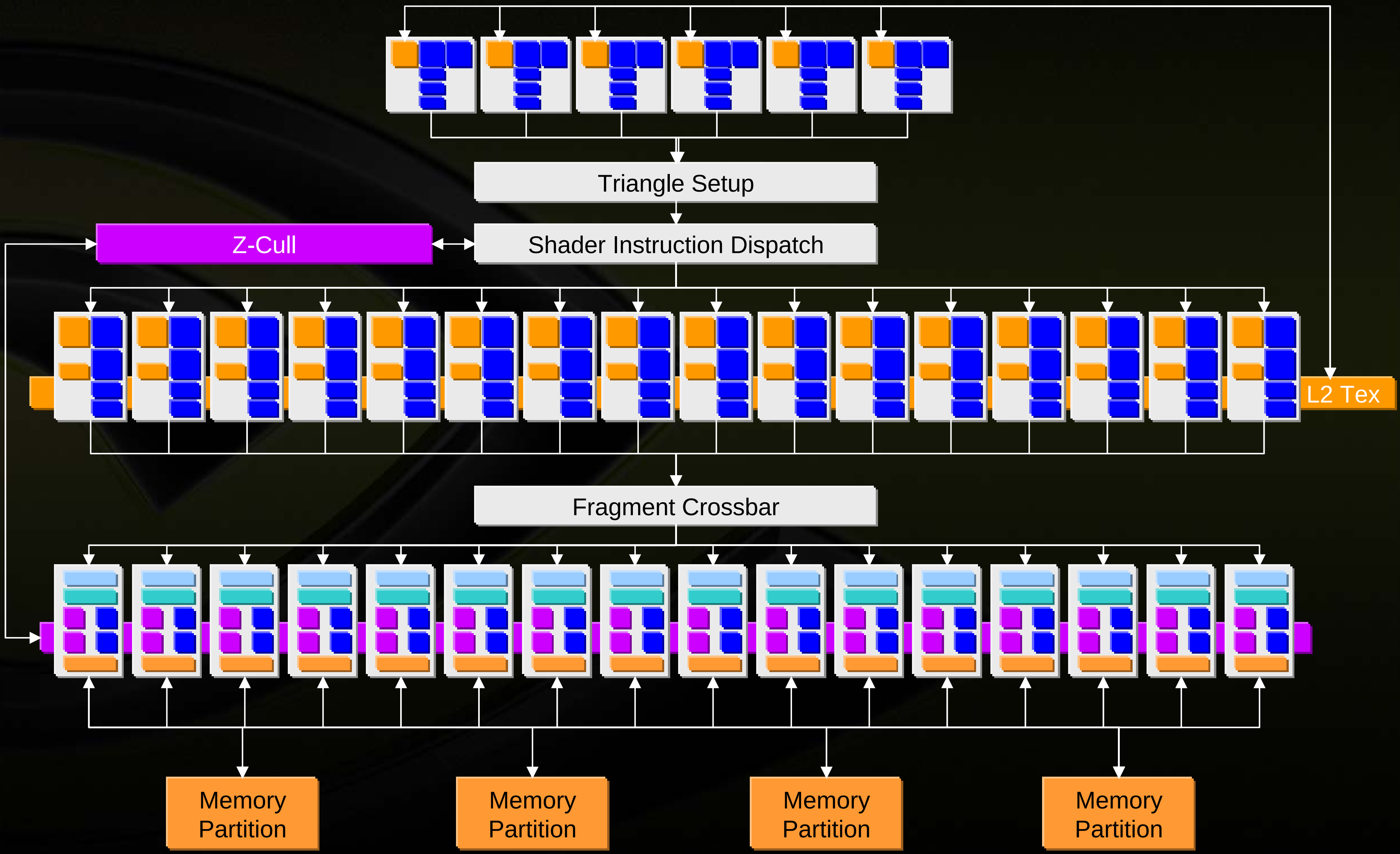
Overview of the 3D graphics pipeline

- Supported primitives: Points, Lines, Triangles
- Vertex positions can be determined by user-specified programs
- Primitives can be texture-mapped and Gouraud shaded
- Per-fragment color can be determined by user-specified programs
- Resultant pixel color can be (non-programmatically) blended with the frame buffer

The Life of a Triangle



GeForce 6800 series 3D Pipeline



Vertex processing

- Primitives are processed one vertex at a time
- Program determines position plus color and other interpolants
- All input types treated as float4 within program
- Instruction set includes MUL, ADD, MAD, SIN, COS, RSQ, (and TEX), etc.
- Up to 512 instruction long program
- Up to 64k instructions executed per vertex
- Output values are interpolated by HW, then fed to fragment processor
- MIMD execution units

Fragment processing

- Programs executed per-fragment, to generate colors to be written to 1 – 4 buffers
- Floating-point precision
- Instruction set includes MUL, ADD, MAD, SIN, COS, RSQ, numerous TEX, etc.
- Up to 16 textures can be bound, unlimited fetches from those texture maps
- Anisotropic filtering, min/mag, rotation, etc.
- Program length up to 8192 instructions
- SIMD execution units, across many fragments

Raster Operations (ROP)

- Z / stencil buffer operations
- Blending with frame buffer (up to fp16)
- Downsampling of AA samples into pixels
- Highly optimized for distributing workload across memory partitions

GeForce 6 speeds and feeds

- Peak FP Performance
  - Vertex Engine (FP32)
    - 6*5*2 * 400 MHz = 24 GFlops
  - Pixel Engine (FP32)
    - 16*4*3 * 400 MHz = 76 GFlops
  - Texture Math Engine (FP16)
    - 16*4*6 * 400 MHz = 154 GFlops
  - FP Blend (FP16)
    - 16*4*3 * 550 MHz = 106 GFlops
- Total = 260 FP16 & 92 FP32 GFlops

- Peak FB Bandwidth
  - 256-bit * 550 MHz DDR = 35.2 GB/s

- Host transfers (PCI Express system)
  - ~2.4 GB/s measured perf in each direction

Moving on to software ...

NVIDIA driver components

OpenGL client library	`libGLcore.so`
GLX	`libGL.so`
X component	`nvidia_drv.o`
kernel-loadable module	`nvidia.ko`

Overall architecture

- Kernel component responsible for
  - Chip init
  - Servicing interrupts
  - Allocating GPU resources to user-space drivers
- Client OpenGL driver
  - Optimize state changes
  - Compile / optimize vertex / fragment programs
  - Queue commands in “fire and forget” command buffer
- GLX
  - Allow OpenGL to interface with window system
  - Allocate visible and offscreen surfaces
  - Management of clip regions

NVIDIA’s kernel module

- Chip initialization
- Allocator for video memory
- Handles interrupts from GPU
- Pins / unpins host memory for DMA
- Raises OS events to user-mode drivers

NVIDIA’s X driver

- Obviously, accelerates X commands
- Surface allocation and management
- Manage window clips for 3D windows
- Acceleration of video decompression (XvMC)

NVIDIA’s OpenGL under the covers (1)

- Initialization
  - Allocation of rendering surfaces
  - Graphics state init
- State change
  - In most cases, update software shadowed copy of state
  - Mark dirty bit(s)
  - Return quickly
  - Texture loads can allocate memory dynamically
  - Matrix operations performed synchronously
  - Some instructions can be quite expensive

NVIDIA's OpenGL under the covers (2)

- Rendering command
  - Clean (validate) state for all dirty bits
    - Compile programs
    - Set potentially hundreds of state vectors
  - Move vertex data into DMA-visible regions
  - Enqueue rendering command
- DMA kickoff begins when:
  - Command buffer starts to fill
  - Driver notices hardware has gone idle
  - Also explicit for glFlush(), glFinish(), SwapBuffers(), etc.

Strengths of the 3D pipeline ...

- Minification / Magnification of textures
- Rotation of textures at full speed
- Blending / compositing
- Able to feed YCbCr or RGB surfaces (to some GPUs)
- Incredible flexibility in many parts of the pipeline

... But there’s also some problems

- State changes of 3D pipeline can be very costly
- High quality font rasterization best done by software
- 3D programs allowed to take arbitrary run time
- Functional variance from vendor to vendor
  - color, position interpolation
  - sub-pixel precision
  - texture sampling

... But wait! There's more!

- Loss of video overlay
- No indexed-8 output support
- Current driver model allows one app to be greedy
- Driver complexity grows significantly
- 3D rasterization rules don't match X rules

Challenge: Memory management

- Many threads can use OpenGL simultaneously
- Any one thread can allocate surfaces, textures, etc.

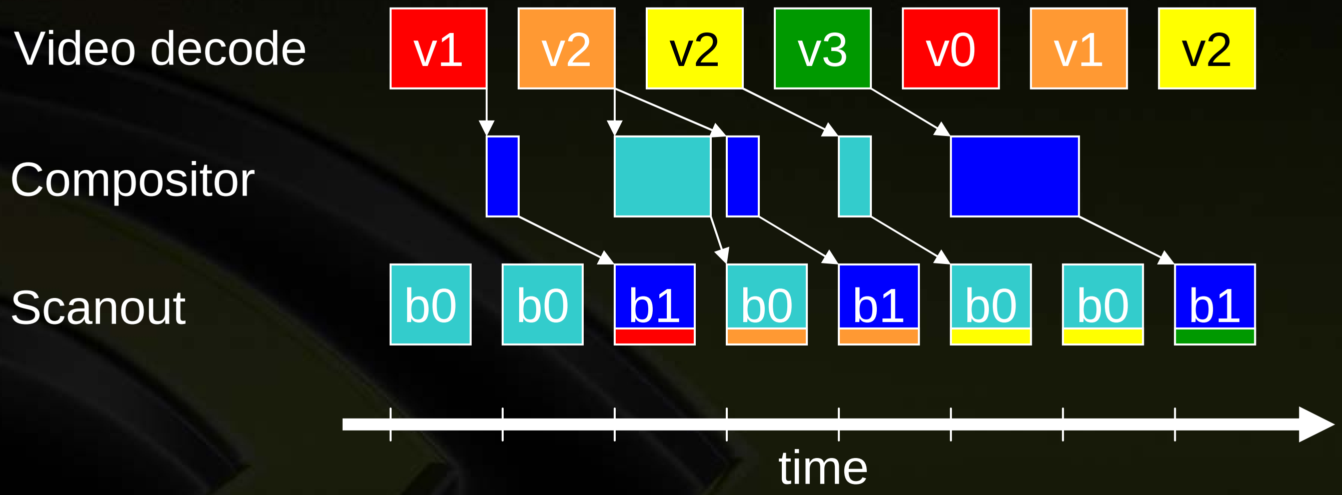
- How do you ensure any one process gets a fair share of the resources?

1. Pre-allocate buffers for a compositor
2. Page out surfaces when one app wants full control
3. Other ideas?

Tying in video

- Most MPEG processing typically done in YUV
- GeForce accelerates conversion of YUV to RGB
- Textures can also be stored in native YCbCr
- Video scaling traditionally performed by video overlay hardware

Challenge: Low latency video



PROBLEMS:

1. **Compositor stalls from only double-buffering**
2. **Video lags behind audio**
3. **Lots of video memory required**
4. **3D pipe is “fire-and-forget”, so compositor latency is not predictable**

Challenge: Keep the desktop responsive

- Some app decides to draw one triangle, with a very complex fragment shader program bound
- One triangle could cover millions of pixels
- A long program can run for tens of thousands of cycles per fragment

- Some simple math
  - 1600 x 1200 x 50000 = 96 billion cycles of work
  - GPU clock is ~400 MHz – 500 MHz
  - GPU processes up to 16 elements per cycle
  - 500 MHz * 16 pipes = 8 billion ops per second
  - 96 billion ops of work / 8 billion ops per second = 12 seconds (to draw one triangle!)

Summary

- With the right software design, layering a windowing system atop 3D is possible
- The 3D pipeline can offer great flexibility and functionality
- There are a lot of very challenging problems which must be addressed
- NVIDIA would be happy to work with you on this design

Contact

Nick Triantos

nick@nvidia.com