

# Optimizing DirectX Graphics

**Richard Huddy**

**European Developer Relations Manager**



# Some early observations

- Bear in mind that graphics performance problems are both commoner and rarer than you'd think...
- The most common problem is that games are CPU-limited
- But you can certainly think of that as a graphics problem...



# Shall we have a target?

- 1280x1024 (at least)
- 85Hz (then lower refresh rates will just work)
- 4xAA or 6xAA - so pixels can look good
- Because of the variability of the platform it makes no sense to ask blindly if we are pixel-limited or vertex-limited etc.

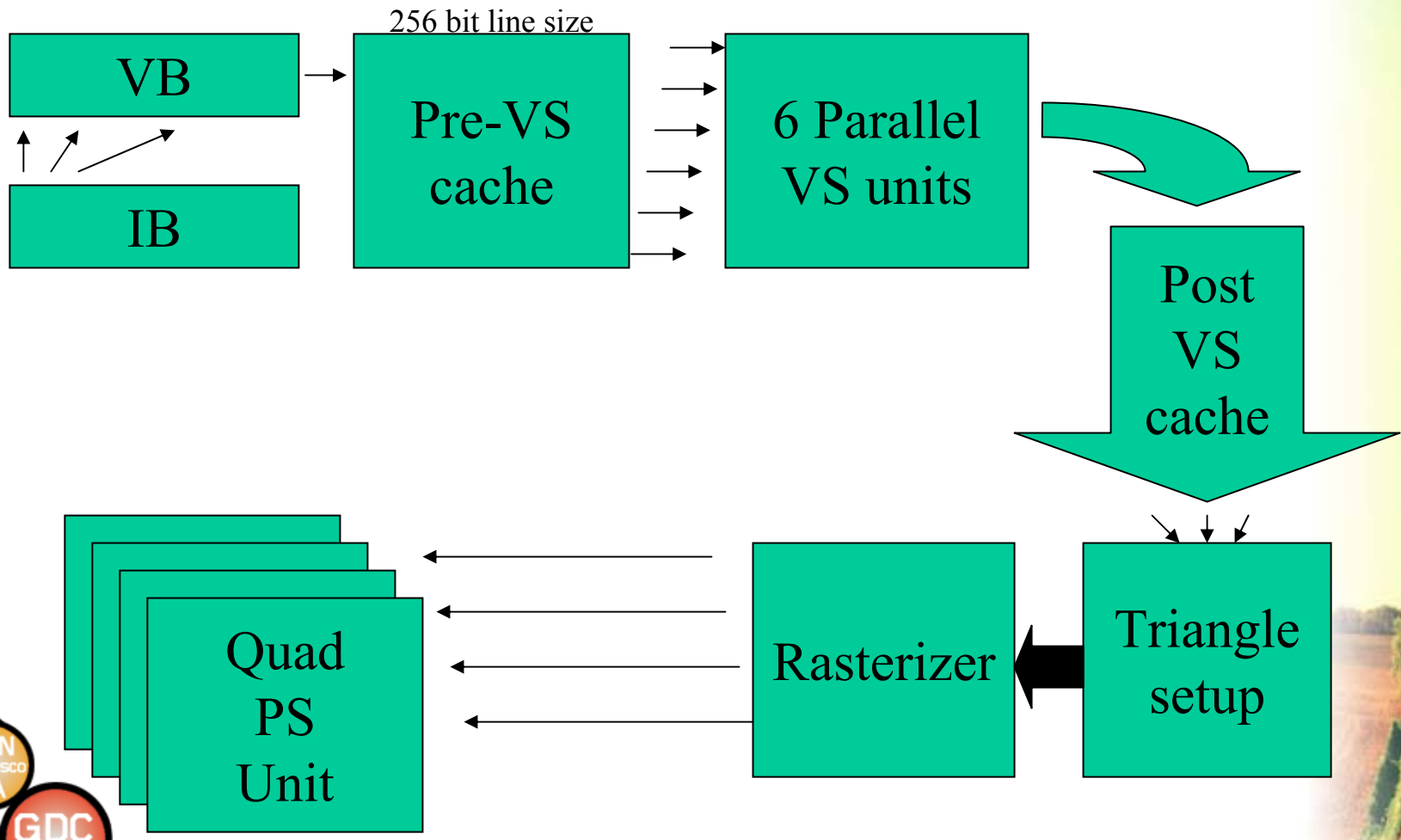


# So, what shall we tune for?

- **Cache re-use**
  - **VFetch, Vertex, texture, Z**
    - All caches are totally independent of each other...
- **Vertex shaders**
- **Pixel shaders**
- **Z buffer**
- **Frame buffer**



# The high-end hardware pipeline...



# The pre-VS cache I

- Is purely a memory cache
- Has a common line size of 256 bits
  - (That's 32 bytes)
- Is accessible by all vertex fetches
- Is why vertex data is best aligned to 32 bytes or 64 bytes
  - 40 is very much worse than 64
  - Truly sequential access would be great...!



# The pre-VS cache II

- **Because it's purely a memory cache...**
  - **Multiple streams can both help and hinder.**
    - **Multiple streams with random access is doubly bad...**
  - **Generally expect 0% to 10% hit for using additional streams**



# Vertex Engines 0

- Consider compressing your vertex data if that helps you line things up with the 32 byte cache line...
  - Decompress in the Vertex Shader
  - Store compressed data in VB
- See previous slide for the point...
- This can be a significant win if it achieves alignment objectives



# Vertex Engines I

- On any modern hardware these generally dwarf the task
- Just don't do mad things here!
- 6 vertex engines is plenty
- Software is another matter:-
  - Integrated solutions...
  - DX7 parts...



# Vertex Engines II

- HLSL is your best approach...
- Expect one op per clock per pipe
  - Sometimes you'll get 2 ops instead...
  - Masking out unused channels helps
  - You can get up to 5 ops at once!
- I've never seen a game which is vertex-throughput limited at interesting resolutions on modern hardware



# The post-VS cache

- Only accessible when using indexed primitives (can give you 'free' triangles)
- Operates as a FIFO
- Use `D3DXOptimizeMesh()`
- ATI: Is 14 entries for triangles, 15 for lines and 16 for points
- NV: Is 16 entries on GF2 & GF4MX, 24 entries on all shader hardware
- Cache Size is independent of vertex format!
- Use highly local wending for best results
- Flushed between `DrawPrim()` calls

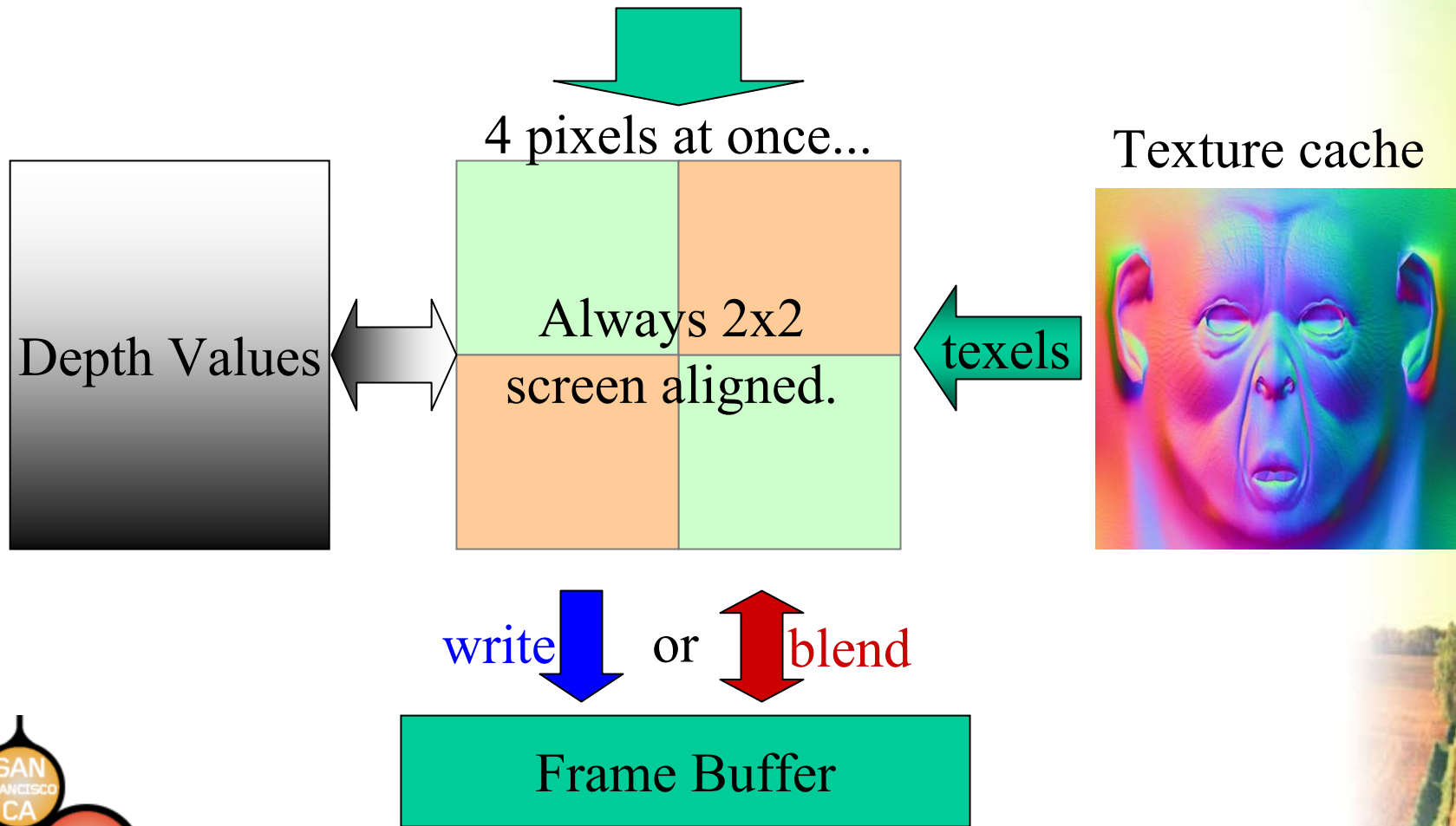


# Triangle setup

- Never a bottleneck
- Just joins vertices into triangles
- Feeds the rasterizer which simply hands out quad pixel blocks to draw



# Each Quad-Pixel Shader Unit



# Texture cache

- Probably smaller than you'd think...
  - Unless you thought “only a few KB”
- Partitioned over all active textures
  - So heavy multi-texturing can really hurt
- Wrecked by random access!
  - Often from bump-map into env-map
  - Needs reuse to show benefits (i.e. don't minify!)
- Usually contains uncompressed data
  - At 8, 16, 32 or more bits per texel
  - NV shader h/w stores DXT1 in compressed format
- Texture fetches are per-pixel



# Making Z work for you...

- **We're faster at rejecting than at accepting...**
  - **So draw roughly front to back**
  - **For complex scenes consider Z pre-pass (not for depth\_complexity=1!)**
  - **Take care to Clear() Z (and stencil)**
- **Although Z is logically at the end of the shader that's not the best way**



# Making Z work for you...

- Note that NV hardware can do double speed Z/Stencil only work when:
  - Color-writes disabled
  - 8-bit/component color buffer bound (not float)
  - No user clip planes
  - AA disabled
    - Good for shadow rendering
  - That's up to 32 zixels per clock



# Making Z work for you...

- Note that ATI hardware can do double speed Z/Stencil only work when:
  - Color-writes disabled
  - AA enabled
    - Good for general rendering
  - That's up to 32 AA samples per clock



# Depth Values

- Can come from:-
  - Actual Z buffer (slow)
  - Compressed Z (fast & lossless)
- Your pixel can be Z-tested away before the shader has run at all!
- If you are performing any Z compare then please try hard not to write to oDepth
- Depth values are per-sample...



# Bashing the depth buffer

- You can reduce the huge(\*) early Z benefits by...
  - Writing oDepth
    - Hits compressed Z and early Z
  - Using alpha-test etc on visible pixels
    - decompresses Z values
  - Changing the Z compare mode (sometimes)
    - Can disable Hi-Z
    - E.g. from LESS to GREATER

(\*) Top class h/w can reject 256 pixels per clock!



# The PS Unit I

- **Shorter shaders generally faster**
  - Older NV hardware also benefits from smaller register footprint
- **At the high end there is roughly 4 times as much ALU power as texture power**
- **This ratio will only go up**
  - Because available bandwidth doesn't rise as fast as chip density
- **So generally push more maths into here**



# The PS Unit II

- Is a 4D vector processor
  - So try to match your math to your needs
    - i.e. Mask out unused channels
- Trust the compilers to schedule things well:-
  - You don't worry about scheduling...
- PS runs once per pixel...



# FB (Fog and) Blend

- Is not part of the PS unit
  - You can think of it as a special function of the memory controller
- Although there are lots of latency hiding tricks here...
  - This is still probably the easiest place to get B/W limited
- So disable blend whenever possible



# Pure FB optimisations

- Fewer bits are written faster...
  - 16BPP > 32BPP > 64BPP > 128BPP
    - (here '>' means faster)
- Blending is slower than not
  - By more than a factor of 2
- ATI: Surfaces are 'faster' if allocated earlier!



# Conclusion...

- **Several classes of optimisation:**
  - **Pushing things back up the pipe:**
    - E.G. Cull early, not late
  - **Getting better parallelism:**
    - E.g. Use write masks to allow SIMD
  - **Doing less is faster than doing more:**
    - E.g. Short shaders are faster
  - **Understand what is cached:**
    - 32 byte vertices are fast! 16 bytes are faster...

