



Optimizing for DirectX Graphics

Richard Huddy
European Developer Relations Manager

Also on today from ATI...

- **Start & End Time:** 12:00pm – 1:00pm
Title: Precomputed Radiance Transfer and Spherical Harmonic Lighting Techniques for Games
Speaker: Chris Oat
- **Start & End Time:** 2:30pm – 3:30pm
Title: Bringing Hollywood to Real-time
Speaker: Abe Wiley
- **Start & End Time:** 4:00pm – 6:15pm (short break at 5:00pm)
Title: Effects Breakdown - How'd They Do That
Speakers: Thorsten Scheuermann, Natalya Tatarchuk, Daniel Ginsburg
- And Jonathan Zarge is presenting our tools strategy on the ATI stand at 6PM Thursday including some new performance tools...



Early observations

- Graphics is a hard problem
- Look at the trends
 - Mem B/W doesn't grow fast enough
 - We're having to generate more heat
 - Graphics Chip Complexity rises much faster than CPU Complexity
- Although we're doing a lot to improve the straight-line speed people want to turn more corners

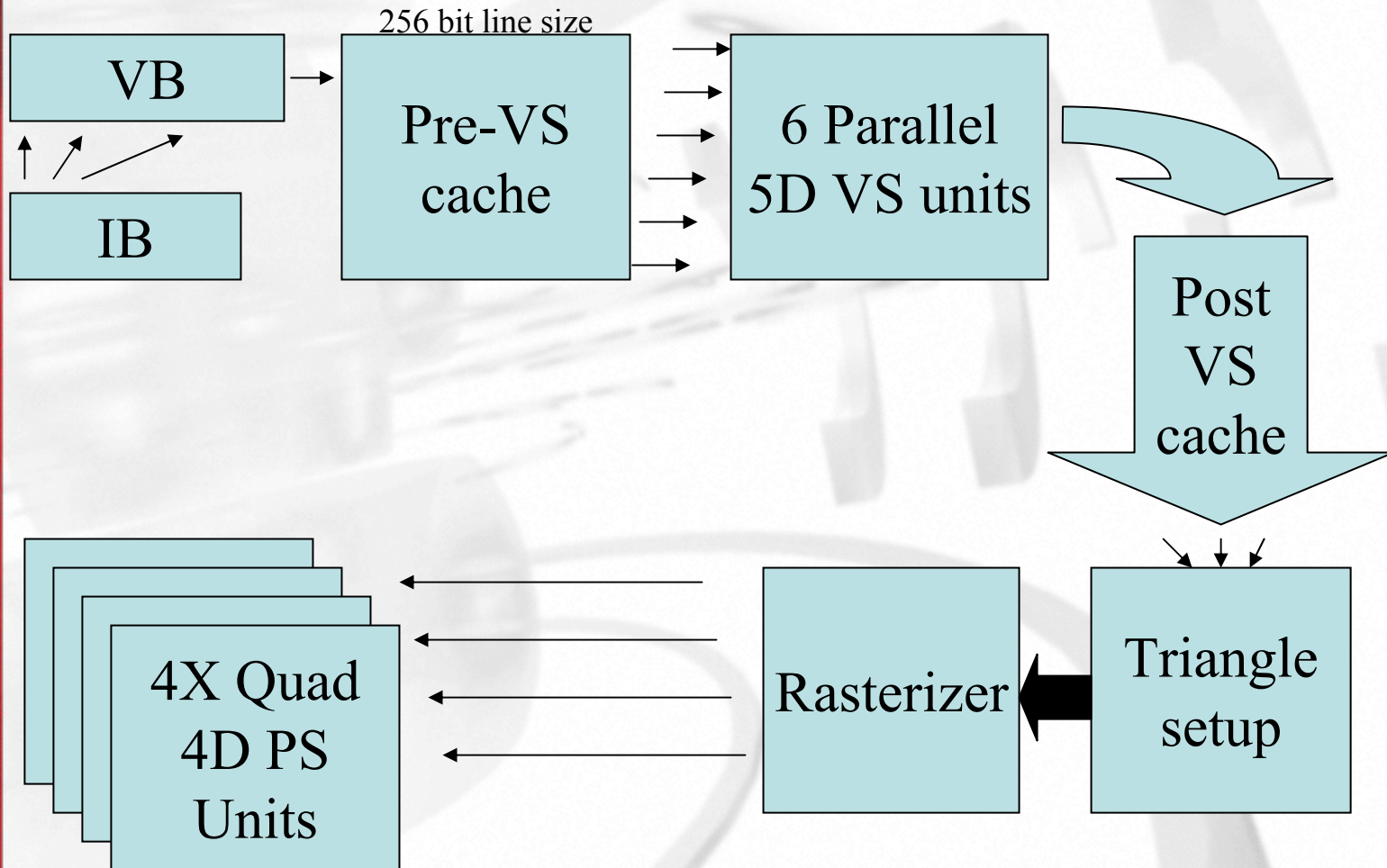
High End hardware's target?

- 1600x1200
- 85Hz (then lower refresh rates will just work)
- 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...
 - i.e. Writing to one never invalidates another
- Vertex shaders
- Pixel shaders
- Z buffer
- Frame buffer

The X800/X850 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
- It is surprisingly easy to get bottlenecked here by unfriendly data
 - That's because unfriendly data really can require 10x as much b/w as friendly data
- Games (and benchmarks) do get stuck here

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
 - delivering over 700 million vertices per second
- Software is another matter:-
 - Integrated solutions...
 - DX7 parts...
 - You may struggle to get 7 million vertices per second

Vertex Engines II

- HLSL is your best approach...
- Expect one op per clock per pipe
 - Often you'll get 2 ops instead...
 - But you won't know that
 - X800 contains both a 4D and 1D vertex ALU which can both do useful work in the same clock cycle
 - So be specific about scalar operations...
 - Masking out unused channels helps
- I've never seen a game which is vertex-throughput limited at interesting resolutions on modern hardware
 - Particularly true at our target settings of 1600x1200 and 6xAA



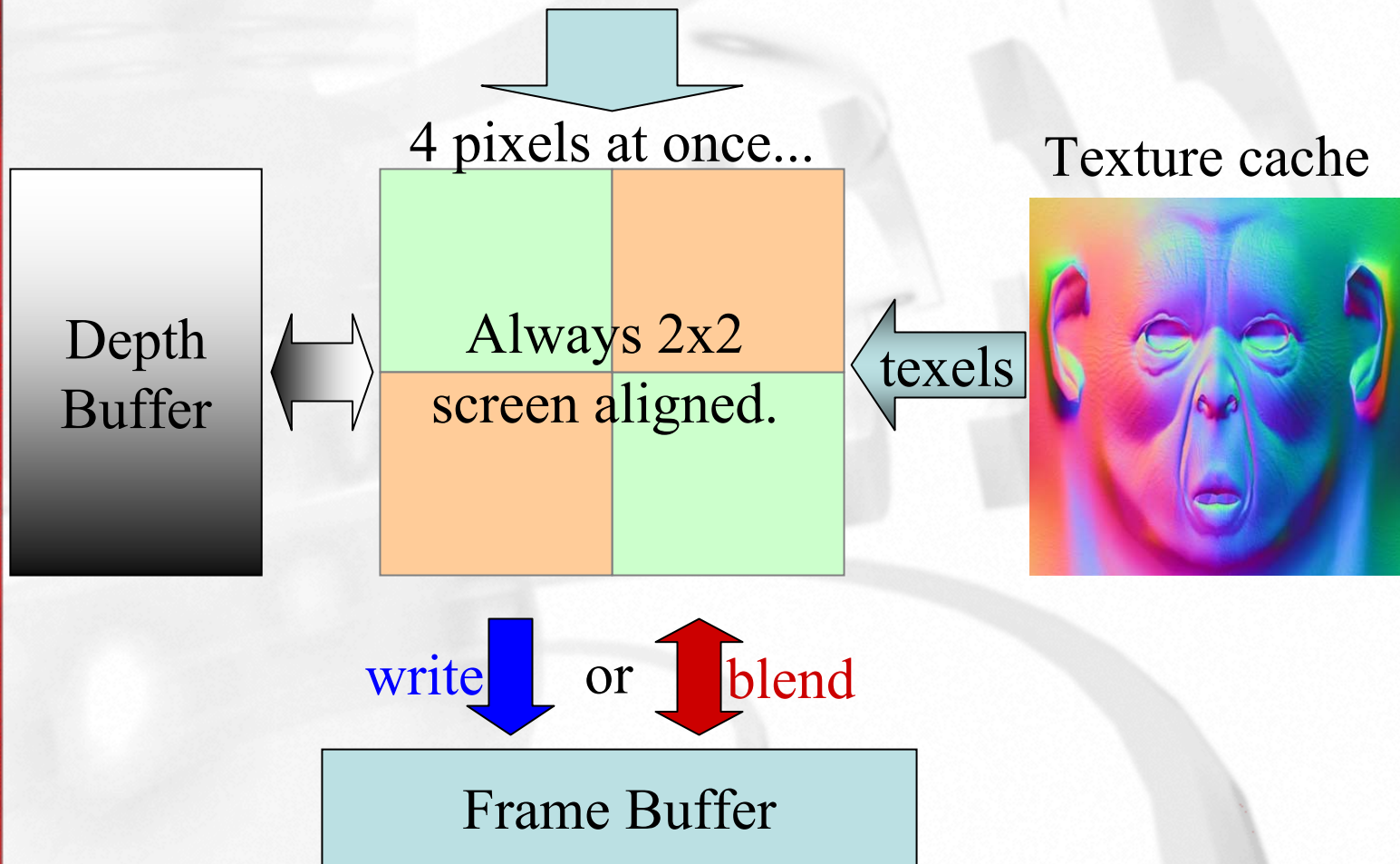
The post-VS cache

- Only accessible when using indexed primitives (can give you 'free' triangles)
- Operates as a FIFO
- Use `D3DXOptimizeMesh()` knows about the current hardware
 - Which means you must not pre-process this!
- Cache has 14 entries for triangles, 15 for lines, 16 for points
- Cache Size is independent of vertex format!
- Use highly local winding 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 I



Each Quad-Pixel Shader Unit II

- The Quad is allocated to just one triangle at a time
 - But each quad might be working on unrelated triangles
 - Which means a quad could be 25% efficient if the triangle is an inconvenient size...
 - Small triangles make good use of the vertex pipe, but at the expense of the pixel pipe efficiency
- Shares all gradient calculations etc

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!)
- Contains uncompressed data
 - At 8, 16, 32 or more bits per texel
- 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 for the hardware to work

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



(*) X800 etc can reject 256 pixels per clock!

The PS Unit 0

- A basic optimisation here is to avoid doing work in the pixel shader which can be done earlier in the pipeline
 - E.g. Never multiply two constants together in the PS, instead do that at the earliest possible point back up the pipe
 - If you can do it in the VS, do it there, not in the PS (because pixels are commoner than vertices)

The PS Unit I

- Shorter shaders generally faster
 - And we store multiple shaders in the shader cache which makes switching faster too
- 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!

Some cases...

- 3DMark05
 - Sometimes Vertex-Fetch limited
 - Large batches, good efficiency
- Un-named games:
 - 85 fps no matter what...
 - 9.9 fps “leaps” to 10.1 fps!
 - Wrong flags push vertex data through S/W Vertex Engine

Conclusion...

- Several classes of optimisation:
 - Push things back up the pipe:
 - E.g. Cull early, not late
 - Get 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. Indexed random access is v bad!