DirectX 9 Performance

Where does it come from, and where does it all go?

Matthias Wloka
MWloka@nvidia.com

Richard Huddy
RHuddy@ati.com

Where does it all come from?

- Complex pixel shader support (2.0 and better)
- Multi-GigaPixel fill-rates
- 100's of millions of triangles per second of geometry throughput
- Multi-sample anti-aliasing
- High Quality texture filtering

Where does it all come from?

- So games should be able to run at 1280x1024 at refresh rate with AA and high quality filtering enabled at all times…

- Yes?

But where does it all go?

- Too many apps are CPU limited…
  - They send too many state changes
  - They don't batch their triangles
  - They misuse vertex buffers
  - They lock critical resources at bad times
- Many games don't push graphics as hard as they could…
  - Which is a shame because reviewers (and future gamers) have high end hardware

Last year I suggested this target

- DX9 style mainstream graphics:
  - > 0.5 million polys per frame
  - < 500 DIP calls
  - < 500 VB changes
  - < 200 texture changes
  - < 200 State change sets
  - "Few" SRT calls (that's single digits...)
  - 1 pass per poly is typical, but 2 is sometimes smart
  - Runs at refresh rate of 80Hz or better
  - That's better than 40 million polys per second
    - And nothing goes through the fixed function pipes

Pass Reduction ("*PR*")

- **Use the most powerful shader available to reduce the total number of passes required to render a given thing to a given standard.**
  - Use ps2.0 to *PR* 1.x techniques
    - Usually n passes -> 1 pass
  - Use ps1.4 to *PR* 1.1 – 1.3 techniques
    - Commonly 2 passes -> 1 pass
  - Use ps1.x to *PR* fixed function techniques

General resource management

- Create your most important resources first. That's targets, shaders, textures, VB's, IB's etc
- "Most important" is defined as "most frequently used"
- Never call Create in your main loop

  - So create the main colour and Z buffers before you do anything else…
    - The "main buffer" is the one through which the largest number of pixels pass…

Sorting

- Sort roughly front to back
  - There's a staggering amount of hardware devoted to making this highly efficient
- Sort by vertex shader

  ...or...

    - Sort by pixel shader, or
    - sort by texture
- When you change VS or PS it's good to go back to that shader as soon as possible...
- Short shaders are faster^2 when sorted

Clearing

- **Ideally use Clear once per frame**
  - **Always clear the whole render target**
  - **Always clear colour, Z and stencil together <u>unless</u> you can just clear Z/stencil**
    - **Don't force us to preserve stencil if you don't need it…**
- **Don't use 2 triangles to clear…**
- **Using Clear() is *<u>the</u>* way to get all the fancy Z buffer hardware working for you**

Vertex Buffers

- Use the standard DirectX8/9 VB handling algorithm with DISCARD & NOOVERWRITE etc
- Specify write-only if possible
- Use POOL_DEFAULT if possible
- Roughly 2 – 4 MB for best performance
  - This allows large batches
  - And gives the driver sufficient granularity to manage memory efficiently

Index Buffers

- **Treat Index Buffers *exactly* as if they were vertex buffers – except always choose the smallest element possible**
  - **i.e. Use 32 bit indices only if you need to**
  - **Use 16 bit indices whenever you can**
- **Much recent hardware treats Index Buffers as 'first class citizens'**
  - **They don't have to be copied about before the chip gets access**
  - **So keep them out of system memory**

Updating Index and Vertex Buffers

- **IBs and VBs which are optimally located need to be updated with sequential DWORD writes.**
- **AGP memory and LVM both benefit from this treatment…**

Handling Render States

- **Prefer minimal state blocks**
  - ‘minimal’ means you should weed out any redundant state changes where possible
    - If 5% of state changes are redundant that’s OK
    - If 50% are redundant then get it fixed!
- **The expensive state changes:**
  - Switching between VS and FF
  - Switching Vertex Shader
  - Changing Texture

How to draw stuff

- DrawIndexedPrimitive(strip or list)
  - Indexing is a big win on real world data
  - Long strips beat everything else
  - Use lists if you would have to add large numbers of degenerate polys to stick with strips (more than ~20% means use lists)
  - Make sure your VB’s and IB’s are in optimal memory for best performance
  - Give the card hundreds of polys per call
    - Small batches murder your performance

Vertex data

- Don’t scatter it around
  - Fewer streams give better cache behaviour
- Compress it if you can
  - 16 bits or less per component
  - Even if it costs you 1 or 2 ops in the shader…
- Try to avoid spilling into AGP
  - Because AGP has high latency
- pow2 sizes help – 32 bytes is best
  - Work the cache on the GPU
- Avoid random access patterns where possible by reordering vertex data before the main loop…
  - That’s at app start up or at authoring time

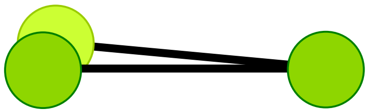
What Is a Batch?

- Every DrawIndexedPrimitive() is a batch
  - Submits n number of triangles to GPU
  - Same render state applies to all tris in batch
  - SetState calls prior to Draw are part of batch
- Assuming efficient use of API
  - No Draw*PrimitiveUP()
  - DrawPrimitive() permissible if warranted
  - No unnecessary state changes
- Changing state means at least two batches

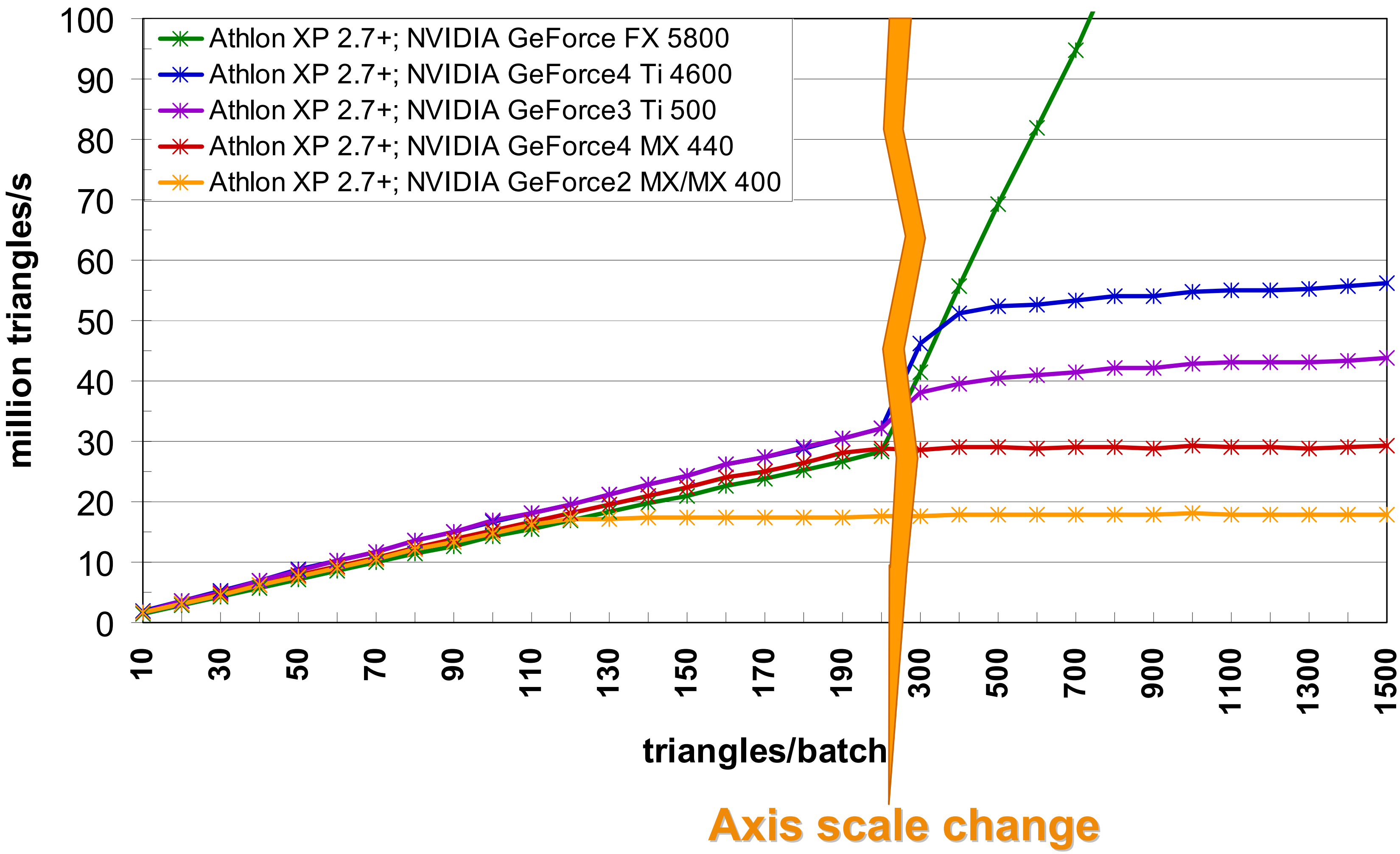
Why Are Small Batches Bad?

- Games would rather draw 1M objects/batches of 10 tris each
  - versus 10 objects/batches of 1M tris each
- Lots of guesses
  - Changing state inefficient on GPUs (WRONG)
  - GPU triangle start-up costs (WRONG)
  - OS kernel transitions (WRONG)
- Future GPUs will make it better!? Really?

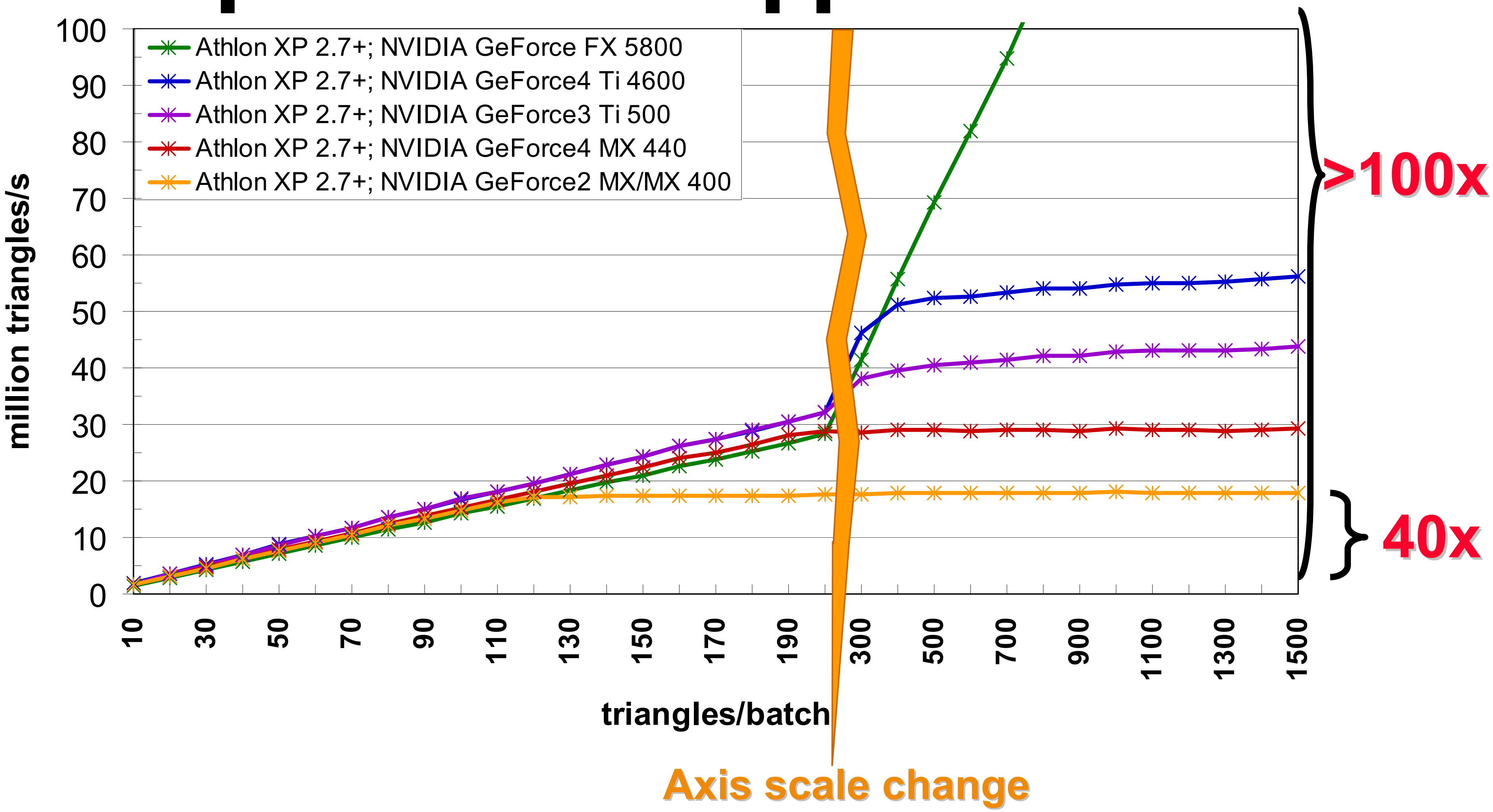
Let's Write Code! Testing Small Batch Performance

- **Test app does…**
  
  - Degenerate triangles (no fill cost)
  - 100% PostTnL cache vertices (no xform cost)
  - Static data (minimal AGP overhead)
  - ~100k tris/frame, i.e., floor(100k/x) draws
  - Toggles state between draw calls:
    (VBs, w/v/p matrix, tex-stage and alpha states)
- **Timed across 1000 frames**
- **Theoretical maximum triangle rates!**

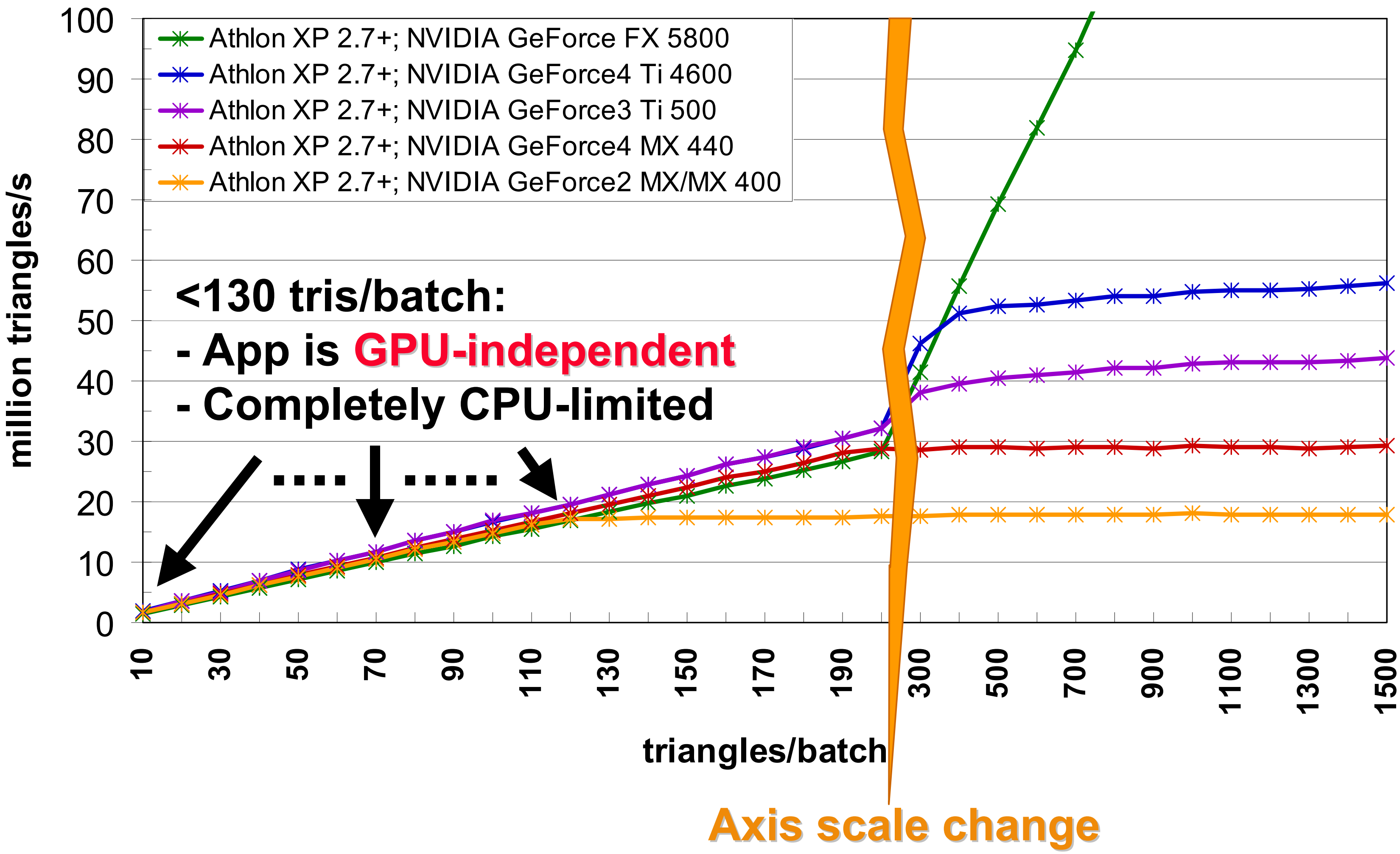
Measured Batch-Size Performance



Optimization Opportunities



Measured Batch-Size Performance



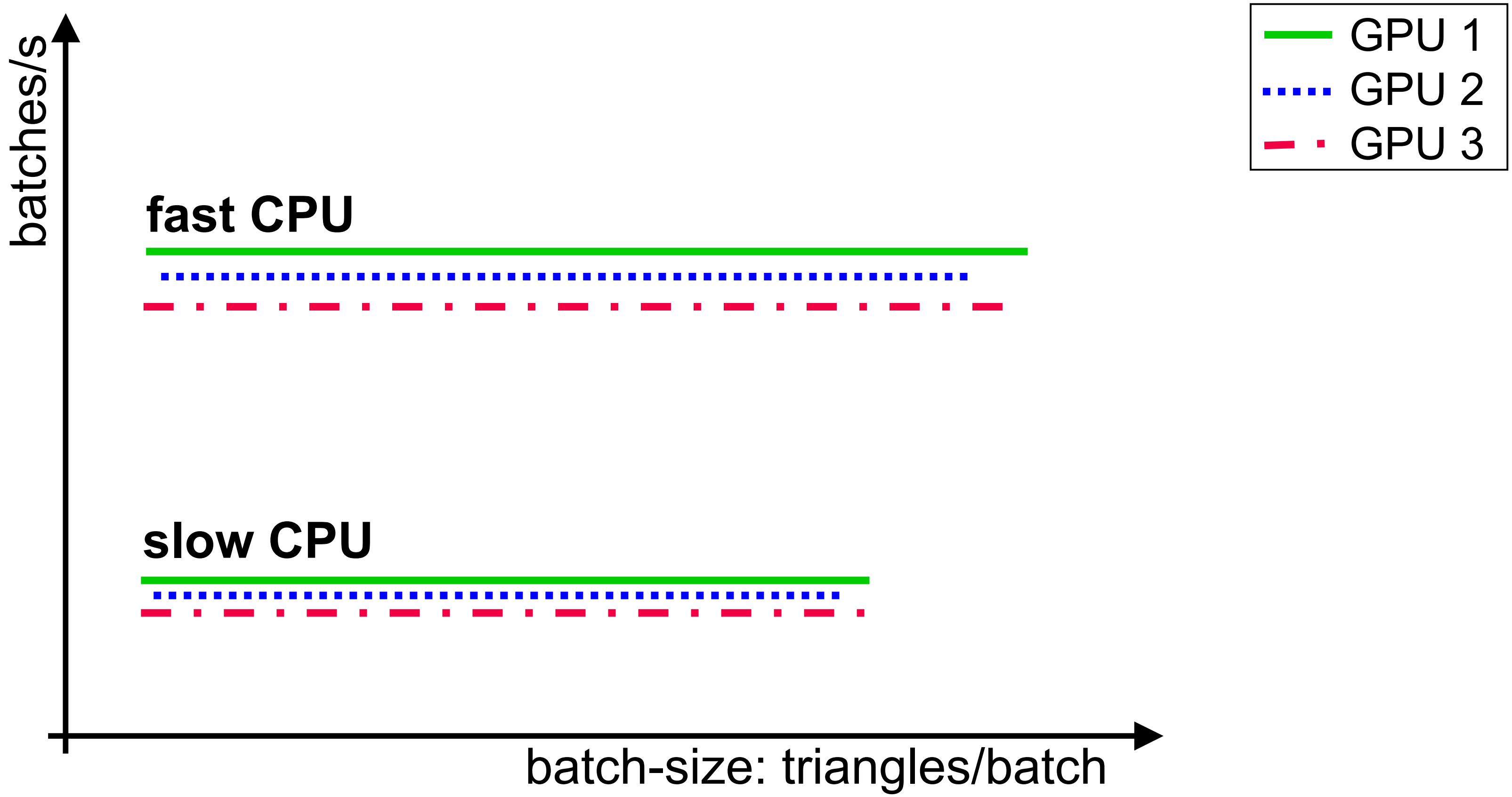
CPU-Limited?

- Then performance results only depend on
  - How fast the CPU is
    - Not GPU
  - How much data the CPU processes
    - Not how many triangles per batch!

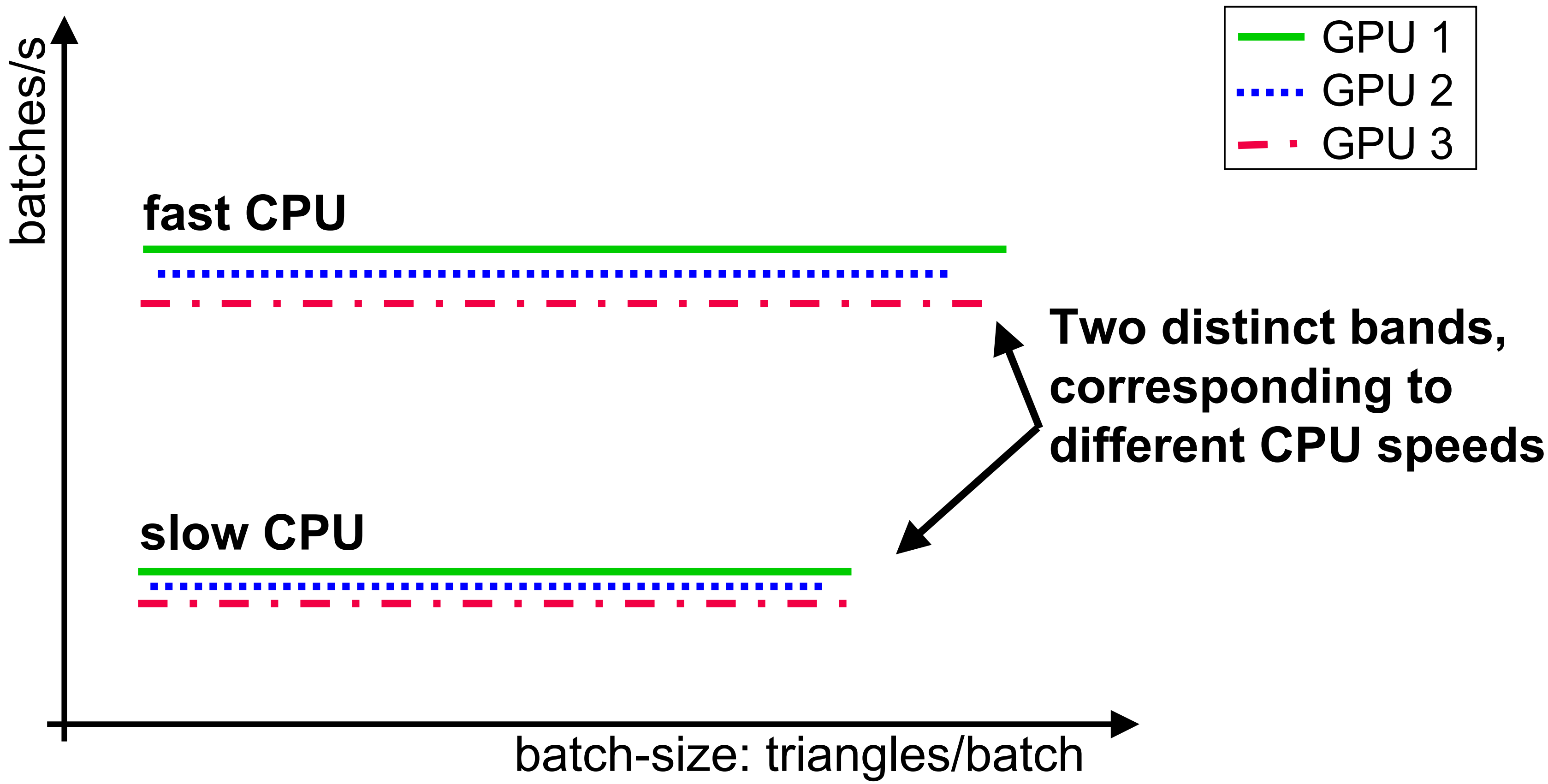
- CPU processes draw calls (and SetStates), i.e., batches

- Let's graph batches/s!

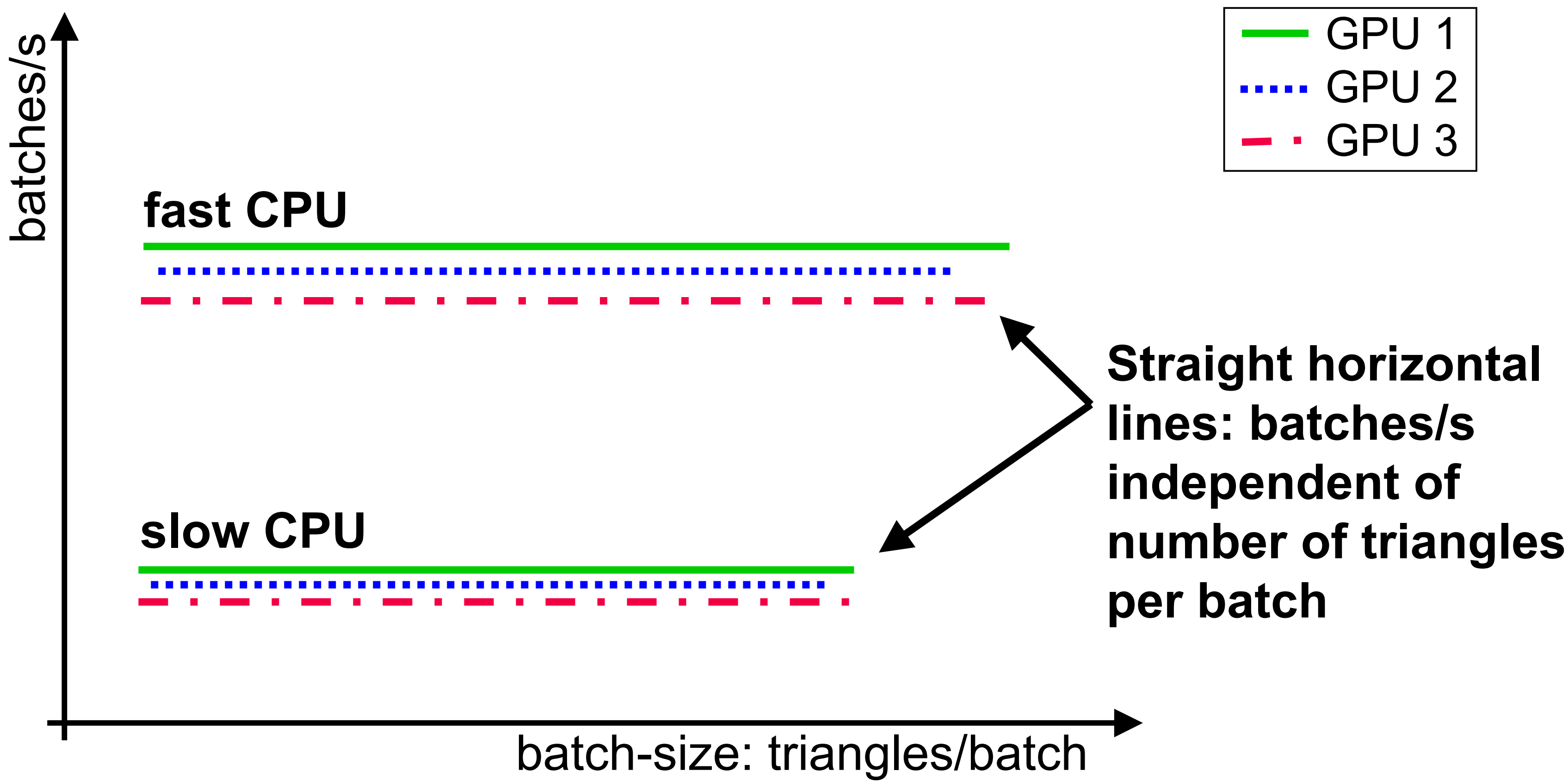
What To Expect If CPU Limited



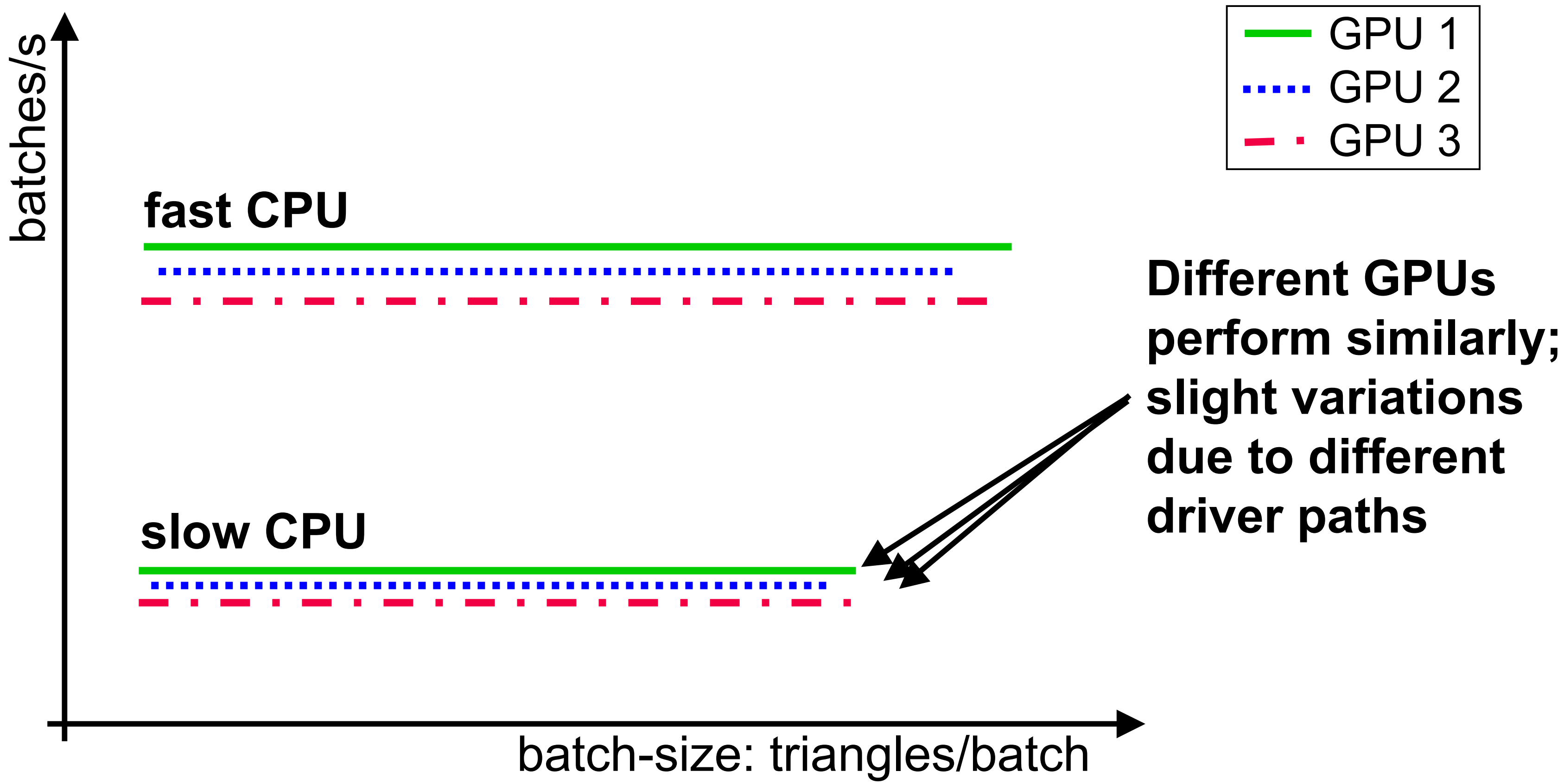
Effects of Different CPU Speeds



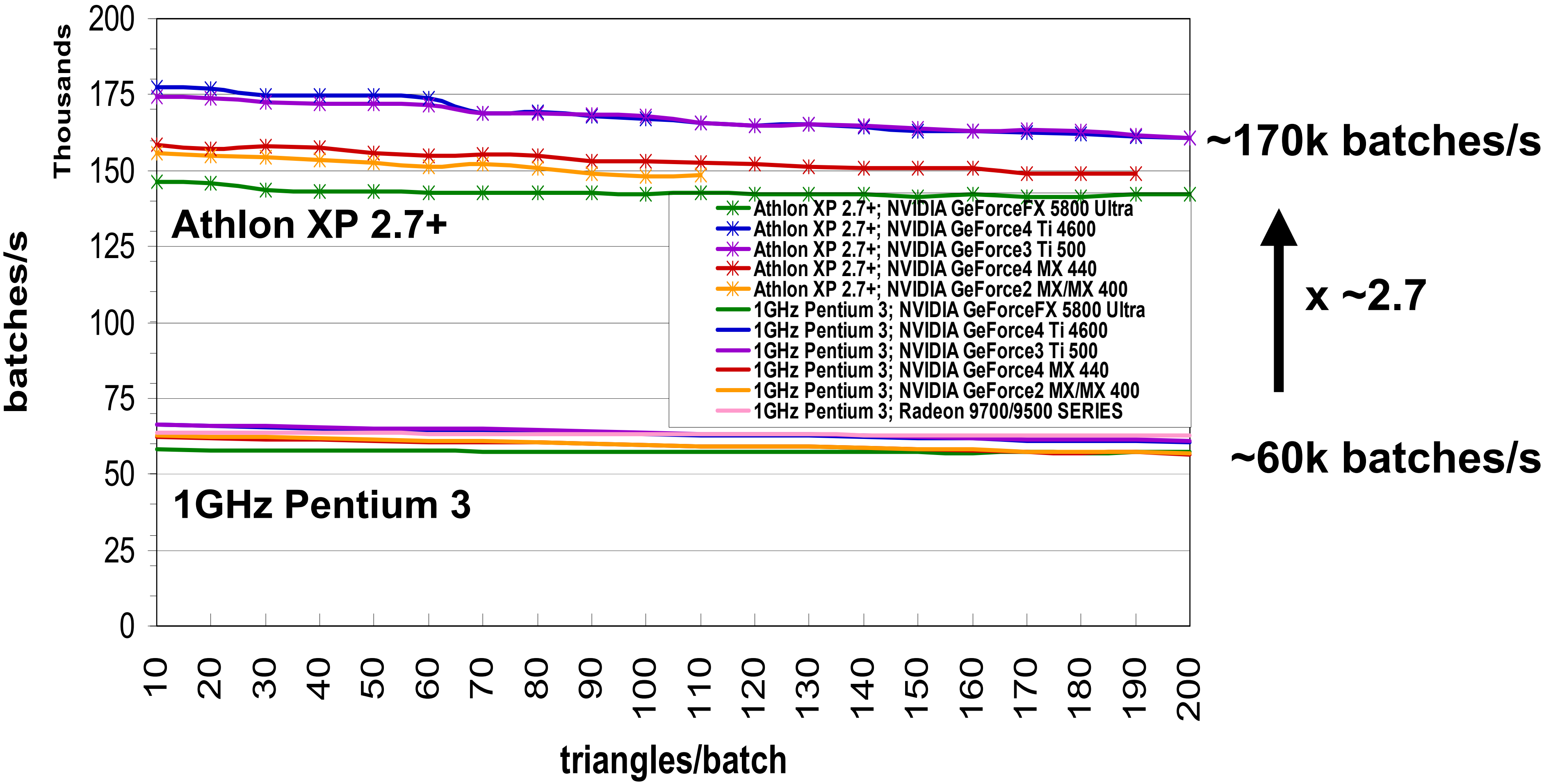
Effects of Number of Tris/Batch



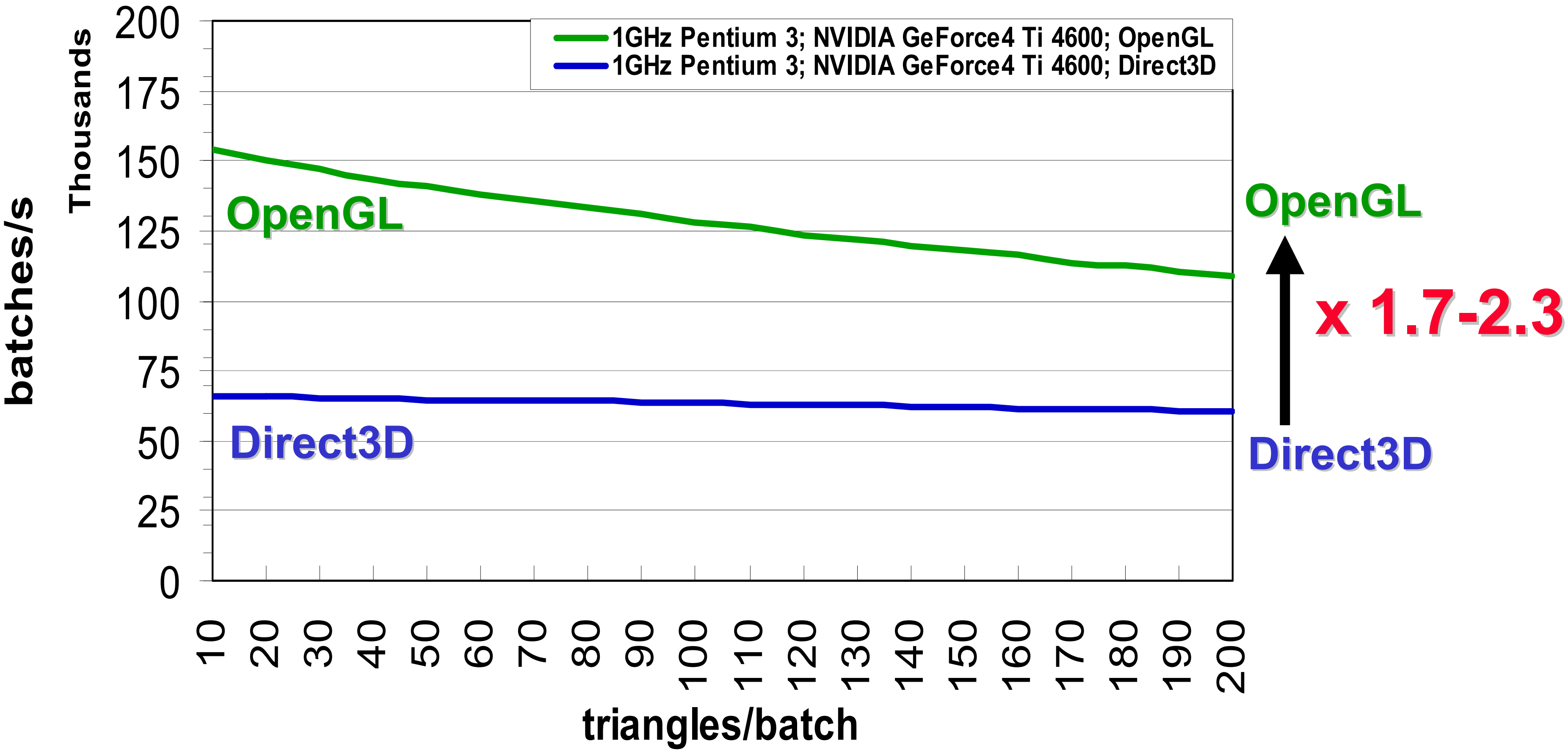
Effects of Different GPUs



Measured Batches Per Second



Side Note: OpenGL Performance



CPU Limited?

- Yes, at < 130 tris/batch (avg) you are
  - completely,
  - utterly,
  - totally,
  - 100%
  - CPU limited!
- CPU is busy doing nothing, but submitting batches!

How 'Real' Is Test App?

- Test app only does SetState, Draw, repeat;
  - Stays in CPU cache
  - No frustum culling, no nothing
  - So pretty much best case
- Test app changes arbitrary set of states
  - Types of state changes?
  - And how many states change?
  - Maybe real apps do fewer/better state changes?

Real World Performance

- 353 batches/frame @ 16% 1.4GHz CPU: 26fps
- 326 batches/frame @ 18% 1.4GHz CPU: 25fps
- 467 batches/frame @ 20% 1.4GHz CPU: 25fps
- 450 batches/frame @ 21% 1.4GHz CPU: 25fps
- 700 batches/frame @ 100% (!) 1.5GHz CPU: 50fps
- 1000 batches/frame @ 100% (!) 1.5GHz CPU: 40fps
- 414 batches/frame @ 20% (?) 2.2GHz CPU: 27fps
- 263 batches/frame @ 20% (?) 3.0GHz CPU: 18fps
- 718 batches/frame @ 20% (?) 3.0GHz CPU: 21fps

Normalized Real World Performance

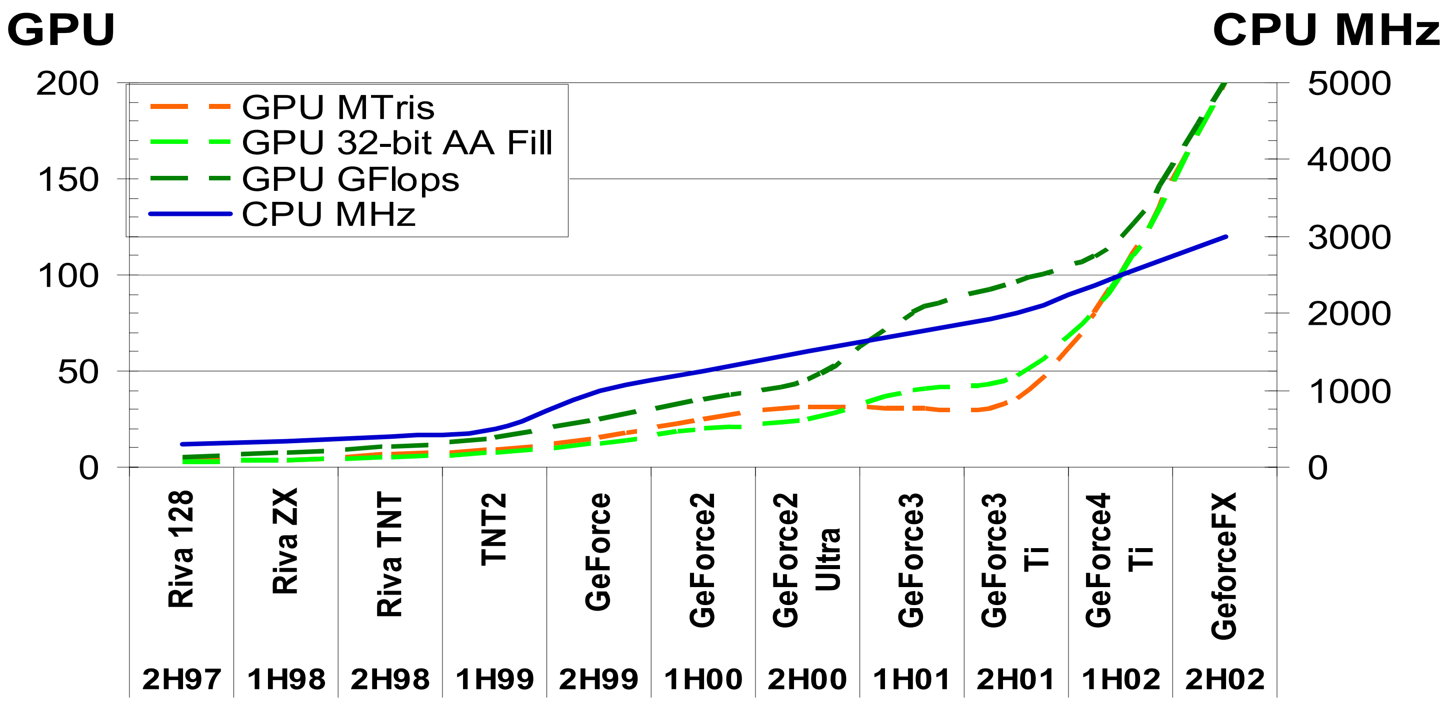
- **~41k** batches/s @ 100% of 1GHz CPU
- **~32k** batches/s @ 100% of 1GHz CPU
- **~42k** batches/s @ 100% of 1GHz CPU
- **~38k** batches/s @ 100% of 1GHz CPU
- **~25k** batches/s @ 100% of 1GHz CPU
- **~25k** batches/s @ 100% of 1GHz CPU
- **~25k** batches/s @ 100% of 1GHz CPU
- **~ 8k** batches/s @ 100% of 1GHz CPU
- **~25k** batches/s @ 100% of 1GHz CPU

10k – 40k batches/s (100% 1GHz CPU)

Small Batches Feasible In Future?

- VTune (1GHz Pentium 3 w/ 2 tri/batch):
  - 78% driver; 14% D3D; 6% Other32; rest noise
- Driver doing little per Draw/SetState, but
  - Little times very large multiplier is still large
- Nvidia is optimizing drivers, but…
- Submitting X batches: O(X) work for CPU
  - CPU (game, runtime, driver) processes batch
  - Can reduce constants but not order O()

GPUs Getting Faster More Quickly Than CPUs



Avg. 18month CPU Speedup: 2.2

Avg. 18month GPU Speedup: 3.0-3.7

GPUs Continue To Outpace CPUs

- CPU processes batches, thus
  - Number of batches/frame MUST scale with:
    - Driver/Runtime optimizations
    - CPU speed increases
- GPU processes triangles (per batch), thus
  - Number of triangles/batch scales with:
    - GPU speed increases
- GPUs getting faster more quickly than CPUs
  - Batch sizes CAN increase

So, How Many Tris Per Batch?

- **500? 1000? It does not matter!**
  - Impossible to fit everything into large batches
  - A few 2 tris/batch do NOT kill performance!
  - N tris/batch: N increases every 6 months
- **I am a donut!** Ask not how many tris/batch, but rather how many batches/frame!
- **You get X batches per frame, depending on:**
  - Target CPU spec
  - Desired frame-rate
  - How much % CPU available for submitting batches

You get X batches per frame,

X mainly depends on CPU spec

What is X?

- **25k batches/s @ 100% 1 GHz CPU**
  - Target: 30fps; 2GHz CPU; 20% (0.2) Draw/SetState:
  - X = 333 batches/frame
- **Formula: 25k * GHz * Percentage/Framerate**
  - GHz = target spec CPU frequency
  - Percentage = value 0..1 corresponding to CPU percentage available for Draw/SetState calls
  - Framerate = target frame rate in fps

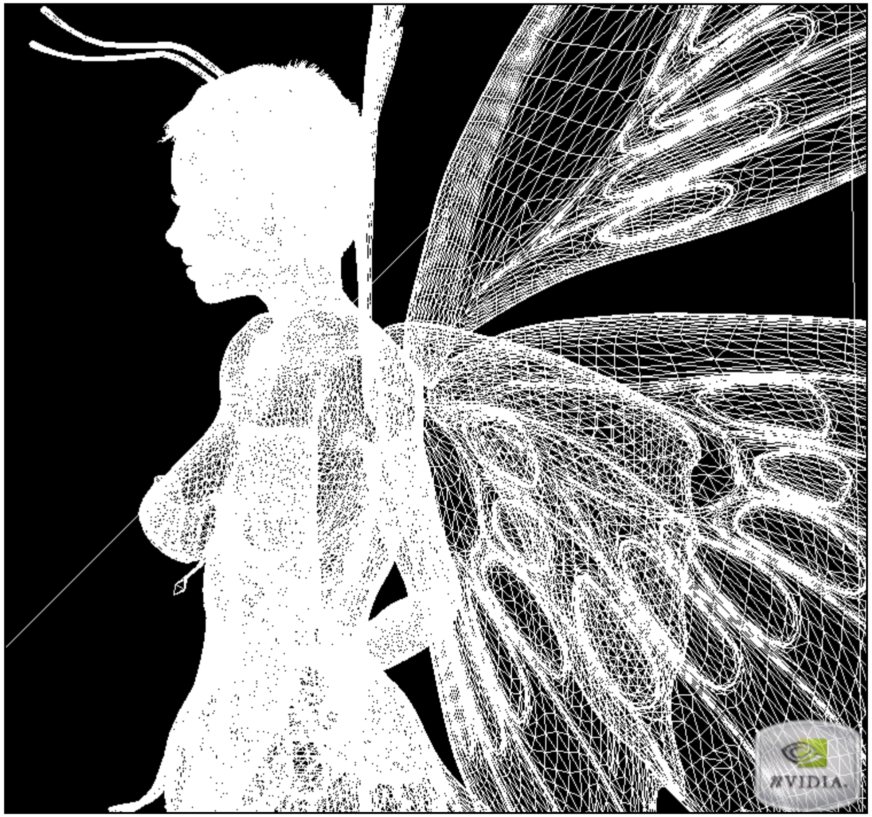
Please Hang Over Your Bed

25k batches/s @ 100% 1GHz CPU

How Many Triangles Per Batch?

- Up to you!
  - Anything between 1 to 10,000+ tris possible
- If small number, either
  - Triangles are large or extremely expensive
  - Only GPU vertex engines are idle
- Or
  - Game is CPU bound, but don't care because you budgeted your CPU ahead of time, right?
  - GPU idle (available for upping visual quality)

GPU Idle? Add Triangles For Free!



GPU Idle?
Complicate Pixel Shaders For Free!

300 Batches Per Frame Sucks

- (Ab)use GPU to pack multiple batches together
- Critical NOW!
  - For increasing number of objects in game world
- Will only become more critical in the future

Batch Breaker: Texture Change

- Use all of 16 textures on DX9 parts
  - Fit 8 distinct dual-textured batches into 1 single batch
- Pack multiple textures into 1 surface
  - Works as long as no wrap/repeat
  - Requires tool support
  - Potentially wastes texture space
  - Potential problems w/ multi-sampling

Batch Breaker: Transform Change

- **Pre-transform static geometry**
  - Once in a while
  - Video memory overhead: model replication
- **1-Bone matrix palette skinning**
  - Encode world matrix as 2 float4s
    - axis/angle
    - translate/uniform scale
  - Video memory overhead: model replication
- **Data-dependent vertex branching**
  - Render variable # of bones/lights in one batch

Batch Breaker: Material Change

- Compute multiple materials in pixel-shaders
  - Choose/Interpolate based on
    - Per-vertex attribute
    - Texture-map

But Only High-End GPUs Have That Feature!?

- Yes, but high-end GPUs most likely CPU-bound
- High-End GPUs most suited to deal with:
  - Longer vertex-shaders
  - Longer pixel-shaders
  - More texture accesses
  - Bigger video memory requirements
- To improve batching

But These Things Slow GPU Down!?

- Remember: CPU-limited
  - GPU is mostly idle
- Making GPU work, so CPU does NOT
- Overall effect: faster game

25k batches/s @ 100% 1GHz CPU

Acknowledgements

- Many thanks to

  Gary McTaggart, Valve

  Jay Patel, Blizzard
  Tom Gambill, NCSoft
  Scott Brown, NetDevil
  Guillermo Garcia-Sampedro, PopTop

Questions, Comments, Feedback?

- Matthias Wloka: mwloka@nvidia.com
- http://developer.nvidia.com

Can You Afford to Loose These Speed-Ups?

- 2 tris/batch
  - Max. of ~0.1 MTriangles/s for 1GHz Pentium 3
    - Factor 1500x away from max. throughput
  - Max. of ~0.4 MTriangles/s for Athlon XP 2.7+
    - Factor 375x away from max. throughput