DirectX8 Performance

Richard Huddy, NVIDIA Corporation

RichardH@nvidia.com

Driving Direct3D

- **<u>The 3 Cardinal Rules of Direct3D</u>**
  1. **"All games are CPU limited"**
     - **That's over 80% of games that I see**
  2. **Batch your polys into groups of 200+**
     - **Many apps send 1 or 2 at a time, all the time**
     - **Apps like this use roughly 1% to 5% of available GPU power**
  3. **Use VBs for everything and use them with care**
     - **Never use DrawPrimitiveUP**
     - **Learn how to use the Create() & Lock() flags**

“All games are CPU limited”

- So what’s wrong with profilers?
  - Nothing, use VTune and the others
    - But it only tells you what the CPU was doing
- NVIDIA also offers:
  - Beta drivers for registered developers
  - NVTune statistics drivers which will tell you what the driver was doing
- Constructive use of H/W TnL frees CPU cycles
  - But batching is even more important here...

Batch your polys into groups of 200+

- Because for small numbers the runtime will copy data around to try to protect the CPU and GPU
  - This is in addition to the usual cost of a call...
- Because the overhead for each call is significant
  - So that on a 1.5GHz machine it's still easy to be CPU limited
- Because it forces you to group state changes
  - Never send a state change that doesn't affect rendering
  - Sort by render target (everyone already does this)
    - Then by texture/Shader

VertexBuffers for everything

- Never use DrawPrimitiveUP
  - Because it guarantees to make it hard to go fast
- Learn how to use the Create() & Lock() flags
  - There are papers on our website for this
  - “The efficient use of vertex buffers” by me...
  - NVTune will help identify this
  - Expect to Lock() with NOOVERWRITE most often
- Make sure the VB goes into the right pool:

  choose carefully between MANAGED and DEFAULT

Where should my VertexBuffers live?

- Static data should be in MANAGED or DEFAULT according to whether you want the system to retain a copy for use when you need to Reset()
  - These will be placed in AGP or local vid mem
- Dynamic data is best placed in DEFAULT and always accessed with WRITEONLY
  - Dynamic data in SYSTEMMEM is always slow...
- GPU reads from system memory run at roughly 10% of theoretical maximum ☹

Managing Dynamic data

:: • **CreateVB(WRITEONLY, 2000+) and new_frame**

A: • **I = 0**

B: • **Space in VB for M vertices?**

- **Yes: Lock(I*fvfSize, M*fvfSize, NOOVERWRITE)**
- **No: GOTO C**

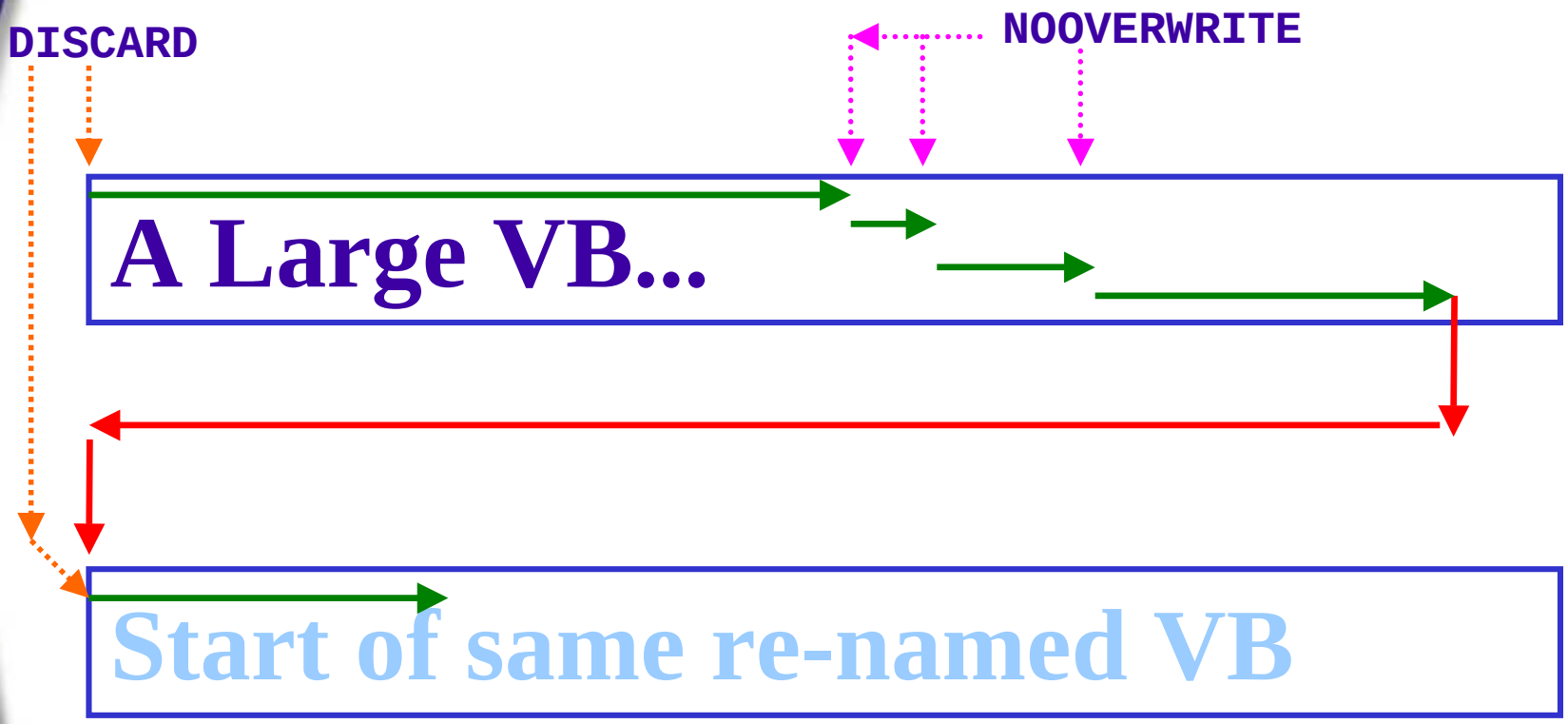
• **Fill in M vertices at index I**

• **Unlock(); DIP(I); I += M; GOTO B**

C: • **Lock(DISCARD) GOTO A**

Use one large VB, discard contents when full

Managing Dynamic data



As many times as you need to...

What FVFs should I use?

- Generally prefer the smallest FVF which encompasses the data you wish to send but...
  - Some FVFs fit hardware especially well:
    - 24, 32 and 64 bytes are the best
    - Very detailed info available:
    - http://developer.nvidia.com/docs/IO/1319/ATT/vb+stats2.xls
- Don't use too many unique FVFs since that will force you to use many VBs, and that implies lots of VB switches
- If you are hardware vertex throughput limited then just using 1 FVF can be a good decision

Vertex shaders – typical use

- Animation
  - Skinning with any number of bones needs access to shaders if you're going to batch
- Making DOT3 lighting work easily
  - Do all the work on the GPU where it's needed
- Instancing
  - Because this helps with batching
- Shadows - completely on the GPU

Note that aggressive use of vertex shaders is very safe. There's great CPU emulation available and on the right hardware it frees up CPU cycles

Pixel shaders – typical use

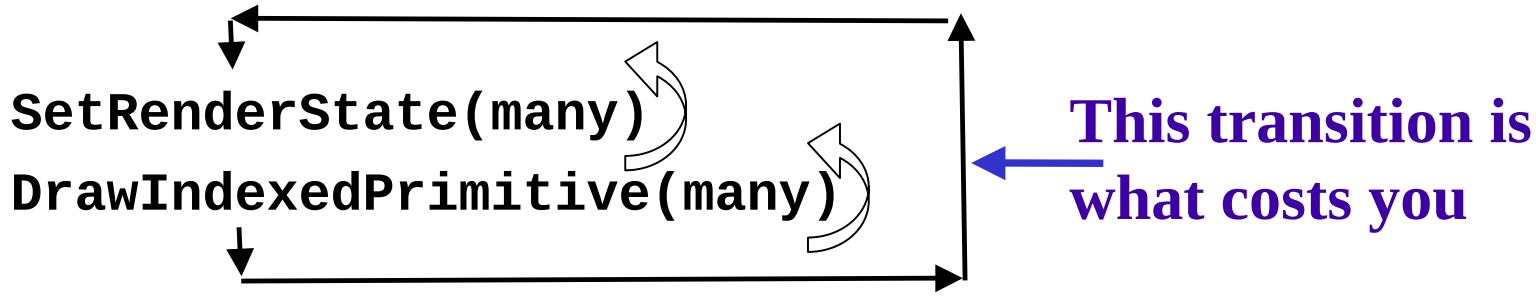
- **Shiny reflective surfaces:**
  - **True “Blinn Bump Mapping” per pixel.**
  - **EMBM style reflective planes**
- **Per pixel normalisation:**
  - **Most obviously for Phong lighting**
    - **In just 2 instructions!**
- **Getting access to all the texture permutations which were hidden in the old ‘classical’ cascade**

Performance considerations

- On hardware, vertex shader execution time is controlled entirely by the number of ops
  - So watch out for macros
  - And watch out for the long versions like ‘exp’
- Pixel shader performance is roughly linear but there are some slower ops. Experiment...
- More apps are shader-limited than bandwidth-limited, so GeForce4 will help a great deal

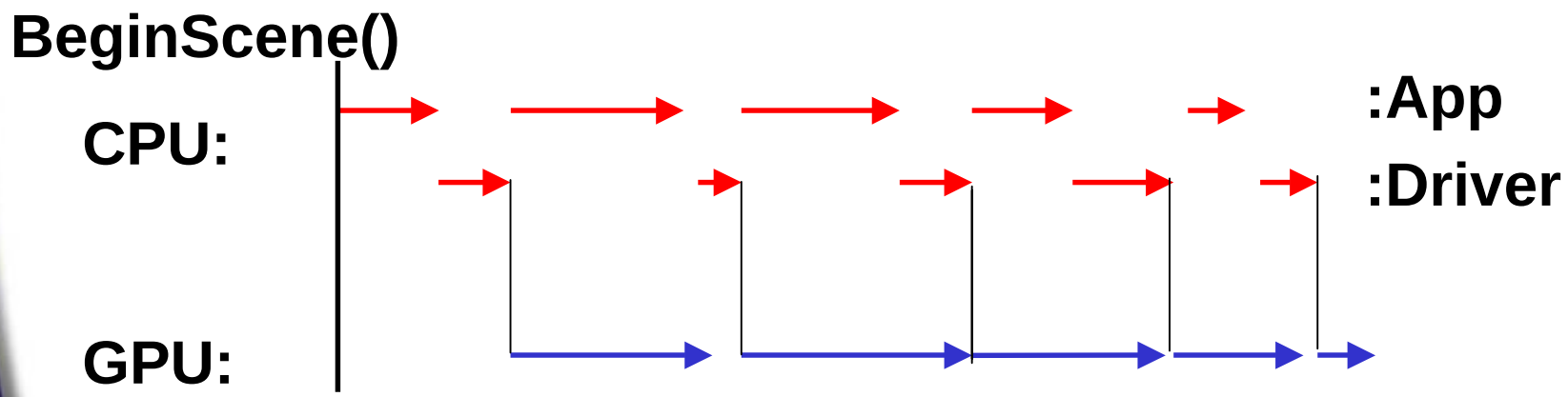
State Changes

- Minimise the sending of state changes whether using the PURE hal or not.
- Batch your state changes (like your polys)
- Minimise the total number of times you send state bundles, not the total number of changes...
  - Aim for fewer than 200 transitions per frame:



What kind of parallelism?

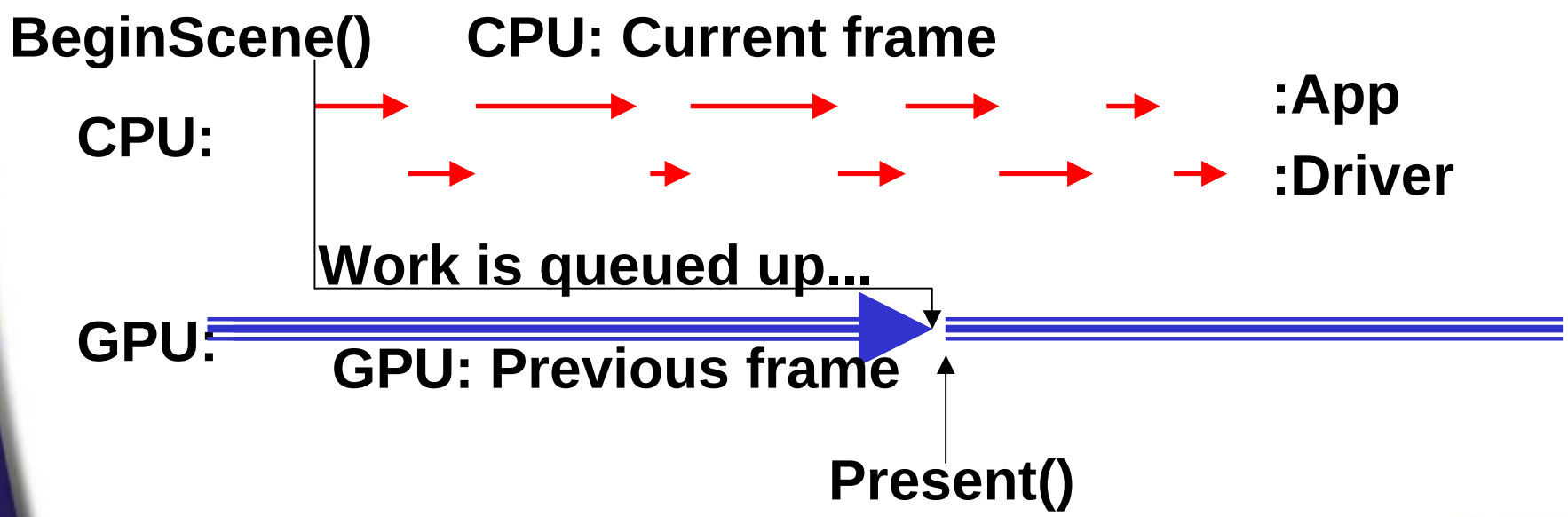
- Lots of people think about it like this:
  - What can I do to make sure that the graphics card is busy between calls to DrawPrim()?
  - Diagrammatically they're trying to do this:



That's the wrong way to think about it...

The right kind of parallelism

- Make the graphics work from the previous frame overlap the CPU work of this frame



So why is Lock expensive?

- Locking the FB or ZB is primarily a CPU cost (while the CPU waits for the GPU to complete)
- For that reason it should be seen as a CPU intensive operation
  - remember the CPU does no useful work while it's waiting
  - but the GPU is productive for most of that time
- So, if you Lock twice in succession is the second Lock cheap or expensive?

Why GeForce4 is faster the GeForce3

- Clock for clock it can do more work
  - Two parallel vertex pipes (that's 2x perf)
  - More efficient and powerful pixel engine
  - More efficient memory interface
- GeForce4 clock speed is roughly 300MHz
  - That's 50% higher than a typical GeForce3
- That's roughly a factor of 3 on vertices
  - 2x times 1.5x
- And roughly a factor of two on pixels
  - But this is sensitive to draw order

What's Z correct bump mapping?

- It is a per-pixel adjustment of the value which gets sent to the Z-buffer
  - It's not displacement mapping
    - So it doesn't change the geometry or silhouette
  - It's like per-pixel "displacement mapping along the line of sight"
    - So it does give good intersections where standard geometry meets bump mapped textures
      - e.g. where waves lap at the water's edge...

Accuview Anti-Aliasing engine

- Performance
  - HRAA Twice as fast as GeForce3
  - Patent-pending Accuview AA technology
    - Quincunx for free – Quincunx runs as fast as 2x AA
- Quality
  - New sub-pixel locations and 4XS AA mode lead to even higher quality AA.
- Flexibility
  - Allows the greatest choice in performance and quality
  - 2x, Quincunx, 4x, and 4XS AA modes
- AA should be a user option for all games...

Anti-Aliasing

- Q: How much memory does AA use?
- A: The algorithm is simple:
  - Take the memory for the *non* AA case
  - Subtract the memory used by the normal Z buffer
  - Add the memory for a single super color and super Z buffer
  - multiply by sample count to get the super scale
  - That's 3 normal buffers, 1 super color buffer and 1 super Z buffer for triple buffering

Anti-Aliasing

- Q: Why should I care?
- A: Because if you run out of local video memory you are likely to have to texture from AGP memory – which is noticeably slower

Note that 1600x1200x32 triple buffered, 4 Sample AA will demand 80.5Mb – So check CreateDevice()!

Why indexed primitives? I

- **Learn what typical hardware can do to help**
  - **TNT has a 16 Vertex Cache**
  - **GeForce has an *effective* 10 Vertex Cache**
  - **GeForce3/4 have an *effective* 18 Vertex Cache**
  - **Other cards will vary, but bigger is always better**
- **NVIDIA's Vertex caches are FIFO's (*not* LRU)**
- **Re-use recent indices**
  - **'Recent' is defined by the cache size**
- **Use highly local winding in your strips**

Vertex caches are only available when using indexing!

Why indexed primitives? II

- Did you just say,

  "Vertex caches are only available when using indexing!

- Yes.
- Which means that indexed primitives are innately much faster (for arbitrary data) than non-indexed data
- And that can be up to three times faster
- Did you just say,

  "Three times faster"

- Yes.

What are the most important things?

- Batching is critical, if you want things to go fast
  - Aim for 200 polys per call...
- Use VBs for everything and use them with care
  - Mis-use of the flags will cost you...
- Use all the available advice to make sure your game is not just CPU limited
  - Always profile at low res...

Questions?

?

Richard Huddy

NVIDIA Corporation

RichardH@nvidia.com