Batching 4EVA

Matthias M Wloka
NVIDIA Corporation
Review: Batch, Batch, Batch

- Batch: state changes & Draw() call

- Lots of batches make you
  - Completely,
  - Utterly
  - CPU limited!

- Overhead caused by
  - ~80% driver
  - ~10% runtime
Measured Batches per Second

- Athlon XP 2.7+; NVIDIA GeForceFX 5800 Ultra
- Athlon XP 2.7+; NVIDIA GeForce3 Ti 500
- Athlon XP 2.7+; NVIDIA GeForce4 MX 440
- Athlon XP 2.7+; NVIDIA GeForce2 MX/MX 400
- 1GHz Pentium 3; NVIDIA GeForceFX 5800 Ultra
- 1GHz Pentium 3; NVIDIA GeForce4 Ti 4600
- 1GHz Pentium 3; NVIDIA GeForce3 Ti 500
- 1GHz Pentium 3; NVIDIA GeForce4 MX 440
- 1GHz Pentium 3; NVIDIA GeForce2 MX/MX 400
- 1GHz Pentium 3; Radeon 9700/9500 SERIES

- 170k batches/s
- x ~2.7
- 60k batches/s
Please Hang over Your Bed

25k batches/s @ 100%
1GHz CPU
Review: Son of a Batch

- All state changes roughly equally bad
  - Multiple state changes worse than changing single state

- Sort by state? Over-constrained problem
  - And only an optimization

- Solution: collapse states
Use Texture Atlases

- Removes `SetTexture()`

- Texture Atlas Tools:
Use Instancing

• Previous session

• “Inside Geometry Instancing,” Francesco Carucci, Lionhead Studios, GPU Gems 2
Most Important: Plan for Batching!

• Oh sh!%$, our game uses 2000 batches/frame
  – Painful to impossible to fix late in development

• Have a batch budget
  – For terrain, characters, etc.
  – Educate and give feedback to your art staff
  – Stick to the plan
Be Aggressive in Moving Stuff to GPU

- All particle systems: 1 Draw() call?!
- Need to alpha blend them?
  - Sort on the GPU!
This Is All Very Complicated...

- Can I just wait until you guys fix this?

- And new cool tech coming out that solves all these problems, right?
  - Dual-core CPUs
  - Longhorn
  - WGF 2.0
GPUs Getting Faster More Quickly

![Graph showing GFLOPS over time](image)

- **GFLOPS (multiplies per second)**

  - **NVIDIA NV30,35,40**
  - **Pentium 4**

**Courtesy Ian Buck, Stanford University**
Multi-Core CPUs to the Rescue!

- Sorry, no...

- Requires thread programming
  - Is your game multi-threaded?
  - Batch overhead is in driver!
  - Batch processing \{SetState; Draw; repeat\} and thus driver inherently serial

- Multi-core GPUs already available:
  - It’s called SLI
Longhorn to the Rescue!

• Sorry, no...

• More efficient runtime and driver
  – Design Goal: 10x improvement
    (WinHEC’04 WGF Slides)

• Does not help your WinXP user base

• Longhorn available: 2006
  – Long time in GPU years
WGF 2.0 to the Rescue!

• You are on to something, but sorry, no...

• Features designed to mend batches, i.e.

• Another ‘simpler’ way to not say
  – Change state
  – Draw triangle
Later Today: “WGF 2.0”

David Blythe, Microsoft

5:15pm
We Are Stuck

1000 batches/frame

4EVA!

Assuming 50% 3GHz CPU @ 33fps
Graphics in the Future?

• Best engine is the one that achieves
  – Most complex
  – Most engaging
  – Most immersive
  – ...

• In 1000 batches/frame or less!

• Make GPU work, so CPU does NOT
To Make Things Worse...
Get a Couple of Flashlights!

• First rule of optimization: Profile! Know your bottleneck!

• PIX

• NVIDIA Performance Analysis Tools

• AMD’s CodeAnalyst
Performance Stalagmites

- Difficult to hit these

- Help available:
  - GPU Programming Guide
  - Tools
  - Your local IHV devtech representative
GPU Performance Advice

- Memory allocation
- Vertex shader optimizations
- Pixel shader optimizations
- Texture
Memory Allocation: Don’ts

- Calling Create() mid-frame
  - Guaranteed a frame-rate hitch
  - Sub-optimal resource placement
  - Expect the call to fail!

- Calling Release() mid-frame
  - Potentially does nothing

- Do your own resource management instead
Allocation Order → Rendering Performance

- Allocate POOL_DEFAULT resources first
  - Render-targets first, sort by pitch
  - Vertex and pixel shaders
  - Textures
  - Vertex and index buffers

- Then POOL_MANAGED
  - If any
**Vertex Shader Optimizations**

- **VS_3_0 dynamic flow control**
  - Go nuts, save batches
  - Not penalty for divergence (MIMD)
  - Driver optimizes short branches

- **VS_3_0 vertex texture fetch (VTF)**
  - 20-30 instructions latency
  - Hide other instructions in latency
  - Dynamically branch over VTFs
  - Pack data into single texture
Great Results with Vertex Texture

Image used with permission from Pacific Fighters.
© 2004 Developed by 1C:Maddox Games. All rights reserved. © 2004 Ubi Soft Entertainment.

“GPU Gems 2 Showcase”
Room 2016
Wednesday, 5:15 - 6:15pm

Arul Asirvatham & Hugues Hoppe

Terrain Rendering Using GPU-Based Geometry Clipmaps
Pixel Shader Optimizations

- Move computations out
  - Remove operations via algebra
  - Pre-compute: use texture as look-up table
  - Into vertex shader: constant, interpolations

- Dynamic branching
  - Driver optimizes
  - Early out
  - Batch materials

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cost (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if / endif</td>
<td>4</td>
</tr>
<tr>
<td>if / else / endif</td>
<td>6</td>
</tr>
<tr>
<td>call</td>
<td>2</td>
</tr>
<tr>
<td>ret</td>
<td>2</td>
</tr>
<tr>
<td>loop / endloop</td>
<td>4</td>
</tr>
</tbody>
</table>
Dynamic Branching: Coherency

- ~1000 pixels, i.e., 30x30 blocks

Incoherent

Coherent
Partial Precision Optimizations

• Compiler/Driver cannot help you here

• Reduces register pressure
  – Critical for GeForce FX
  – 100+ instruction shaders for GeForce 6

• Single cycle half3 normalize()
  – Versus 3 cycle \{dp3; rsq; mul\}
Hardware Shadow Maps

• Support since GeForce 3

• Use:
  – Render to depth format texture (D3DFMT_D24X8, D3DFMT_D16)
  – Use tex2Dproj to sample
  – Automatic shadow map comparison & percentage closer filtering (PCF)

– Explain PCF?!
Hardware Shadow Map Fallback

- Generate depth in shader

- Write to single channel R32F or R16F texture

- Sample texture, compare depths
  - Multiple jittered samples for high quality / soft edges
  - Filter multiple sample via percentage closer
Shadow Map Performance

- HW shadow map comparison half speed
  - No need to compare or filter in the shader
  - PCF of 4 nearest texels if bilinear is on

- Single tap for performance
  - Quality equivalent to 4-tap PCF R32F

- Multiple taps for higher quality
  - 2-tap hw shadow map roughly same speed as 4-tap manual-PCF R32F
Texture Instruction Performance

• Full speed:
  – Regular mipmap, e.g., tex2D(s, t)
  – Scalar bias mipmap, e.g., tex2Dbias(s, t)
  – Explicit mipmap selection

• 1/10th speed:
  – Gradient-based LOD selection, e.g.,
    { ddx(x); ddy(y); tex2Dbias(s, t, ddx, ddy) }
  – But when you need to use it, you need to use it
Common Sense Texture Performance

• **Use mipmaps**
  – GPU fetches local neighbors for each texel

• **Sharper/Crisper textures**
  – Use anisotropic filtering
  – Use better mipmap generation (use texture tools)
  – Do NOT use LOD bias
  – LOD bias is slower and lower quality
Floating Point Texture Performance

- Prefer 32bpp over 64bpp over 128bpp
  - Applies to textures and render targets
  - Bandwidth!

- More importantly: cache coherence
  - Poor cache coherence destroys performance
  - Fp16 textures 2x faster than fp32 if texture bound

- Efficient channel allocation
  - Use R32F buffers for scalar data, not RGBA32F
  - R16G16F for 2-vectors
Conclusion

1000 batches/frame
4EVA!
Questions?

• mwloka@nvidia.com

• Slides available online