Today’s Agenda

• DirectX 9 Features
  – Sim Dietrich, nVidia - Multisample antialising
  – Jason Mitchell, ATI - Shader models and coding tips

• Optimization for DirectX 9 Graphics
  – Mike Burrows, Microsoft - Performance Analysis and Solutions

  Coffee break – 11:00 – 11:15

  – Richard Huddy, ATI - Optimization
  – Ashutosh Rege, nVidia - Optimization

  Lunch break – 12:30 – 2:00

  – Jeff Grills, Sony Online Entertainment - Optimizing Star Wars: Galaxies

• Rendering Techniques
  – Craig Peeper, Microsoft - Advanced D3DX Effects

  Coffee break – 4:00 – 4:15

  – Matthias Wloka, nVidia - Advanced Visual Effects
  – David Gosselin, ATI - ATI Demo Team Visual Effects
  – Shawn Hargreaves, Climax Brighton - Deferred Shading
  – Gary McTaggart, Valve Software - Half-Life 2 / Source Shading
The API

• Since there’s an API between you and the hardware it makes sense to expect that you need to know how to use it

• Abuse of the API can be a mighty expensive option…

• The two commonest failures:
  – 1 Games are CPU limited at target resolutions
  – 2 Games fail to batch sufficiently
Huge Savings...

• SetRenderTarget()
  – Let’s not have too many of these please!
• Lock() – with no flags is a danger signal!
  – Whether that’s a VB that’s being rendered from
  – Or a RenderTarget which was rendered to
• There are milliseconds at stake here!
• Also use ‘DONOTWAIT’ appropriately to reclaim CPU cycles – these are scarce!
Significant savings...

- Every DrawPrim call is a significant cost
  - So make sure you get good value from it

- Every time you set state it costs you significantly
  - Whether you set one or ten...
  - But aggressive state filtering is no longer needed as much in DX9

- One pixel is irrelevant, but millions matter...
  - So draw Front to Back
  - Clear() the Z/stencil buffer to make it work fast
  - Sort by shader
  - Set your shader constants in blocks
Compilers can be smart...

- At ATI we test compilers to make sure that they’re good and help make them better
- Sample Renderman results: (Win, Draw, Lose)
  - HLSL vs Cg on ATI (*) : 5, 7, 2
  - HLSL vs Cg on NV : 16, 7, 0
    - (*) Cg compiler failed to compile 9 samples for SM2.0 even though HLSL compiler succeeded
- So using HLSL seems like the logical choice…
- Not just an industry standard – but the best too
And a PC is complex

• Which is a bit of an understatement

• A 9800 Pro has a similar number of gates as two Pentium4 processors all on one die

• But the highly parallel design allows it to do much more work – of a very specific kind…

• So you’d like to have the CPU and VPU both doing useful work at the same time
  – Luckily the API encourages this…
Which bits are fast?

- **System:**
  - CPU
  - 1 to 1/3 of a nanosecond… (1GHz to 3GHz)
  - System memory
    - High latency compared to the CPU
    - 200 - 800MHz (for moving data about)
  - Virtual memory
    - Takes all week…

- **Graphics card:**
  - VPU core
    - 200 to 500MHz
  - Local video memory
    - 200 to 500MHz

- **AGP 8X Bus:**
  - 266MHz, 2GB per second, with latency like molasses…
Which bits are fast?

• System:
  – CPU
  – So the CPU is fast, but it still has too much to do…
    • “All games are CPU limited”

• Graphics card:
  – VPU core
    • Not a blinding fast clock, but very high throughput

• AGP Bus:
  – Don’t texture from here unless you have to!
Inside the VPU

• You have several units at your disposal…
  – Vertex fetch (memory cache)
  – Vertex shader (xform and lighting)
  – Vertex cache (protecting the shader from abuse)
  – Clipper (so fast it might as well not be there…)
  – Triangle setup
  – Fast Z/stencil reject (quad speed rasterizer rejection)
  – Rasterizer
    • Pixel cache
    • Texture cache
  – Z buffer
  – Blend (Yummy! Read-modify-write)
Inside the VPU

- Because the vertex fetch unit is just reading / caching memory it makes sense to prefer cache-aligned data formats (like 32 bytes or 64 bytes)

- The vertex cache only works for indexed primitives…
  - D3DXOptimizeMesh gets generic benefits!!

- So we recommend that all rendering is done with DrawIndexedPrimitive() and that you submit data in roughly tri-strip order
Saving nanoseconds...

• Use shorter shaders since they’re faster
  – One op per clock is what you should expect
  – ATI hardware can parallelise vector + scalar op pairs
  – NVIDIA hardware prefers a small register footprint

• Shaders are cached on chip too
  – So switching shader can sometimes be very fast

• Hand written assembly isn’t usually a good bet

• ps.1.4 modifiers can be free on ps.2.0 hardware
Saving nanoseconds…

• Prefer the shortest shader which does what you want

• Use the lowest shader model which achieves your target
  – That way you can potentially access the ps1.4 modifiers which run in the same clock cycle

• But please do not sacrifice quality for speed!
  – That can be the user’s choice later on by selecting no-AA, low screen resolution etc
My favourite optimisation stories

1. 9.9 fps to 10.1 fps by using H/W TnL
   How many triangles per call…?
   Just the one huh?

2. My AGP bus is too slow
   We must be drawing too many triangles…
   Nope, you have a pure device using S/W

3. Just draw the damned stuff!
   How productive is cleverness?
   Static LOD is much faster than dynamic!
Questions…

• After Ashu finishes the section please…