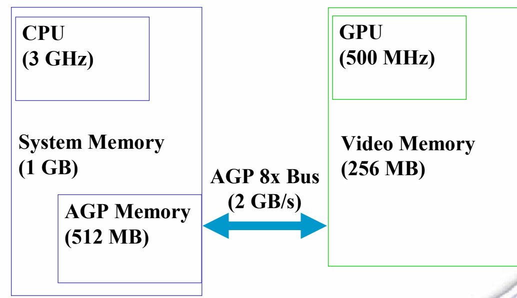
Overview of Graphics Hardware

John Spitzer
NVIDIA Corporation

PC Graphics (Current)



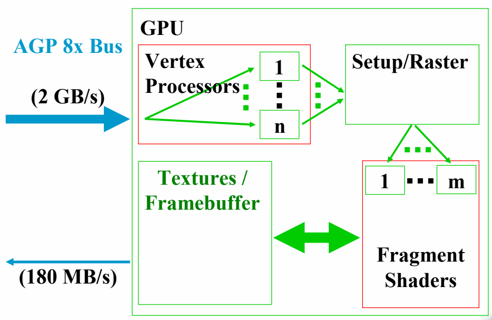
Usual Co-Processor Pitfalls

- Synchronization temporarily idles ALL processors
- Specialized co-processor architecture
  - GPU's deep pipeline means restart is expensive
  - Different mind-set needed to map problems to architecture

GPU as a Co-Processor? Careful!

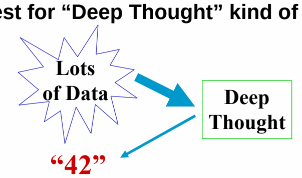
- CPU programmed as von Neumann architecture
- GPU designed to render graphics
  - MAY be able to abuse it for other computations
- GPU is NOT von Neumann architecture
  - Deep pipeline architecture
  - Pipeline stages are multi-pipe SIMD designs
  - Stages are vector-processors
  - Optimized for large table look-ups (textures)
  - AGP interconnect not symmetric

GPU Schematic



AGP Bus Considerations

- Optimized for graphics:
  - CPU hands GPU (lots of) data
  - GPU produces image on monitor
  - AGP read-back (generally) unused
- Best for "Deep Thought" kind of problems:



AGP Performance

- Write AGP data in 32 or 64 byte blocks
  - Otherwise AGP write-combining reads, then writes
- Avoid reading from GPU data-structures
- Communicate intended use to driver
  - Static versus dynamic vertex buffers or textures
  - Declare data as write-only
  - Placement into video-, AGP-, or system-memory
- Allow vertex buffer renaming (avoid syncs)
  - Use discard/no-overwrite and var/fence

Programmable Vertex Processors

- No connectivity info/no access to neighbors (SIMD)
- 1.5 Billion VECTOR operations/s! (~6 GFlops/s)
  - IEEE s23e8 32 bit floating point per component
  - "Simple" operations include dot4, mad, sin, pow, lg2
  - Vector swizzles/conditional writes are free
- Post TnL vertex caches: >>100 Million lit tris/s
- Per-vertex data-dependent:
  - Branches, loops
  - Subroutines

Vertex Processing Performance

- Proportional to number of vertices
- Proportional to number of (assembly) instructions
  - Compute constant expressions on CPU

} Shader compiler takes care

- Post TnL cache critical
  - Much more so than lists versus strips!
  - Must use indexed primitives to access it
  - Allows for drawing up to 1 tri/0.5 vertices computed
  - Free tools reorder your mesh optimally
    - http://developer.nvidia.com

NVIDIA.

Setup/Rasterization

- Collects post TnL vertices into triangles
- Culls and clips
- Rasterizes triangles into fragments
- Per-Vertex data interpolates to per-fragment
  - linearly
  - perspective-correct

NVIDIA.

Setup/Rasterization Performance

- Not much control over it, but…
- Does not matter: rarely the bottleneck
- Degenerate triangles are free
  - Likely that all vertices hit PostTnL cache
  - No rasterization cost
  - Even up to 25% degenerates are okay

NVIDIA.

Programmable Fragment Shader

- No connectivity info/no access to neighbors (SIMD)
- ~8 Billion VECTOR operations/s! (~32GFlops/s)
  - Multiple parallel fragment pipes
  - Parallel RGB vector plus alpha scalar pipe
  - Multiple operations per pipe and clock
  - “Simple” operations include dot4, mad, sin, pow, lg2, table (texture) look-ups
  - Vector swizzles/conditional writes are free

NVIDIA.

Fragment Shader Data Formats

- IEEE s23e8 32 bit floating point per component
- Optional OpenEXR s10e5 16 bit fp per component
  - Same format as endorsed by ILM and other studios
  - In case 16 bit floating point is good enough
  - And performance is critical
- 12 bit fixed point precision

Table (Texture) Look-Ups

- Additional free operations:
  - Bi-Linear filtering for table (texture) look-up
  - Mip-level computations
  - Partial derivative computations
- Shadow maps (free depth compare on read)
- Up to 16 different textures
  - Sampled an arbitrary number of times
- Unlimited dependent texture reads

Texture and Render Target Features

- 1D, 2D, 3D, cube-map, rectangle textures
- Textures and render targets with (per component)
  - 8 bit fixed point
  - OpenEXR 16 bit floating point
  - IEEE s23e8 32 bit floating point
  - Mix and match above
- Free texture compression: HILO and S3TC
- Vertex array render targets

Fragment Shader Performance

- Wider formats more expensive
  - Requires more bandwidth
  - Requires more computation
- More temporaries more expensive
- Longer shaders more expensive

(Shader compiler takes care)

- Non-local texture look-ups more expensive
  - But 2D neighborhood is cached
  - Behavior still better than L1 cache-misses

Other Free Computation Units

- Occlusion queries
- Last century's tech:
  - Frame-Buffer blending and alpha-testing
  - Stencil operations
    - Super-Accelerated via two-sided stencil, stencil-only
  - Z-Buffer operations
    - Super-Accelerated via early z-cull, z-compression

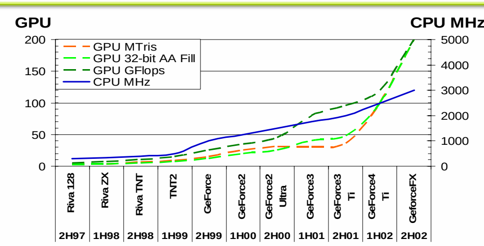
Available Z and Stencil Operations

- Selectable stencil test
  - Test against value in stencil buffer
  - Reject fragment if test fails
  - Perform distinct stencil operation when
    - Stencil-Test fails
    - Z-Test fails
    - Z-Test passes
- Selectable z-test
  - Reject fragment if test fails

Performance Considerations

- Occlusion query: use it asynchronously
- Alpha blending: reads and writes frame buffer
- Stencil-Only pass (no z- or color-writes): extra fast
- Z-Cull: render coarsely sorted front-to-back
- Clear() best way to clear color, stencil, or z
  - Turn off color-, stencil-, or z-writes when unneeded
  - But do not mask individual color components

And the Future Is Blindingly Bright…



Avg. 18month CPU Speedup: 2.2
Avg. 18month GPU Speedup: 3.0-3.7

Last Year's Intro Revisited

- Programmability: Lack of programming tools
- Lack of precision
- Formal models for performance evaluation
- Only a certain class of problems can be mapped to the graphics hardware

Lack of Programming Tools?

- Cg
  - C-Like high-level language
  - Compiles to vertex-/pixel-shader profiles
  - Integrated with OpenGL and/or DirectX
  - Cross-OS support: Windows, Linux, …
  - DirectX HLSL compatible
- DirectX's HLSL (Windows/DirectX only)
- OpenGL's SLang (when spec finalized)

Lack of Precision?

- Yes, limited to 32bit floating point per component
  - No support for doubles
- But 32bit floating point from start to finish of pipe
  - No ifs, buts, or whens
  - At least on NVIDIA's Geforce FX family of GPUs
- Smaller formats available for optimizations
  - When 32bit floating point is overkill

Formal Performance Eval. Models?

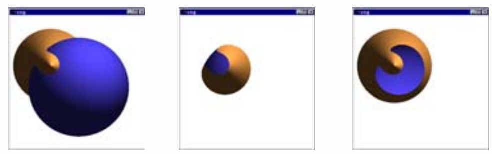
- Not aware; architectures are still changing rapidly
- But: Lots of good stuff available in the trenches
  - Websites, e.g., http://developer.nvidia.com
    - Lots of GPU performance presentations
    - Lots of GPU performance white-papers
  - IHV's Developer Relations
  - Game Developer Conferences
    - Lots of GPU performance talks and discussions
- Shader compilers/drivers optimize for you

Only Certain Problems Map to GPU

- GPU likes
  - Not needing to know about neighbors
  - Closed form solutions (CPU prefers iterative)
  - Table-Lookups (CPU dislikes if causing cache thrash)
  - ‘Deep Thought’ problems
  - Vector operations
  - All pipe processors busy all the time
- GPU dislikes
  - Synchronizing to the CPU (and vice versa!)
  - MIMD
  - Branching

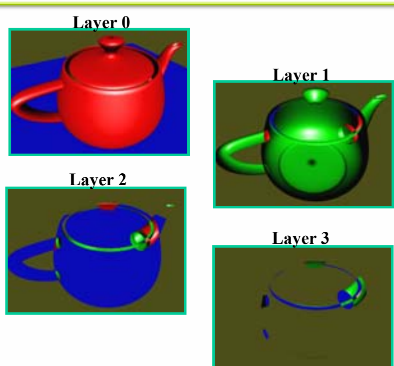
Known GPU (Ab)Uses

- CSG via stencil ops:
  - [Wiegand 1996]
  - [Stewart, Leach, John 1998, 2002]



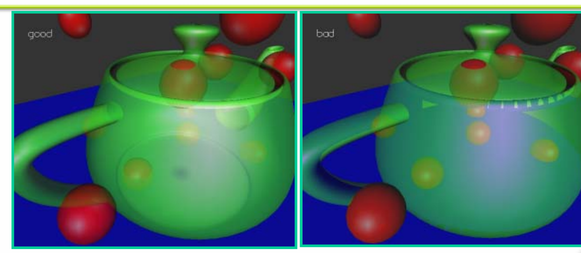
cone ∪ sphere   cone ∩ sphere   cone − sphere

Depth Peeling



- Display pixels at n^{th} layer of depth
- Repeatedly render to depth buffer, but reject pixels previously determined to be ‘closest’

Order Independent Transparency

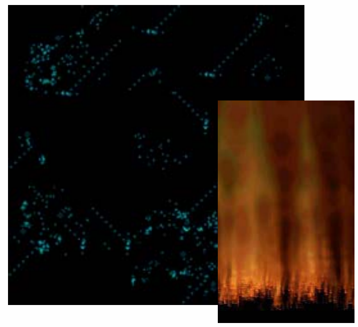


- Corollary to depth peeling [Everitt 2001]:
  - Compute all depth peels
    - Stop when no pixels rendered (occlusion query)
  - Blend depth peels back-to-front

Particle System Physics

- Translate iterative computations to closed form
- Solve closed form physics for every particle (vertex)
- [Wloka 2001]

Game of Life/Fire Simulation



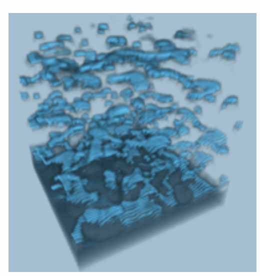
- Sample render-target texture multiple times to determine neighbors' state
- Use dependent 'rule'-texture read to determine new state
- [James 2001]

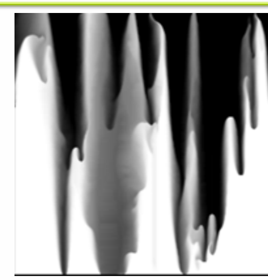
Height-Based Water Simulation

- Simulate height-field dynamics
- Generate normals from height field
- [James 2001], [Elder Scrolls III: Morrowind]

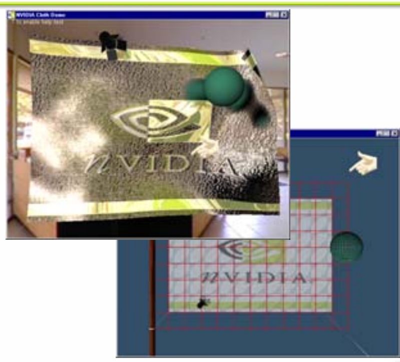
Boiling (2D and 3D)
Rayleigh-Bénard Convection (2D)

- [Harris 2002]





Simple Collision Detection/Response



- Check every vertex for intersection w/ sphere
- Displace vertex out of sphere
- [Wloka 2001]

All the Previous Stuff Runs On…

- Geforce 3, anno early 2001 !!!
  - More restrictive pixel-shaders
    - No floating point formats
    - Only 4 textures, 1 sample per texture (per pass)
    - Maximally 8 math instructions
    - None of the fancy ‘simple’ instructions
  - Much lower performance
- 2003: All features described here available
  - As PC graphics cards
  - At multitude of price-points ($79 and up)
  - Corresponding to performance

Current GPUs Allow

- Ray-Tracing [Purcell et al 2002]
- Cloth simulation via render to vertex-buffer [Green 2002]
- Scientific computations

Advertisement: Implementing a GPU-Efficient FFT

- Case study of:
  - Take a highly CPU-optimized algorithm and …
  - Convert it to run (well) on GPU
- Feasibility checks
- Step-By-Step CPU to GPU conversion
  - Things to avoid
  - Things to strive for
- Optimizing the GPU implementation
  - Taking advantage of GPU’s peculiarities

Thanks to...

- Dinesh Manocha for organizing this course
- Matthias Wloka for writing this presentation

Questions, Comments, Feedback?

- John Spitzer, spit@nvidia.com
- http://developer.nvidia.com