The Quiet Revolution In Interactive Rendering

Matt Pharr
Neoptica

November 9, 2005

Offline Rendering 5 Years Ago

Shrek (PDI/Dreamworks)

Interactive 5 Years Ago

Quake 3 (id Software)

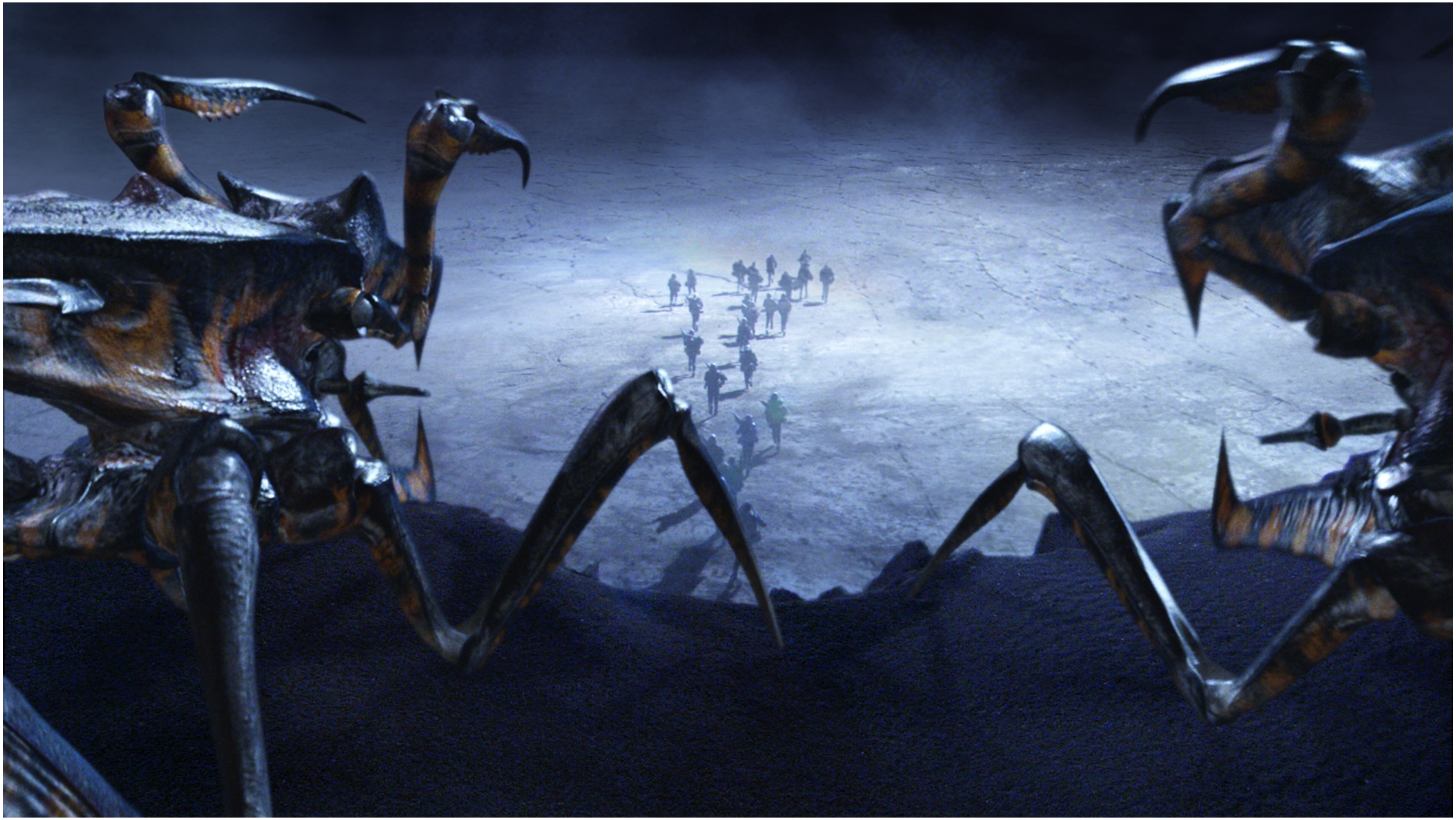
Modern Offline Rendering

Madagascar (PDI/Dreamworks)

Modern Interactive Rendering

The Getaway 3 (SCEE)

Modern Offline Rendering



Starship Troopers (Tippett Studio)

Modern Interactive Rendering

I-8 (Insomniac Games)

Are the offline images 1,000,000 times better than the interactive ones?

What's Happened in 5 Years?

- The remarkable story of modern graphics processing units
  - GPUs have taken much better advantage of semiconductor trends than CPUs
  - Consistent $>$ Moore's law performance growth, no signs of slowing down
- Interplay of GPU capabilities and software R&D
  - New graphics algorithms invented that use GPU optimally
  - Few approaches from offline have been useful for interactive
  - $\rightarrow$ Offline-quality doesn't necessitate using the old offline algorithms

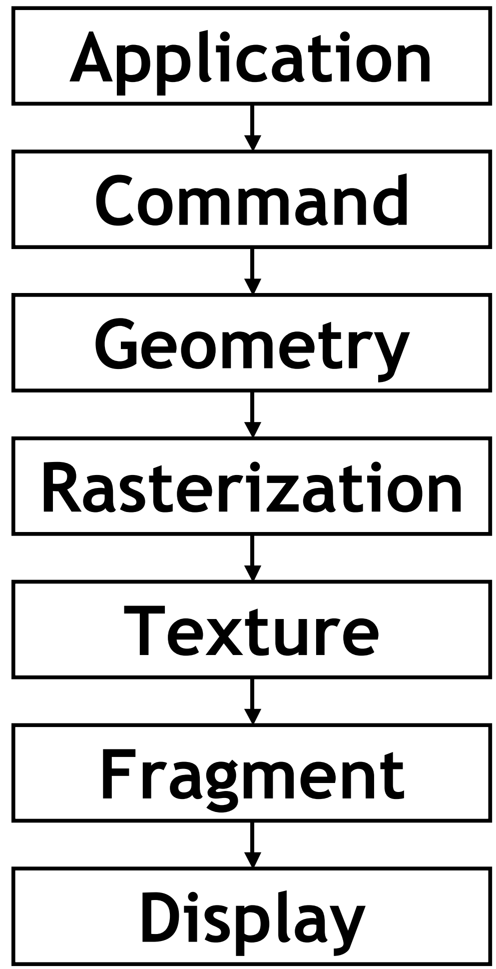
Overview

- Technology trends and graphics hardware
- Characteristics of the two types of rendering
- What factors contribute to the 1,000,000x perf. difference?
  - How efficiently does offline use the CPU?
  - How is innovation in interactive rendering algorithms improving image quality?
    - Hardware's impact on software and algorithms
- Open challenges and the impact of future architectures

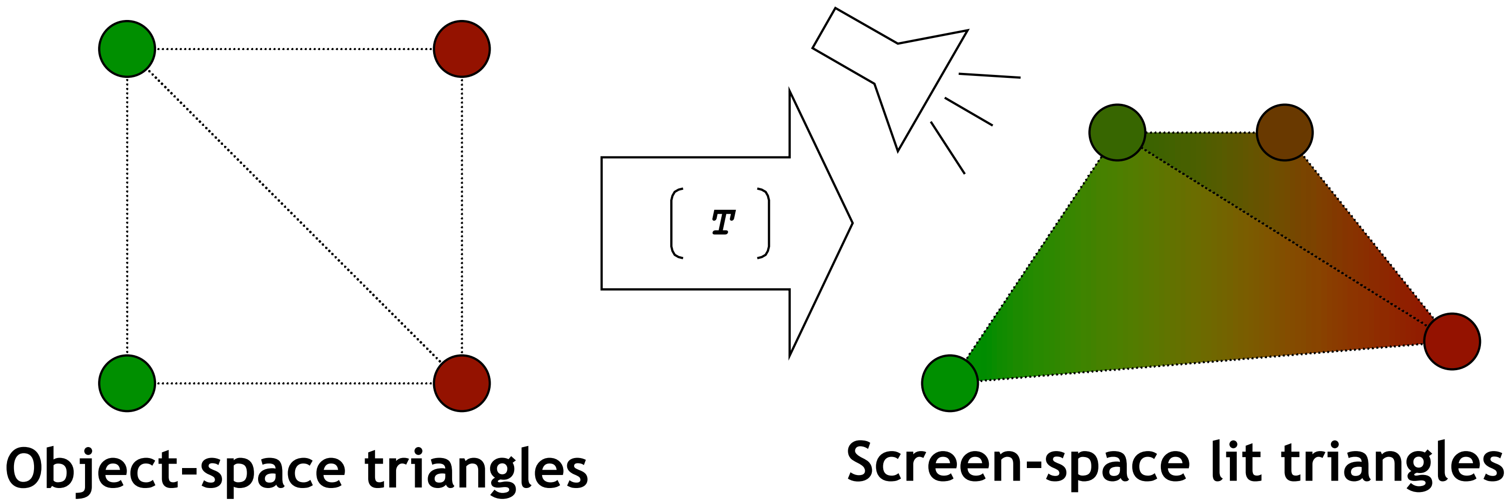
Technology Trends And Graphics Hardware

Project Gotham Racing 3 (Bizarre Creations)

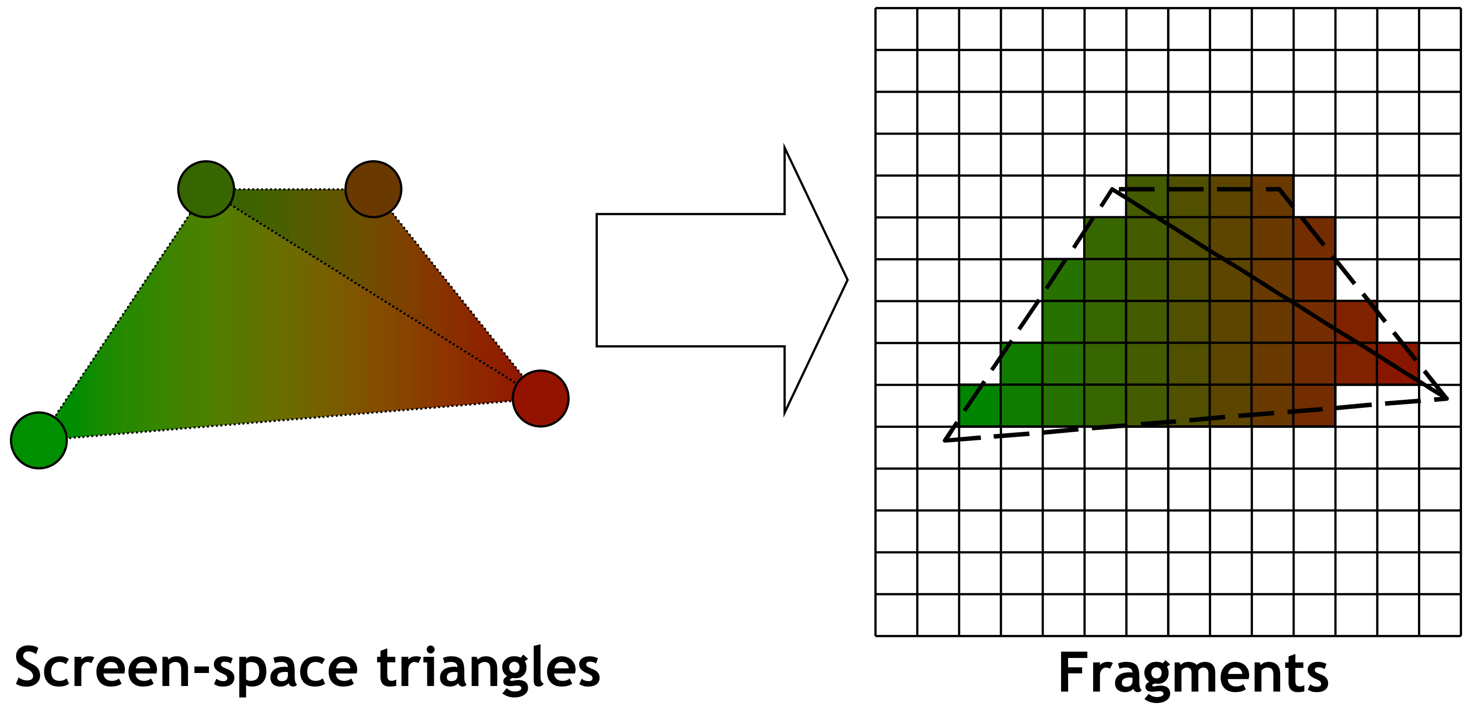
The Basic Hardware Graphics Pipeline



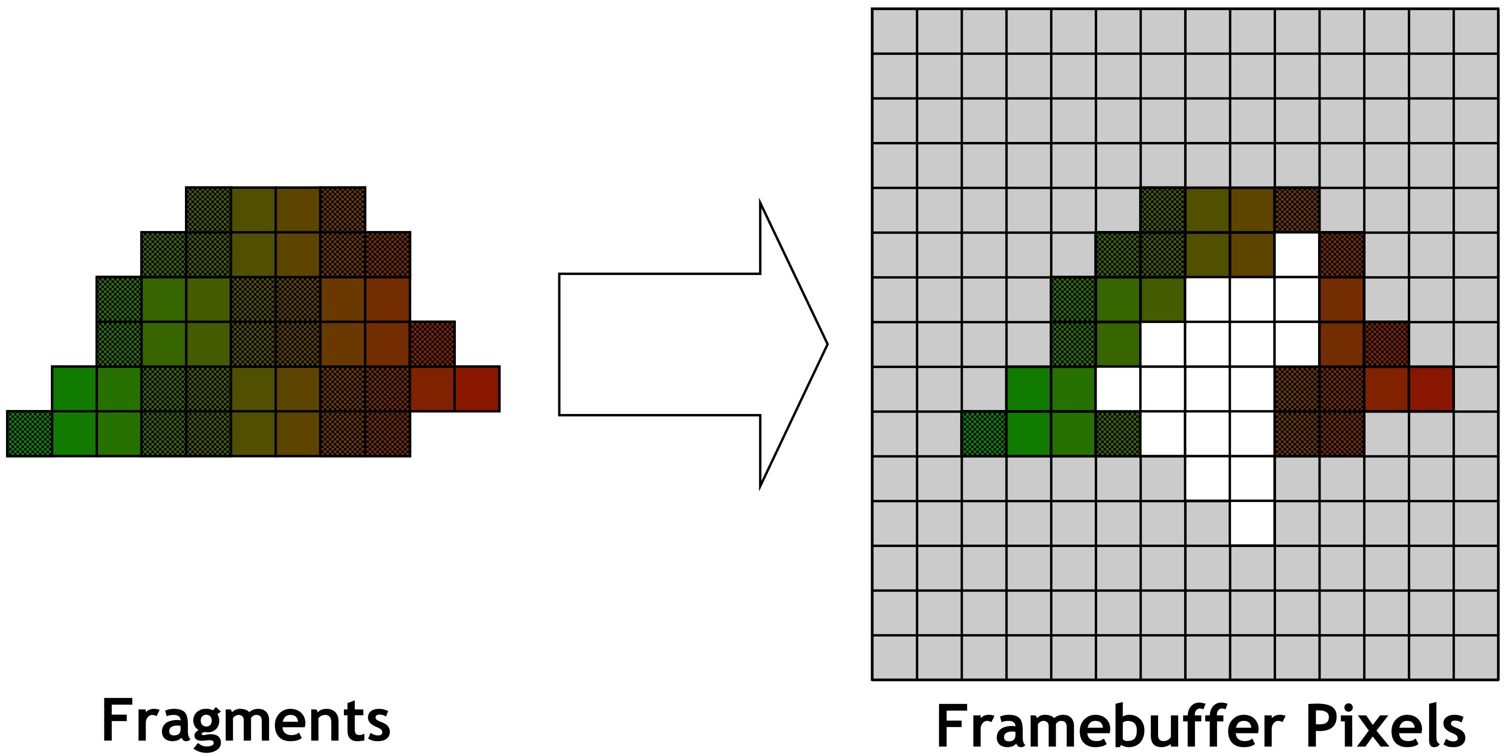
The Basic Hardware Graphics Pipeline



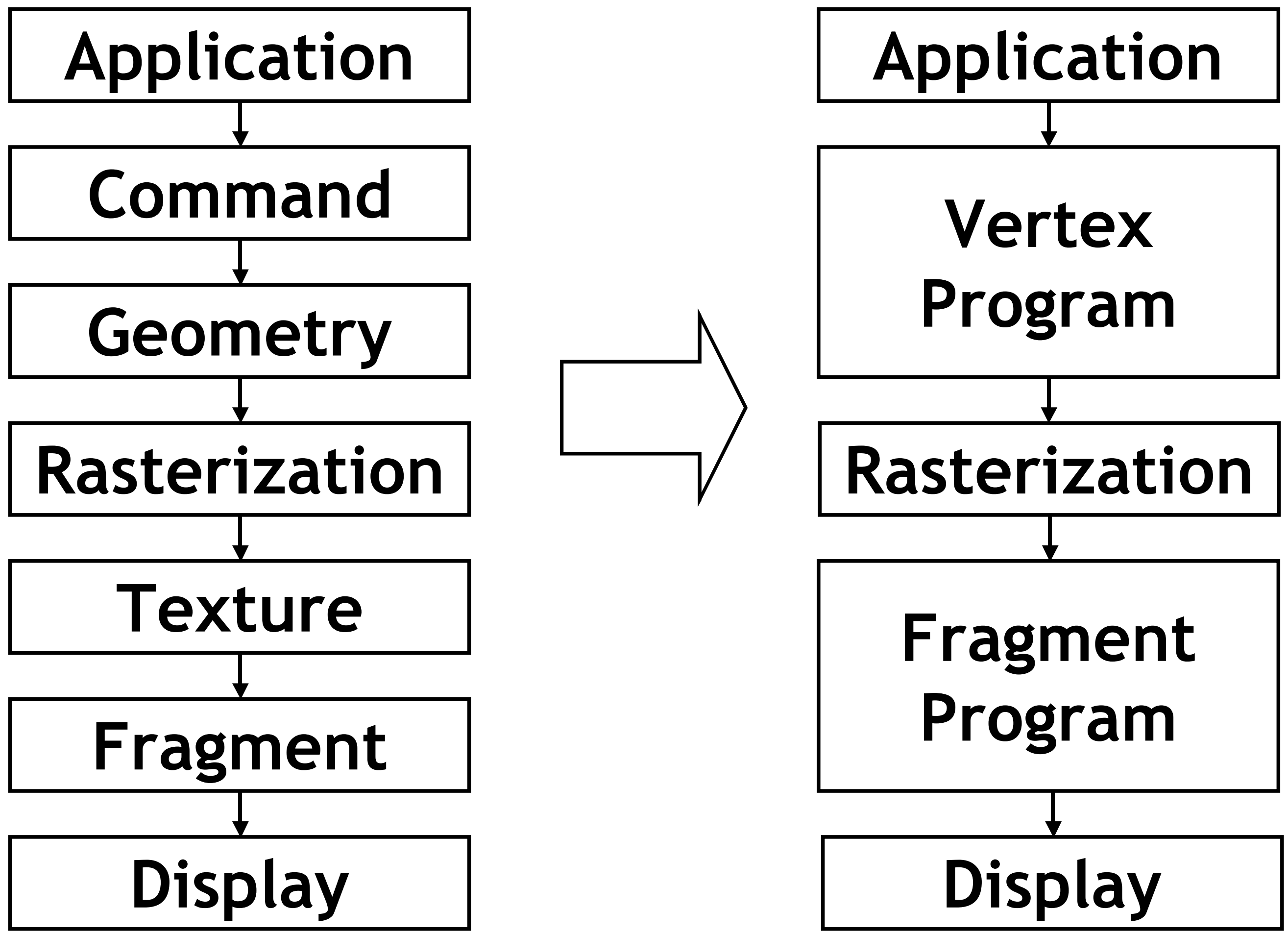
The Basic Hardware Graphics Pipeline



The Basic Hardware Graphics Pipeline



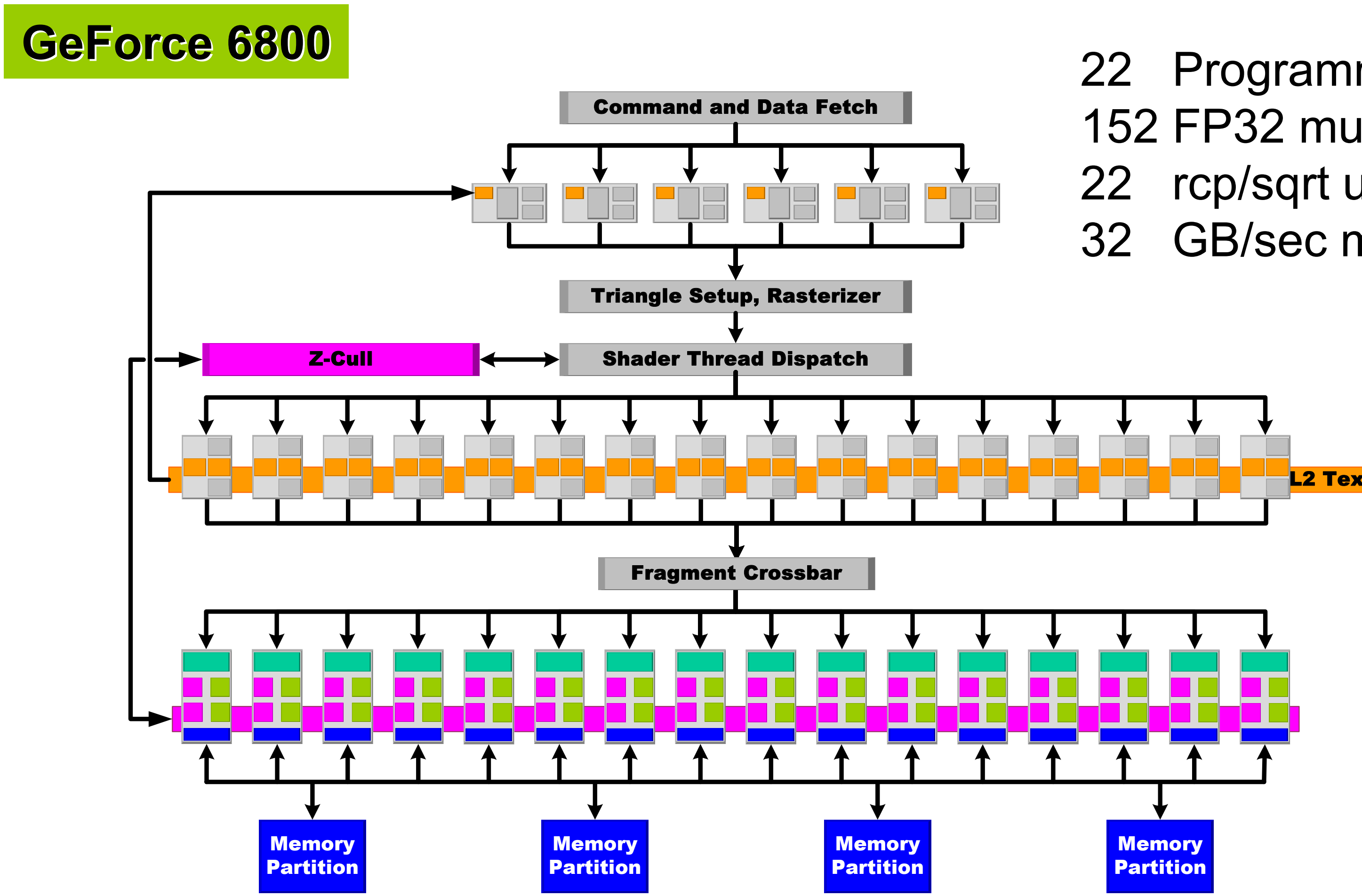
The Programmable Hardware Graphics Pipeline



GPU Architecture

- Highly parallel
- Efficient triangle rasterization
- Separate programmable vertex and pixel processing
  - 32-bit floating point math
  - ADD, MUL, RCP, CALL, RET, ...
  - Arbitrary memory reads. Writes limited.
  - Wasn't programmable at all in 1999!

GPU Architecture



22 Programmable Cores
152 FP32 mult/add units
22 rcp/sqrt units
32 GB/sec memory BW

(Bill Mark and Henry Morteon)

GPU Parallelism

- Task, data, and instruction parallelism all used
- 8 vertex processors, 24 pixel on GeForce 7800
- Vertex
  - 5 FLOPs per clock per processor
  - MIMD
- Pixel
  - 8-12 FLOPs per clock per processor
  - SIMD

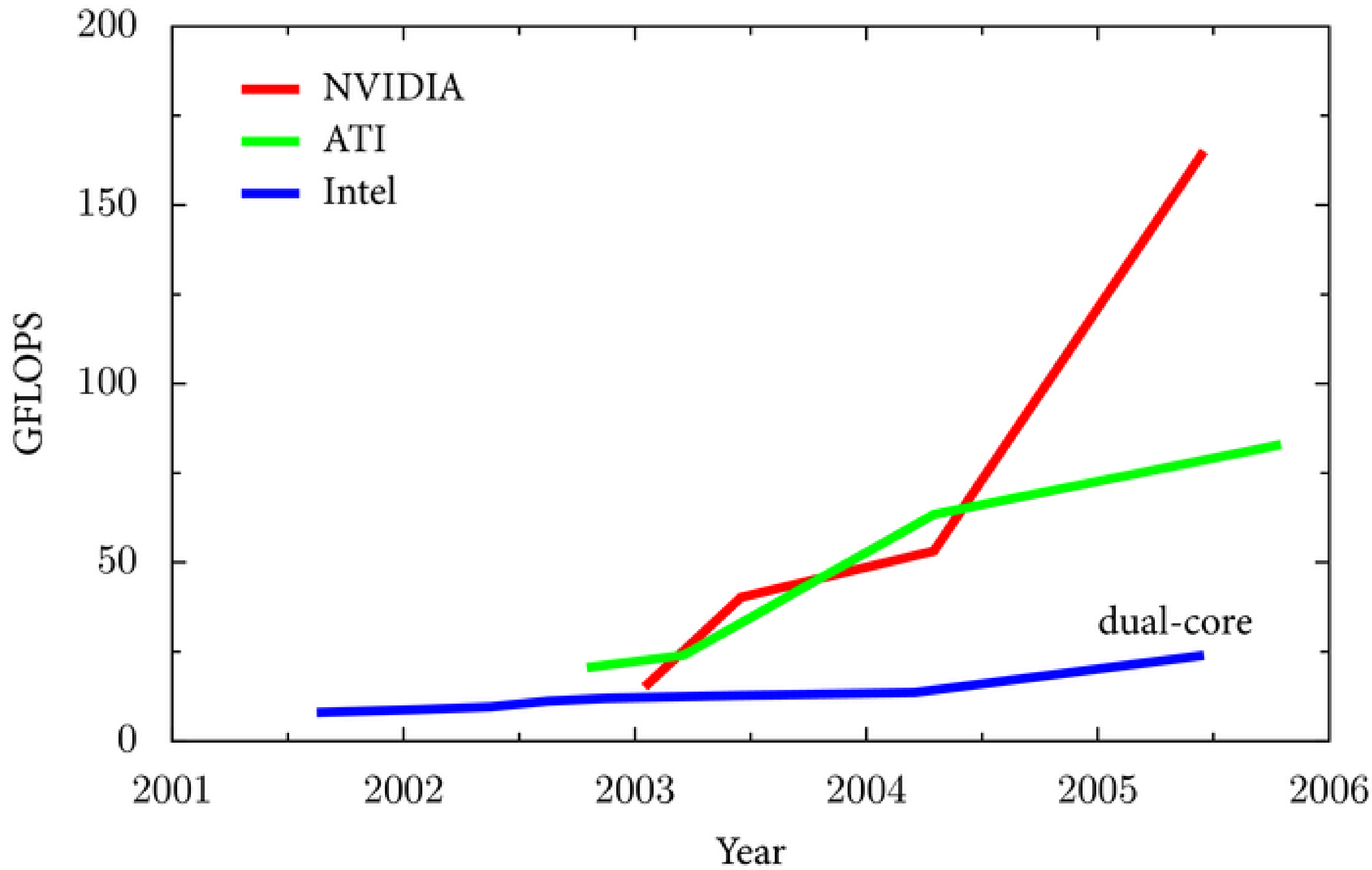
GPU Memory System

- Specialized for streaming linear access
- Arbitrary writes (mostly) not allowed
  - Avoids ordering problems from parallel execution
- Impact of memory latency well-hidden
  - This makes peak perf. easier to get than on a CPU

Good News and Not Enough Good News...

- Transistor density (Moore's law, 50%/yr)
- Clock speed (15%/yr)
  - Together these give +71% per year capability
  - a.k.a. 15x in 5 years.
- DRAM bandwidth increasing at 25% year
- DRAM latency decreasing at 5% year

Implications for CPUs and GPUs



(John Owens, UC Davis)

What Do You Get In a $400 GPU?

- Computation: peak ~150 GFLOPS
- Memory architecture
  - 256MB local memory
  - ~24GB/s to local mem
  - 1-2 GB/s to system mem

Compute/Bandwidth on Modern GPUs

- FLOPs per word of off-chip bandwidth
  - 2002: 2
  - 2003: 2.66
  - 2004: 6
  - 2005: 10
- It's easy to be b/w limited...

Characteristics Of The Two Types of Rendering

Formula One Racing (SCEE)

Rendering As Data Compression

- Start with multi-GB scene description
- Do a significant amount of computation
- Generate a few million RGB pixels (few MB of output)
- How much pre-processing work is worthwhile to speed up computation?

Offline Rendering

- Pre-rendered images
  - Hours per image, no problem
- Generally passive viewer
- Movies, TV, etc.
- Almost completely done on CPUs
  - Flexibility is most important
  - Director is king; may make significant changes late in the process
  - May tweak a character, textures, etc., from shot to shot

The Offline Rendering Problem

- Goals: high quality ("perfect") images
- Throughput generally more important than latency
- Render a handful of times until happy with results
- Then put it on film and you're done

Offline Rendering: Implications

- Scene description can be as complex as necessary
  - Add detail as much as needed to achieve look
- Slow frame causes artists/TDs pain, doesn't matter to consumer
- Optimizing the pipeline has limited benefit: cost/benefit ratio is different than for interactive

The Interactive Rendering Problem

- Latency is the only thing that matters
  - Slow frame is unacceptable: avg of 60 fps doesn't matter if sometimes it's 2 fps
- Almost entirely using graphics processors (GPUs)
- Render billions of times
- Harder than offline:
  - User moves the camera (subject to constraints)
  - World is dynamic/can be changed by the user

Interactive Rendering: Implications

- Scene descriptions are necessarily efficient
- Frame to frame coherence is taken advantage of
- Time spent on precomputation/scene optimization can be amortized over billions of renderings
- Very important to find ways to use the GPU efficiently

How Efficiently Does Offline Rendering Use the CPU?

Resident Evil 5 (Capcom)

Offline Efficiency of CPU Use

- Guesstimates based on Pellacini et al. 2005, "LPICS: A Hybrid Hardware-Accelerated Relighting Engine for Computer Cinematography"
- Video frame 4h9m for 216k pixels
- 13.7M shading calculations (63/pixel!)
- Assuming:
  - Avg. 100k executed shading instructions
  - 90% of time spent on shading
  - 4:1 CPU instruction per shading instruction
- $\rightarrow$ 6 TFLOPs to render image

Offline Efficiency of CPU Use

- 6 TFLOPs to render image
- In 4h9m, CPU can do 179 TFLOPs
- $\rightarrow$ CPU utilization of 3.3%
  - Waiting for data from memory, disk
  - Software overhead in return for flexibility

Offline Efficiency of CPU Use

- 30x from poor CPU utilization
- 10x from GPU FLOPS / CPU FLOPS
  - ˜40x next-gen console FLOPS / GPU FLOPS
- $<$ half the 1M difference
- Still another factor of 1000-3000x to account for
  - Some due to image quality difference
  - Some due to more efficient algorithms in interactive

How Is Innovation in Interactive Rendering Algorithms Improving Image Quality?

S.T.A.L.K.E.R. (GSC Game World)

Three Big Contributing Factors

- GPU performance cliffs are large
  - Must stay close to GPU fast path
  - Easier to achieve good GPU utilization than good CPU utilization?
- The benefits from staying on the GPU fast path are enormous
- Everyone has a GPU
  - Many more developers working on interactive algorithms
  - Millions of GPUs in PCs $\rightarrow$ larger incentive to use efficiently

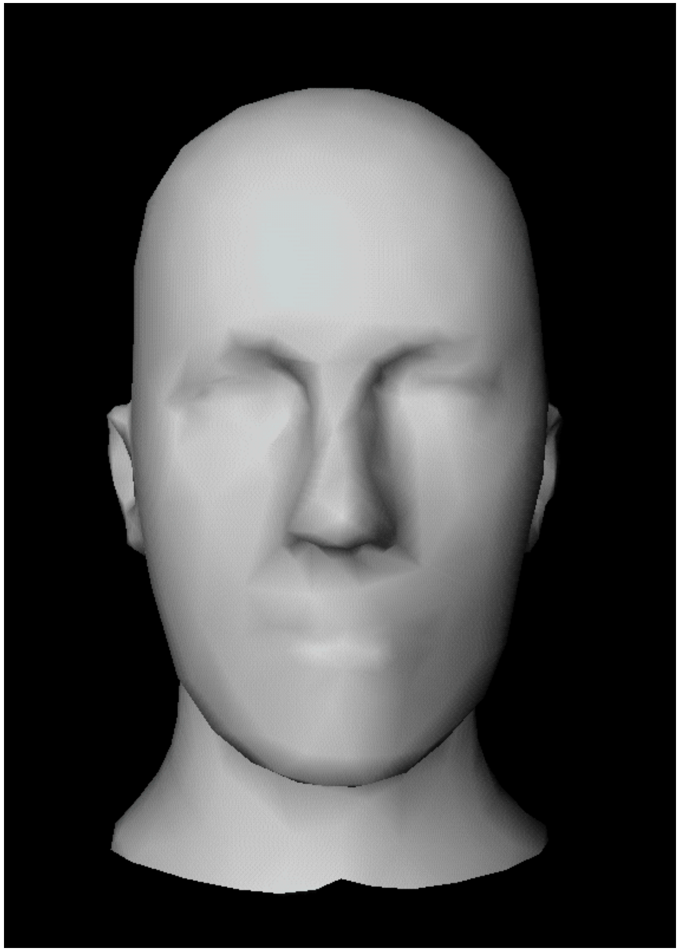
How To Be GPU-Friendly

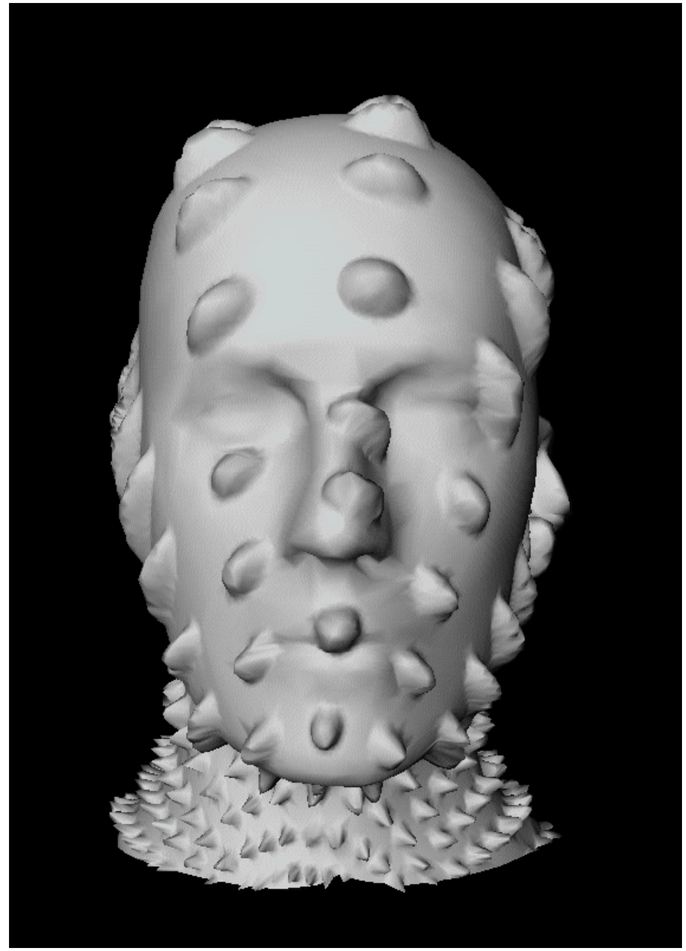
- Move per-vertex work to per-pixel if doing so makes triangles bigger / reduces the number of vertices substantially
- Pre-process now for faster rendering later
  - Visibility / potentially visible sets
  - Model simplification / optimization
  - Precomputed radiance transfer
- Examples
  - Displacement mapping
  - Billboards
  - Mesh simplification
  - Ambient occlusion

Displacement Mapping

- Classic technique from offline for adding fine detail to surfaces
- Texture map defines offset from base surface
- Offline approach:
  - Finely tessellate to pixel-sized triangles
  - Move triangle vertices appropriately
  - Aggressively discard triangles when done with them

Displacement Mapping





(Ivan Neulander, Rhythm and Hues Studios)

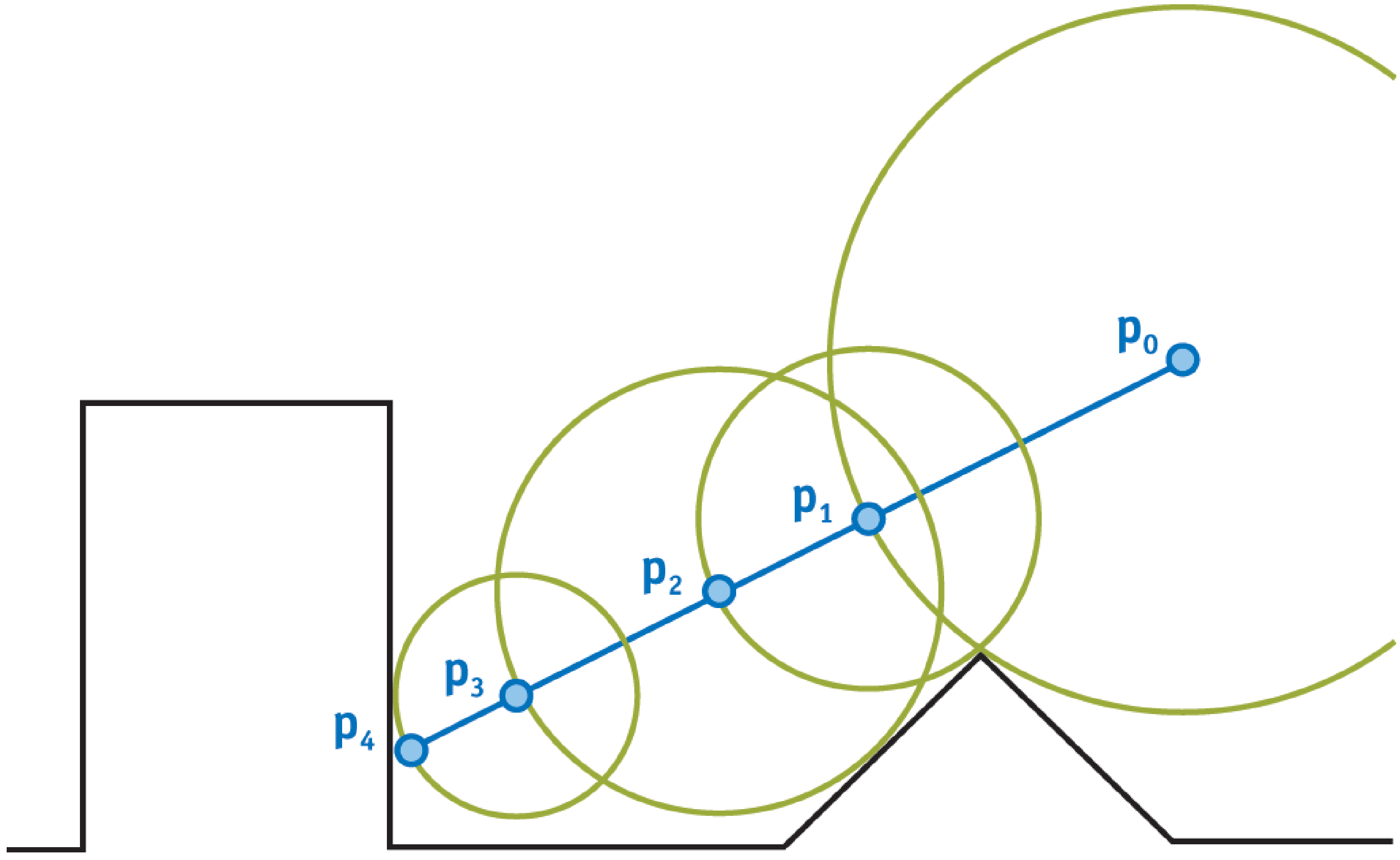
Displacement Mapping

- Offline approach not at all suited to current hardware:
  - GPU is balanced for ~8 pixel big triangles
  - Not enough vertex processing power for many small triangles
  - Very small triangles cause poor utilization throughout the pipeline
- Therefore, invent new techniques that give the same effect but are better suited to the hardware

Displacement Mapping

- GPU has much more pixel processing power than vertex
- Very small triangles are bad all around
- $\rightarrow$ draw bigger triangles, but do work in pixel processor to compute effect of displacement
- Representative approach: Donnelly's distance map-based ray tracing

Distance Map-Based Sphere Tracing



Bump Mapping

(William Donnelly)

Displacement Mapping

(William Donnelly)

How Do You Render A Forest Full of Trees?

- Desired scene complexity is growing faster than image resolution
- Offline: model each tree down to the leaves, even the trees that are a mile away
  - "Wow, how come the renderer is so slow?"
- The above was a slight exaggeration
  - When sufficiently painful, will also model simpler trees for the distance, use billboard impostors
  - This incurs cost that must be worthwhile in the grand scheme of things

How Do You Render A Forest Full of Trees?

- Interactive: pain threshold is much lower than offline
  $\rightarrow$ much quicker to go to more efficient representations
  - Similar issues with very small triangles and bad GPU utilization
  - How can you draw complex objects in a way suited to the GPU?
  - And by the way, dynamic lighting would be nice as well
- More sophisticated billboard representations
  - Small pre-processing cost, significant benefits
  - Represent the model in a way that is friendly to GPU

Billboard Clouds

Behrendt et al.

Billboard Clouds

Behrendt et al.

How Do You Render a 2M Polygon Character?

- Offline: draw 2M polygons
- Interactive: No thanks! Can we simplify without losing detail?

Unreal Engine 3 (Epic Games)

How Do You Render a 2M Polygon Character?

Unreal Engine 3 (Epic Games)

How Do You Render a 2M Polygon Character?

Unreal Engine 3 (Epic Games)

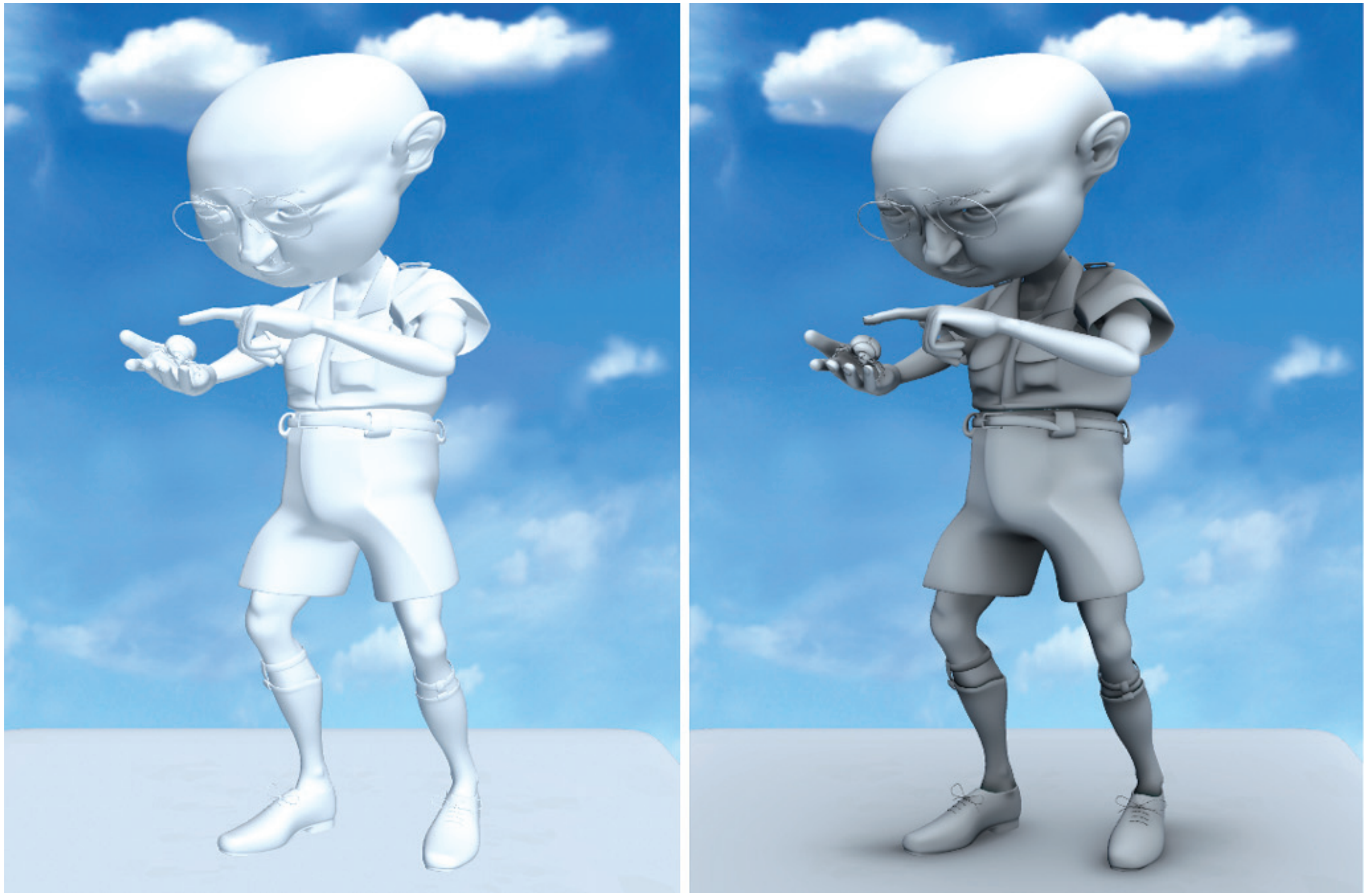
5,287 Triangles Is Much Nicer

Unreal Engine 3 (Epic Games)

Ambient Occlusion

- Technique pioneered by ILM
- Observation: for static model, precompute how exposed to the environment each vertex is
- This value is very useful for shading–makes crevices dark, etc.

Ambient Occlusion

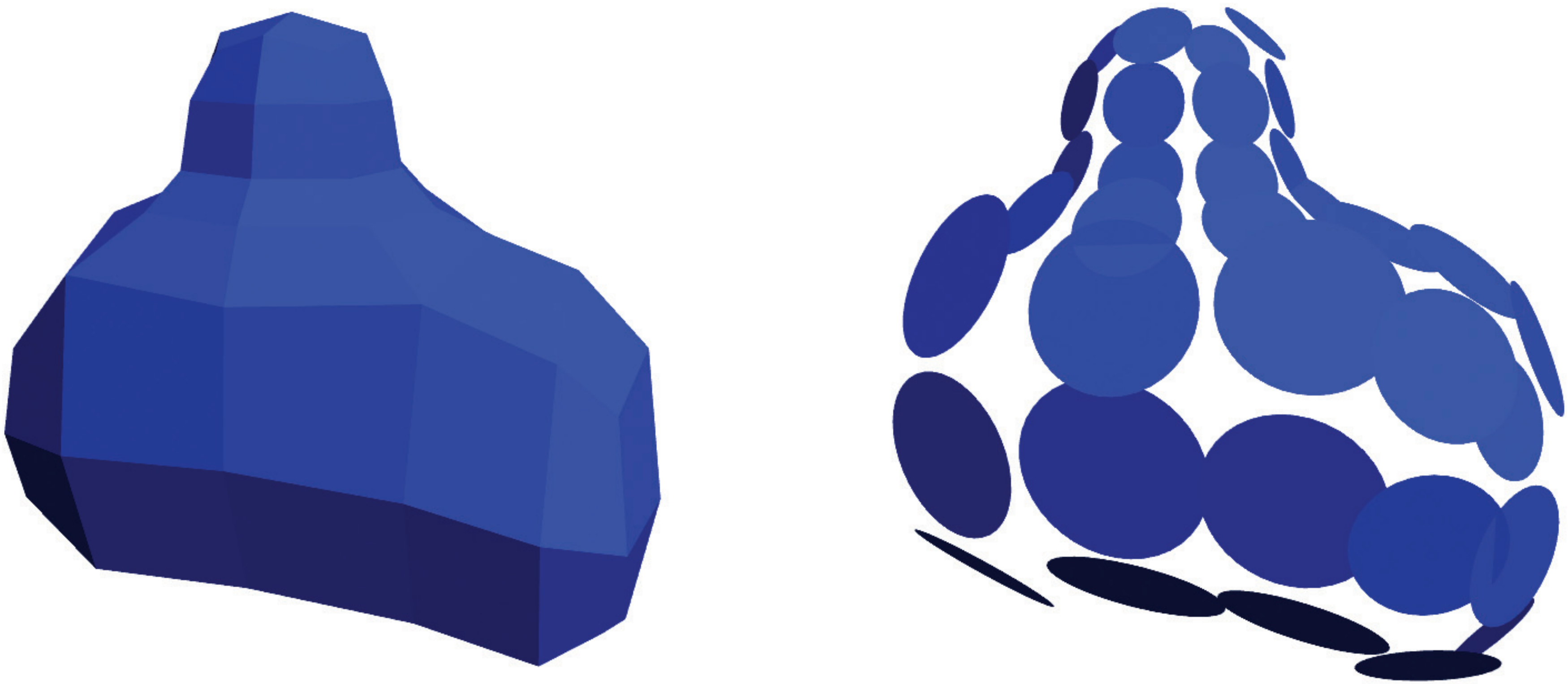


Bunnell

Ambient Occlusion

- Offline approach
  - Compute these values in a preprocess
  - Look them up at render-time
- Directly applicable to interactive
- But what if the model is animated?

Represent Mesh With Oriented Disks



Bunnell

Dynamic Ambient Occlusion

- Represent mesh with oriented disks
- Build tree to represent them hierarchically
  - Far away groups of disks can be merged into a single disk, etc.
- At each point to shade, traverse tree, adaptive termination
  - This is easy to do on a GPU pixel processor
- Result: interactive dynamic ambient occlusion

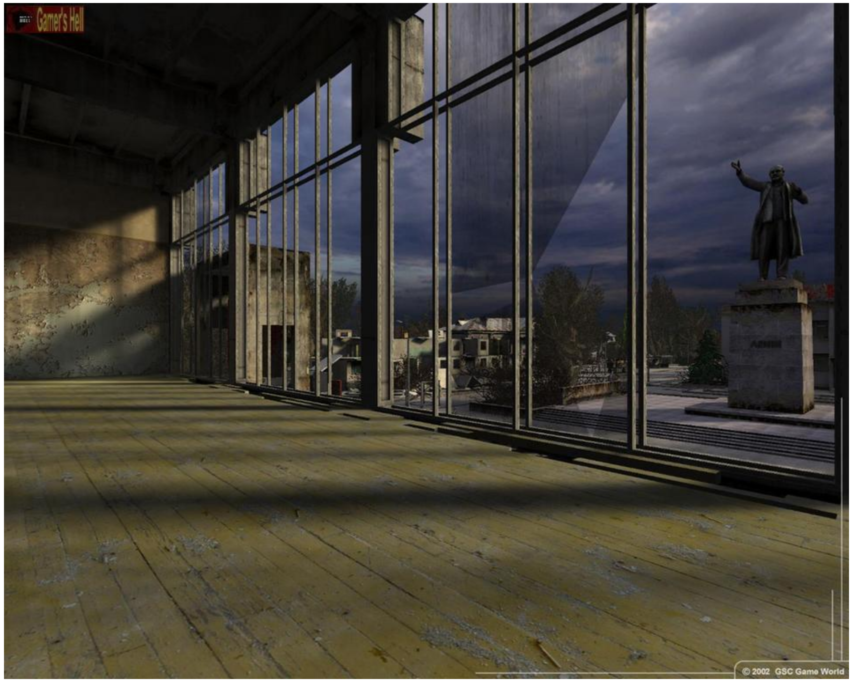
Ambient Occlusion

- Weaknesses: not 100% accurate
- But it does look right, and works well in practice...
- And it's interactive!

The Good News For Interactive Rendering

- The complexity found in offline scenes is not a prerequisite to images of offline quality
  - Number of objects, shaders, textures, etc, only tangentially relevant
- Offline has long claimed the pain they go through is necessary for high-quality images
  - This is demonstrably wrong
  - "Toy Story"-quality will be (has been?) rendered in real time with far fewer FLOPs than were used to render it originally

Open Challenges and Future Architectures



S.T.A.L.K.E.R. (GSC Game World) /
The Courtyard House (Henrik Wann Jensen)

Open Problems in Interactive Rendering

- Solved already in offline
  - No more visible polygons (curved surfaces should look curved)
  - Good transparency solution
  - Anti-aliasing
- Not yet completely solved anywhere
  - Infinitely detailed environments
  - Dynamic lighting in dynamic environments

GPGPU For Graphics

- "General purpose computation on GPUs"
- GPUs have many FLOPs $\longrightarrow$ use them for numerical computation
- Many techniques for abusing the GPU to apply those FLOPs to non-graphics problems
- (See gpgpu.org)

GPGPU For Graphics

- GPU as data parallel processor
- Memory system designed to stream through data
  - Not so good for data reuse though
- GPGPU application areas
  - Protein folding
  - FFT, matrix computation
  - ...

GPGPU For Graphics

- Can use approaches from GPGPU to do different types of graphics on GPU
- Not limited to rasterizing triangles, GPU z-buffer approach
- Purcell et al's and Carr et al's GPU ray tracing, ...
- Rapid improvement in GPU capabilities makes this increasingly appealing

Upcoming Console Architectures

- PlayStation 3 (Cell + RSX)
- XBox 360 (Multicore PPC + GPU)
- 100s of GFLOPs on both CPU and GPU
- Most important, fast connection between the two
  - ~20GB/s bidirectional bus
  - vs. PCI-E 4GB/s peak (not yet seen in practice)

Implications of Console Architectures

- GFLOPs available on both GPU and CPU
  - Can get perf. even with branchy code, small amounts of parallelism
  - Can now consider algorithms not suited to GPU alone
- Bandwidth allows round trips
  - No longer limited by the unidirectional PC graphics pipeline
  - Though N.B. the 40x ratio of FLOPs/float b/w
- What is the future for PCs?

What Can The Two Sides Teach Each Other?

- Offline $\longrightarrow$ interactive
  - Quality and variety of visual effects to strive for
  - Not so much on the algorithms side?
- Interactive $\longrightarrow$ offline
  - What quality is possible from interactive?
  - Can it deliver the last 5% in quality? At what cost?
  - Are there benefits of giving up that last 5%?
    - e.g. artists are more effective

What Is The Future of Offline Rendering?

- Rate of innovation in interactive shows no sign of slowing
- But what is wrong with 12 hour render times, anyway?
  - Only a problem if someone is sitting waiting for it; doesn't directly affect the consumer
- Specialized tools can be effective if they deliver "good enough" for the job at hand
  - e.g. Pixar's LPICS lighting tool
- Is the engineering cost of fixing the pipieline less than the cost of having artists wait?

Future Architectures

- Not much graphics left in graphics hardware
- Will something new change this trend?
  - Hardware ray tracing?
- Or are GPUs soon to be a parallel array of fp units with a DVI connection on the back?
  - CPU and GPU manufacturers are both heading this way from different starting points
- Multi-core/Cell trends will continue on CPU side
  - Where best to put FLOPs and with what programming model still TBD

Future Issues on The Software Side

- How best to use 4,000 FLOPs per pixel?
- Will 40,000 actually lead to better images?
  - Big question for both h/w and s/w side
  - 30 MFLOPs (as in offline) probably never needed?
- Architectures aren't easy to program, debug on
  - Concurrency, asynchronous data transfer, ...
  - How best to use two FLOP heavy processors with very different sweet spots?

Implications

- Many big problems to address on the s/w side
  - s/w is again becoming the main area for innovation in interactive graphics
  - Until the wheel of reincarnation turns again...
- h/w manufacturers (and MSFT) have less control/influence
  - Good for developers
  - MSFT is likely more or less neutral to this
  - Hard for h/w vendors to differentiate on anything other than computation performance

Sources/References

- “Streaming Architectures And Technology Trends”, John Owens, in *GPU Gems 2*
- “Real-Time Programmable Shading”, Bill Mark, in *Texturing And Modeling: A Procedural Approach*
- Pellacini et al. 2005, “LPICS: A Hybrid Hardware-Accelerated Relighting Engine for Computer Cinematography”
- GPGPU slides (Ian Buck, Mike Houston, Pat Hanrahan, ...)
- John Owens Graphics Architecture slides
- Bill Mark Graphics Architecture slides

Acknowledgments

- Craig Kolb
- John Owens
- Bill Mark
- Aaron Lefohn
- Kiril Vidimče
- Doug Epps
- Eric Leven

Questions?