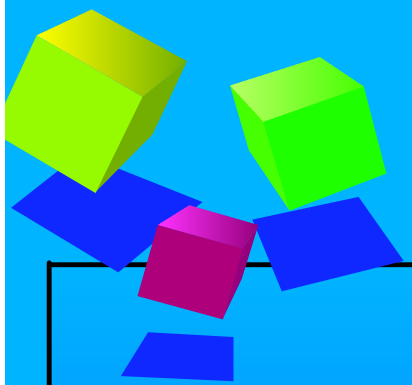
Ray Tracing Hardware

Jason Dale

CS395T - Realtime Graphics

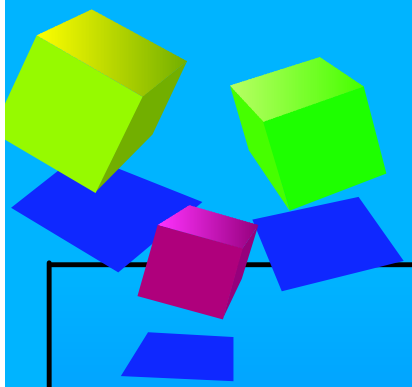
04/17/2003



Overview

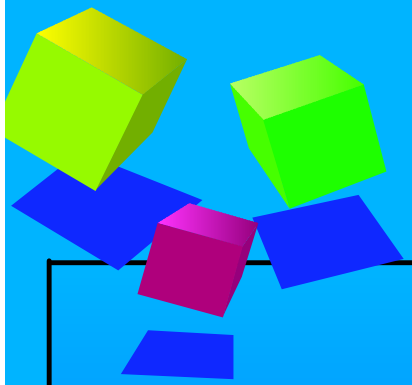
- "Ray Tracing on Programmable Graphics Hardware"
  - Purcell, et al. (Stanford)
  - Commodity GPU
  - Streaming characteristics
- "SaarCOR - A Hardware Architecture for Ray Tracing"
  - Schmittler, et al. (Saarland)
    - From "Interactive Rendering with Coherent Ray Tracing", Wald, et al.
  - Custom Hardware Design
  - Algorithms & tradeoffs
- Discussion

The diagrams in this presentation have been "quoted in bitmap form"



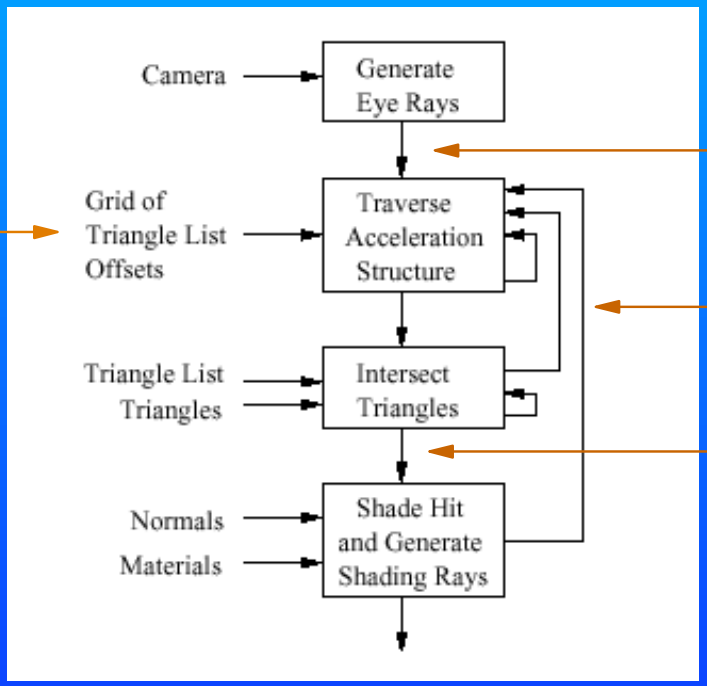
Purcell, et al.

- Ray Tracing with commodity GPU (Carr?)
  - High Level Simulation of GeForce
  - Also an 'Enhanced' version of the GPU:
    - Fragment processor with branch (MIMD) and FP
    - Mode general memory access
    - Expected in the "next year or two"
- Ray Tracing as streaming computation
  - Acceleration structure traversal (wavefront?)
  - Rasterization-optimized hardware

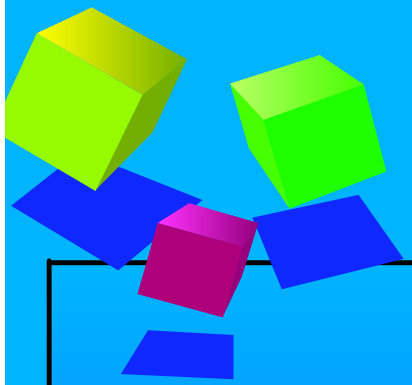


Streaming Ray Tracing

precomputed accel. struct

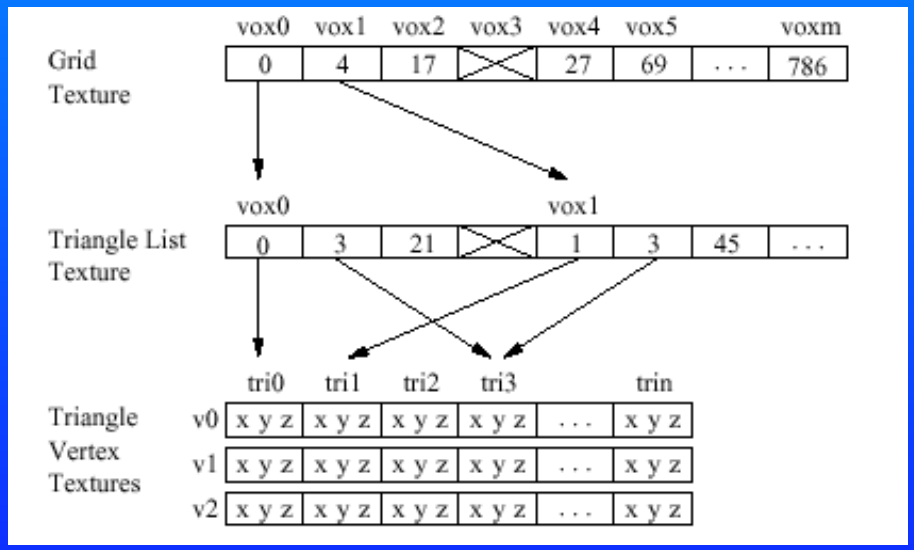


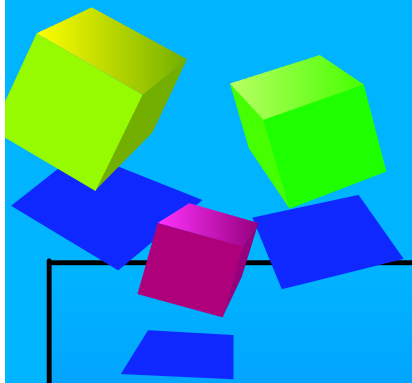
streams



Purcell Algorithm

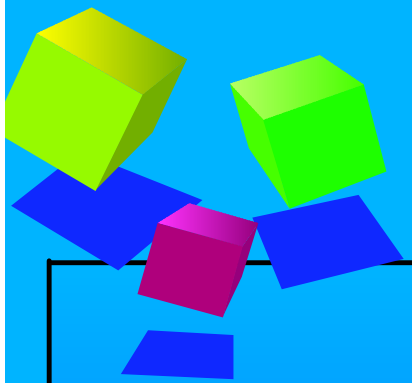
- Acceleration structure (3D-DDA)
  - Uniform grid structure
  - No tree traversal (branches...)
  - No mailboxing (regularity & parallelism)





Base Implementation - Plain Geforce 3

- Clever use of available resources (ala Brook):
  - Texture memory for storage (streaming)
  - No branches? Stencils will do!
    - "Fragment kill" for control
- Streaming
  - Interaction with texture caches
  - Stencil bits for predication
- Multipass:
  - Intermediates + Stencils = Big bandwidth
  - Ultra-SIMD
  - Not covered in detail here *because*... (drumroll, please)

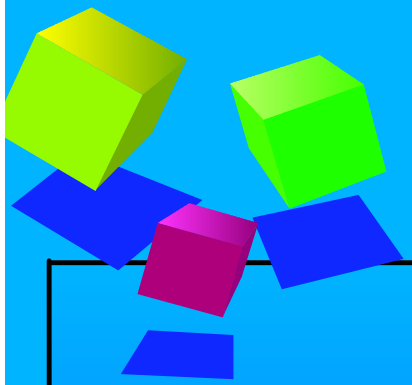


NVIDIA NV35

- Rumors from *www.nvnews.net* (it's *almost* official...)

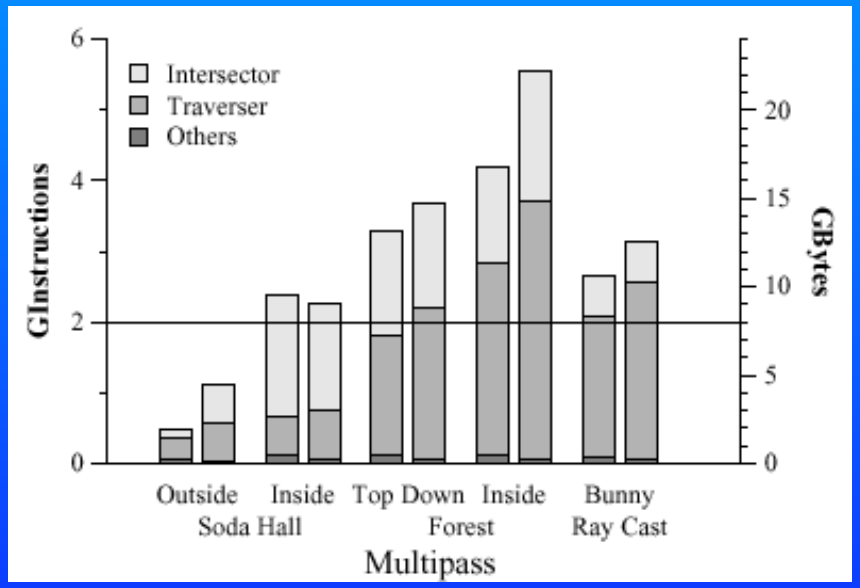
- Branches (Vertex or Fragment?)
- Fragment Programs
  - Up to 1024 instructions
  - Up tp 12 ops / cycle
  - Up to 16 textures / pass
- Vertex Programs
  - Up to 256 static / 64k dynamic insns, 64 loop iterations
  - Memory access?
- Up-to 128 bit FP and 128 bpp

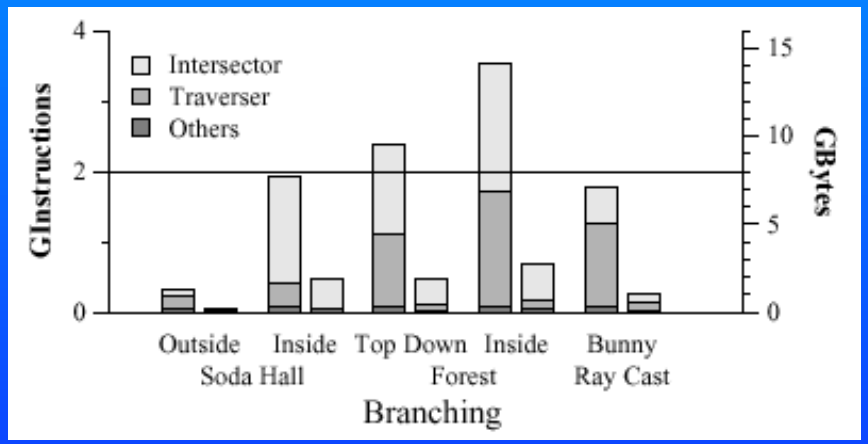
- 120 Watts?



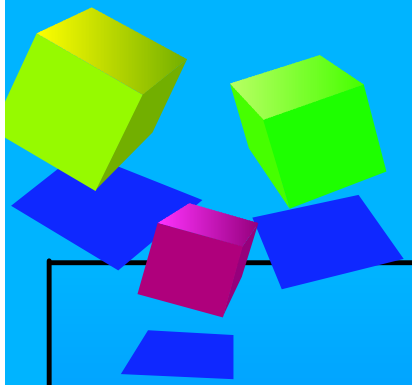
Results

- From cycle-accurate simulator
  - Plus 64k DM texture cache model



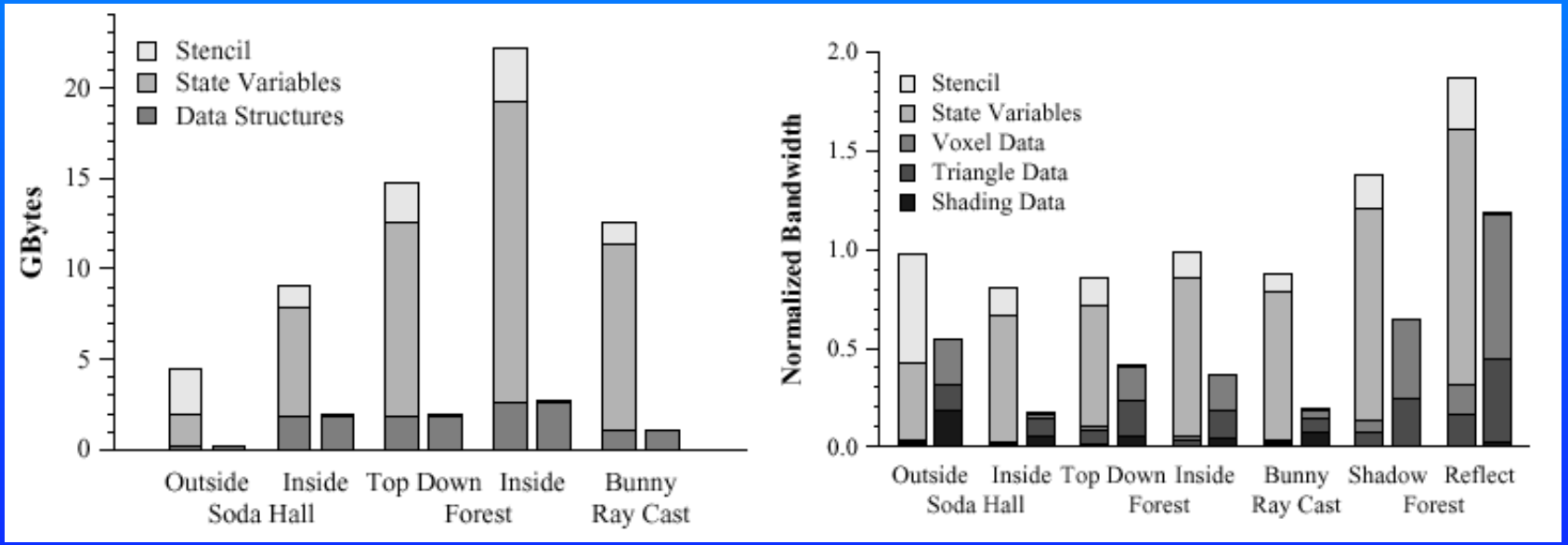


- Remember conditional streams?

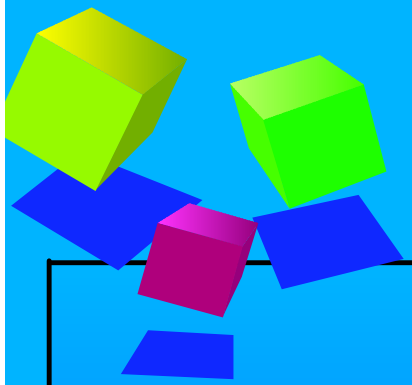


Results (contd)

- Multipass (optimized)
  - ~1000-3000 passes / scene (depending on worst behaving ray)
  - ~6-10% of rays processed per pass



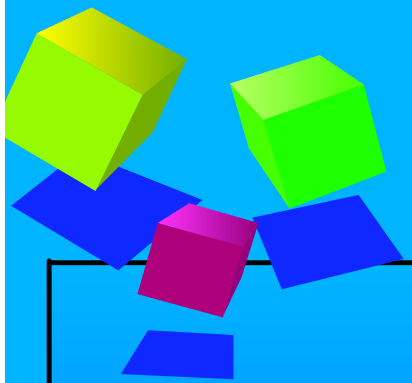
End of first paper -- Questions?



SaarCOR precursor

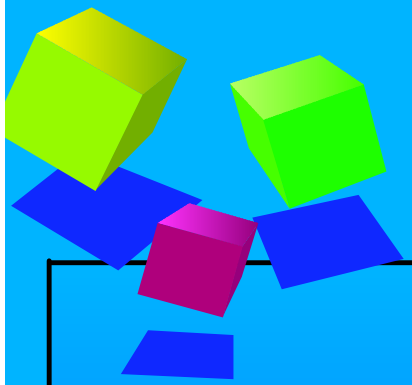
- "Interactive Ray Tracing with Coherent Rendering"
  - Wald, et al.
- Fast!
  - ~3 FPS on a P3 800
  - ~10x speedup over POVray / Rayshade
- BSP tree acceleration structure
- Lots of attention paid to data
  - Prefetching + memory layout
  - Data structure sizes (cache alignment)
- 4-way SIMD tests 4 rays against 1 triangle

- Interesting Conclusion... Memory Bound!



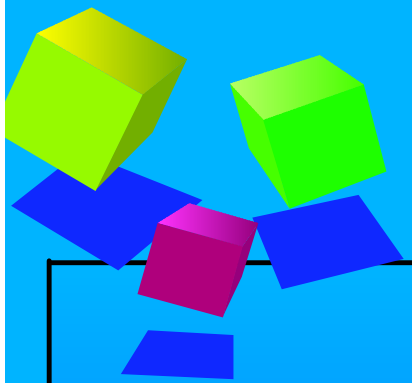
SaarCOR Goals

- Dedicated hardware for interactive ray tracing
- Simpler than general GPU (die + pins + mem)
- Focus on tracing of rays (not shading)
  - Acceleration structure (BSP tree)
  - Ray-triangle intersection tests
- Interaction of hardware + algorithm + data / mem



Algorithm

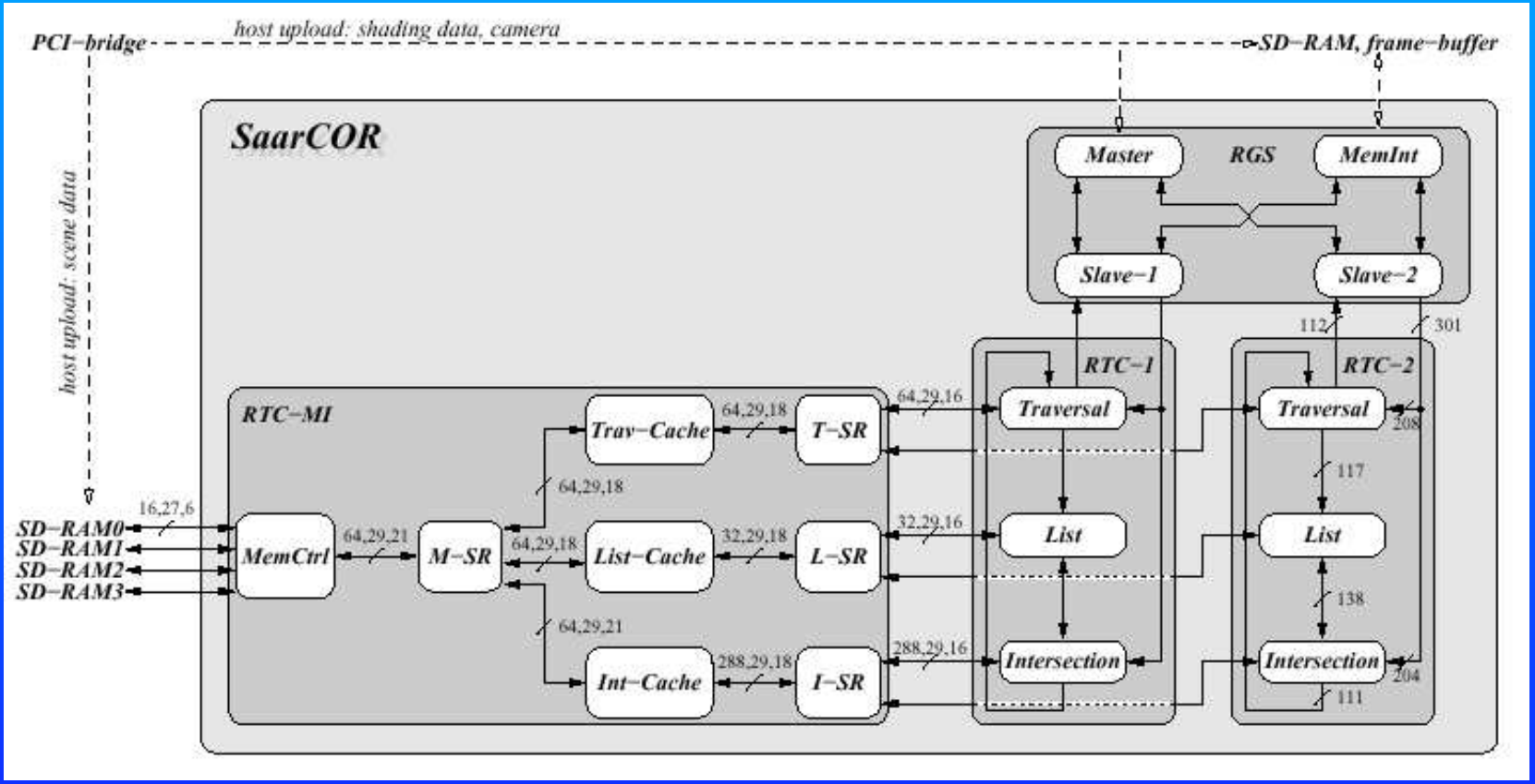
- Rays processed in large SIMD "Bundles"
  - If one ray enters BSP partition, so do the rest
  - If both children in BSP entered, push one onto stack
  - Static bundles (how are rays bundled?)
- Bit-vector to indicate if ray is active in tree
  - Turn off rays that are not part of computation
  - Bundle finished when no rays left active



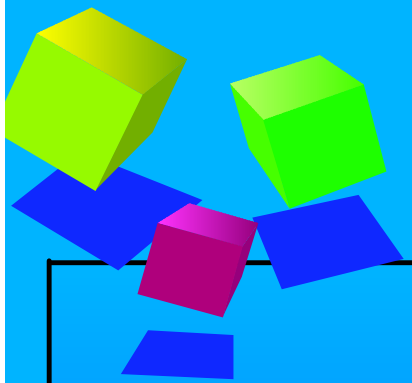
Hardware

- RGS Unit (Ray Generation and Shading)
  - Determines which ray to render next
  - Performs shading (Phong / Bilinear)
- RTC Unit (Ray Tracing Core)
  - Expensive (4-8x hardware of RGS)
  - Traversal subunit: To walk BSP
  - List subunit: To manage triangle lists
  - Intersection subunit: For ray-tri intersection
- RTC-MI Unit
  - Caches
  - Memory Interface

Microarchitecture



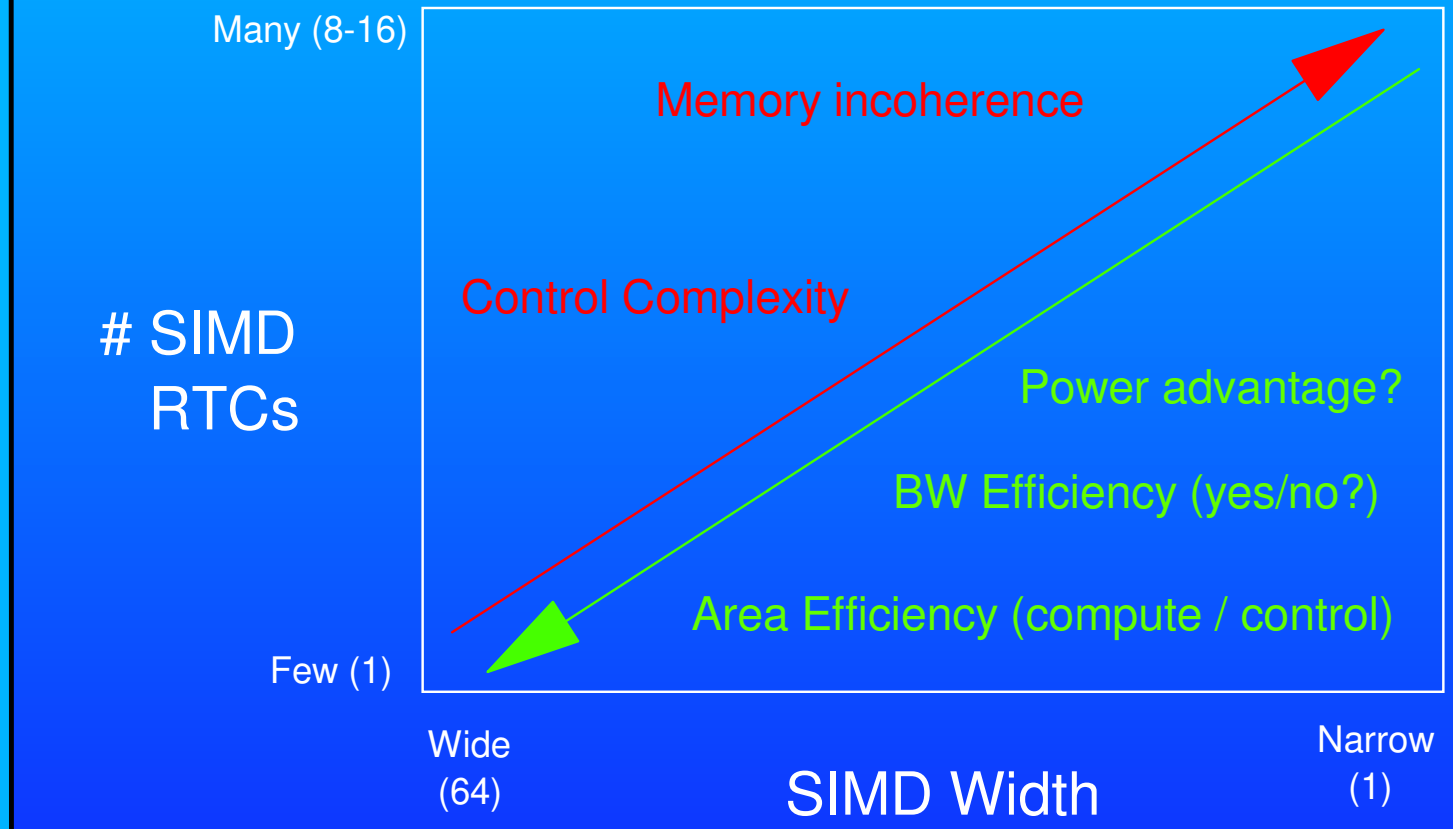
(Flexible Number of RTCs)



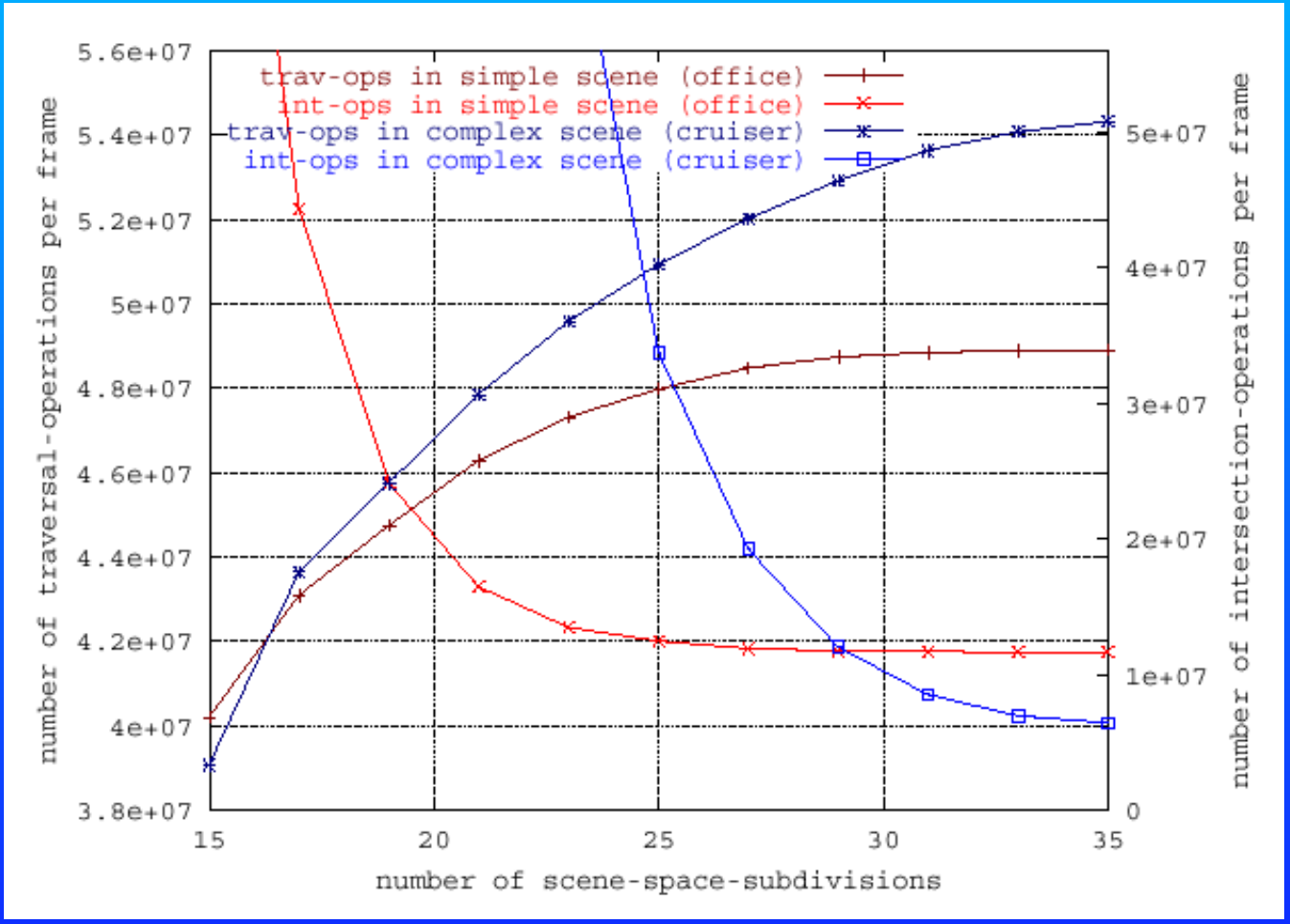
Computation and Data

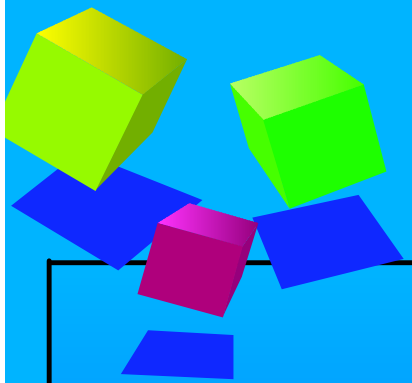
- BSP Traversal
  - Data: 64 Bytes per node
  - FP: 3+, 1*
- Intersection test
  - Data: 288 Bytes per triangle
  - FP: 12+, ~13*
- Shading
  - Data: (Texture Coherence?)
  - FP: ~50+, ~70* (including AGEN)
- Memory
  - 3 DM caches, 64kB - 144kB
  - Simple caching very effective
  - Address Hashing?

Complexity and SIMD



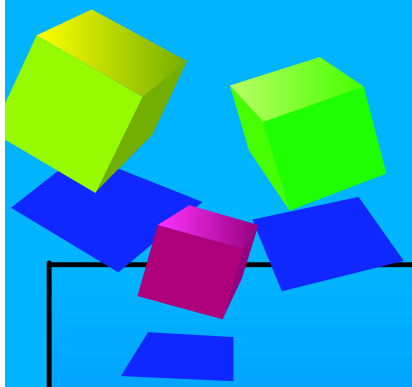
BSP tradeoffs





SaarCOR Results

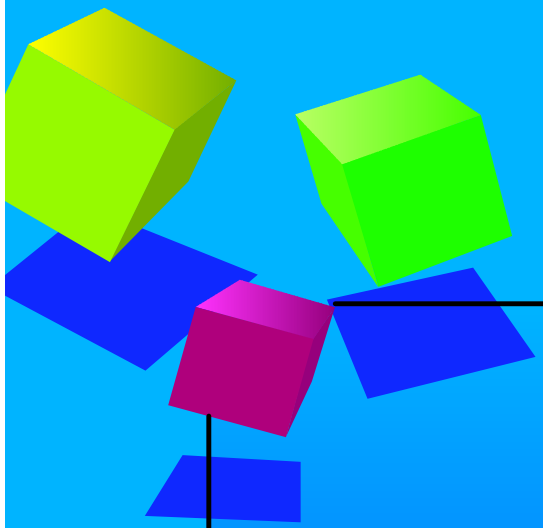
- 20-80 cycles/ray @ 533 MHz
- 10-200 fps (very scene dependant)
  - 1024x768 resolution (no multisampling)
  - Millions of triangles
  - Multiple lights and reflections
- Hardware:
  - ~ 1/2 FP HW of GPU @ 192 Units (pipe stages maybe?)
  - ~1M cache (very reasonable)
  - ~1/10 Mem BW of GPU (<1GB/s)
- Accuracy of the absolute numbers?



Results (contd)

- SaarCOR scalability (% of theoretical maximum)

scene	1 RTC	2 RTCs	4 RTCs
Quake3	76%	76%	78%
Sodahall	80%	79%	79%
CruiserGL	71%	65%	41%
Conf	67%	63%	59%
Cruiser	72%	63%	37%
Office	71%	68%	68%
BQD-1	73%	72%	71%
BQD-2	54%	46%	31%

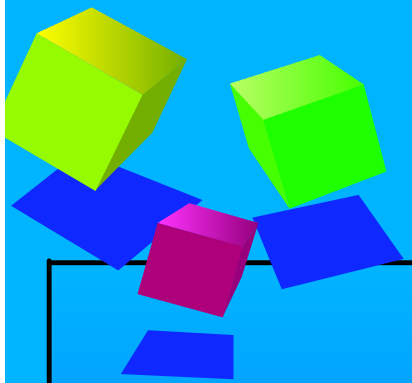


Discussion

Quick comparison

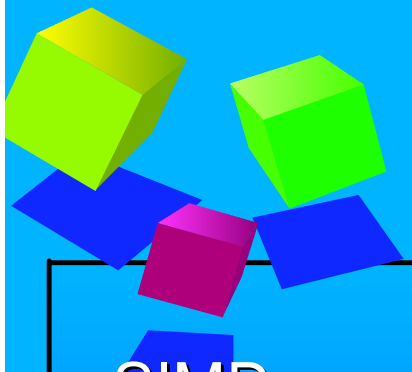
- Similarities:
  - Triangles only
  - Attention to parallelism and data access patterns
- Differences:
  - Migration vs. paradigm shift? (commoditization factor)
  - "Triangles only?" doesn't need to be true

	Purcell	Schmittler
Hardware	Commodity	Custom
Algorithm	Flexible	Fixed
Efficiency	Low	High
Data Mgmt.	Streams	Spatial Ray Coherence
Speed (apple-orange)	~56M intersections / sec (~211M w' branching)	10-200 FPS @ 1024/768



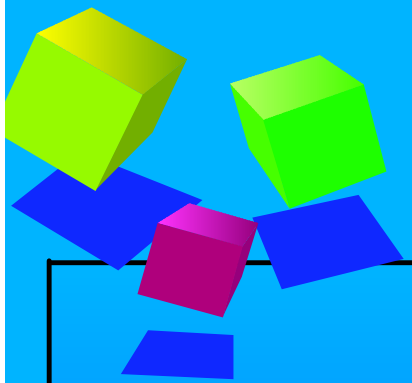
Purcell - GPU changes?

- Fragment Processor:
  - (Branches, Floating Point, etc...)
  - Streaming loads and stores
- Vertex Processor:
  - Lots of idle horsepower...
  - Some ability to access memory
- Making GPU more 'general purpose'
  - Goods: Flexibility, Programmability, Efficiency
  - Bads: "Jack of all trades" (like CPUs?)
  - Where to draw the line?



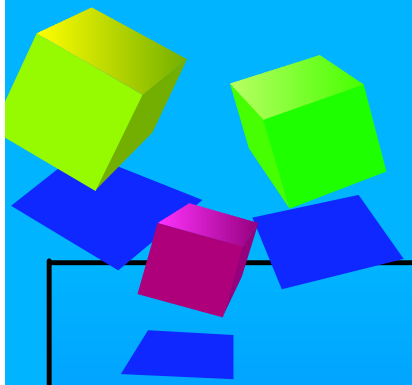
SaarCOR

- SIMD: parallelism vs. waste
  - Data shows ~64 Rays / bundle
  - Improved ray bundling?
  - Number of RTCs (simple bundle splitting?)
- Tradeoff traversal vs. intersection computations
  - BSP tree size - (size / depth) exponential
  - Data shows ~4 traversals / intersection
- Hardware
  - SIMD width vs. Control overhead
  - Fixed vs. Flexible functions
  - Lock step vs. Asynchronous RTCs
  - Multithreading: Good or bad? Tradeoffs?
    - Overheads (caches / data motion? communication?)
    - In SaarCOR (multithreading or just pipelining?)
  - DRAM characteristics?



SaarCOR

- Hardware-based BSP tree generation
  - Axis Aligned?
  - Others?
- Compared to Pharr paper
  - Types of coherence
    - Could both types be exploited?
  - Memory behavior (caching)
    - Acceleration structures
    - Granularity of locality



Parallelism

- Parallelism == Speedup ?
  - Yes: Required!
  - No: Overhead! (Classic DF systems, MP mem scalability)
- Streaming vs. vectors
  - Can insert object into stream
  - Stream size not fixed