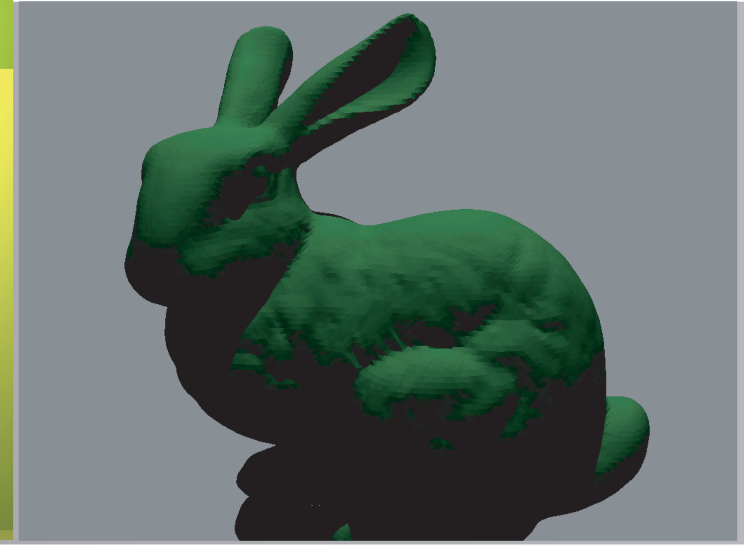
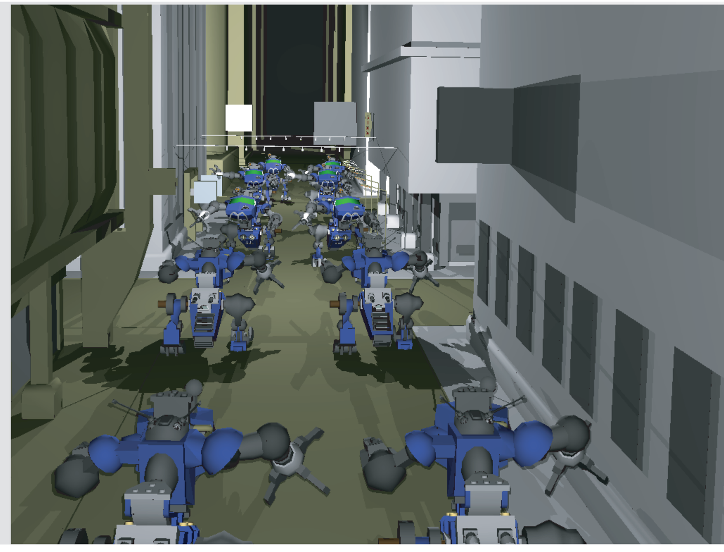
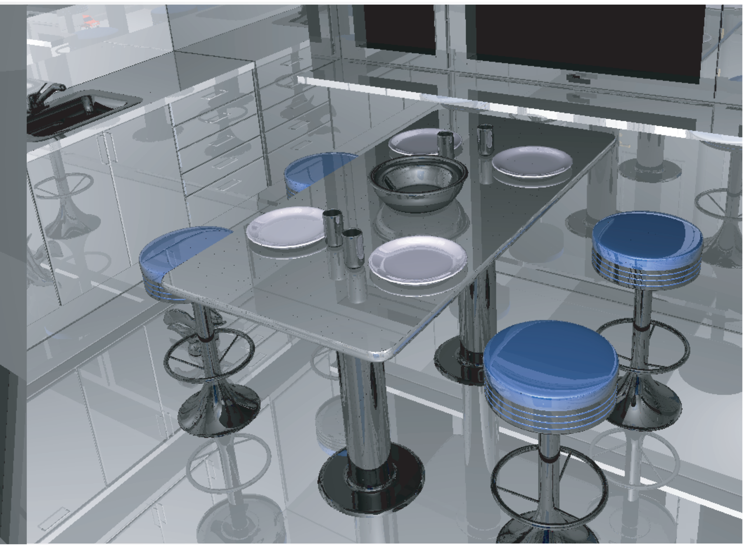


Software Caching for Cell Based Ray Tracing

J. Sugerman T. Foley S. Yoshioka P. Hanrahan

Symposium on Interactive Ray Tracing 2006



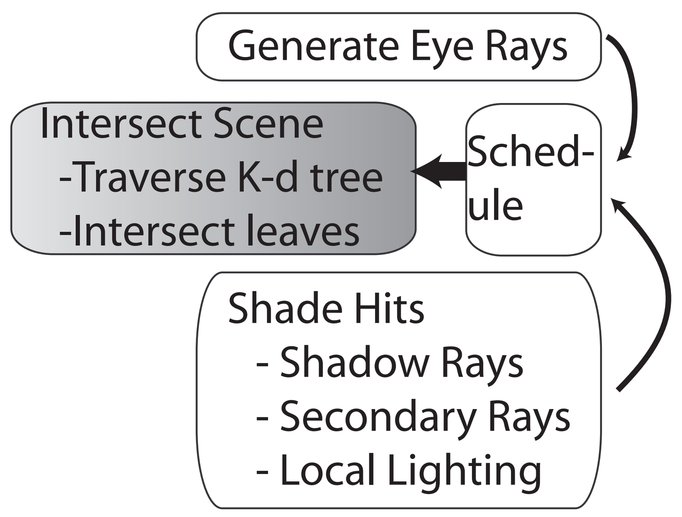


Background

The Cell Broadband Engine [1,2,3] is a novel processor that offers massively more FLOPS than conventional CPUs, but with a more complicated programming model. Performance comes through utilizing the SPUs, eight four-wide SIMD units with no caches, fancy techniques for ILP like branch prediction, and no direct access to system memory for code or data.

Our work implements a ray tracer for Cell focused on evaluating ray - scene intersection for different kinds of rays.

System Structure



- Only the scene intersector uses the SPUs
- Programmed as eight core chip multiprocessor
- Multithreaded basic packet tracer, one thread per SPU
- PPU distributes work (512 ray mega-packets)
- Shading rays are generated and submitted in subsequent passes.

K-d tree traversal data is stored in LS except for triangle and node data, which use software caches.

- Distinct caches for nodes (128K) and triangles (32K)
- 4-way set associative, round-robin replacement
- 256-byte line (32 nodes or 4 triangles)
- Cache miss triggers synchronous DMA
- Cache hit path is only 15 SPU intrinsics per packet

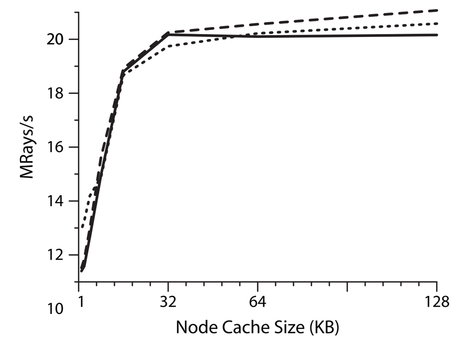
Results

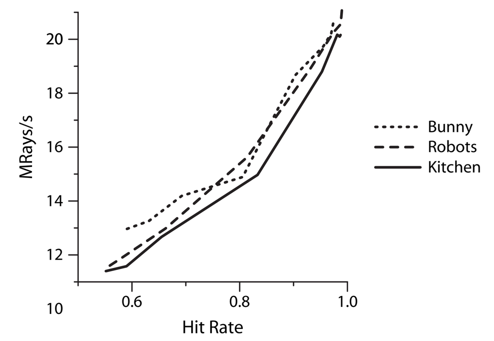
	1 SPU Primary	8 SPU Primary	8 SPU Shadow	8 SPU 1st Bounce	8 SPU 2nd Bounce
Cornell Box	10.83	80.83	48.48	71.76	73.49
Bunny	2.85	20.58	5.41*	5.60*	1.47*
Robots	2.79	21.07	17.38	10.38	6.78
Kitchen	2.70	20.16	17.15	12.17	7.61

- 2.4 GHz Cell blade prototype, 1024x1024 images
- Packet efficiency, not hit rate, turns out to vary between ray types.
- Scales nicely: 7.2x - 7.6x from 1SPU to 8 SPUs.

Cache Analysis

Fixed 32KB triangle cache with varying node cache:





- Nodes cache *really* well:
  - Scenes have 5.3 - 10.4 MB of tree nodes
  - Hit rate over 50% with 1KB cache
  - 96% - 98% with 32KB, 97% - 99% with 128KB
- Expect performance roughly linear with hit rate
- For a given packet, caching only serves as prefetching
- Inter-packet coherence responsible for hit rate
- Cache must be large enough to keep the upper tree nodes resident across packets.

- Triangle caching is a different story
- Most impact is bulk fetching the triangles in a leaf.
- Secondary benefit with a larger cache and inter-packet coherence.
- In practice, high hit rates require large caches
- But, hit rates only influence performance a little
- Our scenes vary less than 5% with 1KB vs. 32KB cache

Cell Development Experience

- Porting SIMD CPU code was a few days' work
  - Naively DMA + stall every tree, triangle fetch
  - No digressions from CMP "create threads and launch"
- Software caching was both simple and effective
  - Straightforward to understand, implement
  - Cheap and amortized over multiple primitives
  - Cached data was read-only
  - Also reduced bus bandwidth, so enabled scaling
- Best CPU ray tracers are already designed to exploit caches for performance
- Cycle-for-cycle our SPU code matches our single-threaded x86 code.
- Easier to stamp out more, faster SPUs than x86es.

Future Work

- Diffuse bounces
- Smaller cache lines and non-power of two sizes
- Methods for building mega-packets of secondary rays
- Combine caching and incremental building
- Efficient models for local shading on SPUs

References

[1] Brian Flachs et al. A streaming processing unit for a CELL processor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, 2005.

[2] Dac Pham et al. The design and implementation of a first-generation CELL processor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, 2005.

[3] Michael Gschwind, Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. A novel SIMD architecture for the CELL heterogeneous chip-multiprocessor. In *Hot Chips 17*, 2005.

[4] Tim Purcell. *Ray Tracing on a Stream Processor*. Ph.D. thesis, Stanford Unviersity, March 2004.

[5] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005.

[6] Gordon Stoll. Optimization techniques. In *Introduction to Real-Time Ray Tracing - SIGGRAPH 2005 Course 38*. 2005.

[7] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. Ph.D thesis, Saarland University, 2004.

[8] Sven Woop, Jorg Schmittler, and Philipp Slusallek. RPU: a programmable ray processing unit for real time ray tracing. *ACM Trans. Graph.* , 24(3):434–444, 2005.