CPU-Style SIMD Ray Traversal on GPUs

Alexander Lier, Marc Stamminger, Kai Selgrad
Computer Graphics Group, University of Erlangen-Nuremberg

August 11, 2018
Same Old, Same Old

- Binary BVH
Same Old, Same Old

- Binary BVH
- Stack based
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
- Test for intersections

CPU-Style SIMD Ray Traversal on GPUs
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
- Test for intersections
- Check distance
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
- Test for intersections
- Check distance
- Push onto stack
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
- Test for intersections
- Check distance
- Push onto stack
- Rinse

CPU-Style SIMD Ray Traversal on GPUs
Same Old, Same Old

- Binary BVH
- Stack based
- Root node on stack
- Test for intersections
- Check distance
- Push onto stack
- Rinse
- Repeat

CPU-Style SIMD Ray Traversal on GPUs
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]

CPU-Style SIMD Ray Traversal on GPUs
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]
  - Kernel code

- **SIMD / CPU**
  - [Wald et al., 2008]
  - Scalar & vectorized code

CPU-Style SIMD Ray Traversal on GPUs
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]
  - Kernel code
  - Many threads
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]
  - Kernel code
  - Many threads

- **SIMD / CPU**
  - [Wald et al., 2008]
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]
  - Kernel code
  - Many threads

- **SIMD / CPU**
  - [Wald et al., 2008]
  - Scalar & vectorized code
Usual Suspects

- **SIMT / GPU**
  - [Aila and Laine, 2009]
  - Kernel code
  - Many threads

- **SIMD / CPU**
  - [Wald et al., 2008]
  - Scalar & vectorized code
  - Few threads
• Schematic SSE
• 4 lanes
- Schematic SSE
- 4 lanes
- Load ray
CPU-Style SIMD Ray Traversal on GPUs

- Schematic SSE
- 4 lanes
- Load ray
- Load nodes
CPU-Style SIMD Ray Traversal on GPUs

- Schematic SSE
- 4 lanes
- Load ray
- Load nodes
- Intersect
SIMD

- Schematic SSE
- 4 lanes
- Load ray
- Load nodes
- Intersect
- Sort hits
CPU-Style SIMD Ray Traversal on GPUs

- Schematic SSE
- 4 lanes
- Load ray
- Load nodes
- Intersect
- Sort hits
- Push candidates
SIMT

- 1 warp / 32 lanes
SIMT

- 1 warp / 32 lanes
- Load rays
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect I
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect I
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect II
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect II
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect II
- Sort hits
SIMT

- 1 warp / 32 lanes
- Load rays
- Load nodes
- Intersect II
- Sort hits
- Push & continue
SIMD vs. SIMT

SIMD
- Ray data

SIMT
- Ray data
- Intersection
- Stack
- Loading / storing
- Sorting
- Intersection
- Stack
- Load / Store
- Sorting
SIMD vs. SIMT

**SIMD**
- Ray data
  - Duplicated

**SIMT**
- Individual
  - Individual

CPU-Style SIMD Ray Traversals on GPUs
SIMD vs. SIMT

SIMD
- Ray data
  - Duplicated
- Intersection

SIMT
- Intersection
  - Individual
- Load / Store
  - Per thread
- Sorting
  - Per thread

CPU-Style SIMD Ray Traversal on GPUs
SIMD vs. SIMT

SIMD
- Ray data
  - Duplicated
- Intersection
  - Vectorized

SIMT
- Ray data
  - Individual
- Intersection
  - Individual
- Stack
  - Exclusive
- Load / Store
  - Per thread
- Sorting
  - Per thread
# SIMD vs. SIMT

<table>
<thead>
<tr>
<th>SIMD</th>
<th></th>
<th>SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray data</td>
<td>Duplicated</td>
<td>Individual</td>
</tr>
<tr>
<td>Intersection</td>
<td>Vectorized</td>
<td>Individual</td>
</tr>
<tr>
<td>Stack</td>
<td>Common</td>
<td>Exclusive</td>
</tr>
</tbody>
</table>
## SIMD vs. SIMT

<table>
<thead>
<tr>
<th>SIMD</th>
<th>SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ray data</strong></td>
<td></td>
</tr>
<tr>
<td>- Duplicated</td>
<td>- Individual</td>
</tr>
<tr>
<td><strong>Intersection</strong></td>
<td></td>
</tr>
<tr>
<td>- Vectorized</td>
<td></td>
</tr>
<tr>
<td><strong>Stack</strong></td>
<td></td>
</tr>
<tr>
<td>- Common</td>
<td>- Exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPU-Style SIMD Ray Traversal on GPUs**
## SIMD vs. SIMT

<table>
<thead>
<tr>
<th>SIMD</th>
<th>SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ray data</td>
<td></td>
</tr>
<tr>
<td>– Duplicated</td>
<td>– Individual</td>
</tr>
<tr>
<td>• Intersection</td>
<td></td>
</tr>
<tr>
<td>– Vectorized</td>
<td>– Individual</td>
</tr>
<tr>
<td>• Stack</td>
<td></td>
</tr>
<tr>
<td>– Common</td>
<td>– Exclusive</td>
</tr>
<tr>
<td>• Loading / storing</td>
<td></td>
</tr>
</tbody>
</table>
# SIMD vs. SIMT

**SIMD**  
- Ray data  
  - Duplicated  
- Intersection  
  - Vectorized  
- Stack  
  - Common  
- Loading / storing  
  - Scalar  

**SIMT**  
- Ray data  
  - Individual  
- Intersection  
  - Individual  
- Stack  
  - Exclusive  
- Loading / storing  
  - Per thread
CPU-Style SIMD Ray Traversal on GPUs

**SIMD vs. SIMT**

**SIMD**
- Ray data
  - Duplicated
- Intersection
  - Vectorized
- Stack
  - Common
- Loading / storing
  - Scalar
- Sorting

**SIMT**
- Ray data
  - Individual
- Intersection
  - Individual
- Stack
  - Exclusive
- Load / Store
  - Per thread
## SIMD vs. SIMT

<table>
<thead>
<tr>
<th>SIMD</th>
<th>SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray data</td>
<td>Individual</td>
</tr>
<tr>
<td>- Duplicated</td>
<td></td>
</tr>
<tr>
<td>Intersection</td>
<td>Individual</td>
</tr>
<tr>
<td>- Vectorized</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Exclusive</td>
</tr>
<tr>
<td>- Common</td>
<td></td>
</tr>
<tr>
<td>Loading / storing</td>
<td>Per thread</td>
</tr>
<tr>
<td>- Scalar</td>
<td></td>
</tr>
<tr>
<td>Sorting</td>
<td>Per thread</td>
</tr>
<tr>
<td>- Scalar</td>
<td></td>
</tr>
</tbody>
</table>
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]

- Process one ray with multiple Threads
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]

- Process one ray with multiple Threads
- One component per thread
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]

- Process one ray with multiple Threads
- One component per thread
- 30 threads used for 10 rays
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]

- Process one ray with multiple Threads
- One component per thread
- 30 threads used for 10 rays
- Applicable for MBVH? SIMD-like?
Inspiration

Ray Tracing with On-The-Fly Subdivision [Binder and Keller, 2015]

- Process one ray with multiple Threads
- One component per thread
- 30 threads used for 10 rays
- Applicable for MBVH? SIMD-like?
  - Team multiple threads
  - Individual intersections
Too Late?

Already considered [Aila and Laine, 2009]
Too Late?

Already considered [Aila and Laine, 2009]

- Did not perform well
Too Late?

Already considered [Aila and Laine, 2009]

- Did not perform well
- Discontinued further analysis
Too Late?

Already considered [Aila and Laine, 2009]
  • Did not perform well
  • Discontinued further analysis
Still promising [Aila and Karras, 2010]
Too Late?

Already considered [Aila and Laine, 2009]
- Did not perform well
- Discontinued further analysis

Still promising [Aila and Karras, 2010]
- No further resources
Too Late?

Already considered [Aila and Laine, 2009]
- Did not perform well
- Discontinued further analysis

Still promising [Aila and Karras, 2010]
- No further resources

Here goes nothing
Concept

- BVH width
Concept

- BVH width → 4
Concept

- BVH width → 4
- Ray data
Concept

- BVH width → 4
- Ray data → duplicate
Concept

- BVH width $\rightarrow$ 4
- Ray data $\rightarrow$ duplicate
- Load
Concept

- BVH width $\rightarrow$ 4
- Ray data $\rightarrow$ duplicate
- Load $\rightarrow$ offset
CPU-Style SIMD Ray Traversal on GPUs

Concept

- BVH width $\rightarrow$ 4
- Ray data $\rightarrow$ duplicate
- Load $\rightarrow$ offset
- Intersection
Concept

- BVH width → 4
- Ray data → duplicate
- Load → offset
- Intersection → vectorized
Concept

- BVH width $\rightarrow 4$
- Ray data $\rightarrow$ duplicate
- Load $\rightarrow$ offset
- Intersection $\rightarrow$ vectorized
- Sort
CPU-Style SIMD Ray Traversal on GPUs

Concept

- BVH width → 4
- Ray data → duplicate
- Load → offset
- Intersection → vectorized
- Sort → shuffle
Concept

- BVH width $\rightarrow$ 4
- Ray data $\rightarrow$ duplicate
- Load $\rightarrow$ offset
- Intersection $\rightarrow$ *vectorized*
- Sort $\rightarrow$ shuffle
- Common stacks
Concept

- BVH width → 4
- Ray data → duplicate
- Load → offset
- Intersection → *vectorized*
- Sort → shuffle
- Common stacks → shared memory
- Kernels [Aila et al., 2012]
• Kernels [Aila et al., 2012]
• Global ray pool
Ray Data

- Kernels [Aila et al., 2012]
- Global ray pool
- Persistent threads
Ray Data

- Kernels [Aila et al., 2012]
- Global ray pool
- Persistent threads
- Redundant load
• Kernels [Aila et al., 2012]
• Global ray pool
• Persistent threads
• Redundant load
• No Dynamic Fetch
Simplified Handling

// g_warpCounter: global ray pool
// num_groups: number of groups per warp
// my_group: thread group index

if (lane == 0)
    ray_idx = atomicAdd(g_warpCounter, num_groups);
ray_idx = __shfl(ray_idx, 0) + my_group;
Node Data

• Load per thread
Node Data

- Load per thread
- Apply offset

CPU-Style SIMD Ray Traversal on GPUs
Node Data

- Load per thread
- Apply offset
- Coalesced access
Coalesced Access

// xy: x, y bounds
// zi: z boundy and child index
// lane_offset: offset within group

int curr_idx = stack[stack_pointer --];
vec4 xy = nodes[curr_idx + lane_offset];
vec4 zi = nodes[curr_idx + 4 + lane_offset];
Intersection

- Duplicated rays
Intersection

- Duplicated rays
- Unique nodes
Intersection

- Duplicated rays
- Unique nodes
- *Vectorized* intersection
Intersection

- Duplicated rays
- Unique nodes
- *Vectorized* intersection
- Count hits
// group_mask: 0x0000000f for first group

bool hit = intersect_box(xy, zi, dist);
unsigned int warp_vote = __ballot(hit);
unsigned int group_hits = __popc(warp_vote & group_mask);
Sort

- Bitonic Sort

3 2 4 1
Sort

- Bitonic Sort
- Shuffle [Demouth, 2013]

![Sorting Example]
Sort

- Bitonic Sort
- Shuffle [Demouth, 2013]
- 3 swaps sufficient
- Bitonic Sort
- Shuffle [Demouth, 2013]
- 3 swaps sufficient
- Exchange node indices
Swap

```c
int child_index = zi.z
float dist = FLT_MAX;
bool hit = intersect_box(xy, zi, dist);
dist = hit ? dist : FLT_MAX;

// bfe: bit field extract
swap(dist, child_index, bfe(lane, 1) ^ bfe(lane, 0));
swap(dist, child_index, bfe(lane, 1));
swap(dist, child_index, bfe(lane, 0));
```

CPU-Style SIMD Ray Traversal on GPUs
Stack

- Shared stack

```
1 2 3 X
```

CPU-Style SIMD Ray Traversal on GPUs
Stack

- Shared stack
- Number of hits known

1 2 3 X ....
Stack

- Shared stack
- Number of hits known
- Insert in stack level
Stack

- Shared stack
- Number of hits known
- Insert in stack level

CPU-Style SIMD Ray Traversal on GPUs
Push

```c
if (group_hits == 0) continue;
stack_pointer += group_hits;

if (dist < FLT_MAX)
    stack[stack_pointer - lane_offset] = child_index;
```
Benchmark

- Extended Aila & Laine Cuda-kernels
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
- 2, 4, and 8-wide BVH
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
- 2, 4, and 8-wide BVH
- GeForce 1070 / Pascal
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
- 2, 4, and 8-wide BVH
- Geforce 1070 / Pascal
- Disabled ray sorting (all)
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
- 2, 4, and 8-wide BVH
- Geforce 1070 / Pascal
- Disabled ray sorting (all)
- No Speculative Traversal (ours)
Benchmark

- Extended Aila & Laine Cuda-kernels
- Compare 4-wide BVH [Guthe, 2014]
- 2, 4, and 8-wide BVH
- Geforce 1070 / Pascal
- Disabled ray sorting (all)
- No Speculative Traversal (ours)
- No Dynamic Fetch (ours)
Average

- MRays per second
- MRays per second
- 100% Aila & Laine

Average
- MRays per second
- 100% Aila & Laine
- Slower primary rays
- MRays per second
- 100% Aila & Laine
- Slower primary rays
- Faster incoherent rays
- MRays per second
- 100% Aila & Laine
- Slower primary rays
- Faster incoherent rays
- Average over 11 scenes
CPU-Style SIMD Ray Traversal on GPUs
Rungholt

CPU-Style SIMD Ray Traversal on GPUs
CPU-Style SIMD Ray Traversal on GPUs
CPU-Style SIMD Ray Traversal on GPUs
Wrap-Up

• 32 registers
Wrap-Up

- 32 registers
- 100% occupancy
Wrap-Up

- 32 registers
- 100% occupancy
- Latency bound
Wrap-Up

- 32 registers
- 100% occupancy
- Latency bound
- Shortage of shared memory
Wrap-Up

- 32 registers
- 100% occupancy
- Latency bound
- Shortage of shared memory
- Slower primary rays
Wrap-Up

- 32 registers
- 100% occupancy
- Latency bound
- Shortage of shared memory
- Slower primary rays
- Faster incoherent rays
void swap(float& dist, int& index, uint mask, uint dir){
    float shfl_dist = __shfl_xor(dist, mask);
    int shfl_index = __shfl_xor(index, mask);
    bool swp = (dist != shfl_dist) && (dist > shfl_dist == dir);
    index = swp ? index : shfl_index;
    dist = swp ? dist : shfl_dist;
}
References

Architecture considerations for tracing incoherent rays.

Understanding the efficiency of ray traversal on gpus.

Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum.

Stackless ray tracing of patches from feature-adaptive subdivision on gpus.

Kepler’s SHUFFLE (SHFL): Tips and Tricks.
GTC ’13.

Latency Considerations of Depth-first GPU Ray Tracing.
In Galin, E. and Wand, M., editors, Eurographics 2014 (Short Papers).

Getting rid of packets – Efficient SIMD single-ray traversal using multi-branching BVHs.