Embree: Ray Tracing Kernels
Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Motivation

• Ray tracing is used heavily for professional graphics
• Implementing a fast ray tracer is difficult

Goal

• Provide the fastest ray tracing kernels to developers
• Address misconceptions about relative performance of CPUs and GPUs for ray tracing
What is Embree?

- The fastest ray tracing kernels for Intel® CPUs
- Designed for Monte Carlo ray tracing
- Easy to integrate into existing applications
- Published under the Apache 2.0 license on ISN: http://software.intel.com/en-us/articles/embree-photo-realistic-ray-tracing-kernels/
Architecture

Professional Graphics Application
CAD, digital content creation, visualization, movie production

Rendering Engine
Distributed ray tracing, path tracing, photon mapping, …

Ray Tracing Kernel
Fast acceleration structure build and traversal

Embree
Status

- Version 1.0 released in November 2011
- Broad adoption by developers and researchers
- 2-5x speed-up over existing implementations
- Version 1.1 released … today!
New Features in Embree 1.1

- 2x lower memory consumption during rendering
- 3x lower memory consumption during BVH build
- Up to 2x faster BVH build
- Improved ray/triangle intersection accuracy
- Support for motion blur
- Support for very large scenes
User Feedback

“I had approached my renderer from the GPU aspect. But once I saw Embree it completely shifted my direction. The CPU with extensions is a more viable platform and thanks to your demonstration / research release of Embree this has totally changed my approach which I am thankful for.”

Gary Herbst
The Embree Example
Renderer
Progressive Path Tracing

1000 x 1200 pixel, rendered on four Intel® Xeon® Processor E7-4860
Model courtesy of Martin Lubich, www.loramel.net
The Embree Kernels
Monte Carlo Ray Tracing
Two Kinds of Ray Distributions

- Coherent Rays
- Incoherent Rays (typical for Monte Carlo)
BVH Acceleration Structure
BVH Acceleration Structure
BVH Acceleration Structure
BVH Acceleration Structure
Solution Space for Vectorized Ray Tracing

Single Ray
- Scalar Traversal
- Packet Traversal

Multi Ray
- Multi Box
  - Single Ray SIMD Traversal
  - Independent Ray Traversal
- Single Box
BVH4 Memory Layout

Traditional BVH Layout

BVH4 Layout
BVH4 Traversal

For each dimension:

Intersect ray with near plane of each box in SIMD

Intersect ray with far plane of each box in SIMD

Clip the near and the far parameters
New Features in Embree 1.1
Memory Consumption During Rendering

Embree 1.0: Precomputed SSE triangles (v0, e1, e2, Ng)

- 55 MB Nodes
- 319 MB Precomputed Triangles

Additional in Embree 1.1: SSE friendly indexed face set

- 55 MB Nodes
- 114 MB Indices
- 40 MB Vertices

Crown (4.8M Triangles) → 44% less data ←
45% less memory for 10% lower performance

<table>
<thead>
<tr>
<th></th>
<th>Precomputed Triangles</th>
<th>Indexed Face Set</th>
<th>Relative Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>44 Mrps</td>
<td>39 Mrps</td>
<td>-11.4 %</td>
</tr>
<tr>
<td></td>
<td>100 Mrps</td>
<td>93 Mrps</td>
<td>-7.0 %</td>
</tr>
<tr>
<td></td>
<td>68 Mrps</td>
<td>61 Mrps</td>
<td>-10.3 %</td>
</tr>
</tbody>
</table>

4x Intel® Xeon® Processor E7-4860
Memory Consumption of BVH Build

- Embree 1.0: *Indices* into Node and Triangle Array
  - Problematic conservative pre-allocations (worst case of 1 out of 4 triangles filled).

- Embree 1.1: *Pointers* to Nodes and Triangles
  - On-demand allocations possible.

→ About 3x lower memory consumption compared to Embree 1.0
Up to 2x Faster BVH Builder

Improvements:

• Optimized allocator for nodes, triangles, and primitive lists.
• Single pass to evaluate heuristic and perform split.
• In-place partition also for parallel splits.
• No pre-allocations at build startup.
• Improved parallelization for spatial split builder.
Ray Triangle Intersection

Möller Trumbore

\[ ABC = \det(dir, v2-v1, v1-v0) \]

\[ A = \det(dir, org-p0, v1-v0) \]

\[ B = \det(dir, org-p0, v2-v1) \]

\[ C = ABC - A - B \]

Improved performance by reducing accuracy of 1 edge test.

Stable Plücker (Additional in Embree 1.1)

\[ ABC = \det(dir, v2-v1, v1-v0) \]

\[ A = \det(dir, org-p0, v1-v0) \]

\[ B = \det(dir, org-p0, v2-v1) \]

\[ C = \det(dir, org-p0, v0-v2) \]

Improved accuracy by performing all 3 tests at high precision.
Performance Impact of Plücker Test

<table>
<thead>
<tr>
<th></th>
<th>Möller Trumbore</th>
<th>Plücker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Performance</td>
<td>-6.8 %</td>
<td>-2.0 %</td>
</tr>
<tr>
<td></td>
<td>44 mrps</td>
<td>100 mrps</td>
</tr>
<tr>
<td></td>
<td>41 mrps</td>
<td>98 mrps</td>
</tr>
</tbody>
</table>

4x Intel® Xeon® Processor E7-4860
Combining Memory and Accuracy Optimizations

Indexed Face Set, Plücker Test

Indexed Face Set, Möller Trumbore

Pre-Gathered Vertices, Plücker Test

Precalculated Möller Trumbore

More Accuracy

Less Memory

More Performance

More Performance
Support for Linear Motion Blur

Linear motions:
- Good approximation for short shutter times.
- Approximated curved motion by piecewise linear motion.

Key Idea:
- Linearly interpolated geometry can be bounded by linearly interpolated bounds.

Algorithm:
- Interpolate bounds at time $t$ during traversal.
- Interpolate triangle vertices at time $t$ during intersection.
## Performance Impact of Motion Blur

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Motion Blur</th>
<th>Relative Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>44 mrps</td>
<td>29 mrps</td>
<td>-34.1 %</td>
</tr>
<tr>
<td></td>
<td>100 mrps</td>
<td>71 mrps</td>
<td>-29.0 %</td>
</tr>
<tr>
<td></td>
<td>68 mrps</td>
<td>25 mrps</td>
<td>-63.2 %</td>
</tr>
</tbody>
</table>

4x Intel® Xeon® Processor E7-4860
Outlook: Embree for Intel® Xeon Phi™
Intel® Xeon® Brand Family

Intel Xeon® Processors E5-1600/2600 Product Family
- High performance computing for mainstream applications
- Accelerating your innovation with exponential performance gains over previous generations

Intel® Xeon® Phi™
- Parallel performance to power breakthrough innovation
- Delivering extremely scalable performance for parallel applications (e.g. simulation, ray tracing and analytics)
Intel® Xeon® Phi™ Architecture

Optimized for highly parallel performance

Groundbreaking differences
• > 50 Smaller, less power consuming cores
• High Memory Bandwidth
• Highly parallel architecture
• Wider vector processing units for greater floating point performance/watt
Embree 2.0 Design Goals

- Two primary goals
  - Goal #1: As easy to use and extend as Embree 1.x
  - Goal #2: High performance on Intel® Xeon Phi™
- Problem: Requires more than just new single-ray kernels for Intel® Xeon Phi™
## Traditional Embree 1.x Architecture

<table>
<thead>
<tr>
<th>User Code</th>
<th>Rendering Engine (C++, scalar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application (C++, scalar)</td>
<td></td>
</tr>
<tr>
<td>E.g. visualization, lighting simulation, …</td>
<td></td>
</tr>
<tr>
<td>E.g. path tracer (Embree path tracer but one example)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Embree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray Tracing Kernels (Intrinsics, SIMD)</td>
</tr>
<tr>
<td>E.g., BVH with single ray traversal</td>
</tr>
</tbody>
</table>
Embree 1.x Issues with wide SIMD

- Intel® Xeon Phi™ : 16-wide SIMD, focus on throughput performance
  → Causes two issues with Embree 1.x Architecture

Problem #1: Harder to use wide SIMD in single-ray kernels
  - E.g.: “16-wide BVH” not 4x as efficient as “4-wide BVH”
  - Instead, prefer working on “packets” of 16 rays in parallel where possible
  - Problem: can’t traverse 16 rays if scalar renderer only produces 1 at a time

Problem #2: “Scalar Renderer” doesn’t make use of vector units
  - Large scalar portion of runtime = diminishing return of wider SIMD (→ Amdahl’s law)
Embree 2.x Approach for wide SIMD

- Solution: Use SPMD compiler to vectorize renderer as well (→ ISPC)

Embree 1.x: Scalar Renderer

- Application (C++, scalar)
  E.g. visualization, lighting simulation, …

- Rendering Engine (C++, scalar)
  E.g. path tracer

- Ray Tracing Kernels (Intrinsics)
  E.g., BVH with single ray traversal

User Code

Low-level RT kernels
Embree 2.x Approach for wide SIMD

- Solution: Use SPMD compiler to vectorize renderer as well (→ ISPC)

Embree 1.x: Scalar Renderer
- Application (C++, scalar)
  E.g. visualization, lighting simulation, …
- Rendering Engine (C++, scalar)
  E.g. path tracer
- Ray Tracing Kernels (Intrinsics)
  E.g., BVH with single ray traversal

Embree 2.0: SPMD Renderer
- Application (C++, scalar)
  E.g. visualization, lighting simulation, …
- User Code
- Ray Tracing Kernels (Intrinsics)
  E.g., hybrid packet/single-ray traversal
- Low-level RT kernels
Embree 2.x Approach for wide SIMD

- Solution: Use SPMD compiler to vectorize renderer as well (→ ISPC)

Embree 1.x: Scalar Renderer

- Application (C++, scalar)
  - E.g. visualization, lighting simulation, …
- Rendering Engine (C++, scalar)
  - E.g. path tracer
- Ray Tracing Kernels (Intrinsics)
  - E.g., BVH with single ray traversal

Embree 2.0: SPMD Renderer

- Application (C++, scalar)
  - E.g. visualization, lighting simulation, …
- Rendering Engine (SPMD → vector!)
  - E.g. path tracer
- Ray Tracing Kernels (Intrinsics)
  - E.g., hybrid packet/single-ray traversal
Embree 2.x Approach for wide SIMD

• Implemented as new Embree “device”
  • Same scalar interface for apps as Embree 1.x

• Use “Intel SPMD Program Compiler (ISPC)” for SPMD renderer *
  • SPMD: User “sees” scalar code (\(\rightarrow\) code as easy to write/maintain as scalar code) ….
  • … but vectorized (one program per SIMD lane) throughout renderer (\(\rightarrow\) performance)

• Use low-level intrinsics kernel for (16-wide!) ray traversal
    * The Intel SPMD Program Compiler, http://ispc.github.com
  • Of course, can also implement one’s own (SPMD-)traversers in ISPC
Embree 2.x Summary

- Fully optional SPMD extension (scalar version on Xeon® still available)
- Uses the right tool for each application layer
- Excellent performance and high programmer productivity
- Code is portable between Xeon® and Intel® Xeon Phi™
- Optional integration of hand-optimized kernels

<table>
<thead>
<tr>
<th>Features</th>
<th>Embree 1.x</th>
<th>Embree 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Core™</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel® Xeon®</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel® Xeon® Phi™</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Demo!