SIGGRAPH2013

Embree Ray Tracing Kernels for the Intel® Xeon® and Intel® Xeon Phi™ Architectures

Sven Woop, Carsten Benthin, Ingo Wald
Intel

Legal Disclaimer and Optimization Notice

Outline

- Embree Overview
- ISPC Overview
- Embree ISPC API
- Implementation Details
- Embree Performance

Embree Overview

Embree Ray Tracing Kernels

Motivation

- Ray tracing is used heavily for professional graphics applications (Movie Industry, Visualization, Digital Content Creation)
- Implementing a fast ray tracer is very difficult
  ➔ Implementations often don't perform optimal on Intel Architecture

Goal

- Provide the fastest ray tracing kernels to application developers

Embree is an open source high fidelity visualization toolkit for application developers who want to create compelling visual applications to deliver an outstanding user experience on current and future computing architectures. Easy to integrate, Embree provides a blueprint for scalable and efficient Ray Tracing capabilities that are demanded by media and entertainment, product design, energy or scientific visualization applications.

Why Ray Tracing?

Physically based:

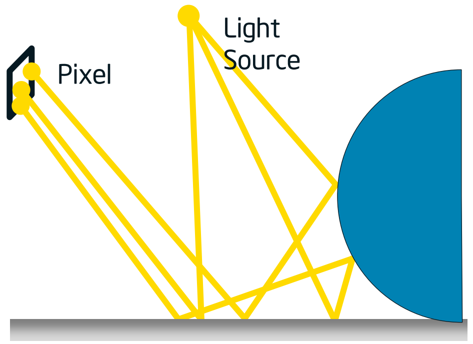
- Imagery can be trusted
- Can generate photorealistic images

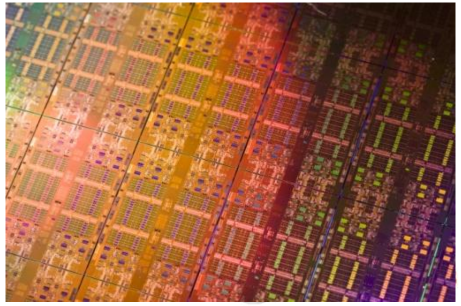
Conceptually simple:

- *Easy* to build renderers with
- Effects combine naturally

Embarrassingly parallel:

- Scales to arbitrary number of compute units
- But also very compute hungry !!!!!





Problem: Writing a Fast Ray Tracer is Hard!

- **Need deep domain knowledge:** many different data structures (kd-trees, octrees, grids, BVH2, BVH4, ..., hybrid structures) and algorithms (single rays, packets, large packets, stream tracing, ...) to choose
- **Need low-level expert knowledge of hardware:** scalability to many cores/CPUs, efficient use of SIMD units, different ISAs (SSE, AVX, Xeon Phi™)
- **Need for multiple implementations:** Different ISAs/CPU types favor different data structures, data layouts, algorithms, and implementations

➔ **Embree is a "middleware" solution!**

What is Embree?

- High fidelity visualization toolkit for application developers
- Easy to integrate, rapid prototyping
- Highly efficient and scalable Ray Tracing features and capabilities
- Compatible with present and future compute platforms
- Available as Open Source on http://embree.github.com (Apache 2.0 license)

Embree 1.x Overview

Support for Intel® Xeon® CPUs:

- Optimized for SSE 3-4.1 (and AVX 1)
- Most professional renders use this platform

"Single ray" Interface to Ray Tracing Kernels:

- Application developers can use scalar C++ code
- Easy to integrate into existing applications

High Performance:

- 1.5x – 6x rendering speedup achievable
- 15M triangles/s high quality spatial index build

Embree 2.0 New Features

- Support for latest Intel® Xeon® Processor family and Intel® Xeon Phi™ coprocessor products
- Support for "Ray Packets" (4, 8, or 16 rays per packet)
- Integration with Intel® SPMD program compiler (ISPC, http://ispc.github.com)
- Two-level Hierarchies and fast BVH builders

Embree 2.0 Released Today!

→ http://embree.github.com

How to use Embree?

- As a benchmark to identify performance issues in your own code
- As a library through the Embree API
- As example code by copying code

Single Ray Tracing not optimal for wide SIMD

Rendering Application

- Shaders leverage 4-wide SSE naturally, e.g. (x,y,z,_) or (r,g,b,a)
- Using wider SIMD units is less efficient and code less readable

Ray Tracing Kernels

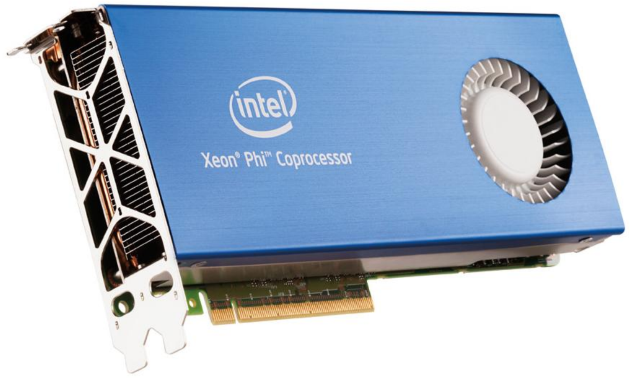
- Single ray kernels work well for 4-wide SSE
- Strongly diminishing return for vector widths wider than 4

How to support Intel® Xeon Phi™ Coprocessor?

Example: Intel® Xeon Phi™ 7120X (→http://ark.intel.com)

- Highly parallel architecture (61 cores, 4 threads per core, 1.238GHz)
- Peak floating point performance >2TFlops SP
- 16 GB of high memory bandwidth (320GB/s)
- 30 MB of L2 cache
- **16-wide SIMD** ISA, 32 SIMD registers
- SIMD + scalar instr can co-issue

→ Great architecture for ray tracing!



Ray Packets and SPMD Programming Model is a Solution

Single Program **M**ultiple **D**ata (SPMD) programming model

- One „program“ per SIMD lane (e.g. one pixel per SIMD lane)
- Masking and sequentialization for diverging control flow
- Code „looks“ like scalar code (e.g. OpenCL)
- Automatic, efficient, and guaranteed vectorization

Embree 2.0 supports SPMD programming model

- Support for “Ray Packets” (4, 8, or 16 rays per packet)
- Integration with Intel® SPMD Program Compiler
  (ISPC, http://ispc.github.com)

ISPC Overview

Intel® SPMD Program Compiler (ISPC)

- Support for **scalar** and **vectorized** control flow and data flow
- Masking done automatically
- Compilation to different vector ISAs (**SSE, AVX, Xeon Phi**™)
- Allows close coupling of C/C++ and ISPC code
  (Data structures shared with C/C++ code)
- Available as Open Source from http://ispc.github.com

ISPC Language

- **C-based syntax** (for, while, if, then else, int, float, …) plus extensions
- Pointers, structs, new, delete, recursion, function pointers, …
- **uniform** and **varying** type qualifiers to express scalars and vectors
- Rich standard library: vectorized transcendentals, atomics, …

Embree ISPC API

Embree ISPC API

- Simple low level Ray Tracing API (build, trace)
- Triangle Meshes and Two Level Scene support
- Support for different acceleration structures
- Support for different traversal algorithms

Embree ISPC Example: Mesh Creation

```
/* create triangle mesh */
uniform RTCGeometry* uniform mesh = rtcNewTriangleMesh (12, 8);

/* fill vertex buffer */
uniform RTCVertex* uniform vertices = rtcMapPositionBuffer(mesh);
vertices[0].x = -1; vertices[0].y = -1; vertices[0].y = -1;
...
rtcUnmapPositionBuffer(mesh);

/* fill triangle index buffer */
uniform RTCTriangle* uniform triangles = rtcMapTriangleBuffer(mesh);
triangles[0].v0  = 0; triangles[0].v1  = 1; triangles[0].v2 = 2;
triangles[0].id0 = 0; triangles[0].id1 = 0;
...
rtcUnmapTriangleBuffer(mesh);

/* launch and wait for build task */
launch rtcBuildAccel (mesh);
sync;

/* query default intersector */
uniform RTCIntersector* uniform intersector = rtcQueryIntersector(mesh);
```

Embree ISPC Example: Rendering

```
/* loop over all screen pixels */
foreach (y=0 ... screenHeight-1, x=0 ... screenWidth-1)
{
  /* create primary ray */
  varying Ray ray;
  ray.org = p;
  ray.dir = normalize(add(mul(x,vx,mul(y,vy),vz));
  ray.tnear = 0;
  ray.tfar = inf;
  ray.id0 = ray.id1 = -1;

  /* trace ray */
  intersector->intersect(intersector,ray);

  /* uv-shading and framebuffer write */
  if (ray.id0 != -1)
    pixels[y*width+x] = make_vec3f(ray.u,ray.v,1.0-ray.u-ray.v);
  else
    pixels[y*width+x] = 0;
}
```

Implementation Details

Algorithms for Intel® Xeon® CPUs

Spatial index structures:

- BVH2, **BVH4 (recommended),** BVH8, BVH4MB (motion blur)

Triangle representations:

- **triangle4 (recommended),** triangle8, triangle4i (less memory), …

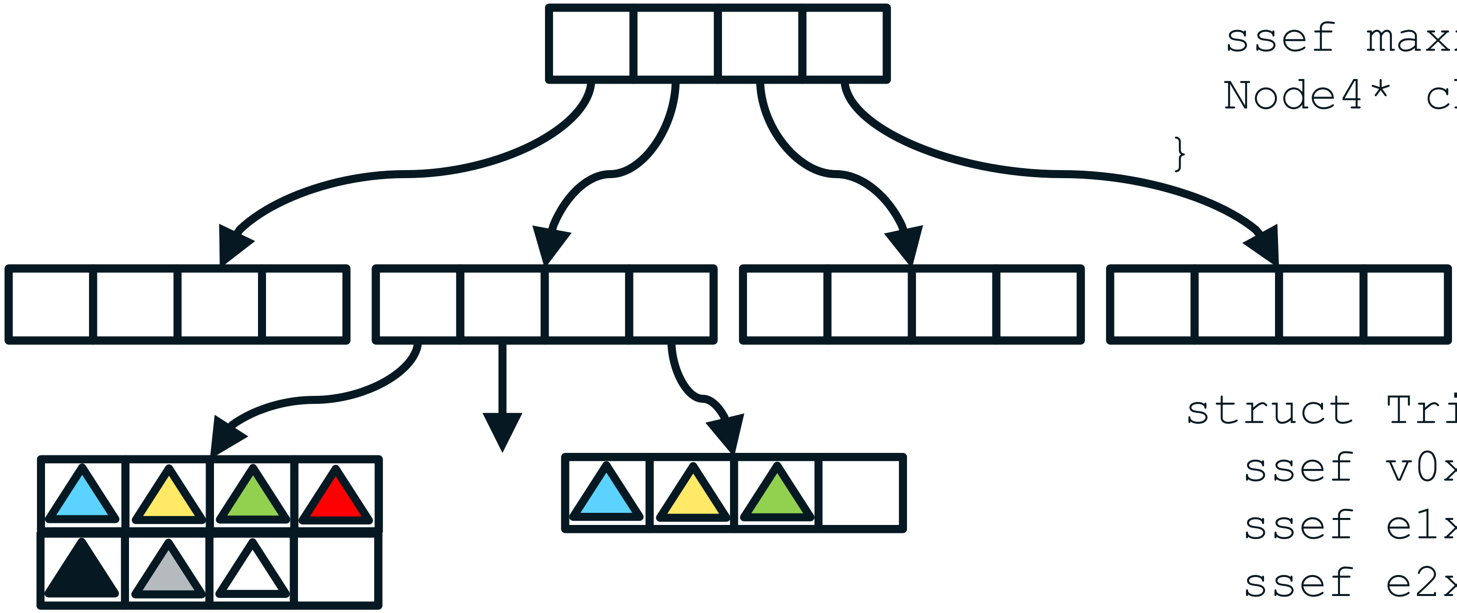
Traversal algorithms:

- **Single ray (recommended for incoherent workloads)**
- SSE packet, AVX packet
- SSE hybrid, AVX hybrid (recommended for coherent workloads)

Ray/triangle intersectors:

- Möller-Trumbore (recommended for performance)
- Plücker variant (recommended for accuracy)

BVH4 Spatial Index Structure



```
struct Node4 {
  ssef minx, miny, minz;
  ssef maxx, maxy, maxz;
  Node4* child[4];
}
```

```
struct Triangle4 {
  ssef v0x,v0y,v0z;
  ssef e1x,e1y,e1z;
  ssef e2x,e2y,e2z;
  ssef Nx,Ny,Nz;
  ssei id0,id1;
}
```

Optimizing BVH4 Traversal for CPUs

- Reduce number of executed instructions
- Reduce data dependencies of critical paths
- Take advantage of special instructions (e.g. SSE, bitscan, etc.)
- Optimize most frequently executed code paths

Optimizing BVH4 Traversal for CPUs (SSE)

- Load front/back plane based on direction sign of the ray.
- Balanced min/max trees
- Bitscans to iterate through hit children
  - Early exit for 0 children hit (20%)
  - 1 child hit (50%): keep next node in register (instead of push/pop sequence)
  - 2 children hit (20%): keep next node in register, sort using a branch

```
while (true) {
  if (isLeaf(node)) goto leaf;
  ssef nearX = (norg.x + node[nearX]) * rdir.x;
  ssef nearY = (norg.y + node[nearY]) * rdir.y;
  ssef nearZ = (norg.z + node[nearZ]) * rdir.z;
  ssef farX  = (norg.x + node[farX ]) * rdir.x;
  ssef farY  = (norg.y + node[farY ]) * rdir.y;
  ssef farZ  = (norg.z + node[farZ ]) * rdir.z;
  ssef near  = max(max(nearX,nearY),
                   max(nearZ,ray.near));
  ssef far   = min(min(farX,farY),
                   min(farZ,ray.far));
  int hitmask = movemask(near <= far);
  if (hitmask == 0) goto pop;
  int c = bitscan(hitmask);
  hitmask = clearbit(hitmask,c);
  if (hitmask == 0) {
    node = node.child[c]; continue;
  } …
```

SIGGRAPH2013

Algorithms for Intel® Xeon Phi™ Coprocessor

Spatial index structure:

- **BVH4AOS**

Traversal algorithms:

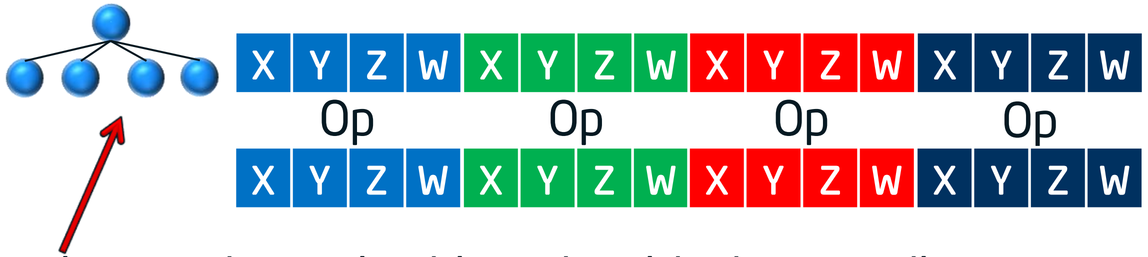
- Single ray traversal
- Packet traversal
- **Hybrid packet/single ray traversal**

Triangle intersector:

- **Möller-Trumbore**

Single Ray Traversal for Intel® Xeon Phi™ Coprocessor

- 4-wide BVH in AoS layout, 2 x 64bit cachelines (4 x box min, 4 x box max)
- Use 16-wide SIMD as 4 x 4-wide 'lanes', ops using AoS layout: 4 x "xyzw"


X Y Z W X Y Z W X Y Z W X Y Z W
Op
Op
Op
Op
X Y Z W X Y Z W X Y Z W X Y Z W

- Horizontal reductions to determine hit node with shortest distance
- Critical path optimization for 0,1, and 2 hit nodes
- Triangle intersection: 1 ray vs. 4 triangles (AoS layout)
- Single ray traversal best for incoherent rays

SIGGRAPH2013

Packet Traversal for Intel® Xeon Phi™ Coprocessor

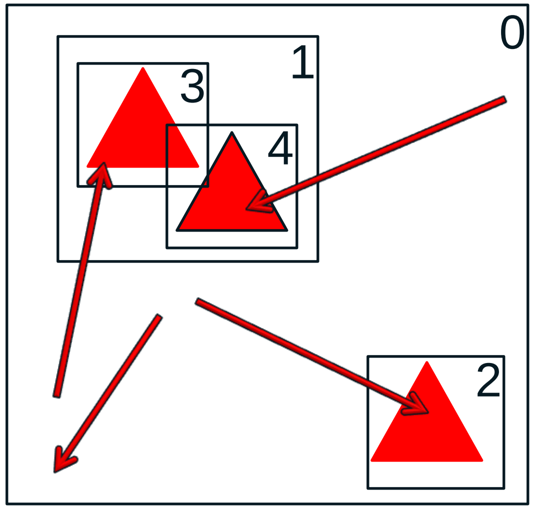
- 16 rays per packet (fits ISA width)
- 4-wide BVH in AoS layout (same layout as for single ray traversal)
- BVH node test: 16 rays vs. 1 box
- Triangle intersection: 16 rays vs. 1 triangle
- Packet traversal best for coherent rays

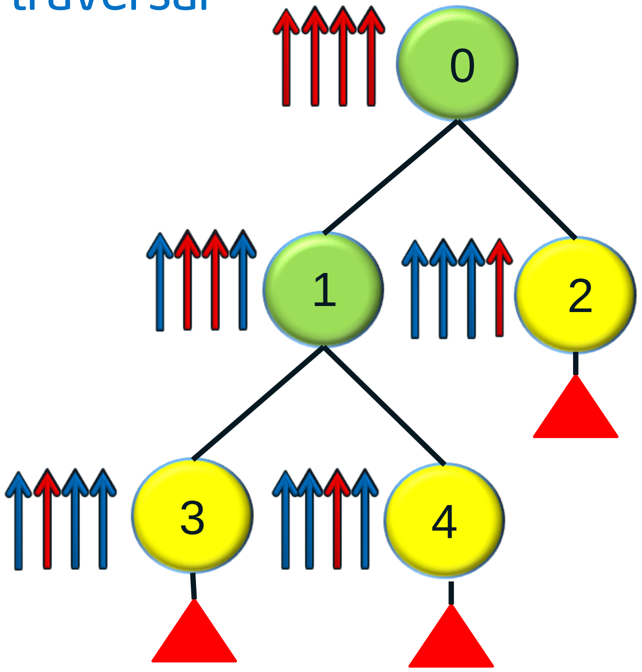
Hybrid Traversal for Intel® Xeon Phi™ Coprocessor

- Real world ray distributions neither fully incoherent nor fully coherent
- Combine packet and single ray traversal into single **hybrid** traversal
- Hybrid traversal
    - Switch between single and packet traversal based on ray coherence
    - Use SIMD utilization as measure for ray coherence (bitcount of mask bits)
    - Low #active rays per packet (incoherent rays) ➔ single ray traversal
    - High #active rays per packet (coherent rays) ➔ packet traversal
    - Switch multiple times per packet (need low switch overhead)

Hybrid Traversal for Intel® Xeon Phi™ Coprocessor

- SIMD util < 2 ➔ single ray traversal





🟢 Packet traversal

🟡 Single ray traversal

Embree Performance

Embree Example Path Tracer

- Flexible modular system design
- Virtual interface to cameras, lights, materials, brdfs, etc.
- Materials build from multiple BRDF components
- Support for HDR environment lighting
- C++ and ISPC implementation of renderer

➔ Entire „Embree 1.x path tracer“ runs on Xeon® and Xeon Phi™

Benchmark Settings

- Intel® Xeon® E5-2690 (8 cores @ 2.9 GHz)
- Intel® Xeon Phi™ 7120 (61 cores @ 1.238 Ghz)
- 1024x1024 image resolution, shading takes 25-40% of total rendering time

	Scene	#triangles
	Imperial Crown of Austria Martin Lubich, www.loramel.net	4.3 M
	Bentley 4.5l Blower (1927)	2.3 M
	Asian Dragon The Stanford 3D Scanning Repository	12.3 M

BVH4 Build Performance on Xeon® and Xeon Phi™

Scene	BVH Build*** [triangles/second]			
	Xeon* SAH (best BVH quality)	Xeon Phi** SAH (best BVH quality)	**Speedup**	
	14.9 M	32.3 M	**2.16x**	
	15.6 M	31.7 M	**2.03x**	
	15.0 M	35.1 M	**2.34x**	

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz
*** including triangle acceleration structure build, excluding memory allocation time

BVH4 Build Performance on Xeon® and Xeon Phi™

Scene	BVH Build*** [triangles/second]			
	Xeon* SAH (best BVH quality)	Xeon Phi** SAH (best BVH quality)	Speedup	Xeon Phi** Morton (reduced BVH qual)
	14.9 M	32.3 M	2.16x	160.1 M
	15.6 M	31.7 M	2.03x	140.4 M
	15.0 M	35.1 M	2.34x	162.1 M

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz
*** including triangle acceleration structure build, excluding memory allocation time

BVH4 Build Performance on Xeon® and Xeon Phi™

Scene	BVH Build*** [triangles/second]			
	Xeon* SAH (best BVH quality)	Xeon Phi** SAH (best BVH quality)	Speedup	Xeon Phi** Morton (reduced BVH qual)
	14.9 M	32.3 M	2.16x	160.1 M
	15.6 M	31.7 M	2.03x	140.4 M
	15.0 M			162.1 M

→ Can rebuild 4.3M crown from scratch **3.5** times per second!

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz
*** including triangle acceleration structure build, excluding memory allocation time

BVH4 Build Performance on Xeon® and Xeon Phi™

Scene	BVH Build*** [triangles/second]			
	Xeon* SAH (best BVH quality)	Xeon Phi** SAH (best BVH quality)	Speedup	Xeon Phi** Morton (reduced BVH qual)
	14.9 M	**32.3 M**	2.16x	160.1 M
	15.6 M	31.7 M	2.03x	140.4 M
	15.0 M	35.1 M		

→ … and over **SEVEN** times per sec on a Xeon Phi!

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz
*** including triangle acceleration structure build, excluding memory allocation time

BVH4 Build Performance on Xeon® and Xeon Phi™

Scene	BVH Build*** [triangles/second]			
	Xeon* SAH (best BVH quality)	Xeon Phi** SAH (best BVH quality)	Speedup	Xeon Phi** Morton (reduced BVH qual)
	14.9 M	32.3 M	2.16x	160.1 M
	15.6 M	31.7 M	2.03x	140.4 M
				162.1 M

→ ... and with reduced quality, at over 35 Hertz

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz
*** including triangle acceleration structure build, excluding memory allocation time

Whitted Style Coherent Rays on Xeon® CPUs*

Scene	Rendering [rays/second]			
	SSE Single Ray	SSE Packets	AVX Packets	**Speedup**
	32.0 M	31.5 M	41.5 M	**1.30x**
	33.5 M	30.0 M	48.0 M	**1.43x**
	33.6 M	31.8 M	43.5 M	**1.29x**

→ Packets and SPMD not much help on SSE …
… but give significant speedup on AVX (wider SIMD)

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz

Path Traced Incoherent Rays on Xeon® CPUs*

Scene	Rendering [rays/second]			
	SSE Single Ray	SSE Packets	AVX Packets	**Speedup**
	18.8 M	15.3 M	17.8 M	**0.94x**
	22.8 M	17.6 M	21.2 M	**0.93x**
	23.1 M	21.3 M	25.5 M	**1.10x**

But: Packets/SPMD won't help much on incoherent rays

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz

Whitted Style Coherent Rays on Xeon Phi™

Scene	Rendering [rays/second]		
	Xeon Phi* **Single Ray**	Xeon Phi* **Hybrid**	**Speedup**
	44.9 M	109.0 M	2.42x
	45.2 M	113.8 M	2.51x
	50.9 M	122.0 M	2.39x

On Xeon Phi (16 wide SIMD) Packets/SPMD even more important than on Xeon with AVX → 2.4-2.5x speedup over single ray for coherent rays

* Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz

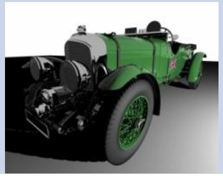
Path Traced Incoherent Rays on Xeon Phi™

Scene	Rendering [rays/second]		
	Xeon Phi* **Single Ray**	Xeon Phi* **Hybrid**	**Speedup**
	34.7 M	62.7 M	**1.80x**
	38.1 M	75.5 M	**1.98x**
	47.5 M	87.5 M	**1.84x**

… and still a ***significant*** 1.8-2x speedup for incoherent rays (→ hybrid)

* Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz

Whitted Style Coherent Rays on Xeon® CPUs and Xeon Phi™

Scene	Rendering [rays/second]		
	Xeon* AVX Packets	Xeon Phi** Hybrid	Speedup
	41.5 M	109.0 M	2.62x
	48.0 M	113.8 M	2.37x
	43.5 M	122.0 M	2.80x

Best on Xeon (AVX, packet) vs best on Xeon Phi (hybrid)
→ Xeon Phi up to 2.8x faster than fastest Xeon for coherent rays

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz

Path Traced Incoherent Rays on Xeon® CPUs and Xeon Phi™

Scene	Rendering [rays/second]		
	Xeon* Single Ray	Xeon Phi** Hybrid	Speedup
	18.8 M	62.7 M	3.33x
	22.8 M	75.5 M	3.31x
	23.1 M	87.5 M	3.78x

Best on Xeon (SSE,single) vs best on Xeon Phi (hybrid)
→ Xeon Phi up to ~3.8x faster than fastest Xeon for incoherent rays

* Intel® Xeon® E5-2690, 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz

Conclusion

- Kernels for both Xeon® (SSE, AVX) and Xeon Phi™
  - With support for both C/C++ and ISPC, fast BVH builds, etc
- Both single-ray **and** packet kernels
  - Single-ray for existing single ray based renderers
  - Packet kernels for packet/SPMD-enabled renderers
  - For Xeon Phi™ : hybrid kernel that is fast for both coherent **and** incoherent rays
- One sample renderer each for both C++ **and** ISPC interfaces
  - Perfect example for how to write a complex renderer in ISPC

➔ **Application developer can freely chose which hardware architecture (Xeon® /Xeon Phi™) and software (C++/SIMD/ISPC) is best for his particular case!**

Questions?

Embree 2.0 available on embree.github.com