Embree Ray Tracing Kernels: *Overview and New Features*

Attila Áfra, Ingo Wald, Carsten Benthin, Sven Woop

Intel Corporation

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.

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

Writing a fast ray tracer is difficult

- Need to multi-thread: easy for rendering but difficult for hierarchy construction
- Need to vectorize: efficient use of SIMD units, different ISAs (SSE, AVX, AVX2, AVX-512)
- 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 to support different CPUs: different ISAs/CPU types favor different data structures, data layouts, and algorithms

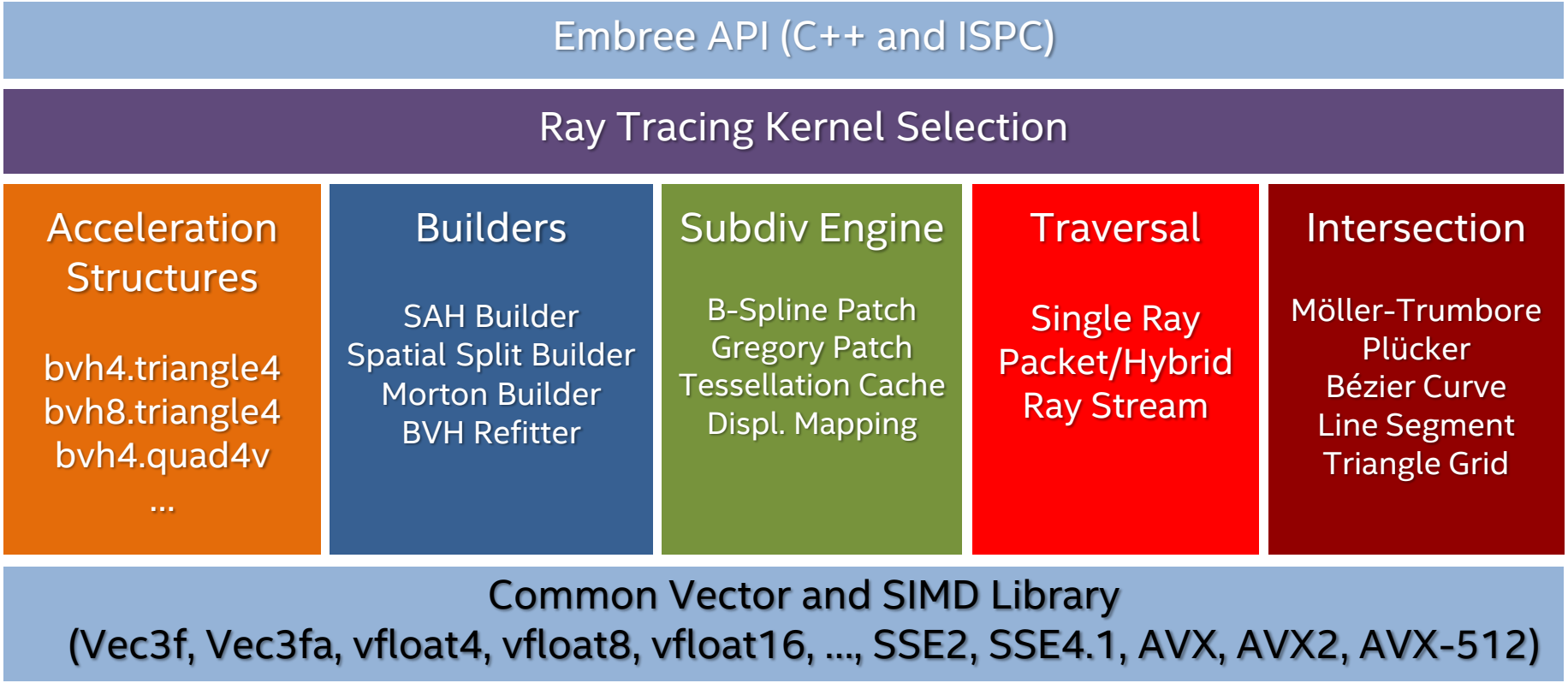
Embree ray tracing kernels

- Provides highly optimized and scalable ray tracing kernels
  - Acceleration structure build and ray traversal
- Targets professional rendering applications
- Highest ray tracing performance on CPUs
  - 1.5–6× speedup reported by users
- Support for latest CPUs
  - Intel® Xeon Phi™ Processor (codenamed *Knights Landing*)
- API for easy integration into applications
- Free and open source under Apache 2.0 license
  - http://embree.github.com

Embree features

- Find closest hit, any hit
  - rtcIntersect, rtcOccluded
- Single rays, ray packets (4, 8, 16), ray streams (N)
- High-quality and high-performance BVH builders
- Intel® SPMD Program Compiler (ISPC) support
- Triangles, quads, subdivs, instances, hair, linear motion blur
- Extensible
  - User defined geometry, intersection filter functions, open source
- Support for Intel® Threading Building Blocks (TBB)

Embree system overview



Embree API

How to use Embree?

Scene

- Scene is a container for set of geometries
- Scene flags passed at creation time
  - Static scene
  - Dynamic scene
  - etc.
- Scene geometry changes have to get commited (rtcCommit), which triggers BVH build

```
// include Embree headers
#include <embree2/rtcore.h>

int main()
{
  // initialize at application startup
  rtcInit();

  // create scene
  RTCScene scene = rtcNewScene
    (RTC_SCENE_STATIC, RTC_INTERSECT1);

  // add geometries
  ... later slide ...

  // commit changes
  rtcCommit(scene);

  // trace rays
  ... later slide ...

  // cleanup at application exit
  rtcExit();
}
```

Triangle mesh

- Contains vertex and index buffers
- Number of triangles and vertices set at creation time
- Linear motion blur supported
  - 2 vertex buffers

```
// application vertex and index layout
struct Vertex { float x, y, z, s, t; };
struct Triangle { int materialID, v0, v1, v2; };

// add mesh to scene
unsigned int geomID = rtcNewTriangleMesh
  (scene, numTriangles, numVertices, 1);

// set data buffers
rtcSetBuffer(scene, geomID, RTC_VERTEX_BUFFER,
  vertexPtr, 0, sizeof(Vertex));
rtcSetBuffer(scene, geomID, RTC_INDEX_BUFFER,
  indexPtr, 4, sizeof(Triangle));

// add more geometries
...

// commit changes
rtcCommit(scene);
```

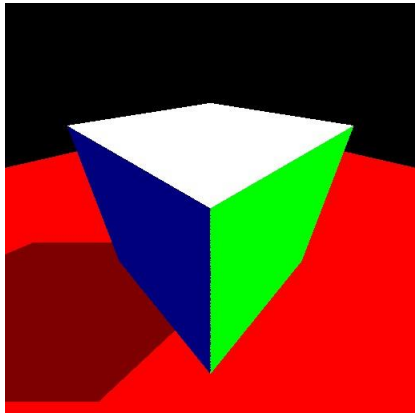
Rendering (ISPC)

```
// loop over all screen pixels
foreach (y=0 ... screenHeight-1, x=0 ... screenWidth-1) {
```

```
    // create and trace primary ray
    RTCRay ray = make_Ray(p, normalize(x*vx + y*vy + vz), eps, inf);
    rtcIntersect(scene, ray);
```

```
    // environment shading
    if (ray.geomID == RTC_INVALID_GEOMETRY_ID) {
      pixels[y*screenWidth+x] = make_Vec3f(0.0f); continue;
    }
```

```
    // calculate hard shadows
    RTCRay shadow = make_Ray(ray.org+ray.tfar*ray.dir, neg(lightDir), eps, inf);
    rtcOccluded(scene, shadow);
```



```
    if (shadow.geomID == RTC_INVALID_GEOMETRY_ID)
      pixels[y*width+x] = colors[ray.primID]*(0.5f + clamp(-dot(lightDir, normalize(ray.Ng)), 0.0f, 1.0f));
    else
      pixels[y*width+x] = colors[ray.primID]*0.5f;
}
```

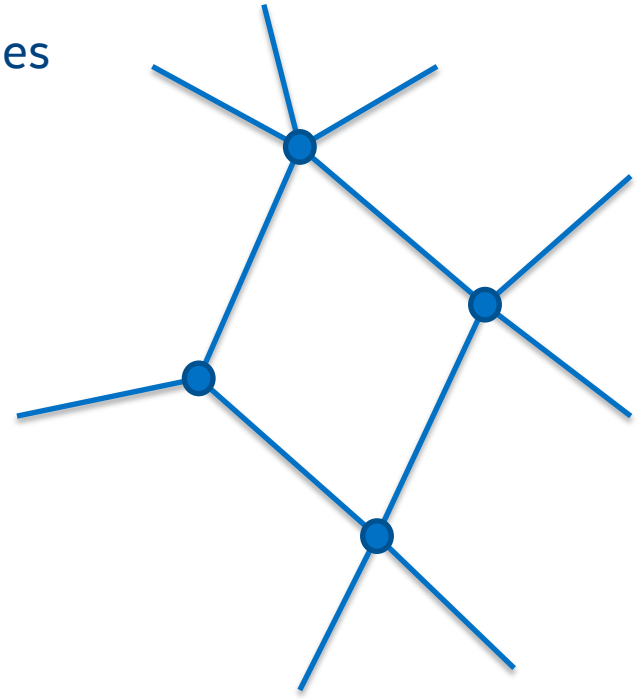
New/Advanced Features

Since the initial publication of Embree [Wald et al. 2014]

Quad meshes (Embree 2.8)

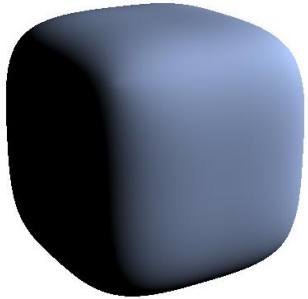
- Most 3D modeling packages produce quad meshes
- No need to convert them to triangles anymore!
- rtcNewQuadMesh

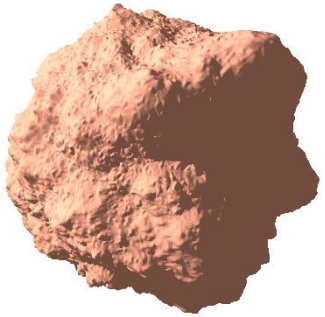
- Up to 2× lower memory usage
- Faster BVH building
- Higher ray intersection throughput



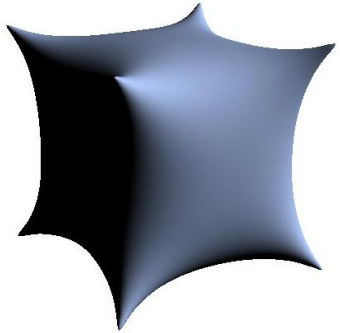
Subdivision surfaces (Embree 2.4)

- Catmull-Clark subdivision surfaces
- Compatible with OpenSubdiv 3.0
- Displacement mapping



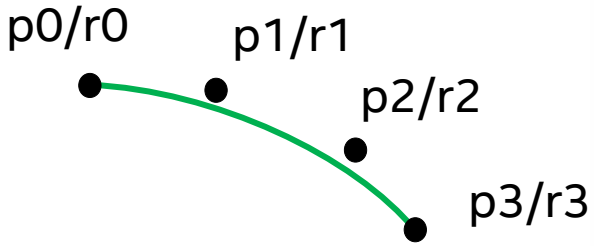


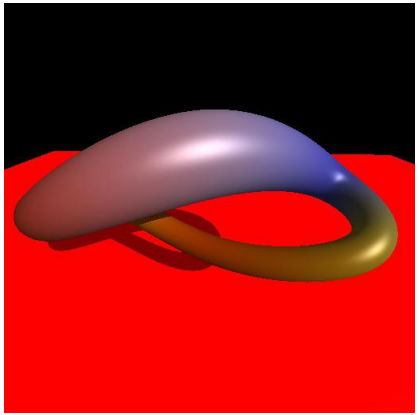
- Tessellation cache [Benthin et al. 2015]
  - Low memory usage
  - Real-time rendering performance



Hair

- Three hair geometry types:
  - Cubic Bézier hair (Embree 2.3)
  - Line segment (Embree 2.8)
  - Cubic Bézier curve (Embree 2.10)
    - Sweep surface of a circle along a Bézier curve

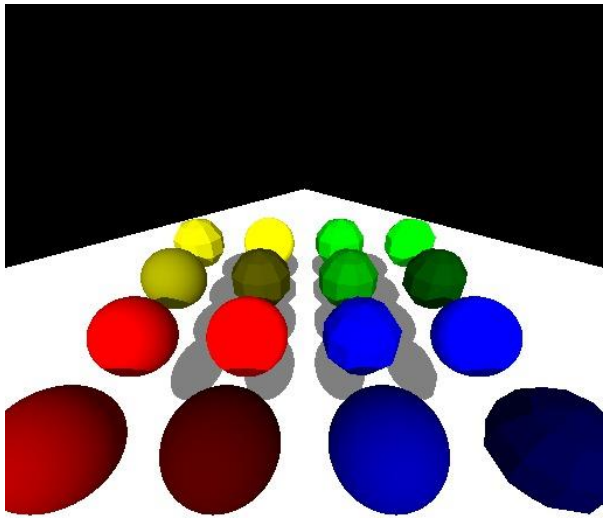




- High performance through use of oriented bounding boxes [Woop et al. 2014]
- Low memory consumption through direct ray/curve intersection

User defined geometries

- Enables implementing custom primitives not provided by Embree
  - e.g., point, disk
- User provides:
  - Bounding function
  - Intersect and occluded functions
- Linear motion blur support (Embree 2.8)



Ray streams (Embree 2.9)

- Intersect many rays together
  - e.g., 1K-4K
- rtcIntersect1M, rtcIntersectNM, rtcIntersectNp
- Enables better coherence extraction than packets
  - Improves both traversal and shading [Áfra et al. 2016] performance
- Novel stream traversal algorithm
  - Based on hiding memory access latency
- Improves performance by 10-30% depending on coherence

Embree 2.10.0 Performance

Ray tracing performance

Test setup

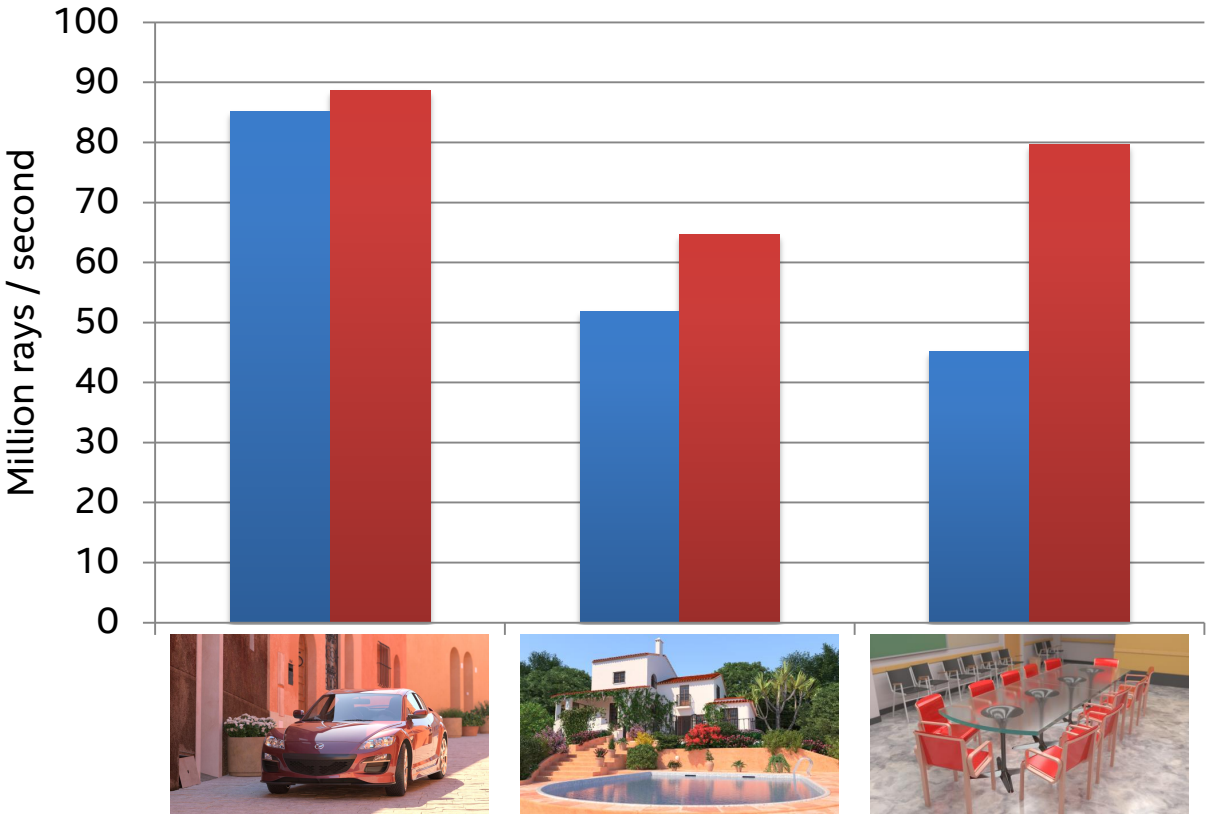
- Path tracer with complex materials and shaders (including procedural)
- Ray stream tracing with local shading coherence extraction [Áfra et al. 2016]
- Hardware:
  - Dual-socket Intel® Xeon® E5-2699 v3 (*Haswell*, 2×18 cores, 2.3 GHz, AVX2), 64 GB DDR4
  - Intel® Xeon Phi™ 7210 (*Knights Landing*, 64 cores, 1.3 GHz, AVX-512), 96 GB DDR4

Mazda / 5.7M triangles

Villa / 37.7M triangles

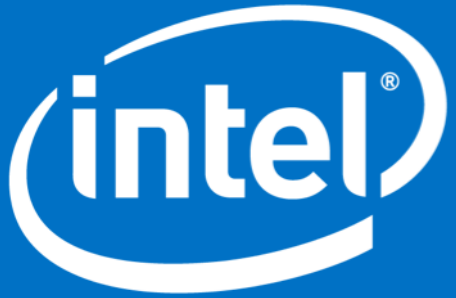
Conference / 0.3M triangles

Ray tracing performance (including shading)



Q&A

Visit the Intel booth for a live Embree demo running on Intel® Xeon Phi™!


intel