Embree Ray Tracing Kernels

Sven Woop
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
Outline

- Embree Overview
- Embree Performance
- Embree API
- Catmull Clark Subdivision Surfaces
Embree Overview
• Movie industry transitioning to ray tracing (better image quality, faster feedback)
• High quality rendering for commercials, prints, etc.
• Provides higher fidelity for virtual design (automotive industry, architectural design, etc.)
• Various kind of simulations (lighting, sound, particles, collision detection, etc.)
• Prebaked lighting in games
• etc.
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, KNCNI)
- **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
Observations

- Ray tracers are often not sufficiently optimized
- Ray traversal consumes a lot of cycles of renderer (often over 70%)
- Ray tracing can be expressed by small number of commonly used operations (build and traversal)

⇒ Ray tracing kernel library has potential to speed up many rendering applications
Embree Ray Tracing Kernels

- Provides highly optimized and scalable Ray Tracing Kernels (data structure build and ray traversal)
- Targets application developers in professional rendering environment
- Highest ray tracing performance on CPUs (1.5x – 6x speedup reported by users)
- Support for latest CPUs (e.g. AVX512 support)
- API for easy integration into applications
- Free and Open Source under Apache 2.0 license (http://embree.github.com)
Embree Features

- Find closest and any hit kernel (rtcIntersect, rtcOccluded)
- Single Rays and Ray Packets (4, 8, 16)
- High quality and high performance hierarchy builders
- Intel® SPMD Program Compiler (ISPC) supported
- Triangles, Instances, Hair, Linear Motion blur
- Extensible (User Defined Geometry, Intersection filter functions, Open Source)
- Support for Intel Threading Building Blocks (TBB)
New Embree Features

- Catmull Clark Subdivision Surfaces
  - Smooth surface primitive
- Vector Displacement Mapping
  - Add geometric detail
- Interpolation
- Initial AVX512 support
  - 16 wide AVX512 traversal kernels
  - Full AVX512 optimizations will come when hardware available!

31/08/15
### Embree System Overview

#### Embree API (C++ and ISPC)

#### Ray Tracing Kernel Selection

<table>
<thead>
<tr>
<th>Accel. structure</th>
<th>Builders</th>
<th>Subdiv Engine</th>
<th>Traversal</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>bvh4.triangle4,</td>
<td>SAH builder</td>
<td>B-Spline Patch</td>
<td>Single ray (SSE2, AVX, AVX2), packet (SSE2), hybrid (SSE4.2), ...</td>
<td>MöllerTrumbore, Plücker Variant, Bezier Curve, Triangle Grids</td>
</tr>
<tr>
<td>bvh8.triangle8,</td>
<td>Spatial split builder</td>
<td>Gregory Patch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bvh4aos.triangle1,</td>
<td>Morton code builder</td>
<td>TessellationCache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bvh4.grid</td>
<td>build</td>
<td>Displ. Mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>ers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Common Vector and SIMD Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Vec3f, Vec3fa, float4, float8, float16, SSE2, SSE4.1, AVX, AVX2, AVX512)</td>
</tr>
</tbody>
</table>
Why Ray Tracing on CPUs?

- High ray tracing performance for photorealistic rendering
- Large memory capacity to render really complex models
- Runs on any CPU through well defined ISA
- No special hardware requirements
- Robust tools to develop and debug rendering application
- Large shading and rendering applications are executed efficiently
Why should I use Embree?

- Hides complexity of writing high performance ray tracing kernels
  - gives you more time for innovation of your renderer
- High performance on latest Intel® Xeon® Processor family and Intel® Xeon Phi™ coprocessor products
- Embree always up to date with latest ISA instruction sets
- High potential performance gain
  (1.5x – 6x rendering speedup reported by Embree users)
How can I use Embree?

- As a benchmark to identify performance issues in existing applications
- Adopt algorithms from Embree to your code
  - However Embree internals change frequently!
- As a library through the Embree API (recommended)
  - Benefit from future Embree improvements!
Embree v2.6.1 Performance
Performance Methodology

- Models and illumination effects representative for professional rendering environment
- Path tracer with different material types, different light types, about 2000 lines of code
- Evaluation on typical Intel® Xeon® rendering workstation* and Intel® Xeon Phi™ Coprocessor**
- Compare against state of the art GPU*** methods (using OptiX™ 3.8.0 and CUDA® 7.0.28)
- Identical implementations in ISPC (Xeon®), ISPC (Xeon Phi™), OptiX™ (GTX™ Titan X)

* Dual Socket Intel® Xeon® E5-2699 v3 2x18 cores @ 2.30GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz *** NVIDIA® GeForce® GTX™ Titan X
Build Performance for Static Scenes

SAH Build (high quality)

- Intel® Xeon® E5-2699 v3 Processor
  2 x 18 cores, 2.3 GHz

- Intel® Xeon Phi™ 7120 Coprocessor
  61 cores, 1.28 GHz

<table>
<thead>
<tr>
<th>Million Triangles/Second</th>
<th>Intel® Xeon® E5-2699 v3</th>
<th>Intel® Xeon Phi™ 7120</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40 (32.3)</td>
<td>41 (31.7)</td>
</tr>
<tr>
<td>50</td>
<td>45 (35.1)</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Build Performance for Dynamic Scenes

Million Triangles/Second

<table>
<thead>
<tr>
<th></th>
<th>Intel® Xeon® E5-2699 v3 Processor</th>
<th>Intel® Xeon Phi™ 7120 Coprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>2 x 18 cores, 2.3 GHz</td>
<td>61 cores, 1.28 GHz</td>
</tr>
<tr>
<td>Performance</td>
<td>160.1 Million Triangles/Second</td>
<td>140.4 Million Triangles/Second</td>
</tr>
<tr>
<td></td>
<td>162.1 Million Triangles/Second</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Ray Tracing Performance (incl. Shading)

<table>
<thead>
<tr>
<th>Million Rays/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon® E5-2699 v3 Processor</td>
</tr>
<tr>
<td>2 x 18 cores, 2.3 GHz</td>
</tr>
<tr>
<td>107.2</td>
</tr>
<tr>
<td>64.96</td>
</tr>
<tr>
<td>29.472</td>
</tr>
<tr>
<td>Intel® Xeon Phi™ 7120 Coprocessor</td>
</tr>
<tr>
<td>61 cores, 1.28 GHz</td>
</tr>
<tr>
<td>129.6</td>
</tr>
<tr>
<td>75.36</td>
</tr>
<tr>
<td>35.04</td>
</tr>
<tr>
<td>NVIDIA® GeForce® GTX™ Titan X Coprocessor</td>
</tr>
<tr>
<td>12 GB RAM</td>
</tr>
<tr>
<td>134.98</td>
</tr>
<tr>
<td>82.62</td>
</tr>
<tr>
<td>38.76</td>
</tr>
</tbody>
</table>

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.*
Embree API
Scene Object

- Scene is container for set of geometries
- Scene flags passed at creation time
- Scene geometry changes have to get committed \((\texttt{rtcCommit})\) which triggers BVH build

```c
/* 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 ();
}
```
Scene Types

✝ Static Scenes
  – Geometry cannot get changed
  – High quality BVH build (SAH) ➔ faster ray traversal
  – For final frame rendering

✝ Dynamic Scenes
  – Geometries can get added, modified, and removed
  – Faster build (Morton) ➔ slower ray traversal
  – Preview mode during geometric modeling
Triangle Mesh

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

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

/* fill data buffers */
... later slide ...

/* add more geometries */
...

/* commit changes */
rtcCommit (scene);
```
 Buffer Sharing

- Recommended to use buffer sharing
- Reduces memory consumption
- Application manages buffers (buffer has to stay alive as long as geometry is alive)
- Support for stride and offset allows application flexibility in its data layout
/* 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);

/* share buffers with application */
rtcSetBuffer (scene, geomID, RTC_VERTEX_BUFFER, vertexPtr, 0, sizeof(Vertex));
rtcSetBuffer (scene, geomID, RTC_INDEX_BUFFER, indexPtr, 4, sizeof(Triangle));
Tracing Rays

- `rtcIntersect (scene, ray)` reports first intersection
- `rtcOccluded (scene, ray)` reports any intersection
- Packet versions for ray packets of size 4, 8, and 16
rtcIntersect: Ray Structure Inputs

- Ray origin and direction (org, dir)
- Ray interval (tnear, tfar)
- Time used for motion blur [0,1]

```c
struct RTCRay {
    Vec3f org;
    Vec3f dir;
    float tnear;
    float tfar;
    float time;

    Vec3f Ng;
    float u;
    float v;
    int geomID;
    int primID;
    int instID;
}
```
 rtcIntersect: Ray structure Outputs

- Hit distance (tfar)
- Unnormalized geometry normal (Ng)
- Local hit coordinates (u,v)
- Geometry identifier of hit geometry (geomID)
- Index of hit primitive of geometry (primID)
- Geometry identifier of hit instance (instID)

No shading normals, texture coordinates, etc.

```
struct RTCRay
{
    Vec3f org;
    Vec3f dir;
    float tnear;
    float tfar;
    float time;

    Vec3f Ng;
    float u;
    float v;
    int geomID;
    int primID;
    int instID;
}
```
Intel® SPMD Program Compiler (ISPC)

- Simplifies writing vectorized renderer
- C-based language plus vector extensions
- Scalar looking code that gets vectorized automatically
- Guaranteed vectorization
- Compilation to different vector ISAs (SSE, AVX, AVX2, AVX512, Xeon Phi™)
- Available as Open Source from [http://ispc.github.com](http://ispc.github.com)
/* 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;
    }
}
Intersection Filter Functions

- Per geometry callback that is called during traversal for each primitive intersection
- Callback can **accept** or **reject** hit
- Can be used for:
  - Trimming curves (e.g. modeling tree leaves)
  - Transparent shadows (reject and accumulate)
  - Find all hits (reject and collect)
/* procedural intersection filter function */
void intersectionFilter(void* userPtr, RTCRay& ray) {
    Vec3fa h = ray.org + ray.dir*ray.tfar;
    float v = abs(sin(4.0f*h.x)*cos(4.0f*h.y)*sin(4.0f*h.z));
    float T = clamp((v-0.1f)*3.0f,0.0f,1.0f);
    if (T > 1.0f) return; // accept hit
    ray.geomID = RTC_INVALID_GEOMETRY_ID; // reject hit
}

/* set intersection filter for the cube */
rtcSetIntersectionFilterFunction(scene, geomID, (RTCFilterFunc)&intersectionFilter);
rtcSetOcclusionFilterFunction (scene, geomID, (RTCFilterFunc)&intersectionFilter);
rtcSetUserData (scene, geomID, NULL);
Hair Geometry

- Hair curves represented as cubic bezier curves with varying radius
- High performance through use of oriented bounding boxes
- Low memory consumption through direct ray/curve intersection
Catmull Clark Subdivision Surfaces
Catmull Clark Subdivision Surfaces

- Converts coarse mesh into smooth surface by subdivision
- Generalization of bi-cubic B-Spline surfaces to arbitrary topology
- Embree is compatible with OpenSubdiv 3.0
Catmull Clark Subdivision
CC Subdivision Surface Advantages

- Low resolution base mesh controls high resolution surface
- Smoothness always guaranteed (C2 continous almost everywhere)
- Support for arbitrary topology (no trimming required as with NURBS)
- Creases allow introducing sharp features
- Support in most modeling tools
- Established as standard in movie production
Semi-sharp edge creases
Semi-sharp vertex creases
Vertex attribute interpolation
Tessellation level per edge
Non-manifolds and holes
Boundary modes
Triangles, Quads, Pentagons, ...
Vector Displacement mapping
Embree Subdivision Example

unsigned geomID = rtcNewSubdivisionMesh (scene, RTC_GEOMETRY_STATIC,
   numFaces, numIndices, numVertices,
   numEdgeCreases, numVertexCreases, numHoles);

rtcSetBuffer (scene,geomID,RTC_VERTEX_BUFFER, vertices, 0, sizeof(float3));
rtcSetBuffer (scene,geomID,RTC_INDEX_BUFFER , indices, 0, sizeof(int));
rtcSetBuffer (scene,geomID,RTC_FACE_BUFFER , faces, 0, sizeof(int));
rtcSetBuffer (scene,geomID,RTC_LEVEL_BUFFER , levels, 0, sizeof(float));

rtcSetBuffer (scene,geomID,RTC_EDGE_CREASE_INDEX_BUFFER,...);
rtcSetBuffer (scene,geomID,RTC_EDGE_CREASE_WEIGHT_BUFFER,...);

rtcSetBuffer (scene,geomID,RTC_VERTEX_CREASE_INDEX_BUFFER,...);
rtcSetBuffer (scene,geomID,RTC_VERTEX_CREASE_WEIGHT_BUFFER,...);

rtcSetBuffer (scene,geomID,RTC_HOLE_BUFFER,holes,0,sizeof(char));
Embree Subdivision Implementation

- Tessellate and Cache
  - limited memory consumption
  - trade memory for performance
- Parallel Shared Tessellation Cache
- Grid evaluation through feature adaptive subdivision into B-Spline patches and Gregory patches

Feature adaptive subdivision into B-Spline patches (green) and Gregory Patches (blue)
## Embree Subdivision Performance

<table>
<thead>
<tr>
<th></th>
<th>Patches</th>
<th>Edge Creases</th>
<th>Micro Quads</th>
<th>Same View</th>
<th>Walkthrough</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>0</td>
<td>1048k</td>
<td>105 fps</td>
<td>40 fps</td>
</tr>
<tr>
<td></td>
<td>52k</td>
<td>0</td>
<td>831k</td>
<td>84 fps</td>
<td>72 fps</td>
</tr>
<tr>
<td></td>
<td>53k</td>
<td>30k</td>
<td>837k</td>
<td>100 fps</td>
<td>80 fps</td>
</tr>
</tbody>
</table>

**Intel® Xeon® E5-2690**
2.9 GHz
2x 8 cores
1024 x 1024 pixels
Vertex Data Interpolation

- Interpolates arbitrary user data over geometries (non-trivial for subdivision geometries)
- Interpolated data \( P \) as well as \( dPdu \) and \( dPdv \) can be calculated at arbitrary location
- Enables smooth normals and anisotropic texture lookups
- Different rules for interpolation of texture coordinates supported (by evaluation of second subdiv mesh)
Vertex Data Interpolation Example

```c
rtcNewScene (RTC_STATIC, RTC_INTERSECT1 | RTC_INTERPOLATE);
...
unsigned geomID = rtcNewSubdivisionMesh (...);
rtcSetBuffer (scene,geomID,RTC_INDEX_BUFFER, indices, 0, sizeof(int));
rtcSetBuffer (scene,geomID,RTC_VERTEX_BUFFER, vertices, 0, sizeof(float3));
rtcSetBuffer (scene,geomID,RTC_USER_VERTEX_BUFFER, vertex_colors, 0, sizeof(float3));
...
rtcCommit (scene);
...
rtcIntersect (scene, ray);
...
float3 P, dPdu, dPdv;
rtcInterpolate (scene, ray.geomID, ray.primID, ray.u,ray.v, RTC_VERTEX_BUFFER, &P, &dPdu, &dPdv, 3);

float3 color;
rtcInterpolate (scene, ray.geomID, ray.primID, ray.u,ray.v, RTC_USER_VERTEX_BUFFER, &color, 0,0, 3);
```
Displaced Subdivision Surface

- Support for vector displacement
- Tessellation approach enables displacements
- Callback function displaces vertex positions
- Smooth normals possible through approximation

\[ Q = P + D*Ng \]
\[ \frac{dQ}{d\xi} \approx \frac{dP}{d\xi} + \frac{dD}{d\xi} * Ng \]
\[ \frac{dQ}{d\eta} \approx \frac{dP}{d\eta} + \frac{dD}{d\eta} * Ng \]
void displacementFunction(
  void* ptr, int geomID, int primID,
  const float* u, const float* v,
  const float* nx, const float* ny, const float* nz,
  float* px, float* py, float* pz,
  size_t N)
{
  for (size_t i = 0; i<N; i++) {
    float D = displacement(...);
    px[i] += D*nx[i];
    py[i] += D*ny[i];
    pz[i] += D*nz[i];
  }
}

BBox3fa bounds(...);
rtcSetDisplacementFunction (scene, geomID, displacementFunction, &bounds);
Summary

- Embree delivers highest ray tracing performance on CPUs
- Embree is easy to use through its API
- Subdivision surface support compatible to OpenSubdiv 3.0
- Free and Open Source (https://embree.github.com)
Questions?

https://embree.github.io
embree@googlegroups.com