Embree Ray Tracing Kernels

Tutorial and Application at Dreamworks Animation and in Autodesk Maya

Talks

- Building Ray Tracing Applications with Embree (40min)
  Sven Woop (Intel)
- Interactive Light Transport Simulation at DreamWorks Animation (35min)
  Louis Feng (Intel), Evan Smyth (DreamWorks Animation)
- Interactive Embree-Based Ray Tracing in Autodesk Maya (40min)
  Charles Congdon (Intel)

Building Ray Tracing Applications with Embree

Sven Woop

Legal

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Optimized Intel® HD Graphics P3000 only available on select models of the Intel® Xeon® processor E3 family. To learn more about Intel Xeon processors for workstation visit www.intel.com/go/workstation.

HD Graphics P4000 introduces four additional execution units, going from 8 in the HD P3000 to 12 in the HD P4000. Optimized Intel® HD Graphics P4000 only available on select models of the Intel® Xeon® processor E3-1200 v2 product family. For more information, visithttp://www.intel.com/content/www/us/en/architecture-and-technology/hdgraphics/hdgraphics-developer.html

Iris™ graphics is available on select systems. Consult your system manufacturer.

Any code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.

Performance claims: 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. For more information go to : http://www.Intel.com/performance

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

Outline

- Embree Overview
- Embree Performance
- Embree API Tutorial

After this talk you …

… will be able to write high performance CPU ray tracing applications using Embree!

Embree Overview

Usage of Ray Tracing Today

- Movie industry transitioning to ray tracing (better image quality, faster feedback)
- High quality rendering for commercials, prints, etc.
- Virtual design in automotive industry, architectural design, …
- 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, Xeon Phi™)
- **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 (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 applications

Embree

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

Embree Features

- Find closest hit kernel (**rtcIntersect**)
- Find any hit kernel (**rtcOccluded**)
- Single Ray, Ray Packets (4, 8, 16)
- High quality and high performance hierarchy builders
- Intel® SPMD Program Compiler (ISPC) supported
- Triangles, Instances, Hair
- Extensible (User Defined Geometry, Intersection filter functions, Open Source)

Embree System Overview

Embree API (C++ and ISPC)

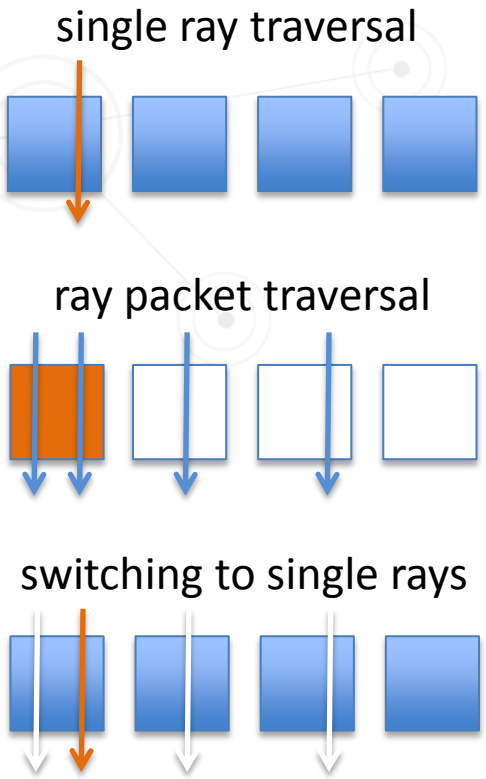
Ray Tracing Kernel Selection

Accel. structure	Builders	Traversal	Intersection
bvh4.triangle4, bvh4.triangle8, bvh8.triangle8, bvh4aos.triangle1, …	SAH builder Spatial split builder Morton code builder BVH Refitter	Single ray (SSE2), single ray (SSE4.1), single ray (AVX), single ray (AVX2), packet (SSE2), hybrid (SSE4.2), ...	Möller Trumbore, Plücker Variant, Bezier Curve

Common Vector and SIMD Library
(Vec3f, Vec3fa, ssef, avxf, SSE2, SSE4.1, AVX, AVX2)

Vectorized Ray Traversal in Embree

- Bounding Volume Hierarchy with fanout of 4 (BVH4) for fast single ray traversal
- Packets of rays for 2x faster coherent ray traversal
- Hybrid packet/single ray traversal also fast for incoherent rays



Why Ray Tracing on CPUs?

- High ray tracing performance for photorealistic rendering
- Large memory capacity to render really complex models
- Robust tools to develop and debug complex applications
- Complex shading and rendering applications are executed efficiently (e.g. light cuts with large per pixel state)

Why should I use Embree?

- Hides complexity of writing high performance ray tracing kernels, and gives you more time developing your renderer
- High performance on latest Intel® Xeon® Processor family and Intel® Xeon Phi™ coprocessor products
- 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 into your code
  - However Embree internals change frequently!
- As a library through the Embree API (recommended)
  - Benefit from future Embree improvements!

Supported Software and Hardware

- Windows XP, Windows 7, Windows 8, Mac OS X 10.X, Linux
- 64 bit (recommended) and 32 bit
- SSE2, …, AVX, AVX2, Xeon Phi™ Instructions
- Visual Studio 2010 - 2013
- ICC (recommended), GCC, CLANG, MSVC

Embree Performance

Performance Methology

- Models and illumination effects representative for professional rendering environment
- Evaluation on typical Intel® Xeon® rendering workstation* and Intel® Xeon Phi™ Coprocessor**
- Compare against state of the art GPU*** methods (using OptiX™ 3.5.1 and CUDA® 5.5)
- Path tracer with different material types, different light types, about 2000 lines of code
- Identical implementations in C++ (Xeon®), ISPC (Xeon Phi™), OptiX™ (GTX™ Titan)

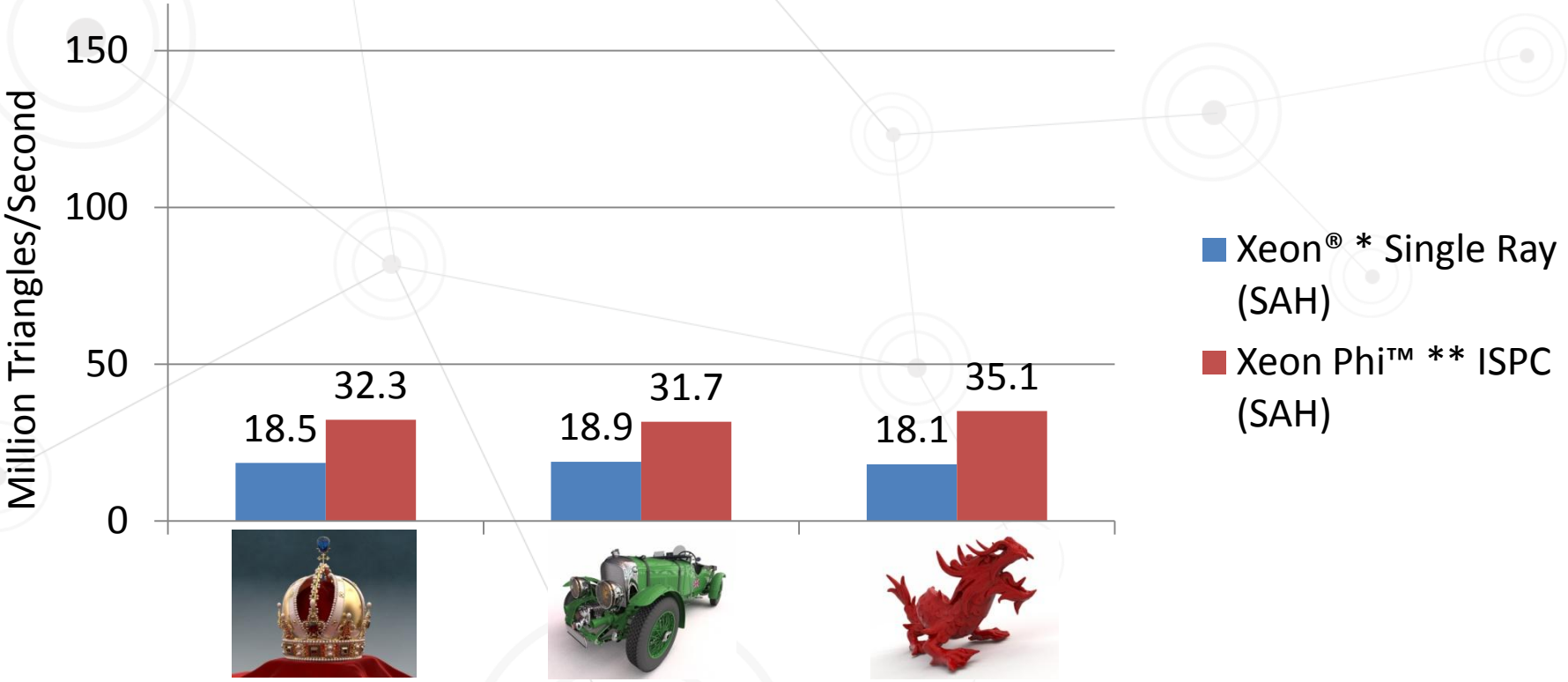
Imperial Crown of Austria
4.3M triangles

Bentley 4.5l Blower (1927)
2.3M triangles

Asian Dragon
12.3M triangles

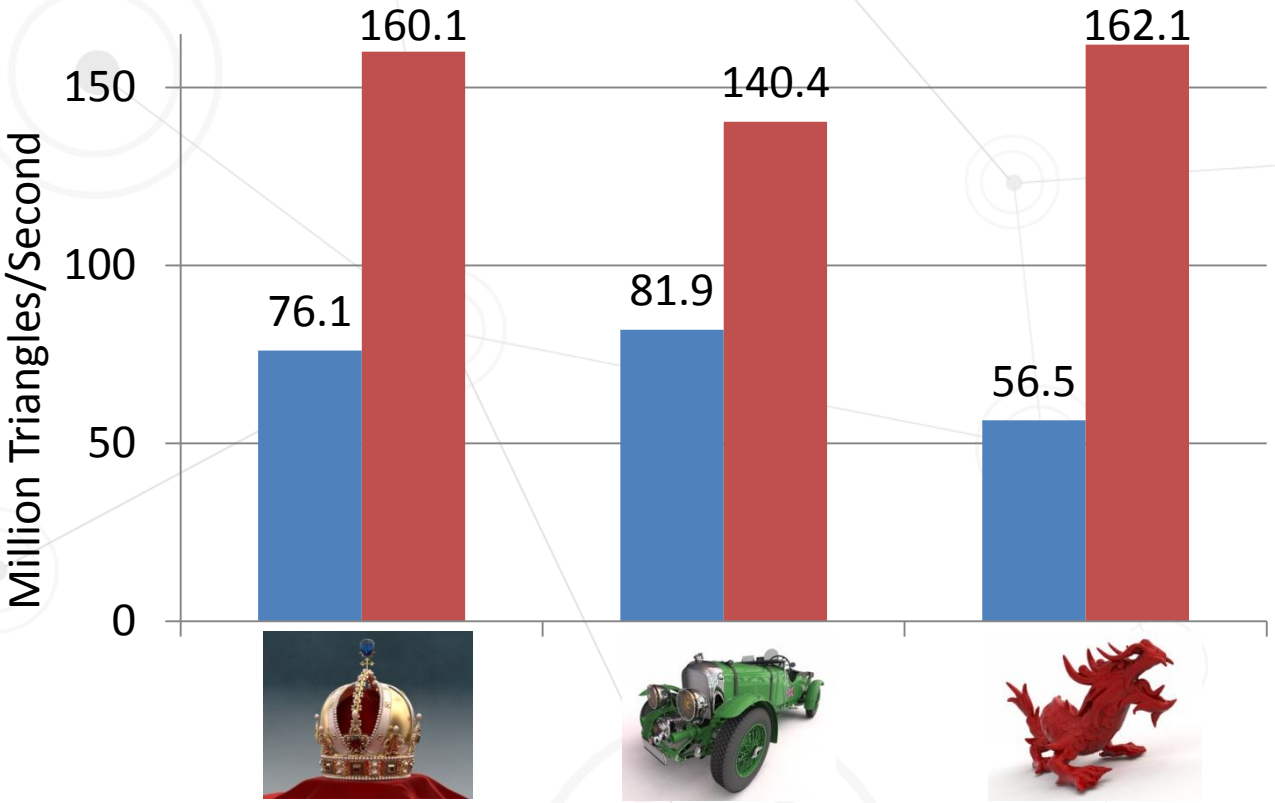
* Dual Socket Intel® Xeon® E5-2690, 2x 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz *** NVIDIA® GeForce® GTX™ Titan

Build Performance for Static Scenes



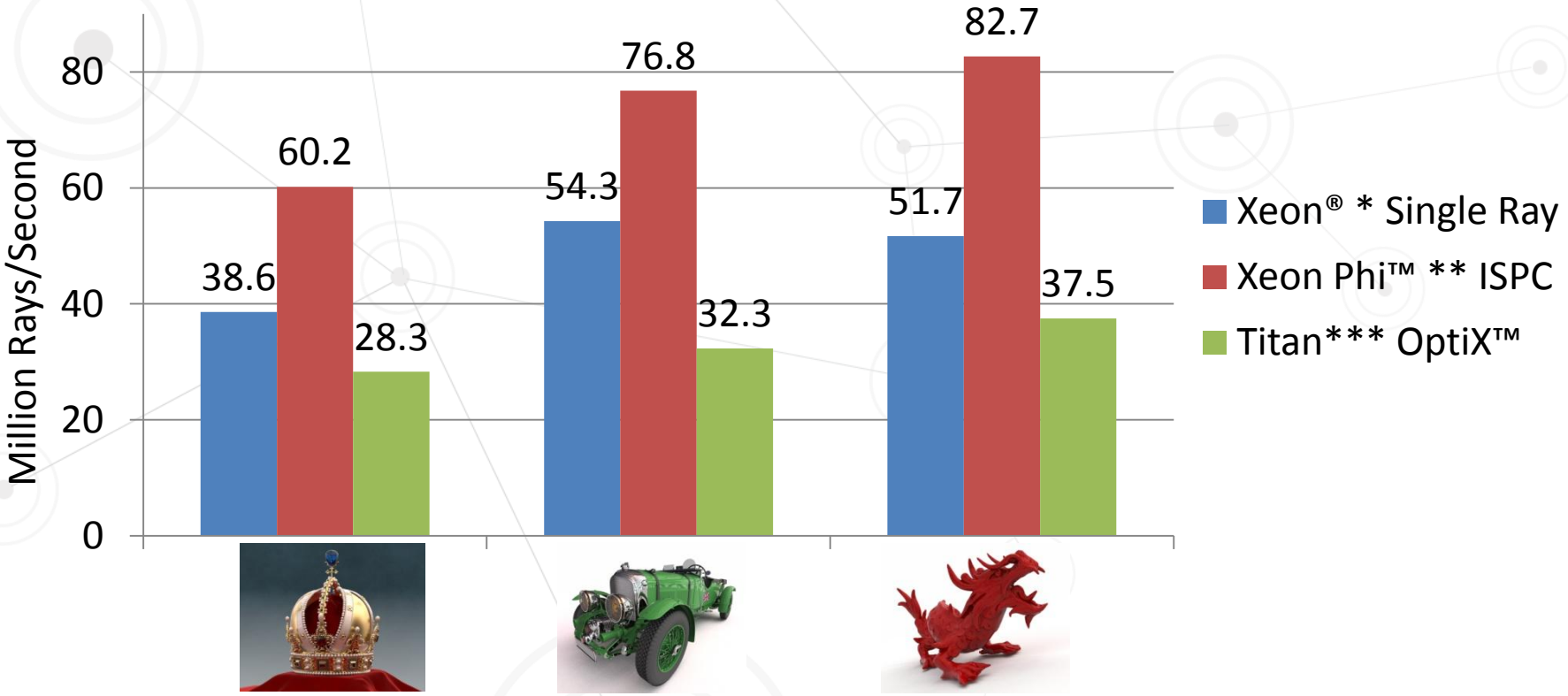
* Dual Socket Intel® Xeon® E5-2690, 2x 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz *** NVIDIA® GeForce® GTX™ Titan

Build Performance for Dynamic Scenes



* Dual Socket Intel® Xeon® E5-2690, 2x 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz *** NVIDIA® GeForce® GTX™ Titan

Ray Tracing Performance (incl. Shading)



* Dual Socket Intel® Xeon® E5-2690, 2x 8 cores @ 2.9 GHz ** Intel® Xeon Phi™ 7120, 61 cores @ 1.238 GHz *** NVIDIA® GeForce® GTX™ Titan

Embree API Tutorial

Embree API Overview

- Version 2 of the Embree API
- Compact and easy to use
- C++ and ISPC version
- Hides implementation details (such as different spatial index structures)

Scene

- Scene is container for set of geometries
- Scene flags passed at creation time
- 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 ();
}
```

Scene Types

- Static Scenes
  - Geometry cannot get changed
  - High quality BVH build (SAH), fast trace
  - For final frame rendering
- Dynamic Scenes
  - Geometries can get added, modified, and removed
  - Faster build (Morton), slower trace
  - Preview mode during geometric modeling

Geometries

- Geometries created inside a scene and always belong to that scene
- Geometries of scene can share the same or have separate spatial index structures internally
- Accessed via geometry identifier (geomID) of compact integer range
- Supported are: Triangle meshes, hair geometry, instances, and „user defined geometry“

Triangle Mesh

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

```
/* 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 during rendering and as long as geometry is alive)
- Support for stride and offset allows application flexibility in its data layout

Buffer Sharing Example

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

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

Ray Structure: Inputs

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

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

  Vec3f Ng;
  float u;
  float v;
  int geomID;
  int primID;
  int instID;
}
```

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;
}
```

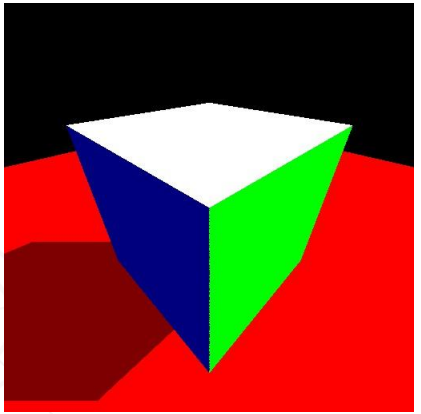
Tracing Rays

- **rtcIntersect** reports first intersection
  in tfar, Ng, u, v, geomID, primID, (instID)
- **rtcOccluded** reports any intersection
  by setting geomID to 0

Intel® SPMD Program Compiler (ISPC)

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

Embree Rendering: ISPC Example

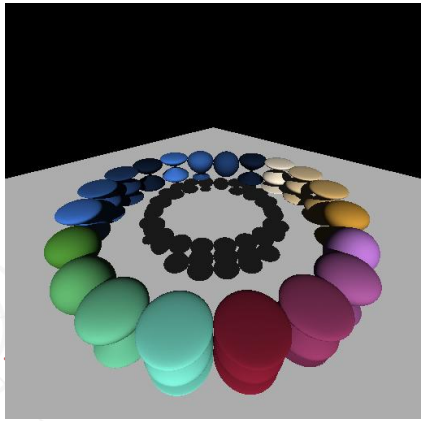


```
/* loop over all screen pixels */
foreach (y=0 ... screenHeight-1, x=0 ... screenWidth-1)
{
    /* create and trace primary ray */
    RTCRay ray = make_Ray(p,normalize(x*vx + y*vy + vz),eps,inf);
    rtcIntersect(scene,ray);

    /* 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;
}
```

Dynamic Scenes

- Create scene with RTC_SCENE_DYNAMIC flag
- Report modified meshes with **rtcUpdate** call
- Possibly enable (**rtcEnable**), disable (**rtcDisable**), add (**rtcNewXX**), and delete (**rtcDeleteGeometry**) geometries

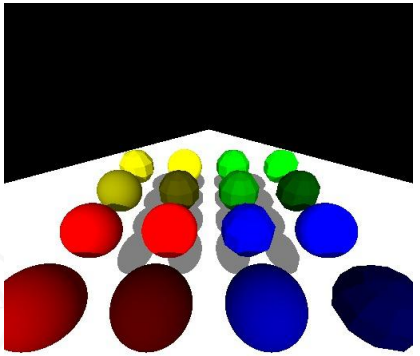
```
for each frame
{
  for each dynamic mesh
  {
    /* modify shared buffers
    modify mesh->indices
    modify mesh->vertices

    /* signal that mesh got updated */
    rtcUpdate(scene,mesh);
  }

  /* commit changes */
  rtcCommit (scene);

  /* trace rays */
  ...
}
```



User Defined Geometry

- Allows to extend Embree with new primitives types (spheres, subdivision surfaces, etc.)
- User has to provide bounding, intersect, and occluded functions
- User Geometry represents array of primitives (to reduce memory overheads)
- Maintains fast path for natively supported primitives

User Defined Geometry Example

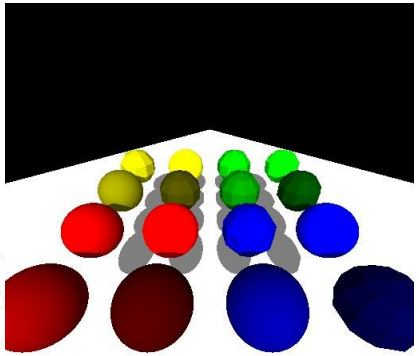


```
/* user defined sphere representation */
struct Sphere {
  Vec3f p;   //!< position of the sphere
  float r;   //!< radius of the sphere
};
```

```
/* create application representation of spheres  */
Sphere* spheres = new Sphere[N];
for (int i=0; i<N; i++) spheres[i] = createSphere(i);
```

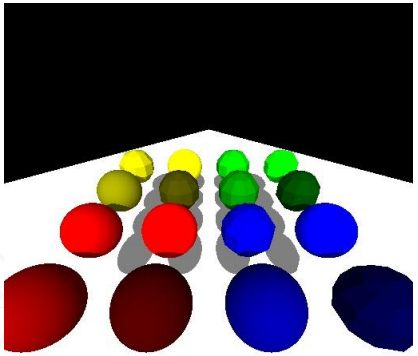
```
/* add user geometry to scene */
unsigned int geomID = rtcNewUserGeometry(scene,N);
rtcSetBoundsFunction    (scene,geomID,(RTCBoundsFunc)    &sphereBoundsFunc);
rtcSetIntersectFunction(scene,geomID,(RTCIntersectFunc)&sphereIntersectFunc);
rtcSetOccludedFunction  (scene,geomID,(RTCOccludedFunc )&sphereOccludedFunc);
rtcSetUserData          (scene,geomID, spheres);
```

User Defined Geometry Bounds



```
void sphereBoundsFunc(const Sphere* spheres,
                      uniform size_t N,
                      RTCBounds* bounds_o)
{
  const Sphere& sphere = spheres[N];
  bounds_o->lower_x = sphere.p.x-sphere.r;
  bounds_o->lower_y = sphere.p.y-sphere.r;
  bounds_o->lower_z = sphere.p.z-sphere.r;
  bounds_o->upper_x = sphere.p.x+sphere.r;
  bounds_o->upper_y = sphere.p.y+sphere.r;
  bounds_o->upper_z = sphere.p.z+sphere.r;
}
```

User Defined Geometry Occluded

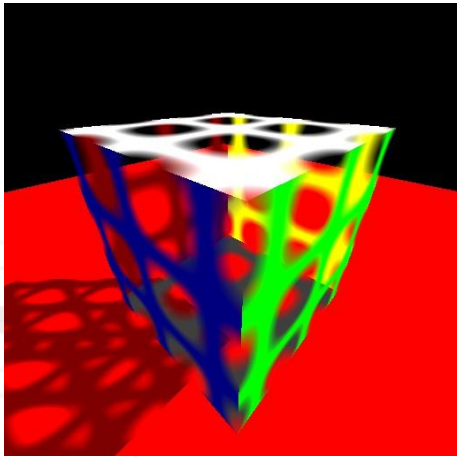


```
void sphereOccludedFunc(const Sphere* spheres, RTCRay& ray, size_t N)
{
  const Sphere& sphere = spheres[N];
  const Vec3fa v = ray.org-sphere.p;
  const float A = dot(ray.dir,ray.dir);
  const float B = 2.0f*dot(v,ray.dir);
  const float C = dot(v,v) - sqr(sphere.r);
  const float D = B*B - 4.0f*A*C;
  if (D < 0.0f) return; // reports miss
  const float Q = sqrt(D);
  const float rcpA = rcp(A);
  const float t0 = 0.5f*rcpA*(-B-Q);
  const float t1 = 0.5f*rcpA*(-B+Q);
  if ((ray.tnear < t0) & (t0 < ray.tfar)) {
    ray.geomID = 0; // report hit
  }
  if ((ray.tnear < t1) & (t1 < ray.tfar)) {
    ray.geomID = 0; // report hit
  }
}
```

Intersection Filter Functions

- Per geometry callback that is called during traversal for each primitive intersection
- Can accept or reject hit (by setting geomID to -1)
- Can be used for:
  - Trimming curves (e.g. modeling tree leaves)
  - Transparent shadows (reject and accumulate)
  - Find all hits (reject and collect)

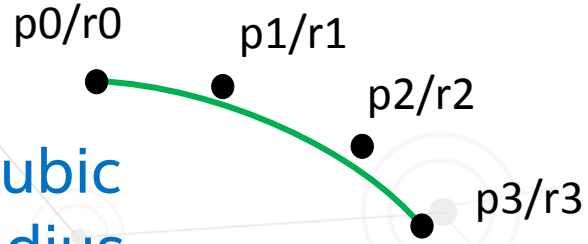
Filter Function Example



```
/* 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

Summary

- Embree delivers high ray tracing performance on CPUs
- Embree has potential to speed up many ray tracing applications
- Embree is easy to use through its API
- Free and Open Source (https://embree.github.com)

Embree at Siggraph

- **Embree Ray Tracing Kernels: Tutorial and Application at Dreamworks Animation and in Autodesk Maya**

  Wednesday 2pm – 4:15pm (West Building, Rooms 208-209)

- **Embree – A Ray Tracing Kernel Framework for Efficient CPU Ray Tracing**

  Thursday 9am - 10:30am (East Building, Ballroom B-C)

- **Embree Demo at Exhibition**

  Tuesday – Thursday, Intel® Booth 1001 (West Building, Hall B/C)

Questions?

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