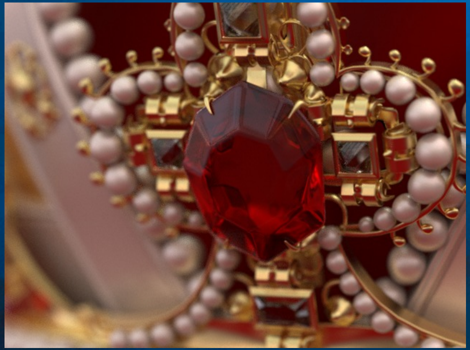
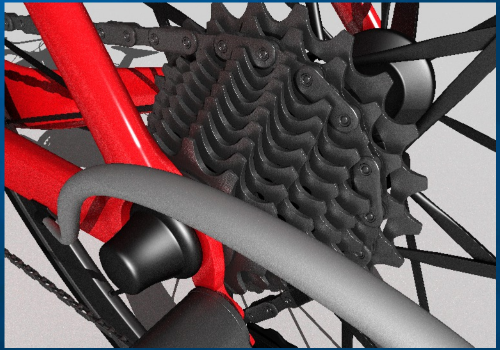
# Embree Ray Tracing Kernels

*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

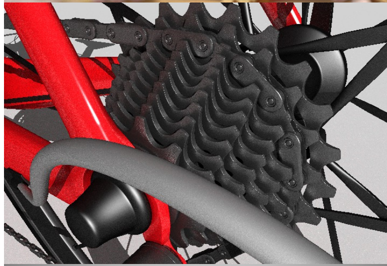
# Outline

- Embree Overview
- Embree Performance
- Embree API
- Catmull Clark Subdivision Surfaces

# 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.
- 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

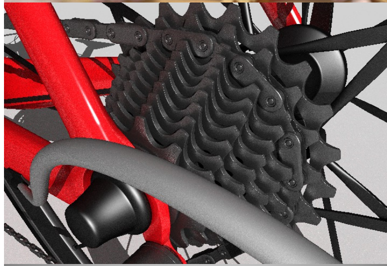
# Observations

- 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)

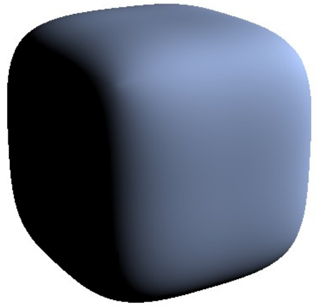


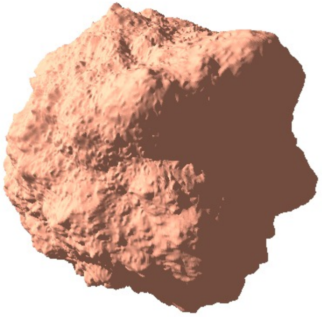
# Embree Features

- 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 detai
- Interpolation
- Initial AVX512 support
  – 16 wide AVX512 traversal kernels
  – Full AVX512 optimizations will come when hardware available!





# Embree System Overview

Embree API (C++ and ISPC)

Ray Tracing Kernel Selection

| Accel. structure | Builders | Subdiv Engine | Traversal | Intersection |
|---|---|---|---|---|
| bvh4.triangle4, bvh8.triangle8, bvh4aos.triangle1, bvh4.grid … | SAH builder<br>Spatial split builder<br>Morton code builder<br>BVH Refitter | B-Spline Patch<br>Gregory Patch<br>TessellationCache<br>Displ. Mapping | Single ray (SSE2, AVX, AVX2), packet (SSE2), hybrid (SSE4.2), … | MöllerTrumbore, Plücker Variant, Bezier Curve, Triangle Grids |

Common Vector and SIMD Library
(Vec3f, Vec3fa, float4, float8, float16, SSE2, SSE4.1, AVX, AVX2, AVX512)

# 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 Methology

- 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)

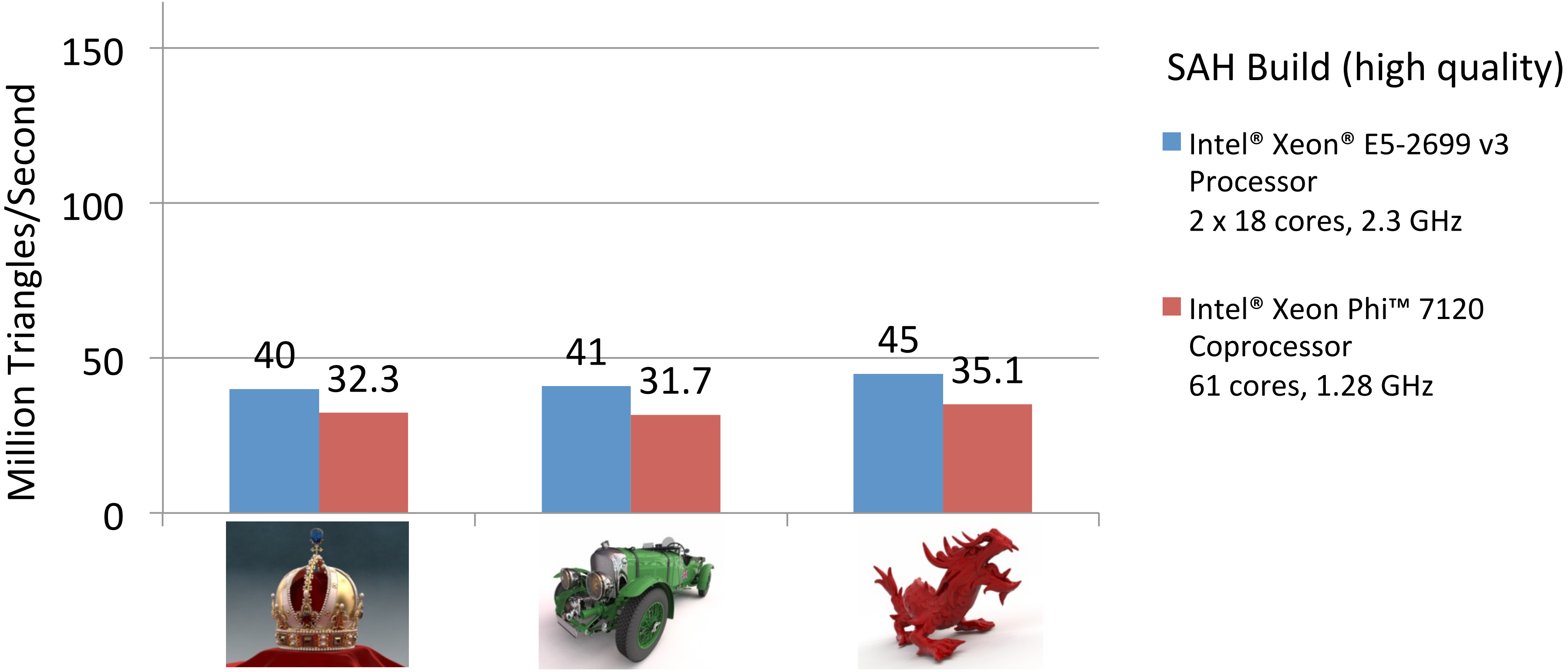
Imperial Crown of Austria
4.3M triangles

Bentley 4.5l Blower (1927)
2.3M triangles

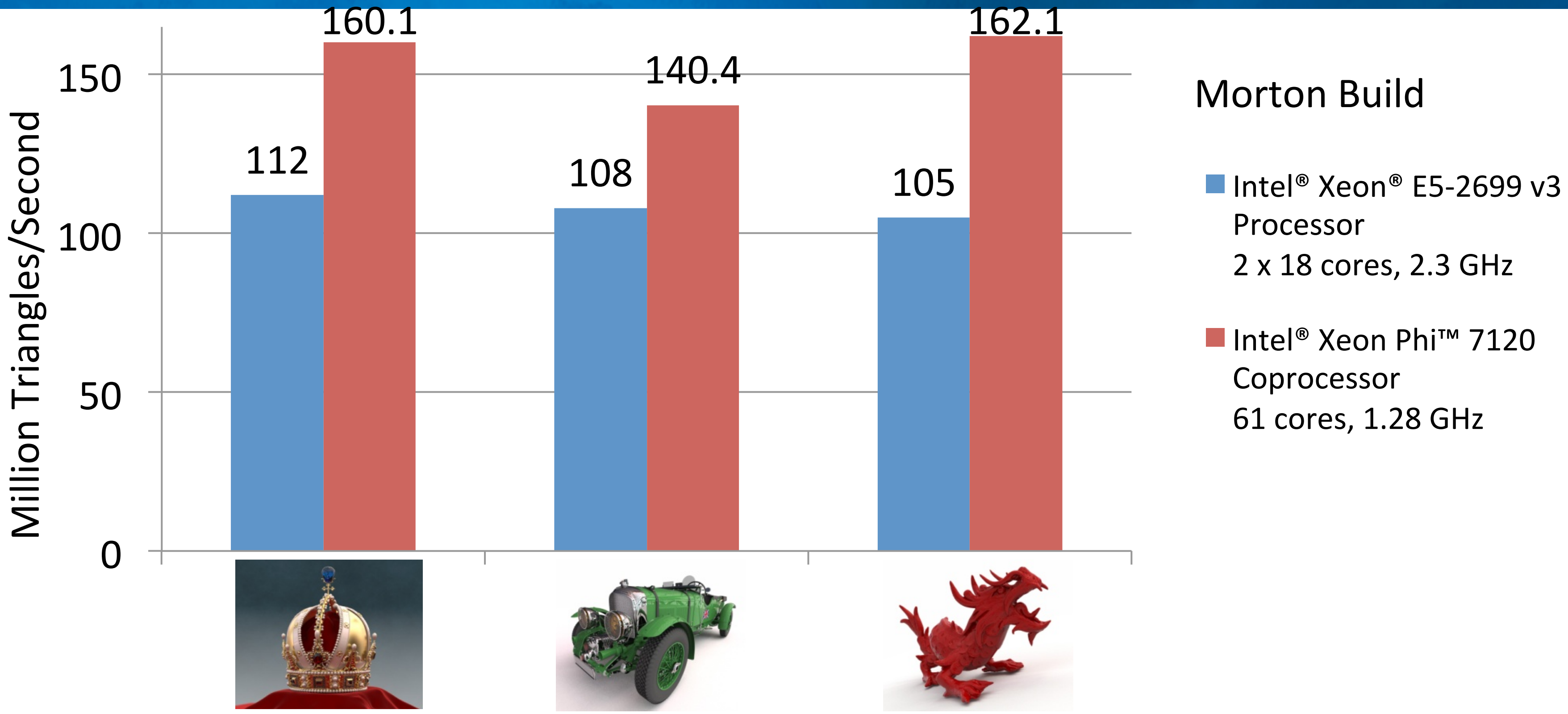
Asian Dragon
7.3M triangles

* 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

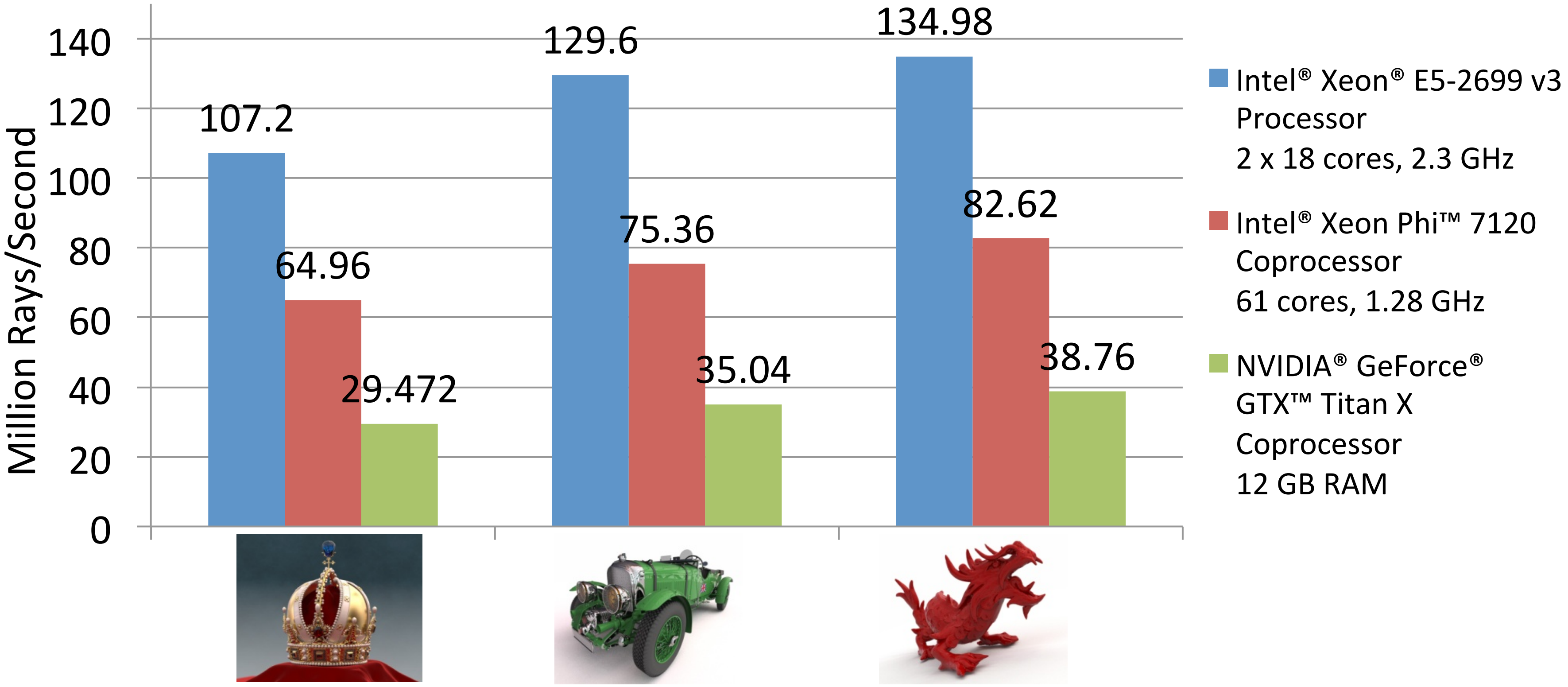
# Build Performance for Static Scenes



# Build Performance for Dynamic Scenes



# Ray Tracing Performance (incl. Shading)



# Embree API

# Scene Object

- 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) ➔ 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)

```
/* 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

# Buffer Sharing Example

```
/* 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]

```
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.

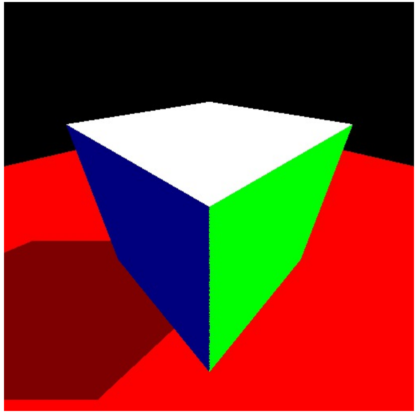
```
struct RTCRay
{
  Vec3f org;
  Vec3f dir;
  float tnear;
  float tfar;
  float time;

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

# Intel® SPMD Program Compiler (ISPC)

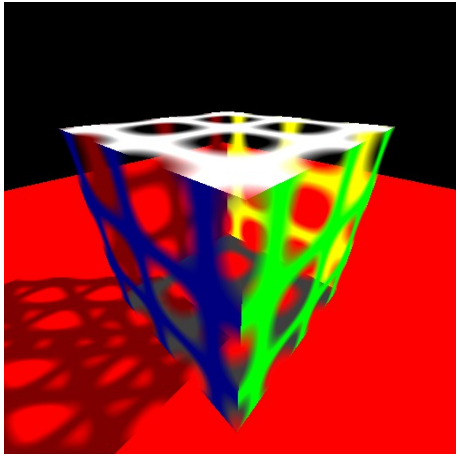
- Simplifies writing vectorized renderer
- C-based language plus vector extensions
- Scalar looking code that gets vectorized automatically
- Guaranteed vectorization
- Compilation to different vector ISAs (SSE, AVX, AVX2, AVX512, Xeon Phi™)
- Available as Open Source from http://ispc.github.com

# 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;
}
```



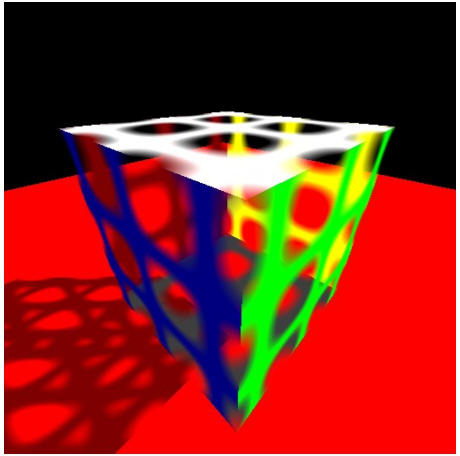
# Intersection Filter Functions



- 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)

# Filter Function Example

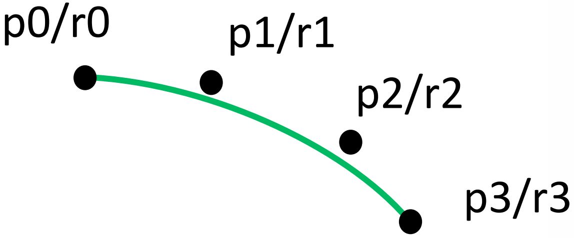
```
/* procedural intersection filter function */
void intersectionFilter(void* userPtr, RTCRay& ray)
{
    Vec3fa h = ray.org + ray.dir*ray.tfar;
    float v = abs(sin(4.0f*h.x)*cos(4.0f*h.y)*sin(4.0f*h.z));
    float T = clamp((v-0.1f)*3.0f,0.0f,1.0f);
    if (T > 1.0f) return;                    // 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

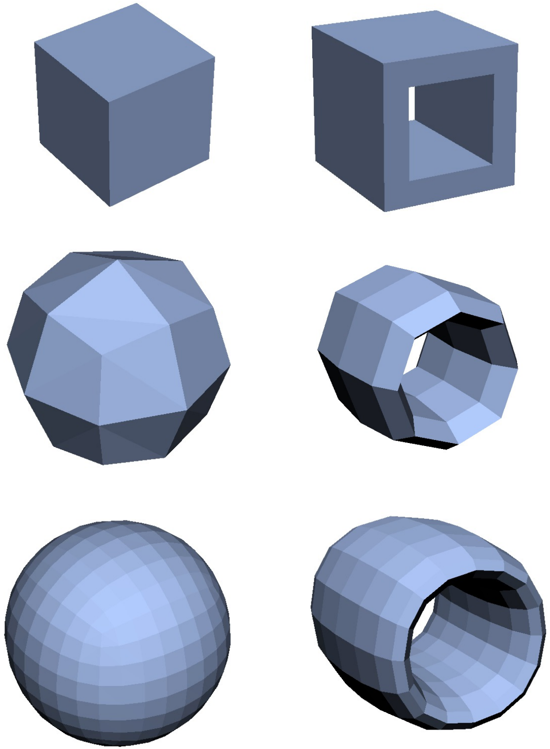
- Hair curves represented as cubic bezier curves with varying radius
- High performance through use of oriented bounding boxes
- Low memory consumption through direct ray/curve intersection



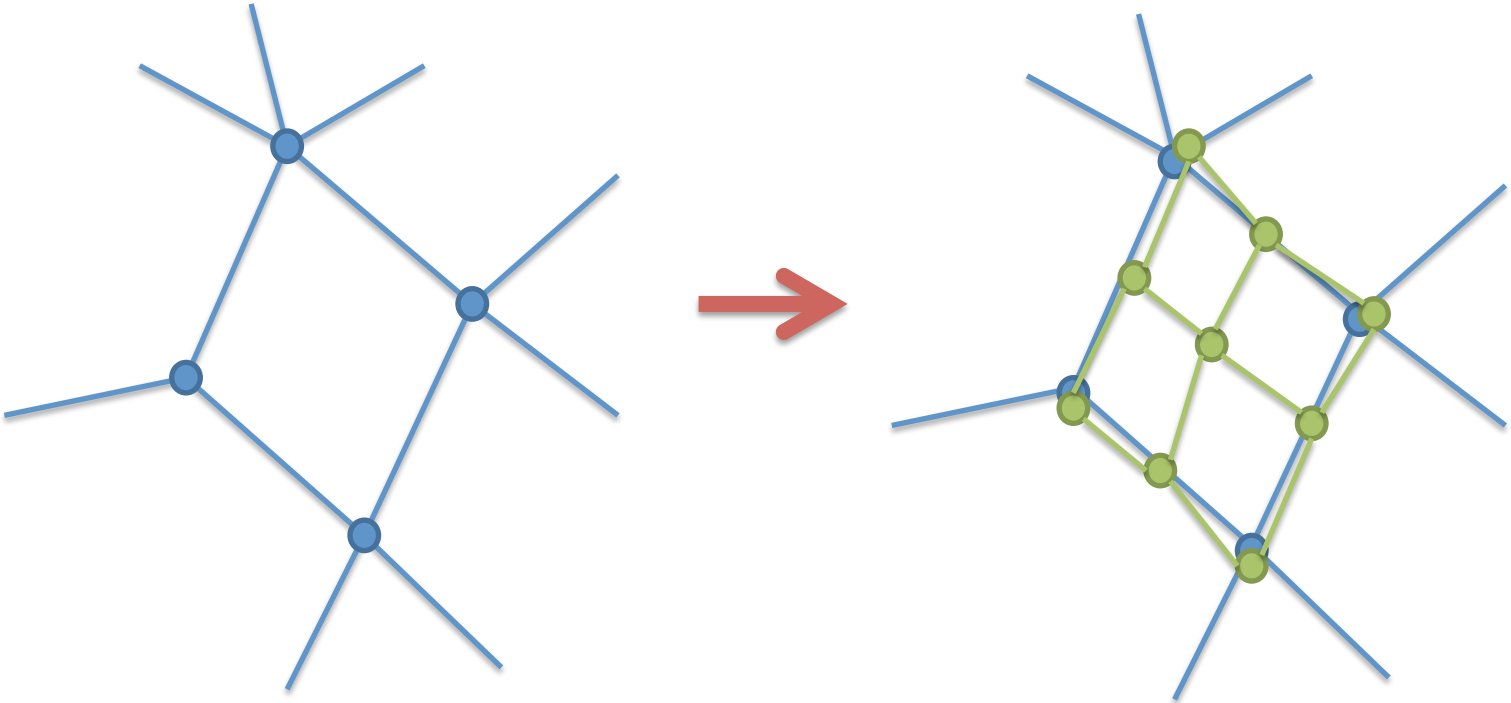

# Catmull Clark Subdivision Surfaces

# 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



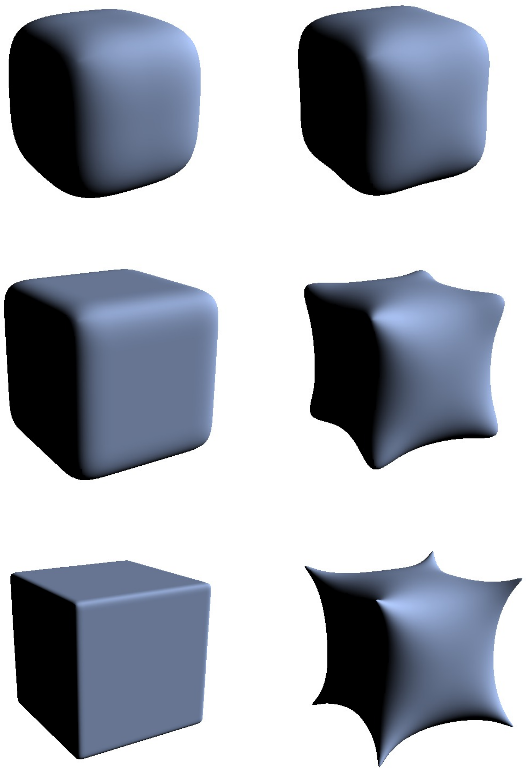
# CC Subdivision Surface Advantages

- 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

# Embree Subdivision Features

- 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

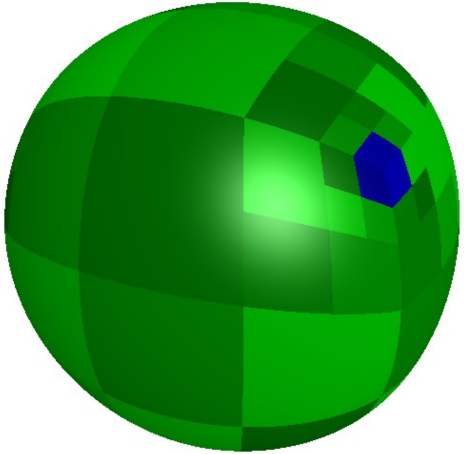
```
unsigned geomID = rtcNewSubdivisionMesh (scene, RTC_GEOMETRY_STATIC,
  numFaces, numIndices, numVertices,
  numEdgeCreases, numVertexCreases, numHoles);
```

```
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));
```

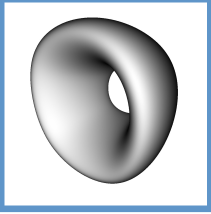
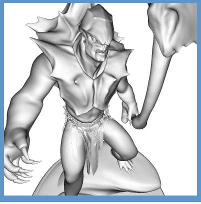
# Embree Subdivision Implemention

- 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)

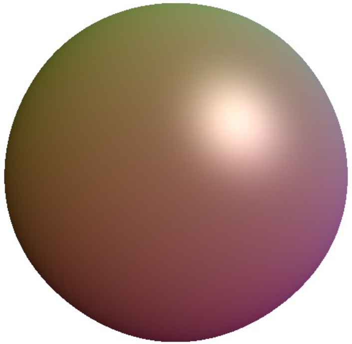
# Embree Subdivision Performance

| |  |  |  |
|---|---|---|---|
| Patches | 16 | 52k | 53k |
| Edge Creases | 0 | 0 | 30k |
| Micro Quads | 1048k | 831k | 837k |
| Same View | 105 fps | 84 fps | 100 fps |
| Walkthrough | 40 fps | 72 fps | 80 fps |

Intel® Xeon® E5-2690
2.9 GHz
2x 8 cores
1024 x 1024 pixels

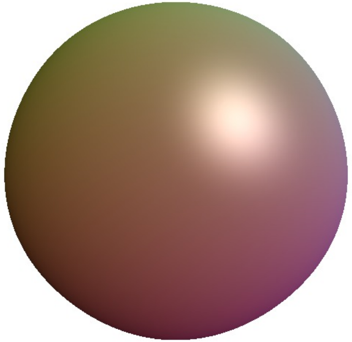
# Vertex Data Interpolation

- 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

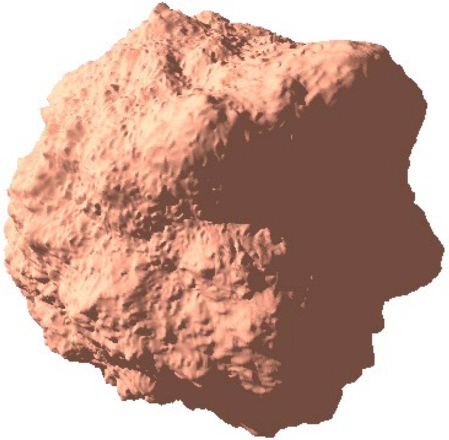
```
rtcNewScene (RTC_STATIC, RTC_INTERSECT1 | RTC_INTERPOLATE);
...
unsigned geomID = rtcNewSubdivisionMesh (...);
rtcSetBuffer (scene,geomID,RTC_INDEX_BUFFER,  indices,  0, sizeof(int));
rtcSetBuffer (scene,geomID,RTC_VERTEX_BUFFER, vertices, 0, sizeof(float3));
rtcSetBuffer (scene,geomID,RTC_USER_VERTEX_BUFFER, 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

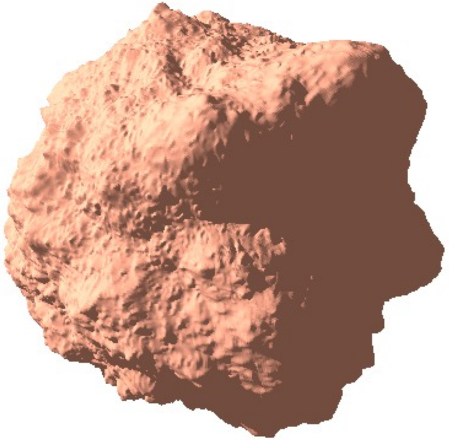
- Support for vector displacement
- Tessellation approach enables displacements
- Callback function displaces vertex positions
- Smooth normals possible through approximation

```
Q = P + D*Ng
dQdu ≈ dPdu + dDdu*Ng
dQdv ≈ dPdv + dDdv*Ng
```



# Displaced Subdivision Surface Example

```
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);
```

# Demo

# 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