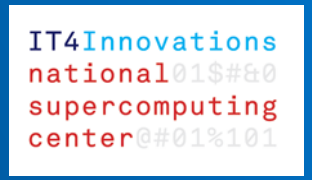


Rendering in Blender Cycles Using AVX-512 Vectorization

IT4Innovations, VŠB – Technical University of Ostrava, Ostrava, Czech Republic

Milan Jaroš, Petr Strakoš, Lubomír Říha

Outline

Part I.

- Blender Cycles introduction
- Acceleration Structures for Path-tracing
- CyclesPhi

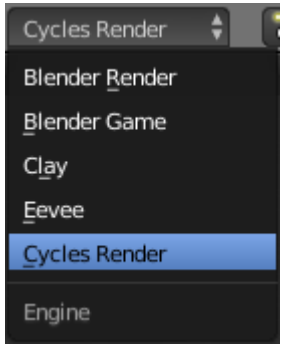
Part II.

- Intel® AVX-512
- Embree integration
- Benchmarks

Blender & Cycles Render

- **Blender** is an open source 3D creation suite. It has five render engines in the new version 2.8: Blender Internal, Blender Game, Clay, Eevee and Cycles.
- **Cycles** is a path-tracing based render engine with support for interactive rendering, shading node system, and texture workflow.

The Daily Dweebs

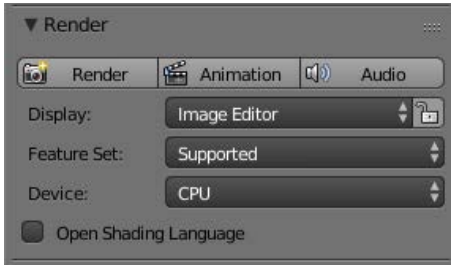


Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

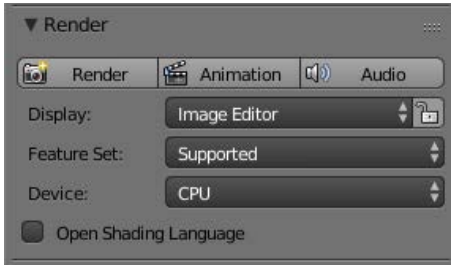
```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

1.) Write a C++ function for rendering

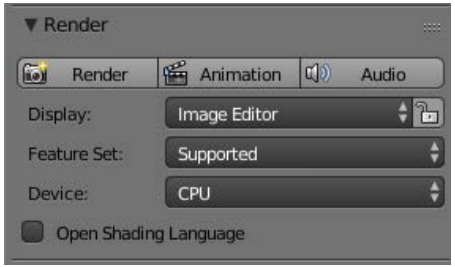
```
//blender/intern/cycles/blender/blender_session.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

2.) Write a Python-callable function

```
//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject
*value) {
 BlenderSession *session =
(BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
```

1.) Write a C++ function for rendering

```
//blender/intern/cycles/blender/blender_sessio
n.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

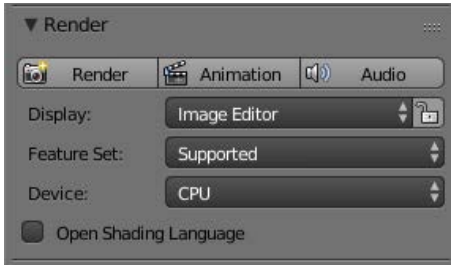
More at: http://intermediate-and-advanced-software-carpentry.readthedocs.io/en/latest/c++-wrapping.html

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

3.) Register this function within a module's symbol table

```
//intern/cycles/blender/blender_python.cpp
static struct PyModuleDef module = {
 PyModuleDef_HEAD_INIT,
 "_cycles",
 "Blender cycles render integration", -1,
 methods,
 NULL, NULL, NULL, NULL
};
```

```
//intern/cycles/blender/blender_python.cpp
static PyMethodDef methods[] = {
  //...
 { "render", render_func, METH_O, "" },
 { "bake", bake_func, METH_VARARGS, "" },
 { "draw", draw_func, METH_VARARGS, "" },
 //...
 { NULL, NULL, 0, NULL },
};
```

2.) Write a Python-callable function

```
//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject
*value) {
 BlenderSession *session =
(BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
```

1.) Write a C++ function for rendering

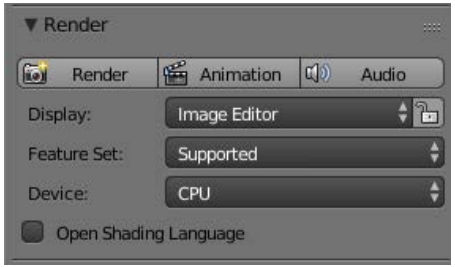
```
//blender/intern/cycles/blender/blender_sessio
n.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

3.) Register this function within a module's symbol table

```
//intern/cycles/blender/blender_python.cpp
static struct PyModuleDef module = {
 PyModuleDef_HEAD_INIT,
 "_cycles",
 "Blender cycles render integration", -1,
 methods,
 NULL, NULL, NULL, NULL
};
```

4.) Write an init function for the module

```
//intern/cycles/blender/blender_python.cpp
void *CCL_initPython() {
 PyObject *mod =
PyModule_Create(&ccl::module);
 //...
 return (void*)mod;
```

```
//intern/cycles/blender/blender_python.cpp
static PyMethodDef methods[] = {
  //...
 { "render", render_func, METH_O, "" },
 { "bake", bake_func, METH_VARARGS, "" },
 { "draw", draw_func, METH_VARARGS, "" },
 //...
 { NULL, NULL, 0, NULL },
};
```

2.) Write a Python-callable function

```
//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject
*value) {
 BlenderSession *session =
(BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
```

1.) Write a C++ function for rendering

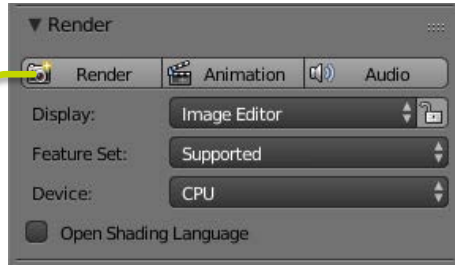
```
//blender/intern/cycles/blender/blender_sessio
n.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

3.) Register this function within a module's symbol table

```
//intern/cycles/blender/blender_python.cpp
static struct PyModuleDef module = {
 PyModuleDef_HEAD_INIT,
 "_cycles",
 "Blender cycles render integration", -1,
 methods,
 NULL, NULL, NULL, NULL
};
```

```
//intern/cycles/blender/blender_python.cpp
static PyMethodDef methods[] = {
  //...
 { "render", render_func, METH_O, "" },
 { "bake", bake_func, METH_VARARGS, "" },
 { "draw", draw_func, METH_VARARGS, "" },
 //...
 { NULL, NULL, 0, NULL },
};
```

4.) Write an init function for the module

```
//intern/cycles/blender/blender_python.cpp
void *CCL_initPython() {
 PyObject *mod =
PyModule_Create(&ccl::module);
 //...
 return (void*)mod;
```

2.) Write a Python-callable function

```
//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject
*value) {
 BlenderSession *session =
(BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
```

1.) Write a C++ function for rendering

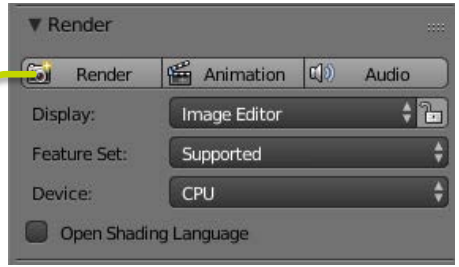
```
//blender/intern/cycles/blender/blender_sessio
n.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

```
//source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }};
```

Blender GUI



Starting the rendering plugin

```
//intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```

3.) Register this function within a module's symbol table

```
//intern/cycles/blender/blender_python.cpp
static struct PyModuleDef module = {
 PyModuleDef_HEAD_INIT,
 "_cycles",
 "Blender cycles render integration", -1,
 methods,
 NULL, NULL, NULL, NULL
};
```

```
//intern/cycles/blender/blender_python.cpp
static PyMethodDef methods[] = {
  //...
 { "render", render_func, METH_O, "" },
 { "bake", bake_func, METH_VARARGS, "" },
 { "draw", draw_func, METH_VARARGS, "" },
 //...
 { NULL, NULL, 0, NULL },
};
```

4.) Write an init function for the module

```
//intern/cycles/blender/blender_python.cpp
void *CCL_initPython() {
 PyObject *mod =
PyModule_Create(&ccl::module);
 //...
 return (void*)mod;
```

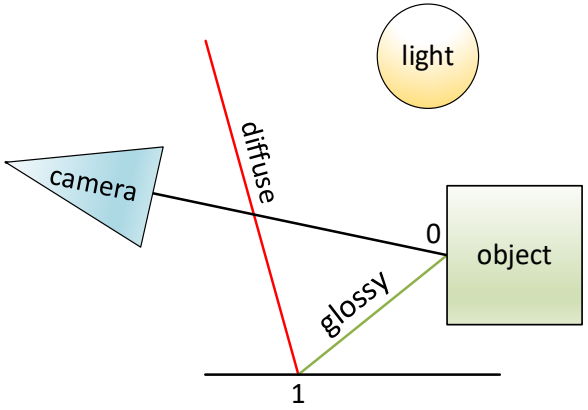
2.) Write a Python-callable function

```
//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject
*value) {
 BlenderSession *session =
(BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
```

1.) Write a C++ function for rendering

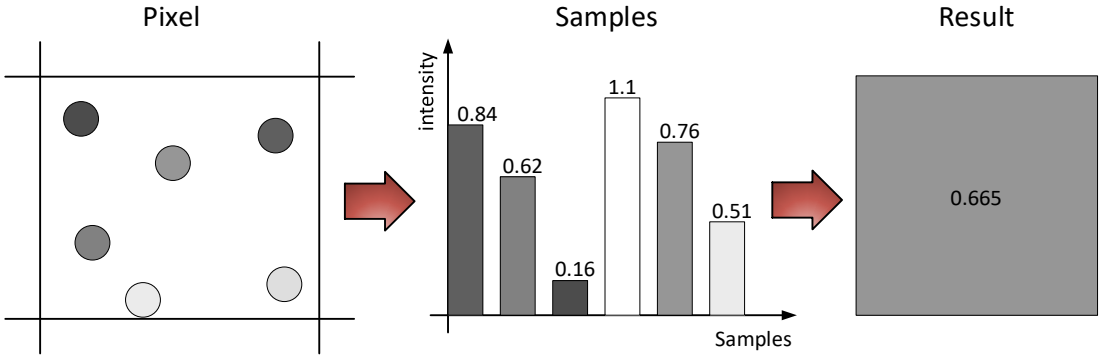
```
//blender/intern/cycles/blender/blender_sessio
n.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
```

Path-tracing (unbiased vs. biased)



Acceleration methods:

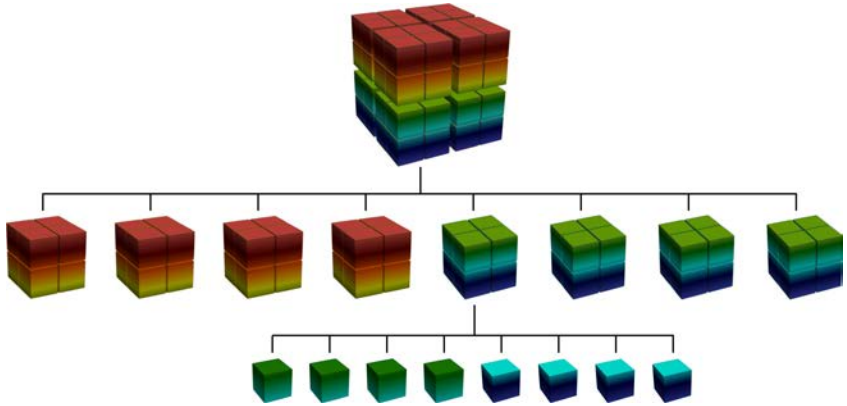
- Adaptive rendering (unbiased->biased)
- Filtering (unbiased->biased)
- Acceleration structures (BVH, kd-tree, …)
- Distributed rendering



Acceleration Structures

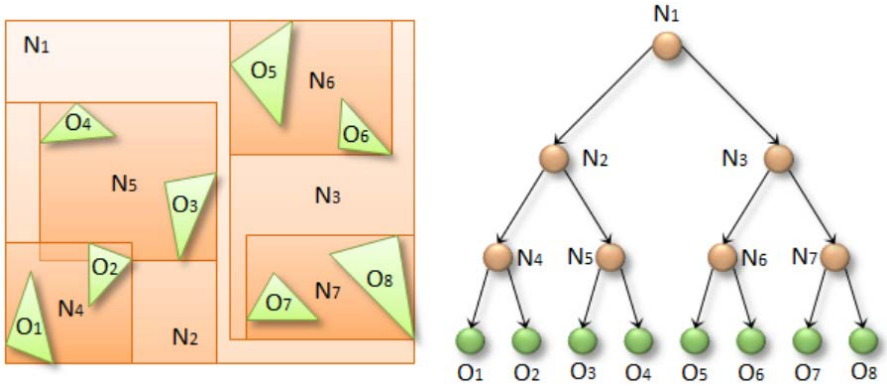
Space partitioning

- octree
- kd-tree (binary tree)



Bounding Volume Hierarchy (BVH)

- BVH2
- BVH4
- BVH8

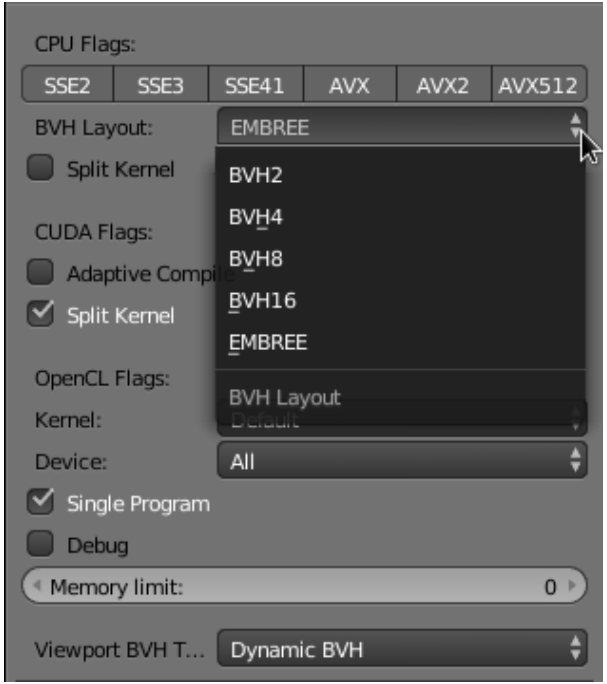


https://devblogs.nvidia.com/parallelforall/thinking-parallel-part-ii-tree-traversal-gpu/

Bounding Volume Hierarchy (BVH)

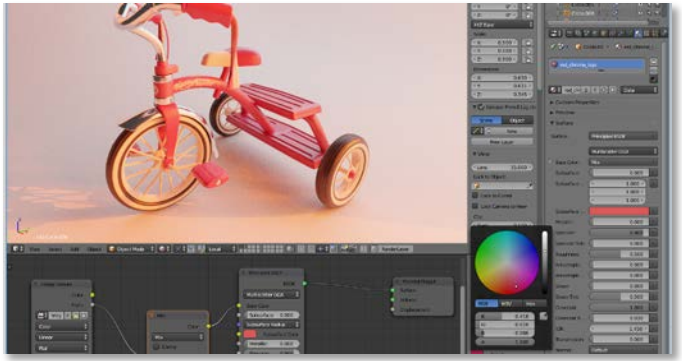
The raytracing acceleration structure used is a bounding volume hierarchy. The original code is based on an implementation from Nvidia (BVH2) and Embree (Hair BVH builder - oriented SAH, BVH4 traversal, BVH nodes intersection, Triangles intersection).

- BVH2 (wo, SSE)
- BVH4 (SSE2)
- BVH8 (AVX(2))
- BVH16 (AVX-512) – in progress
- EMBREE – BVH4 (SSE)
- EMBREE – BVH8 (AVX(2), AVX-512)

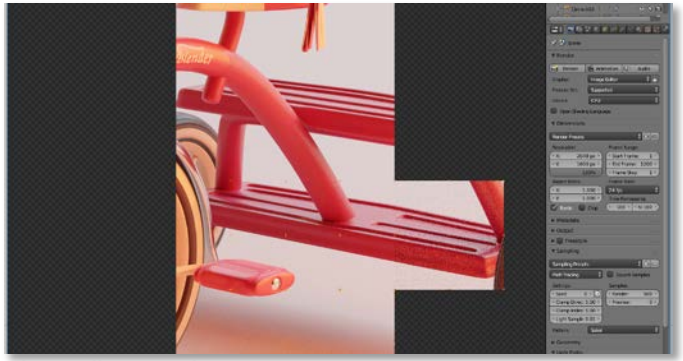


Bounding Volume Hierarchy (BVH)

- **Dynamic BVH (interactive)** allows you to move objects around in the scene with quick viewport updates in rendered mode. But the image will take longer to clear up.
- **Static BVH (offline)** will create a completely new BVH tree whenever an object is moved around but the noise will clear up faster.



The Daily Dweebs

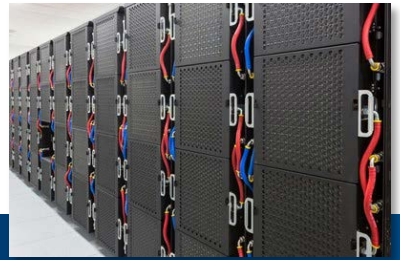


The Daily Dweebs

IT4Innovations

Salomon

- 1008 nodes
- 2x Intel Xeon E5-2680v3, 2.5 GHz, 12 cores (Hasswell)
- 128GB
- InfiniBand FDR56 / 7D Enhanced hypercube
- 576 nodes w/o accelerator
- 432 nodes with 2x Intel Xeon Phi 7120P, 61 cores, 16 GB RAM



Anselm

- 209 nodes
- 2x Intel Xeon E5-2665, 2.4 GHz, 8 cores (Sandy Bridge)
- 64GB
- InfiniBand FDR56 / 7D Enhanced hypercube
- 180 nodes w/o accelerator
- 4 nodes with MIC accelerator (1x Intel Xeon Phi 5110P)
- 23 nodes with NVIDIA Kepler K20

CyclesPhi

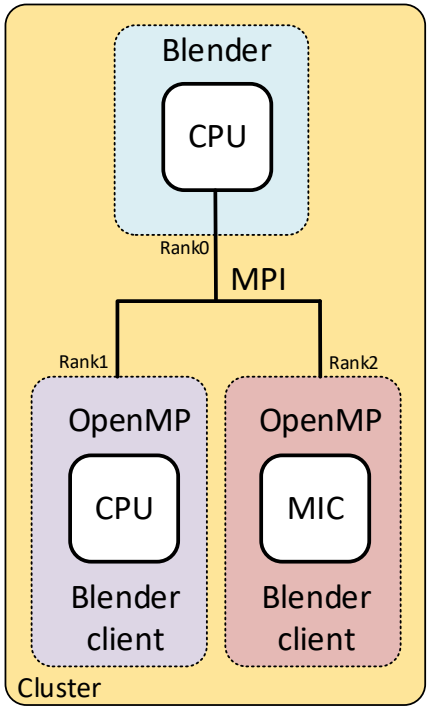
We have modified the kernel of the Blender Cycles rendering engine and then extended its capabilities to support the HPC environment. We call this version the CyclesPhi and it supports following technologies:

- OpenMP
- MPI
- Sockets
- Intel® Xeon Phi™ with Symmetric mode (KNC, KNL)
- Intel® Xeon and Intel® Xeon Phi™ with Intel® AVX-512

MPI, Hybrid and Native mode

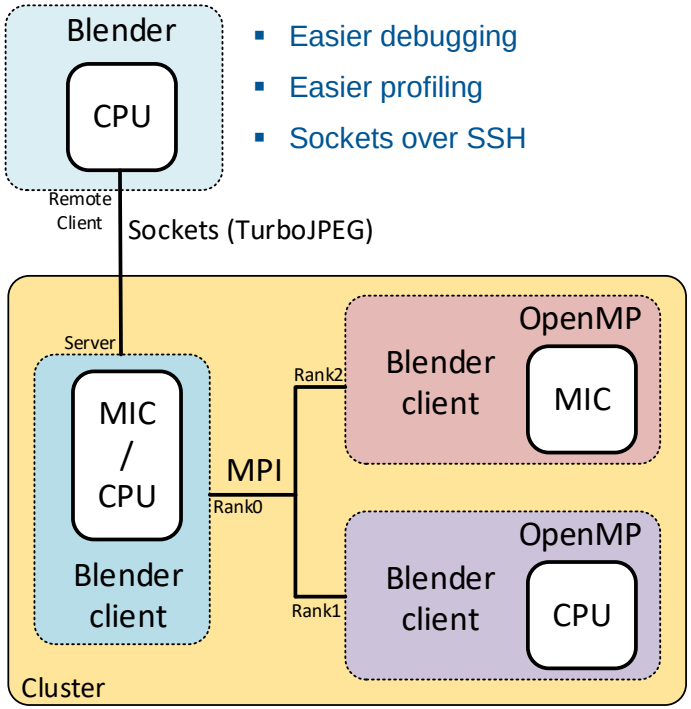
Distributed rendering

MPI mode (KNC, KNL, Skylake)



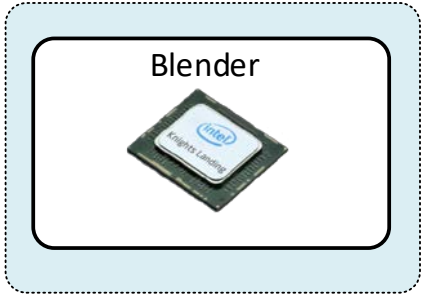
Hybrid mode (KNC, KNL, Skylake)

- Easier debugging
- Easier profiling
- Sockets over SSH

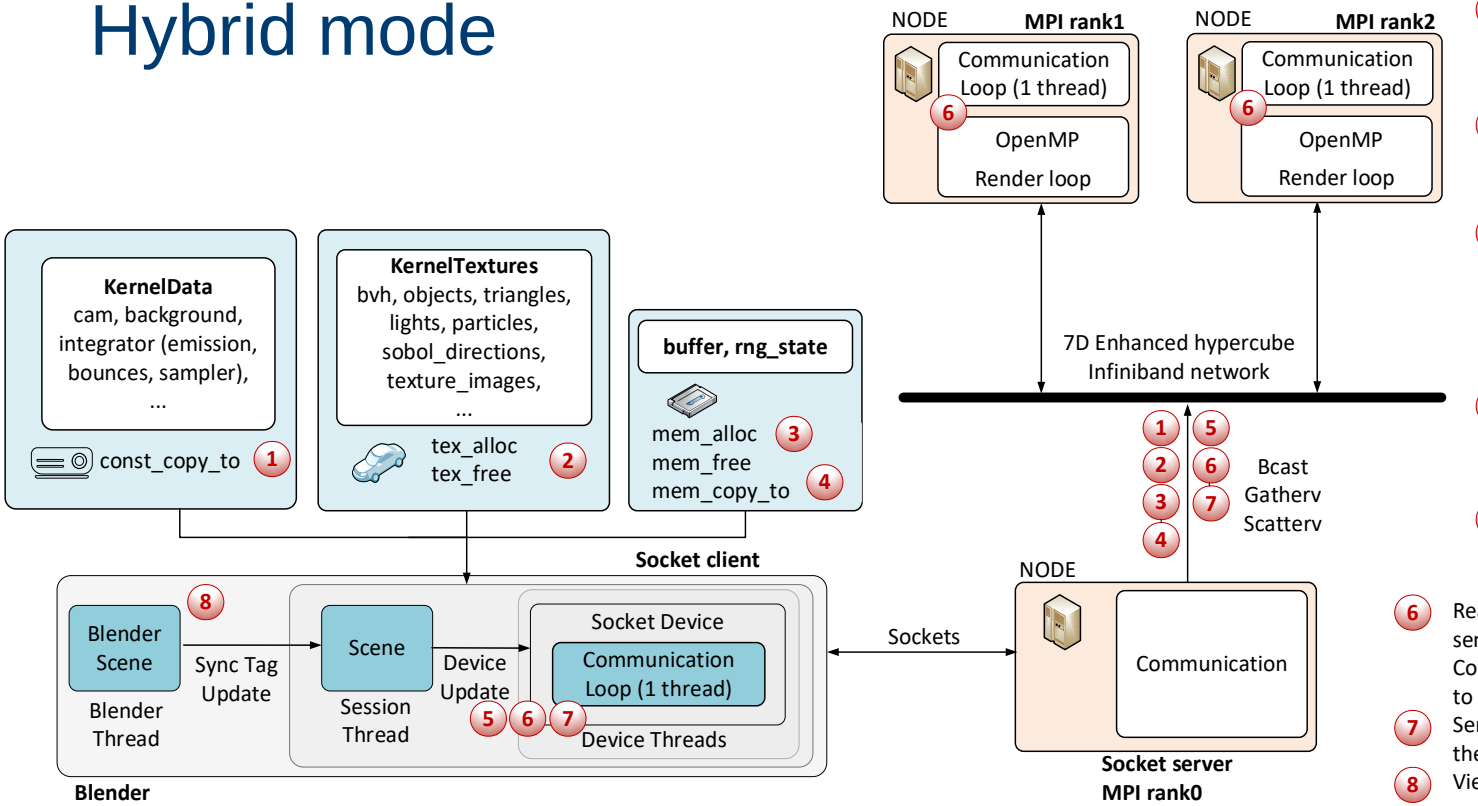


Single node rendering

Native mode (KNL, Skylake)



Rendering using Hybrid mode



1. Send **KernelData** (ex. camera properties) to Socket server and from there to all nodes with **Bcast.**
2. Send **KernelTextures** (ex. triangles) to Socket server and from there to all nodes with **Bcast.**
3. Send the information about the size of **buffer** (rendered pixels) and size of **rng_state** (random number generator state) to Socket server and from there to all nodes with **Bcast.**
4. Send the initial values of **rng_state** to Socket server and from there to all nodes with **Bcast.**
5. Start rendering in Socket client and distribute the command via Socket server by **Bcast** message.
6. Read the current results from node and send the **buffer** to root with **Gatherv**. Compress the results by JPEG and send it to Socket client.
7. Send new jobs to Socket server and from there to clients with **Scatterv.**
8. View results in Blender.

Outline

Part I.

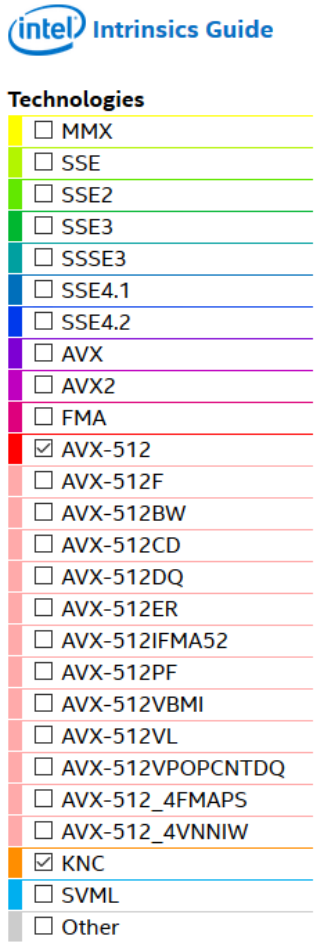
- Blender Cycles introduction
- Acceleration Structures for Path-tracing
- CyclesPhi

Part II.

- Intel® AVX-512
- Embree integration
- Benchmarks

AVX-512 Vectorization and Intrinsics Instructions

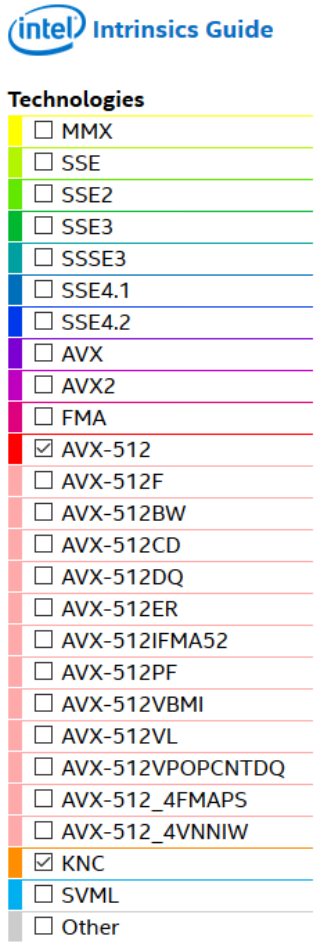
- AVX-512 = Intel® Advanced Vector Extensions 512
- KNL and Skylake support AVX-512
- KNC supports many instructions from AVX-512



AVX-512 Vectorization and Intrinsics Instructions

- AVX-512 = Intel® Advanced Vector Extensions 512
- KNL and Skylake support AVX-512
- KNC supports many instructions from AVX-512

```
float dst[16], a[16], b[16];
// The intialization of a, b
// ...
for (int j = 0; j < 16; j++) {
   dst[j] = a[j] * b[j];
}
```

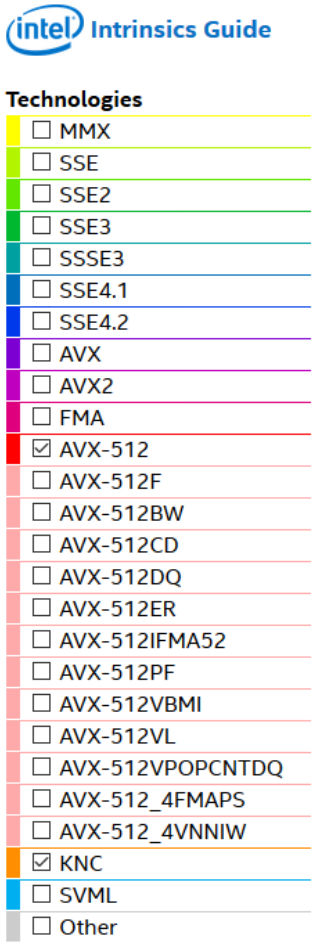


AVX-512 Vectorization and Intrinsics Instructions

- AVX-512 = Intel® Advanced Vector Extensions 512
- KNL and Skylake support AVX-512
- KNC supports many instructions from AVX-512

```
float dst[16], a[16], b[16];
// The intialization of a, b
// ...
for (int j = 0; j < 16; j++) {
   dst[j] = a[j] * b[j];
}
```

```
__m512 dst, a, b;
// The intialization of a, b
// ...
dst = _mm512_mul_ps(a, b);
```



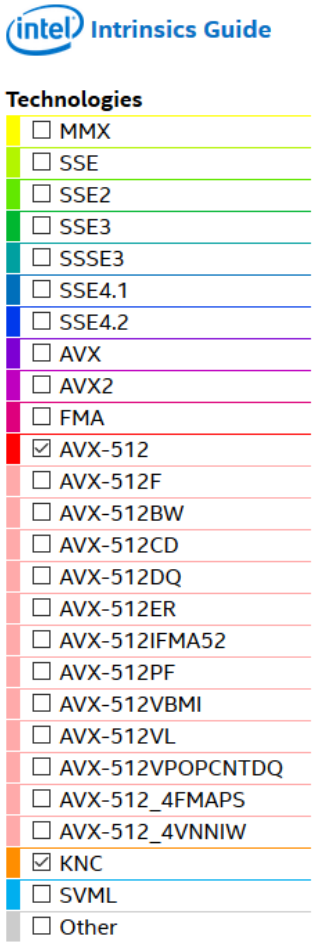
AVX-512 Vectorization and Intrinsics Instructions

- AVX-512 = Intel® Advanced Vector Extensions 512
- KNL and Skylake support AVX-512
- KNC supports many instructions from AVX-512

```
float dst[16], a[16], b[16];
// The intialization of a, b
// ...
for (int j = 0; j < 16; j++) {
   dst[j] = a[j] * b[j];
}
```

```
__m512 dst, a, b;
// The intialization of a, b
// ...
dst = _mm512_mul_ps(a, b);
```

```
struct avx512f {
  union {
    __m512 v;
    float f[16];
  };
  __forceinline avx512f operator*(
    const avx512f& a, const avx512f& b) {
      return _mm512_mul_ps(a, b);
  }
}

avx512f dst, a, b;
// The intialization of a, b
// ...
dst = a * b;
```



API from Embree: avx512f, avx512i, avx512b

```
/* 16-wide AVX-512 float type */
struct avx512f
{
  enum  { size = 16 }; // number of SIMD elements
  union {              // data
    __m512 v;
    float f[16];
    int i[16];
}; }

/* 16-wide AVX-512 integer type */
struct avx512i
{
  enum  { size = 16 }; // number of SIMD elements
  union {              // data
    __m512i v;
    int i[16];
}; }

/* 16-wide AVX-512 bool type */
struct avx512b
{
  enum { size = 16 };   // number of SIMD elements
  __mmask16 v;          // data
}
```

- Constructors, Assignment & Cast Operators
- Loads and Stores
- Constants
- Array Access
- Unary Operators
- Binary Operators
- Ternary Operators
- Operators with rounding
- Assignment Operators
- Comparison Operators + Select
- Rounding Functions
- Movement/Shifting/Shuffling Functions
- Reductions
- Memory load and store operations

Example of AVX-512 Vectorization: Ray-Triangle Intersection

```
/* Calculate vertices relative to ray origin. */
const float3 v0 = tri_c - ray_P;
const float3 v1 = tri_a - ray_P;
const float3 v2 = tri_b - ray_P;

/* Calculate triangle edges. */
const float3 e0 = v2 - v0;
const float3 e1 = v0 - v1;
const float3 e2 = v1 - v2;

/* Perform edge tests. */
const float U = dot(cross(v2 + v0, e0), ray_dir);
const float V = dot(cross(v0 + v1, e1), ray_dir);
const float W = dot(cross(v1 + v2, e2), ray_dir);

const float minUVW = min(U, min(V, W));
const float maxUVW = max(U, max(V, W));

if(minUVW < 0.0f && maxUVW > 0.0f) return false;

/* ... */
*isect_u = U * inv_den;
*isect_v = V * inv_den;
*isect_t = T * inv_den;
```

```
avx512f P_512(ray_P.x, ray_P.y, ray_P.z, ray_P.w);
avx512f dir_512(ray_dir.x, ray_dir.y, ray_dir.z, ray_dir.w);

avx512f tri_abc_512 = loadu(float_verts);
avx512f tri_abc_512_shuff = shuffle4<3,1,0,2>(tri_abc_512);

/* Calculate vertices relative to ray origin. */
avx512f v012_512 = tri_abc_512_shuff - P_512;
avx512f v012_512_shuff = shuffle4<3,1,0,2>(v012_512);

/* Calculate triangle edges. */
avx512f e012_512_min = v012_512_shuff - v012_512;
avx512f e012_512_plus = v012_512_shuff + v012_512;

/* Perform edge tests. */
avx512f cross_512 = lcross_xyz(e012_512_plus, e012_512_min);
const avx512f UVW = ldot3_xyz(cross_512, dir_512);
const float minUVW = reduce_min(UVW);
const float maxUVW = reduce_max(UVW);

if (minUVW < 0.0f && maxUVW > 0.0f) return false;

/* ... */
```

Blender & Embree – Data Preparation

```
// bvh_embree.h
class BVHEmbree : public BVH
{
  RTCScene scene;
  RTCDevice device;
  //...
  BVHEmbree(/*...*/) : BVH(/*..*/) { device = rtcNewDevice("verbose=1"); }
  virtual ~BVHEmbree() { rtcDeleteScene(scene); rtcDeleteDevice(device); }
} };
```

```
// bvh_embree.cpp
void BVHEmbree::build(/*...*/) {
    // ...
    RTCSceneFlags flags = RTC_SCENE_HIGH_QUALITY | /*...*/ ;
    scene = rtcDeviceNewScene(device, flags, RTC_INTERSECT1);
    foreach(Object *ob, objects) { add_object(ob, i); ++i; // ... }
    rtcCommit(scene);
}

void BVHEmbree::add_instance(Object *ob, int i) {
    BVHEmbree *instance_bvh = (BVHEmbree*) (ob->mesh->bvh);
    int geom_id = rtcNewInstance3(scene, instance_bvh->scene,
num_motion_steps, i*2);
    int geom_id_device = 0;
    //...
    rtcSetTransform2(scene, geom_id, …);
    rtcSetUserData(scene, geom_id, (void*) instance_bvh->scene);
    rtcSetMask(scene, geom_id, ob->visibility);
}
```

```
// bvh_embree.cpp
void BVHEmbree::add_object(Object *ob, int i) {
    Mesh *mesh = ob->mesh;
    int geom_id = 0;
    int geom_id_device = 0;
     //...
    add_triangles(mesh, i, geom_id, geom_id_device);
    rtcSetUserData(scene, geom_id, (void*) prim_offset);
    rtcSetOcclusionFilterFunction(scene, geom_id, rtc_filter_func);
    rtcSetMask(scene, geom_id, ob->visibility);
}


void BVHEmbree::add_triangles(Mesh *mesh, int i, int &geom_id, int &geom_id_device) {
    //...
    geom_id = rtcNewTriangleMesh2(scene, RTC_GEOMETRY_DEFORMABLE,
num_triangles, num_verts, num_motion_steps, i*2);

    void* raw_buffer = rtcMapBuffer(scene, geom_id, RTC_INDEX_BUFFER);
    unsigned *rtc_indices = (unsigned*) raw_buffer;
    for (size_t j = 0; j < num_triangles; j++) {
        Mesh::Triangle t = mesh->get_triangle(j);
        rtc_indices[j * 3] = t.v[0];
        //...
    }
    rtcUnmapBuffer(scene, geom_id, RTC_INDEX_BUFFER);
    update_tri_vertex_buffer(geom_id, geom_id_device, mesh);
    //...
}
```

Our modifications is based on code from Stefan Werner:
https://github.com/tangent-animation/blender278/tree/master_cycles-embree

Blender & Embree & MPI – Data Preparation

```
// bvh_embree.h
class BVHEmbree : public BVH
{
  RTCScene scene;
  RTCDevice device;
  //...
  BVHEmbree(/*...*/) : BVH(/*..*/) { devi MPI NewDevice("verbose=1"); }
  virtual ~BVHEmbre MPI DeleteScene(sce MPI DeleteDevice(device); }
} };
```

```
// bvh_embree.cpp
void BVHEmbree::build(/*...*/) {
   // ...
   RTCSceneFlags flags = RTC_SCENE_HIGH_QUALITY | /*...*/ ;
   sce MPI DeviceNewScene(device, flags, RTC_INTERSECT1);
   foreach(Object *ob, objects) { add_object(ob, i); ++i; // ... }
   rtcCommit(scene);
}
```

```
void BVHEmbree::add_instance(Object *ob, int i) {
   BVHEmbree *instance_bvh = (BVHEmbree*) (ob->mesh->bvh);
   int geom_ MPI NewInstance3(scene, instance_bvh->scene,
num_motion_steps, i*2);
   int geom_id_device = 0;
   //
MPI SetTransform2(scene, geom_id, …);
   SetUserData(scene, geom_id, (void*) instance_bvh->scene);
   SetMask(scene, geom_id, ob->visibility);
}
```

```
// bvh_embree.cpp
void BVHEmbree::add_object(Object *ob, int i) {
   Mesh *mesh = ob->mesh;
   int geom_id = 0;
   int geom_id_device = 0;
    //...
   add_triangles(mesh, i, geom_id, geom_id_device);
MPI SetUserData(scene, geom_id, (void*) prim_offset);
   SetOcclusionFilterFunction(scene, geom_id, rtc_filter_func);
   SetMask(scene, geom_id, ob->visibility);
}
```

Each architecture can have a different type of BVH

```
void BVHEmbree::add_triangles(Mesh *mesh, int i, int &geom_id, int &geom_id_device) {
   //...
   geom_ MPI NewTriangleMesh2(scene, RTC_GEOMETRY_DEFORMABLE,
num_triangles, num_verts, num_motion_steps, i*2);

   void* raw_buffer = rtcMapBuffer(scene, geom_id, RTC_INDEX_BUFFER);
   unsigned *rtc_indices = (unsigned*) raw_buffer;
   for (size_t j = 0; j < num_triangles; j++) {
      Mesh::Triangle t = mesh->get_triangle(j);
      rtc_indices[j*3] = t.v[0];
      //...
   }
MPI
   rtcUnmapBuffer(scene, geom_id, RTC_INDEX_BUFFER);
   update_tri_vertex_buffer(geom_id, geom_id_device, mesh);
   //...
}
```

Our modifications is based on code from Stefan Werner:
https://github.com/tangent-animation/blender278/tree/master_cycles-embree

Blender & Embree - Rendering

```
// bvh.h
bool scene_intersect(KernelGlobals *kg, const Ray ray, const uint visibility,
Intersection *isect, uint *lcg_state, float difl, float extmax) {
  if(kernel_data.bvh.scene) {
    isect->t = ray.t;
    CCLRay rtc_ray(ray, kg, visibility, CCLRay::RAY_REGULAR);
    rtcIntersect(kernel_data.bvh.scene, rtc_ray);
    if(rtc_ray.geomID != RTC_INVALID_GEOMETRY_ID && rtc_ray.primID !=
RTC_INVALID_GEOMETRY_ID) {
      rtc_ray.isect_to_ccl(isect);
      return true;
  }
  return false; } // … }


// bvh_embree_traversal.h
struct CCLRay : public RTCRay {
  typedef enum {
    RAY_REGULAR = 0, RAY_SHADOW_ALL = 1, RAY_SSS = 2,
    RAY_VOLUME_ALL = 3,
  } RayType;
  // cycles extensions:
  ccl::KernelGlobals *kg;
  RayType type;
  // for shadow rays
  ccl::Intersection *isect_s;
  int max_hits;
  int num_hits;
// … }
```

```
// bvh_embree_traversal.h
CCLRay::CCLRay(const ccl::Ray& ray, ccl::KernelGlobals *kg_, const ccl::uint visibility,
RayType type_) {
  org[0] = ray.P.x; org[1] = ray.P.y; org[2] = ray.P.z;
  dir[0] = ray.D.x; dir[1] = ray.D.y; dir[2] = ray.D.z;
  tnear = 0.0f; tfar = ray.t; time = ray.time; mask = visibility;
  geomID = primID = instID = RTC_INVALID_GEOMETRY_ID;
  // …
}


void CCLRay::isect_to_ccl(ccl::Intersection *isect) {
  const bool is_hair = geomID & 1;
  isect->u = 1.0f - v - u;
  isect->v = u;
  isect->t = tfar;
  if(instID != RTC_INVALID_GEOMETRY_ID) {
    RTCScene inst_scene = (RTCScene) rtcGetUserData(kernel_data.bvh.scene, instID);
    isect->prim = primID + (intptr_t) rtcGetUserData(inst_scene, geomID) +
kernel_tex_fetch(__object_node, instID/2);
    isect->object = instID/2;
} else {
  isect->prim = primID + (intptr_t) rtcGetUserData(kernel_data.bvh.scene, geomID);
  isect->object = OBJECT_NONE;
}
  isect->type = kernel_tex_fetch(__prim_type, isect->prim);
} };
```

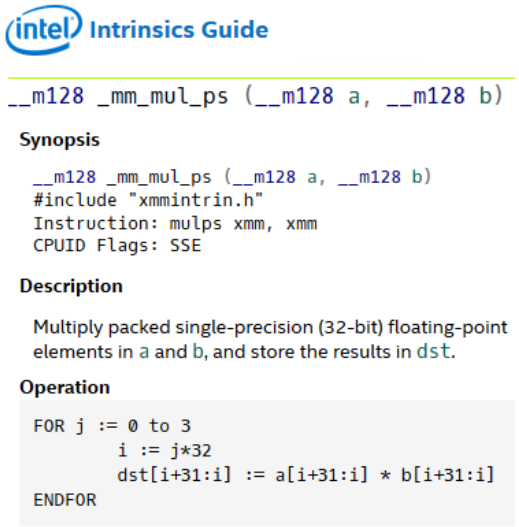
Our modifications is based on code from Stefan Werner:
https://github.com/tangent-animation/blender278/tree/master_cycles-embree

Embree & KNC

- Embree versions: Embree v3.0.0 (beta), Embree v2.17.2 (release), Embree v2.9.0 (KNC)
- SSE emulator

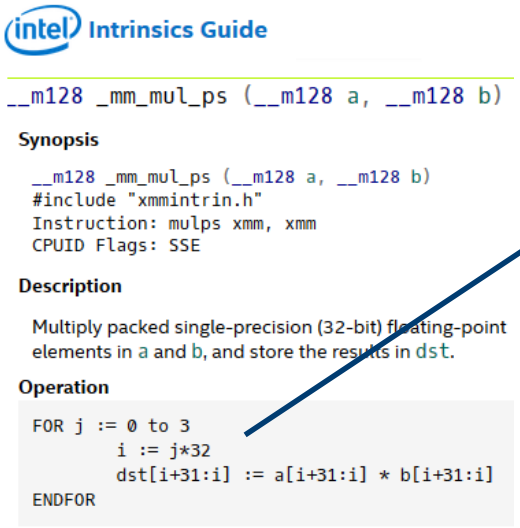
Embree & KNC

- Embree versions: Embree v3.0.0 (beta), Embree v2.17.2 (release), Embree v2.9.0 (KNC)
- SSE emulator



Embree & KNC

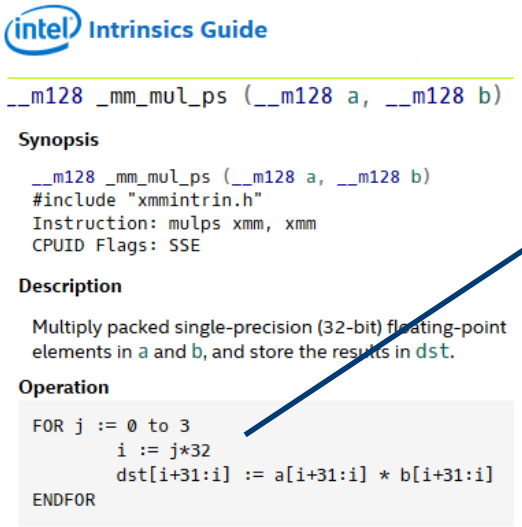
- Embree versions: Embree v3.0.0 (beta), Embree v2.17.2 (release), Embree v2.9.0 (KNC)
- SSE emulator




```
#define _MM_mul_ps _simd_mul_ps

//Multiply packed single-precision (32-bit) …
INTRINSICS_EMUL_INLINE
__simd128 _simd_mul_ps (__simd128 a, __simd128 b)
{
    INTRINSICS_EMUL_ASSUME_ALIGNED(a);
    INTRINSICS_EMUL_ASSUME_ALIGNED(b);

    __simd128 dst;
    for (int j = 0; j < 4; j++) {
        dst.f[j] = a.f[j] * b.f[j];
    }

#ifdef SIMD_DEBUG
    __m128 __a;
    memcpy(&__a, a, 16);
    __m128 __b;
    memcpy(&__b, b, 16);
    __m128 __dst = _mm_mul_ps(__a, __b);
    if (memcmp(&dst, &__dst, 16) != 0)
        printf("_mm_mul_ps:%d\n",
       (memcmp(&dst, &__dst, 16) != 0) ? 0 : 1);
#endif
    return dst;
}
```

Embree & KNC

- Embree versions: Embree v3.0.0 (beta), Embree v2.17.2 (release), Embree v2.9.0 (KNC)
- SSE emulator




```
#define _MM_mul_ps _simd_mul_ps

//Multiply packed single-precision (32-bit) …
INTRINSICS_EMUL_INLINE
__simd128 _simd_mul_ps (__simd128 a, __simd128 b)
{
    INTRINSICS_EMUL_ASSUME_ALIGNED(a);
    INTRINSICS_EMUL_ASSUME_ALIGNED(b);

    __simd128 dst;
    for (int j = 0; j < 4; j++) {
        dst.f[j] = a.f[j] * b.f[j];
    }

#ifdef SIMD_DEBUG
    __m128 __a;
    memcpy(&__a, a, 16);
    __m128 __b;
    memcpy(&__b, b, 16);
    __m128 __dst = _mm_mul_ps(__a, __b);
    if (memcmp(&dst, &__dst, 16) != 0)
        printf("_mm_mul_ps:%d\n",
       (memcmp(&dst, &__dst, 16) != 0) ? 0 : 1);
#endif
    return dst;
}
```

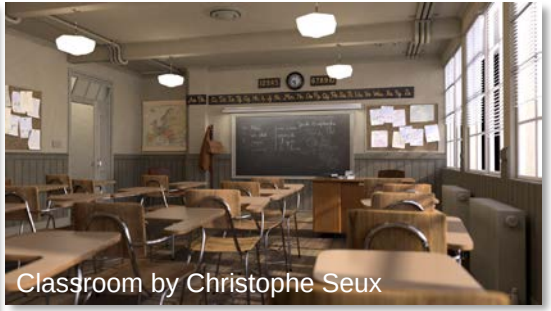
```
struct __simd128 {
    union {
        __m512 m;
        int8_t c[64];
        int16_t s[32];
        float f[16];
        int32_t i[16];
        double d[8];
        int64_t l[8];
    };
    __simd128() {};
    __simd128(const __m512 &m_):
                        m(m_) {}
} __attribute__((aligned(64)));
```

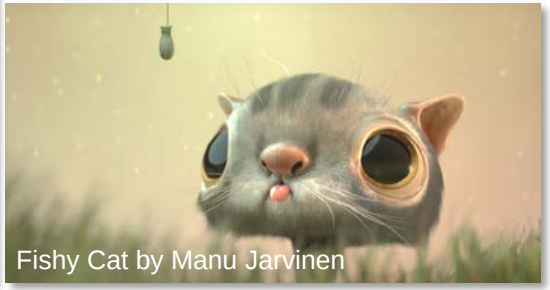
```
#ifdef SIMD_AVX512
    return _mm512_mul_ps(a.m, b.m);
#endif
```

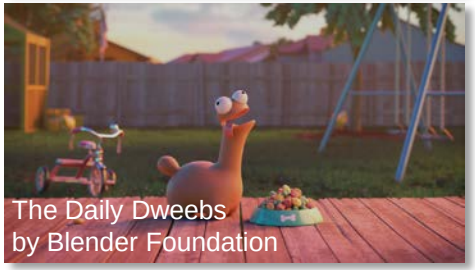
Benchmarks

- Salomon IT4Innovations
  - Intel® Xeon™ E5-2680v3 (Haswell)
  - Intel® Xeon Phi™ 7120P (KNC)
  - NVIDIA® GeForce® GTX TITAN X
- Marconi CINECA
  - Intel® Xeon Phi™ 7250 (KNL)
  - Intel® Xeon 8160 (SKL)


Crown by Martin Lubich

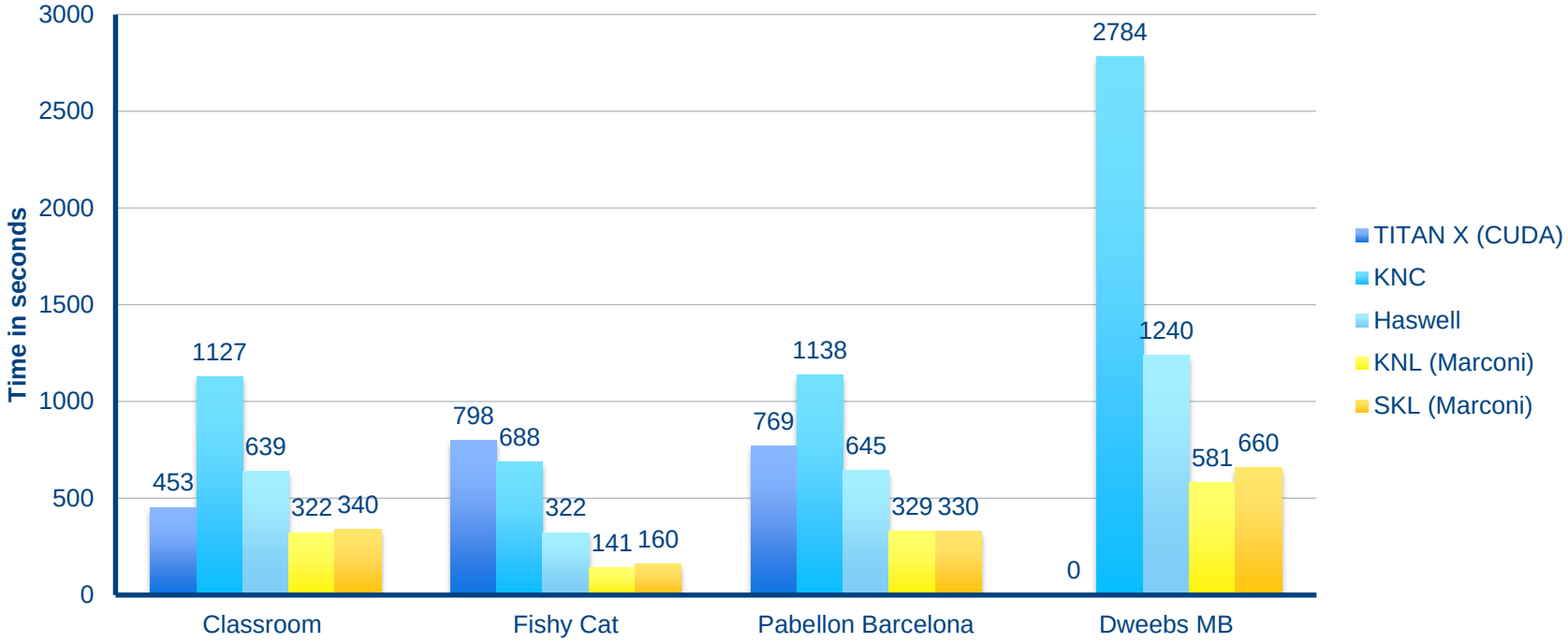

Classroom by Christophe Seux


Fishy Cat by Manu Jarvinen


The Daily Dweebs
by Blender Foundation

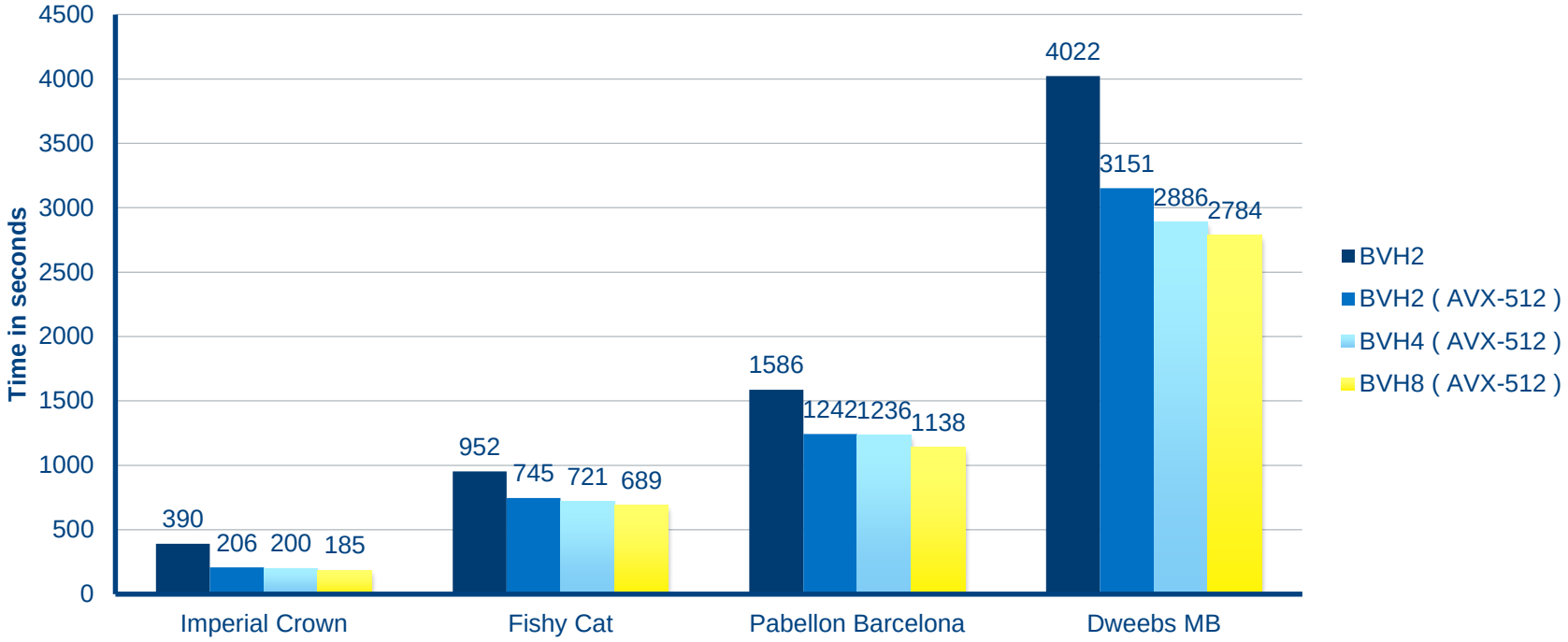

Pabellon Barcelona by Claudio Andres

Performance comparison - Rendering



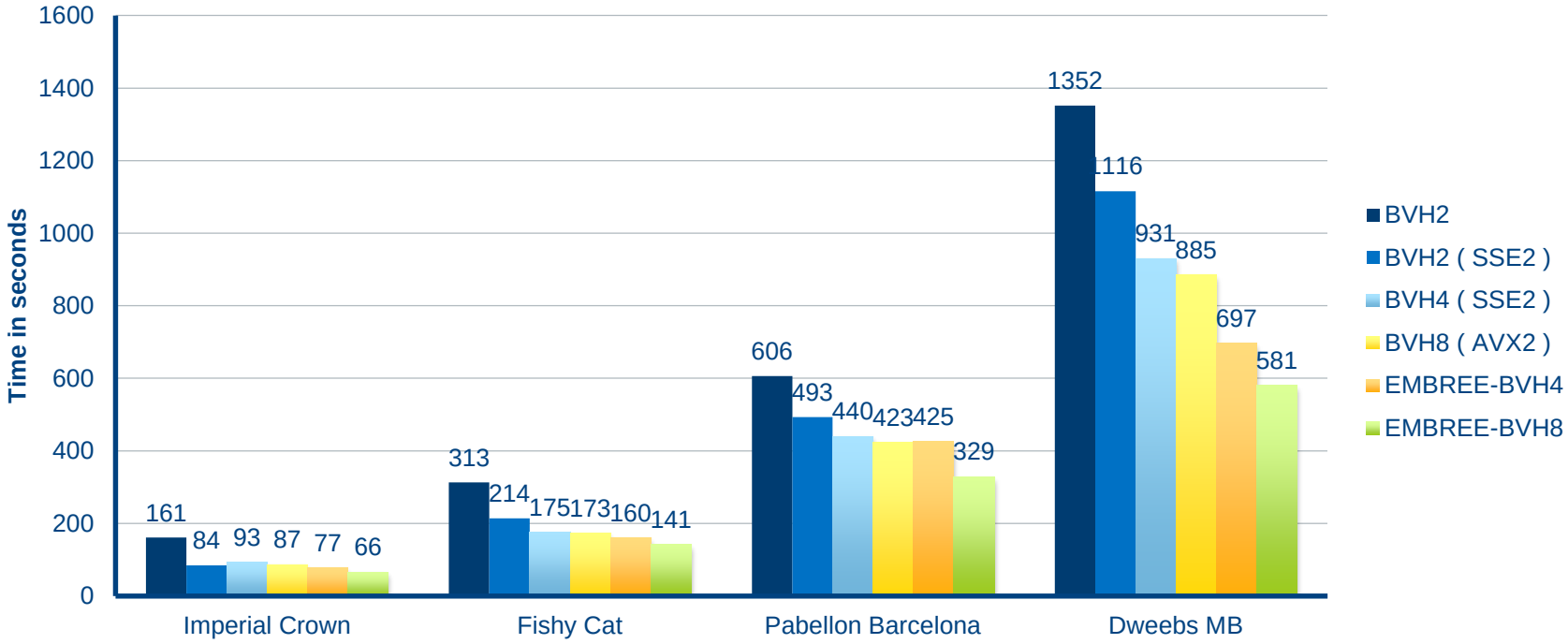
OpenMP threads: KNC 244x, Haswell 12x, KNL 272x, SKX 24x. KNC uses BVH8. EMBREE-BVH8 is used for Haswell, KNL and Skylake.

Performance comparison of BVH - KNC

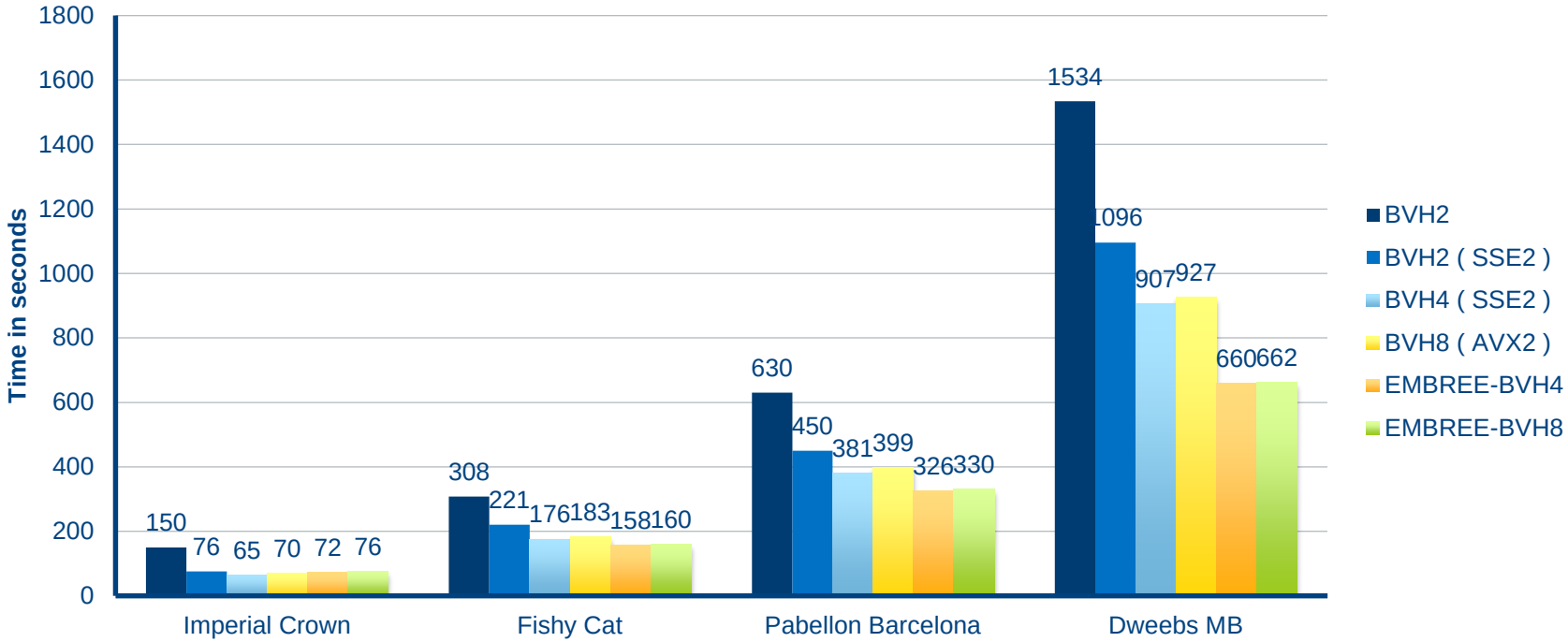


BVH2, BVH4 – ssef was replaced by avx512f. BVH8 – avxf was replaced by avx512f. BVH2, BVH4, BVH8 contain Ray Triangle Intersection with avx512f.

Performance comparison of BVH – KNL (Marconi)



Performance comparison of BVH – SKL (Marconi)

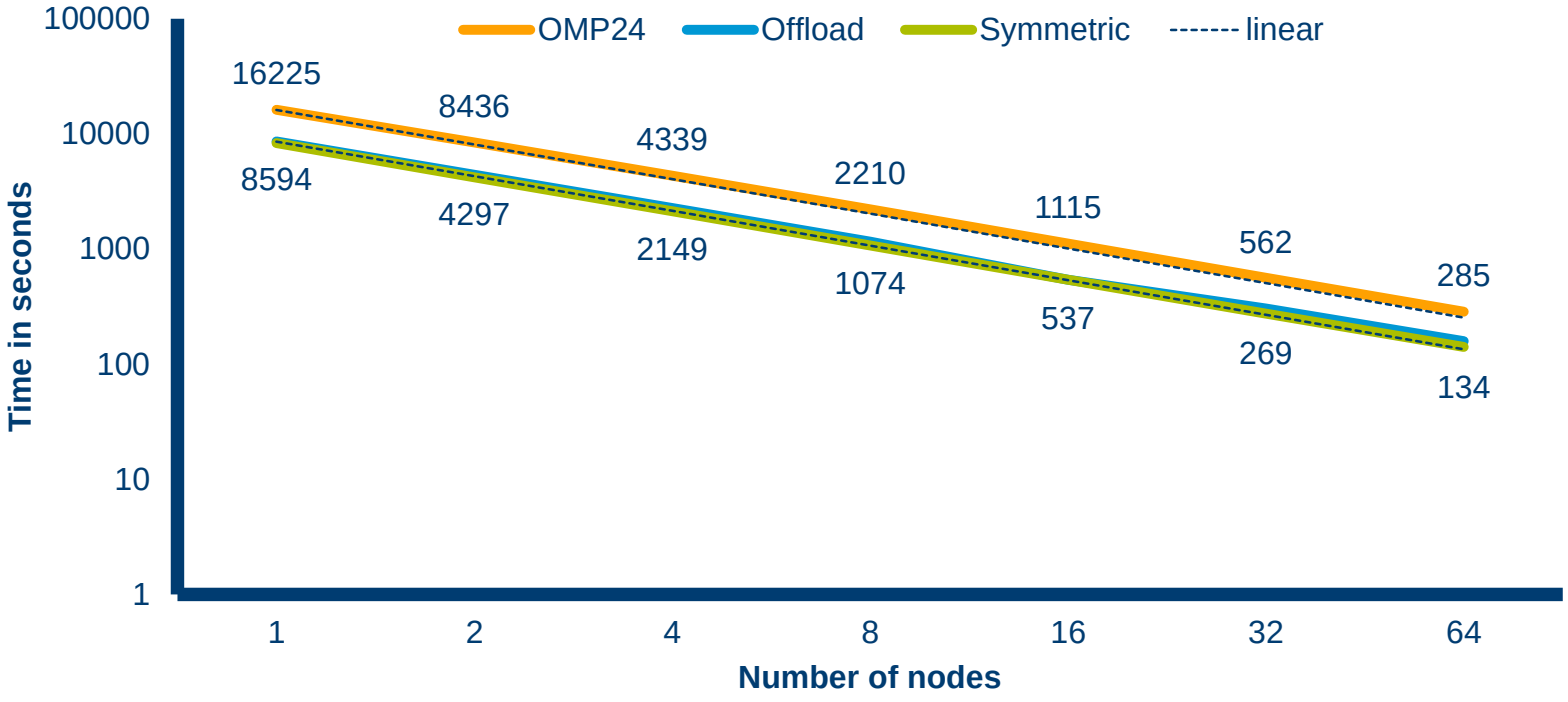


Benchmark Worm: Strong Scalability MPI Test (offline)

- **The benchmark** was run on 64 computing nodes of the Salomon supercomputer equipped with two Intel® Xeon™ E5-2680v3 CPUs and two Intel® Xeon Phi™ 7120P.
- **Worm scene** has 13.2 million triangles.
- Resolution: 4096x2048, Samples: 1024

Cosmos Laundromat - First Cycle

Benchmark Worm: Strong Scalability MPI Test (offline)

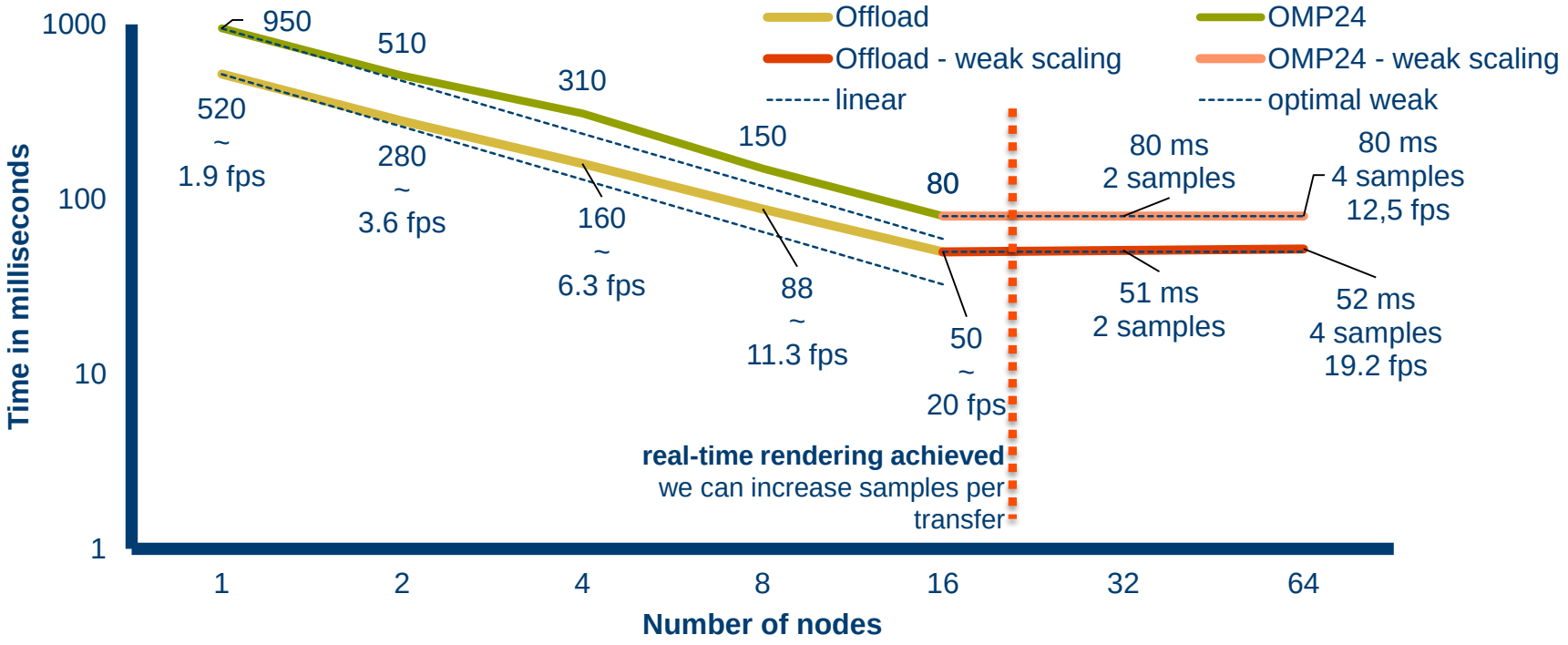


Benchmark Tatra T87: Strong Scalability MPI Test (interactive)

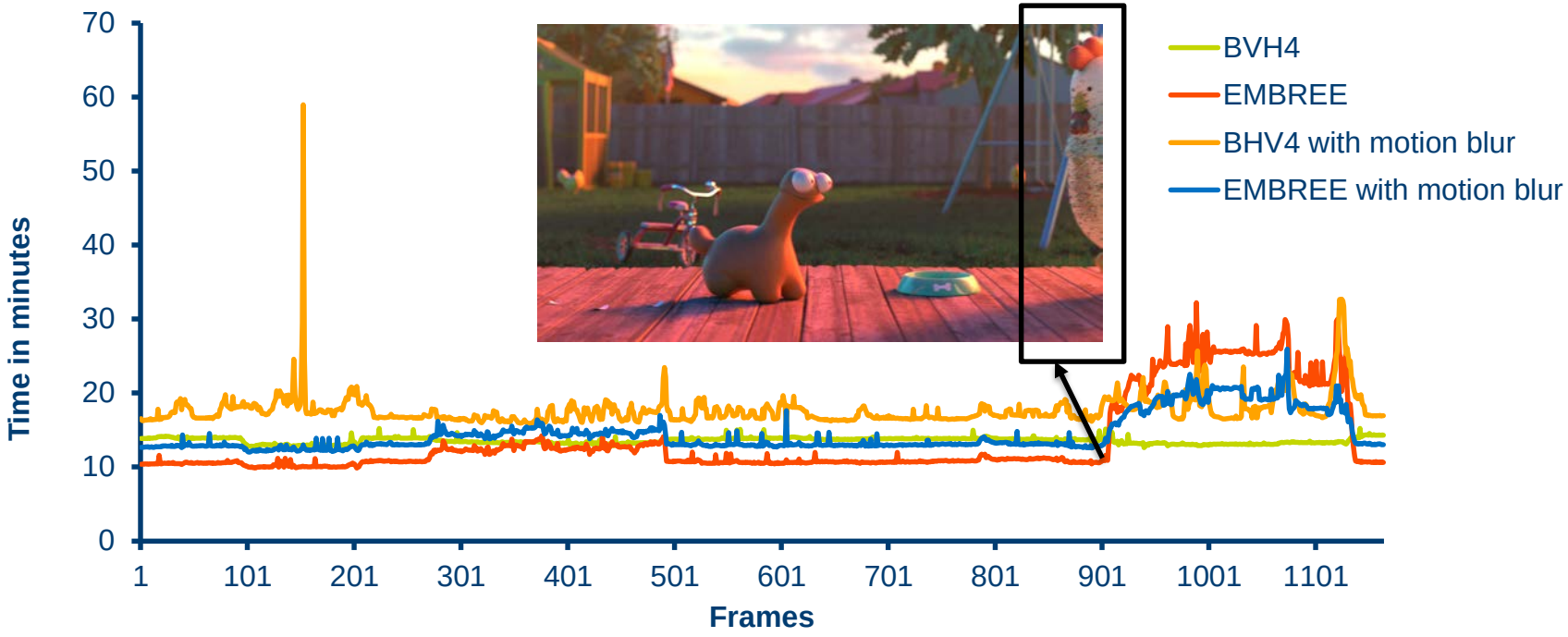
- **The benchmark** was run on 64 computing nodes of the Salomon supercomputer equipped with two Intel® Xeon™ E5-2680v3 CPUs
- **Tatra T87** has 1.2 million triangles and uses the HDRI lighting.
- Resolution: 1920x1080, Samples: 1

Tatra T87 by David Cloete

Benchmark Tatra T87: Strong Scalability MPI Test (interactive)



The Daily Dweebs: Performance comparison of w/o Motion Blur on 2x Intel® Xeon™ E5-2680v3 (Haswell)



https://cloud.blender.org/blog/the-dweebs-a-mini-open-movie-project

References

- Jaros, M., Riha, L., Strakos, P. , et al.: Rendering in blender cycles using MPI and intel® xeon phi™. Paper presented at the ACM International Conference Proceeding Series, Part F130523 doi:10.1145/3110224.3110236, 2017
- Jaros, M., Riha, L., Strakos, P. , et al.: Acceleration of Blender Cycles Path-Tracing Engine Using Intel Many Integrated Core Architecture, CISIM 2015, Warsaw, Poland, p. 86-97, 2015
- http://blender.it4i.cz

Acknowledgement

- *This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project „IT4Innovations National Supercomputing Center – LM2015070“.*