Rendering in Blender Cycles Using Intel® Xeon Phi™ (Code Named Knights Landing)

IT4Innovations National Supercomputing Centre, Ostrava, Czech Republic

Milan Jaroš, Lubomír Říha
Outline

- Blender Cycles introduction
- Algorithm for image rendering
- CPU rendering in original code
- New OMP Device for rendering
- New MPI Device for rendering
- Embree integration
- Benchmarks
Blender Cycles

- **Blender** is an open source 3D creation suite. It has two render engines: Blender Internal and Cycles.

- **Cycles** is a raytracing based render engine with support for interactive rendering, shading node system, and texture workflow.
Cycles is internal plugin (Extending Python w. C++)

Blender start – register C++ modules into Python

1.) Write a C++ function for rendering

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

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();
    //...
    Py_RETURN_NONE;
}
```

3.) Include the Python module in the Blender GUI

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

4.) Register this function within a module’s symbol table

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

5.) Write an init function for the module

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

Rendering Equation

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) (\omega_i \cdot n) \ d\omega_i
\]

- \( \omega_o \) is direction of outgoing ray
- \( \omega_i \) is direction of incoming ray
- \( L_o \) is spectral radiance emitted by the source from point \( x \) in direction \( \omega_o \)
- \( L_e \) is emitted spectral radiance from point \( x \) in direction \( \omega_o \)
- \( \Omega \) is the unit hemisphere in direction of normal vector \( n \) with center in \( x \), over which we integrate
- \( L_i \) is spectral radiance coming inside to \( x \) in direction \( \omega_i \),
- \( f_r(x, \omega_i, \omega_o) \) is distribution function of the image (BRDF) in point \( x \) from direction \( \omega_i \) to direction \( \omega_o \),
- \( \omega_i \cdot n \) is angle between \( \omega_i \) and surface normal.
Path tracing

- For each pixel a ray is cast into a scene.
- A ray from a camera hits a glossy surface (0), then a diffuse surface (1), and it bounces into a random direction.
- The color of the ray is calculated depending on all materials of the surfaces.
- This process is repeated by the value of samples.
- The mean value of all samples is used for the color of the pixel.
Bounding Volume Hierarchy (BVH)

The raytracing acceleration structure used is a bounding volume hierarchy. The original code is based on an implementation from Nvidia (BHV2, BHV4) and Embree (Hair BVH builder - oriented SAH, QBVH traversal, BVH nodes intersection, Triangles intersection).

- **BVH2**
- **BVH4 (QBVH)**
- **BVH8 (by maxim_d33)**
- **EMBREE (by Stefan Werner)**
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 BHV tree whenever an object is moved around but the noise will clear up faster.
Original Blender parallelization

- POSIX Threads
- Communication via sockets

CPU rendering in original code (Pthreads)

- **KernelData**
  - cam, background, integrator (emission, bounces, sampler), ...
  - const_copy_to

- **KernelTextures**
  - bvh, objects, triangles, lights, particles, sobol_directions, texture_images, ...
  - tex_alloc
  - tex_free

- **CPUDevice**
  - decompose task to subtasks
  - thread_run

- **Pthreads**
  - MEM
  - CPU

- **ONE NODE**
  - mem_alloc
  - mem_free
  - mem_copy_from
  - mem_copy_to
CPU rendering in original code (Pthreads)

The decomposition of synthesized image with resolution $x(r) \times y(r)$ to tiles with size $x(t) \times y(t)$ by original implementation. The one tile is computed by one POSIX thread on one CPU core for $x(t) \times y(t)$ pixels. This is an example of CPU4.
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
- Intel® Xeon Phi™ with Offload concept (KNC)
- Intel® Xeon Phi™ with Symmetric mode (KNC, KNL)
- And their combinations
Native, Offload and Symmetric modes
Native mode, Offload mode and Symmetric mode

**Distributed rendering**
Symmetric mode (KNC, KNL)
- Blender
  - CPU
  - MPI
  - OpenMP
  - MIC0
  - MIC1
  - Blender client

**Single node rendering**
Offload mode (KNC)
- Blender
  - CPU
  - MIC0
  - MIC1
  - OpenMP+Offload

Native mode (KNL)
- Blender
  - MIC0

**Blender**
- Client

---

**OpenMP**
- MIC0
- MIC1

**MPI**

---

**General**
- OpenCL:
  - Clip Alpha: 0.004
- Virtual Pixel Mode:
  - Native
- Frame Server Port: 8080
- Console Scrollback: 256
- Selection:
  - Automatic
- Anisotropic Filtering: 2x
- Window Draw Method:
  - Automatic
  - No Multisample
  - Region Overlap

**ScreenCap**
- Compute Device: 10
- CPU
  - Device No: 0, Device Name: mic0
  - Device No: 0, Device Name: mic1
  - Device No: 1, Device Name: mic1
- CPU+Device No: 0, Device Name: mic0
- CPU+Device No: 0, Device Name: mic1
- CPU+Device No: 1, Device Name: mic1
- Time Out: 120
- Collection Rate: 60
Parallelization using OpenMP

OMPDevice

KernelData
- cam, background, integrator
- (emission, bounces, sampler), ...

const_copy_to

KernelTextures
- bvh, objects, triangles, lights,
- particles, sobol_directions,
- texture_images, ...

tex_alloc

tex_free

OMPDevice

buffer, rng_state

mem_alloc

mem_free

mem_copy_from

mem_copy_to

ONE NODE

OpenMP+Offload(KNC)

OpenMP (KNL)

OpenMP (CPU)

OMPDevice

decompose task to subtasks

Pthreads

tile

thread_run

Parallelization using OpenMP
Rendering using OpenMP and MPI
Rendering using OpenMP and MPI

**KernelData**
- cam, background, integrator (emission, bounces, sampler), ...
- `const_copy_to` (1)

**KernelTextures**
- bvh, objects, triangles, lights, particles, sobol_directions, texture_images, ...
- `tex_alloc` (2)
- `tex_free`

**buffer, rng_state**
- `mem_alloc`
- `mem_free`
- `mem_copy_to` (3)

---

**OpenMP Render loop**

**7D Enhanced hypercube Infiniband network**

**Send **KernelData**(ex. camera properties) to all nodes with Bcast**

**Send **KernelTextures**(ex. triangles) to all nodes with Bcast**

**Send the information about the size of buffer (rendered pixels) and size of rng_state (random number generator state) to all nodes with Bcast**

**Send the initial values of rng_state to all nodes with Bcast**

**Start rendering with Bcast message**

**Read the current results from node and send the buffer to root with Gather**

**Send new jobs to clients with Scatter**

**View results in Blender**
Blender & Embree

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

---

```cpp
// bvh_embree.h
class BVHEmbree : public BVH
{
  RTCScene scene;
  RTCDevice device;

  BVHEmbree() { device = rtcNewDevice("verbose=1"); }
  virtual ~BVHEmbree() {
    rtcDeleteScene(scene);
    rtcDeleteDevice(device);
  }

  void add_triangles(Mesh *mesh, int i, int &geom_id, int &geom_id_device)
  {
    RTCindices tri_indices = rtcNewTriangleMesh2(scene, RTC_GEOMETRY_DEFORMABLE, num_triangles, num_verts, num_motion_steps, i*2);
    rtcSetTransform2(scene, geom_id, RTC_MATRIX_ROW_MAJOR, (const float*) &ob->tfm);
    rtcSetUserData(scene, geom_id_device, (void*) instance_bvh->scene);
    rtcSetMask(scene, geom_id, ob->visibility);
  }

  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, RTC_MATRIX_ROW_MAJOR, (const float*) &ob->tfm);
    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_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);

    rtcSetTransform2(scene, geom_id, RTC_MATRIX_ROW_MAJOR, (const float*) &ob->tfm);
    rtcSetUserData(scene, geom_id, (void*) instance_bvh->scene);
    rtcSetMask(scene, geom_id, ob->visibility);
  }
```

bool scene_intersect(KernelGlobals *kg, const Ray ray, const uint visibility, Intersection *isect, uint *lgc_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; // ...
    }
    return false; // ...
}

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(primID != RTC_INVALID_GEOMETRY_ID) {
        RTCScene inst_scene = (RTCScene) rtcGetUserData(kernel_data.bvh.scene, instID);
        isect->object = instID/2;
    } else {
        isect->object = OBJECT_NONE;
    }
    isect->type = kernel_tex_fetch(__prim_type, isect->prim);
}
Benchmarks

- **IT4Innovations**
  - Salomon
    - Intel® Xeon™ E5-2680v3 (Haswell)
    - Intel® Xeon Phi™ 7120P (KNC) (for MPI tests)
    - NVIDIA® GeForce® GTX TITAN X
  - Workstation
    - NVIDIA® GeForce® GTX 970

- **Marconi-A2 CINECA**
  - Intel® Xeon Phi™ 7250 (KNL)
The preprocessing for GTX 970 and for TITAN X is on CPU. This time includes the data transfers. EMBREE-BVH8 is used for Haswell and KNL.
Performance comparison - Rendering

GTX 970 and TITAN X use CUDA technology. KNL uses 4x Pthreads and 68x OpenMP threads. EMBREE-BVH8 is used for Haswell and KNL.
Performance comparison of building of BVH

Classroom | Fishy Cat | Koro | Pabellon Barcelona
---|---|---|---
Time in seconds

- BHV2 - 2x Haswell
- BHV4 - 2x Haswell
- BHV8 - 2x Haswell
- EMBREE - 2x Haswell
- BHV2 - KNL
- BHV4 - KNL
- BHV8 - KNL
- EMBREE - KNL
Performance comparison of BVH - Rendering

![Bar chart showing time in seconds for different BVH and EMBREE methods on various datasets.](chart)

- **Classroom**: BHV2 - 2x Haswell (413), BHV4 - 2x Haswell (319), BHV8 - 2x Haswell (376), EMBREE - 2x Haswell (349), BHV2 - KNL (315), BHV4 - KNL (343), BHV8 - KNL (382), EMBREE - KNL (413)
- **Fishy Cat**: BHV2 - 2x Haswell (231), BHV4 - 2x Haswell (195), BHV8 - 2x Haswell (213), EMBREE - 2x Haswell (177), BHV2 - KNL (217), BHV4 - KNL (177), BHV8 - KNL (295), EMBREE - KNL (245)
- **Koro**: BHV2 - 2x Haswell (482), BHV4 - 2x Haswell (401), BHV8 - 2x Haswell (416), EMBREE - 2x Haswell (375), BHV2 - KNL (321), BHV4 - KNL (314), BHV8 - KNL (301), EMBREE - KNL (301)
- **Pabellon Barcelona**: BHV2 - 2x Haswell (379), BHV4 - 2x Haswell (376), BHV8 - 2x Haswell (349), EMBREE - 2x Haswell (247), BHV2 - KNL (217), BHV4 - KNL (177), BHV8 - KNL (122), EMBREE - KNL (122)
The Daily Dweebs: Performance comparison of w/o Motion Blur on 2x Intel® Xeon™ E5-2680v3 (Haswell)
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
Benchmark Worm: Strong Scalability MPI Test (offline)

- OMP24
- Offload
- Symmetric
- linear

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16225</td>
</tr>
<tr>
<td>2</td>
<td>8436</td>
</tr>
<tr>
<td>4</td>
<td>4339</td>
</tr>
<tr>
<td>8</td>
<td>2210</td>
</tr>
<tr>
<td>16</td>
<td>1115</td>
</tr>
<tr>
<td>32</td>
<td>562</td>
</tr>
<tr>
<td>64</td>
<td>285</td>
</tr>
</tbody>
</table>

- 1 2 4 8 16 32 64
- 1 10 100 1000 10000 100000

Intel
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)

Time in milliseconds vs. Number of nodes graph with data points:
- 1 node: 950 ms, 1 sample
- 2 nodes: 510 ms, 1 sample
- 4 nodes: 310 ms, 1 sample
- 8 nodes: 150 ms, 2 samples
- 16 nodes: 80 ms, 2 samples
- 32 nodes: 51 ms, 4 samples
- 64 nodes: 52 ms, 4 samples

Lines indicate:
- Offload
- Offload - weak scaling
- Linear
- Optimal weak

Real-time rendering achieved: we can increase samples per transfer.

FPS values:
- 950 ms ~ 1.9 fps
- 510 ms ~ 3.6 fps
- 310 ms ~ 6.3 fps
- 150 ms ~ 11.3 fps
- 80 ms ~ 20 fps

Overall performance highlighted with Intel branding.
Agent 327: Operation Barbershop
Agent 327: Operation Barbershop
References

- Frederik Steinmetz, Gottfriend Hofmann: The Cycles Encyclopedia
- https://wiki.blender.org
- https://cloud.blender.org/blog/cycles-turbocharged-how-we-made-rendering-10x-faster
- https://www.youtube.com/watch?v=mN0zPOpADL4 (Agent327)
- https://www.youtube.com/watch?v=RJnKaAtBPhA (The Daily Dweebs)