Embree 2.17.2

Installation and verification

Luís Paulo Santos, February, 2018

Throughout this course we will use Embree as the ray tracing engine. Embree is a collection of optimized ray tracing kernels, developed by Intel, with support for vector processing such as SSE, AVX, AVX2 and AVX-512. Embree selects at runtime ray traversal and BVH build algorithms that best match the instruction set of your CPU.

This tutorial is just a brief summary of the extended information you can find at:

https://embree.github.io/

Installation

During the semester we will be using Embree 2.17.2, since version 3.0.0 is still a beta release.

Follow instructions available at https://embree.github.io/downloads.html for your operating system.

Even though pre-built versions of the library can be found and used I suggest you build Embree from the sources. We will need to have a look and eventually modify the source code of the tutorials, and this is one way of getting access to it.

You will need Intel Thread Building Blocks. It might either be available on your system or you might have to download it from https://www.threadingbuildingblocks.org/download

Also install CMake (version 2.8.11 or higher).

Note that if you have opted to not use Intel ISPC then you have to uncheck "EMBREE_ISPC_SUPPORT" in CMake.

After configuring and generating with CMake you should be able to build Embree using your compiler and, eventually, associated IDE.

After the build you should find all the Embree libraries and tutorials applications on your selected target directory.

Verify your installation running for instance the path tracer tutorial:

- make sure, depending on the operating system, that the application has access to the Embree and tbb libraries (static or dynamic, depending on the build)
- make sure you identify the directory where the geometric models are made available (copy the models folder to the folder where the tutorials have been built, from the $EMBREE_SOURCES/tutorials/models folder)
- execute the path tracer (you can change the loaded model by writing: -c <model path>) – check for instructions at https://embree.github.io/tutorials.html)

Tutorial Code Walkthrough

Let's have a quick look at the viewer tutorial rendering code, in order to get a feeling of how a ray tracer rendering method can be organized.

NOTE: the code description done here is very high level and some of the modifications suggested are not generalizable for more complex shaders.

Open the `viewer_device.cpp` file and locate the method responsible for shading one pixel:

```
Vec3fa renderPixelStandard(float x, float y, const ISPCCamera& camera,
RayStats& stats)
```

Note that a primary ray is created that originates in the camera position and goes through the current pixel `(x,y)`:

```
/* initialize ray */
  RTCRay ray;
  ray.org = Vec3fa(camera.xfm.p);
  ray.dir = Vec3fa(normalize(x*camera.xfm.l.vx + y*camera.xfm.l.vy +
camera.xfm.l.vz));
  ray.tnear = 0.0f;
  ray.tfar = inf;
```

The ray is traced through the scene – `Embree` is informed that this ray is spatially coherent:

```
/* intersect ray with scene */
  RTCIntersectContext context;
  context.flags = g_iflags_coherent;
  rtcIntersect1Ex(g_scene,&context,ray);
```

If the ray doesn't intersect any geometry then its color is set to (0., 0., 0.):

```
/* shade background black */
  if (ray.geomID == RTC_INVALID_GEOMETRY_ID) {
    return Vec3fa(0.0f);
  }
```

Shading consists simply on assigning the material diffuse color to the ray result:

```
/* shade */
  if (g_ispc_scene->materials[materialID]->type == MATERIAL_OBJ) {
    ISPCOBJMaterial* material = (ISPCOBJMaterial*) g_ispc_scene-
>materials[materialID];
    color = Vec3fa(material->Kd);
  }

  return color*dot(neg(ray.dir),dg.Ns);
```

Depth Shading

Let's change the code such that the ray color is the object depth. Comment the lines shown in the last paragraph, which correspond to material shading.

Knowing that a ray has the attribute `tfar`, which is set to the depth of the first intersection change the code to:

```
        /* shade background black */
        if (ray.geomID == RTC_INVALID_GEOMETRY_ID) {
          return Vec3fa(0.0f);
        }
        // for depth shading
        else {
         return (Vec3fa(ray.tfar*((float)1.e-4)));
        }
```

Direct Lighting – no shadows

Knowing that the Cornel Box has one point light source (see file cornell_box.ecs) and that you can access the attributes `g_ispc_scene->numLights` and `g_ispc_scene->lights[i]` , include the code below to do direct light shading in your scene.

Do not forget to comment the results from the previous exercise!

```
  /* iterate over lights */
  if (g_ispc_scene->materials[materialID]->type == MATERIAL_OBJ) {
    ISPCOBJMaterial* material = (ISPCOBJMaterial*)g_ispc_scene->materials[materialID];
    for (size_t i = 0; i < g_ispc_scene->numLights; i++)
    {
        const Light* l = g_ispc_scene->lights[i];
        Light_SampleRes ls = l->sample(l, dg, RandomSampler_get2D(sampler));
        float cos_L_N = dot(ls.dir, dg.Ns);
        if (cos_L_N > 0.f) {
                color += ls.weight * material->Kd * dot(ls.dir, dg.Ns);
        }
    }
  }
  return color;
```

Direct Lighting

Now add shadows by shooting shadow rays!

```
// direct light with shadows
// iterate over lights
if (g_ispc_scene->materials[materialID]->type == MATERIAL_OBJ) {
  ISPCOBJMaterial* material = (ISPCOBJMaterial*)g_ispc_scene->materials[materialID];
  context.flags = g_iflags_incoherent;
  for (size_t i = 0; i < g_ispc_scene->numLights; i++)
  {
    const Light* l = g_ispc_scene->lights[i];
    Light_SampleRes ls = l->sample(l, dg, RandomSampler_get2D(sampler));
    float cos_L_N = dot(ls.dir, dg.Ns);
    if (cos_L_N > 0.f) {
      RTCRay shadow = RTCRay(dg.P, ls.dir, dg.tnear_eps, ls.dist, ray.time);
      shadow.transparency = Vec3fa(1.0f);
      rtcOccluded1Ex(g_scene, &context, shadow);
      if (max(max(shadow.transparency.x, shadow.transparency.y), shadow.transparency.z) > 0.0f){
          color += ls.weight * material->Kd * dot(ls.dir, dg.Ns) *shadow.transparency;
      }
    }
  }
}
return color;
```