Tutorial 3
OSPRay – Distributed Ray Tracing

Iluminação e Visualização II

Luís Paulo Santos, Maio 2021

Throughout this tutorial you will develop a distributed ray tracer.

Introduction

Distributed ray tracing is based on the idea of stochastically distributing the rays over the domain (or domains) that are being sampled. Typically rays are stochastically distributed over:

1. the light sources' areas, therefore enabling soft shadows (umbra + penumbra);
2. the pixel area, enabling image plane antialiasing;
3. the camera lens area, enabling depth of field;
4. the BRDF lobe, enabling less directional light transport phenomena such as glossy reflections;
5. time, enabling motion blur.

Throughout this tutorial we will explore points 1 and 4 (soft shadows and glossy reflections).

Edit the `CMakeLists.txt` file in `ospray-vi2/src/rtlibrary` and make sure that you use the `MyRenderer-T3.ih` and `MyRenderer-T3.ispc` files for this project:

Universidade do Minho

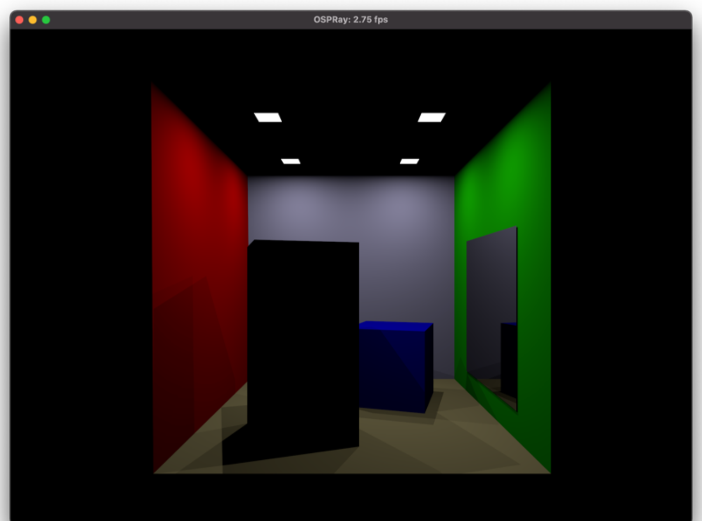
```
ispc_target_add_sources(ospray_module_rtlibrary

  ModuleInit.cpp

  render/MyRenderer.h
  render/MyRenderer.cpp
  render/MyRenderer.ih
  render/MyRenderer.ispc
  render/MyMaterial.h
  render/MyMaterial.cpp
  render/MyMaterial.ih
  render/MyMaterial.ispc
  render/MyRenderer-T3.ih
  render/MyRenderer-T3.ispc
)
```

Also make sure that in `MyRenderer.cpp` the `commit()` method is setting the lights for the Cornell box scene. Build the renderer and run it:

```
>./cornell_VI2
```



Note that the ambient light has been switched off. The renderer you are using is completely deterministic.

The shadow rays are always shot towards the center of the light sources. Therefore, they behave as if they were point light sources and there are **no soft shadows**. All shadows have hard, sharp edges.

Area Light Sources and Soft Shadows

Open the file `MyRenderer-T3.ispc` and locate the function `direct_illumination()`. This function shoots one shadow ray towards each light source. The code fragment below selects a point on the light source's area:

```
const vec2f s = make_vec2f(0.0f); // sample center of area lights
// stochastically sample the area lights
//const vec2f s = make_vec2f(frandom(rng),frandom(rng));

const Light_SampleRes light = l->sample(l, dg, s);
```

The `sample(l, dg, s)` function returns the sampled point (together with some additional data). The point to be sampled is selected according to the parameter `s`, which is a vector of 2 floats (each in the interval 0 …1). Since `s` is (0, 0) the area center is always selected.

Instead generate a pair of random numbers, by commenting the first line and uncommenting the line below (the function `frandom()` uses `ispc` pseudo random numbers generator to return a float in the interval 0 .. 1). The shadow rays will now be distributed over the light source area.

Build the renderer and render the Cornell box again. Comment on the shadows obtained now. What happens when you change the view point?

The fact that there are 4 small area light sources doesn't help in perceiving the soft shadows. Let's include a single wide area light source. Edit the `defineLightsCornell()` method in `MyRenderer.cpp`. Change the following line

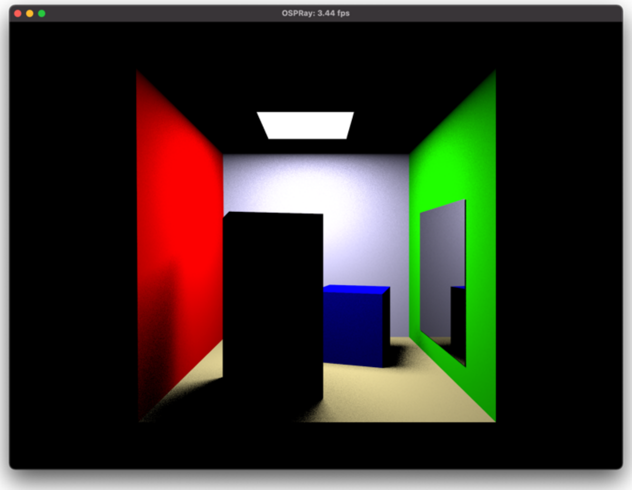
```
if (1) {  // 4 light sources
```

to

```
if (0) {  // 4 light sources
```

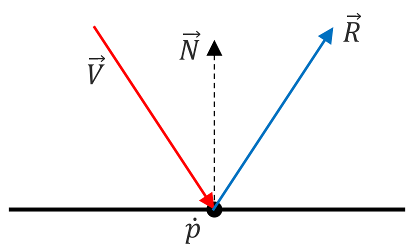
Build the renderer and render the Cornell box again. Can you see the soft shadows now?

Universidade do Minho



Glossy reflections

Our renderer only supports ideal specular reflections: to a given view direction $\vec{V}$, there is only and only one direction $\vec{R}$ from which radiance is reflected in a specular manner. The following figure illustrates this and Equation 1 presents the expression for indirect specular radiance on point $\dot{p}$:



Equation 1- specular reflected radiance

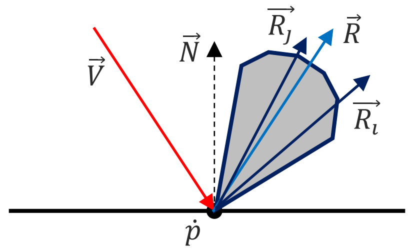
$$L_r(\dot{p}, \rightarrow \vec{V}) = L_i(\dot{p}, \leftarrow \vec{R}) * k_s * \cos(\vec{R}, \vec{N})$$

But real materials are not ideal specular reflectors. In fact, light is reflected along a glossy lobe centered on the ideal specular reflection direction. The Phong model allows us to simulate this effect according to Equation 2; the greater the value of n_s, the sharper the lobe, converging towards ideal specular.

Equation 2 - glossy reflected radiance

$$L_r(\dot{p}, \rightarrow \vec{V}) = L_i(\dot{p}, \leftarrow \vec{R_i}) * k_s * \cos(\vec{R_i}, \vec{N}) * \cos^{n_s}(\vec{R}, \vec{R_i})$$

Distributed ray tracing allows us to distribute the reflected rays across the glossy lobe, as depicted below.



In file `MyRenderer-T3.ispc` look for `secondary_classic()` . You will see that the direction of the reflected ray is computed as:

```
// compute reflection direction
vec3f wi = 0.f - ray.dir;
vec3f Rs = 2.f * dot (dg.Ns, wi) * dg.Ns - wi;
// set reflection ray
setRay(specRay, dg.P, Rs, dg.epsilon, inf, ray.time);
```

Change it such that it is stochastically selected according to a cosine lobe, where the cosine is to the power of n_s:

```
// compute reflection direction
vec3f wi = 0.f - ray.dir;
vec3f Rs = 2.f * dot (dg.Ns, wi) * dg.Ns - wi;

// sampling the BRDF glossy lobe
// get a random direction distributed over a cosine lobe to the power of N
// this direction is local, i.e. the lobe is centered around the Z axis
const vec2f s = make_vec2f(frandom(rng), frandom(rng));
const vec3f local_glossy_dir = powerCosineSampleHemisphere(Ns, s);
// rotate the local direction
// to a lobe centered around the ideal specular reflection direction
const vec3f glossy_dir = frame(Rs) * local_glossy_dir;

// set reflection ray
setRay(specRay, dg.P, glossy_dir, dg.epsilon, inf, ray.time);
```

Build the renderer and render the Cornell box.

There is a reason why you cannot see any glossy reflections. The only specular reflecting object in the scene is the mirror in the wall; its `Ns` parameter is very high (`100000`), therefore the reflection is very close to an ideal specular reflection. Open the file with the materials definition for the cornel box: `~/ospray-vi2/build/models/cornell_box_VI2/cornell_box_VI2.mtl`

You will find the `my_mirror` material defined as

```
newmtl my_mirror
Ka 0 0 0
```

```
Kd 0 0 0
Ks 0.85 0.85 0.85
Ns 100000
```

Change `Ns` to 100, save the file and render the scene again. Can you see the glossy reflections now?

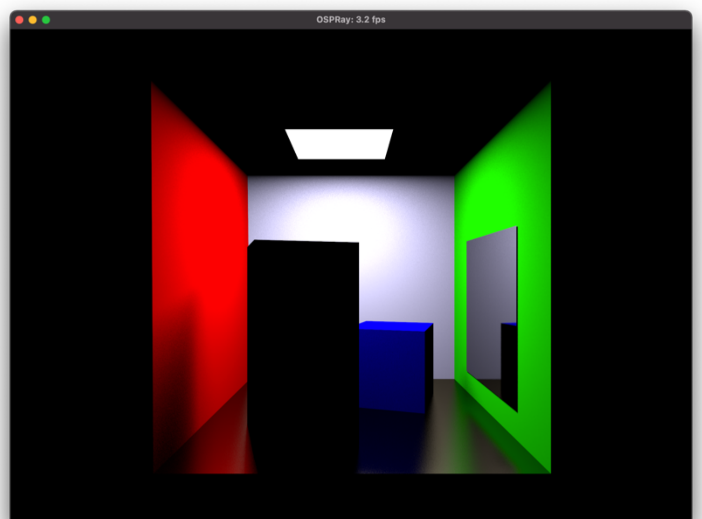
Revert the `my_mirror` material back to the original value. Let's play with the floor. It is made of `gold` and its definition is:

```
newmtl gold
Ka 0 0 0
Kd 0.733 0.631 0.310
Ks 0 0 0
Ns 100
```

Exchange the definition of `Ks` and `Kd`, such that the floor reflects light in a glossy manner:

```
newmtl gold
Ka 0 0 0
Ks 0.733 0.631 0.310
Kd 0 0 0
Ns 100
```

Render the scene again. Can you understand what is going on? Play with different values for `Ns`.



That's all, folks!

Universidade do Minho