pbrt v3 – Distributed Ray Tracing

Luís Paulo Santos – Dep. de Informática

Universidade do Minho

May, 2022

Distributed ray tracing proposes that the samples are stochastically distributed over the integration domain. This allows the simulation of soft light transport phenomena such as:

Anti-aliasing	Distribute samples over the image plane
Soft shadows	Distribute samples over the light sources area
Glossiness	Distribute samples over the BRDF lobe
Depth of field	Distribute samples over the lens area
Motion blur	Distribute samples over time

Anti-aliasing and soft shadows have already been addressed in previous tutorial. This tutorial will focus on depth of field and glossiness.

Depth of Field

`pbrt` allows simulation of depth of field (dof) by using the perspective camera and distribution of primary rays over the camera lens.

Download the file `dof-dragons.zip` from the elearning system and unzip it to a given folder. Open the `pbrt` scene description file and carefully verify the camera description. You'll find two parameters that control the dof effect:

Name	Type	Default	Description
`lensradius`	`float`	0	The radius of the lens. Used to render scenes with depth of field and focus effects. The default value yields a pinhole camera.
`focaldistance`	`float`	10^30	The focal distance of the lens. If "`lensradius`" is zero, this has no effect. Otherwise, it specifies the distance from the camera origin to the focal plane.

Render the scene with the suggested `lensradius` (0.02, 0.0075 and 0.002) and comment the results.

Notice that the image is focused on the second visible dragon. Play with the parameter `focaldistance` (for `lensradius` =0.02) such that the camera becomes focused on the third visible dragon.

Set `lensradius` to 0 and rerender the scene. Was there any significant change in the rendering time compared to your previous rendering? Since now dof has not been simulated, how do you explain that the rendering time doesn't change?

Glossy Reflections

Download the file `cornellGlossy.pbrt` from the elearning system and open it. Note that the floor of the Cornell Box has been changed such that it is a glossy metal (gold, in fact).

Render the scene and comment on the aspect of the floor. Are there any glossy reflections?

Note that the floor is completely black because it should reflect light through glossy interactions and these are not being simulated. Note that the reflected floor in the mirror is also black as expected.

Open the integrator `VI-Tutorial2.cpp`[1]. On the `Li()` method and locate the part where reflections are handled:

```
if (ray.depth + 1 < maxDepth) {
   // Trace rays for specular reflection and refraction
   L += SpecularReflect(ray, isect, scene, sampler, arena,depth);
   L += SpecularTransmit(ray, isect, scene, sampler, arena, depth);
}
```

Notice that each of the above methods only handles specular reflections and transmissions, repectively.

Open the `core/integrator.cpp` file and locate the `SpecularReflect()` function. It defines

```
BxDFType type = BxDFType(BSDF_REFLECTION | BSDF_SPECULAR);
```

and then calls

```
Spectrum f = isect.bsdf->Sample_f(wo, &wi, sampler.Get2D(), &pdf, type);
```

which stochastically selects a direction (`wi`) on the hemisphere to send a ray, returns the probability with which that direction was selected (`pdf`) and returns the value of the BRDF for the pair of directions (`wo, &wi`). Actually sending a ray happens below when the `Li()` method is recursively called:

```
[…] Li(rd, scene, sampler, arena, depth+1) […]
```

The reason why `ViTutorial2` does not include glossy reflections is that `Sample_f` is explicitly told to sample only the specular peak: `BxDFType(BSDF_REFLECTION | BSDF_SPECULAR)`. To sample the glossy lobe this should be parameterised with

```
BxDFType(BSDF_REFLECTION | BSDF_GLOSSY)
```

To add glossy reflections to our integrator copy the `SpecularReflect()` function to the `VI-Tutorial2.cpp` file and change its name to `GlossyReflect()`.

Now change the new function such that it samples the glossy lobe withy 4 rays. To achieve this add a `for` loop and make sure that the returned value (L) is properly initialized, the final radiance value is accumulated and then averaged (i.e., divided by the number of glossy reflection rays).

Finally, change the method `Li()` such that it calls the new function. In order to maintain the rendering time reasonable let's evaluate glossy reflections only for primary rays:

---

[1] This file is made available in the elearning system together with this tutorial, just in case you missed Tutorial 2. But it should be in your `$PBRT3/integrators` folder. Use that one!

```
if (ray.depth<1) {
    L += GlossyReflect(ray, isect, scene, sampler, arena,depth);
}
```

Render the scene and check whether or not glossy reflections are now simulated.

Look at the specular reflections on the mirror. Are these correct? Why?

Can you find the way to change your code such that the reflections on the mirror correctly depict the glossy floor? What happened to execution time?

Simulating glossy light transport by resorting to stochastic sampling often results on noisy images – that is the variance among pixels. The brute force approach to reduce this variance is to increase the number of samples.

There are two forms of doing this in our code:

. you can increase the number of rays used to sample the glossy lobe. Try doing that at the `GlossyReflect` function;

. you can increase the number of primary rays per pixel. Try changing your sampler to

```
Sampler "stratified" "integer xsamples" [8] "integer ysamples"
[8] "bool jitter" ["true"]
```

Compare the resulting images and respective rendering times.