Tutorial 8 - Gaussian Stratified Sampling - extending PBRT with your own algorithms

Computer Graphics

Kartic Subr and Martin Asenov

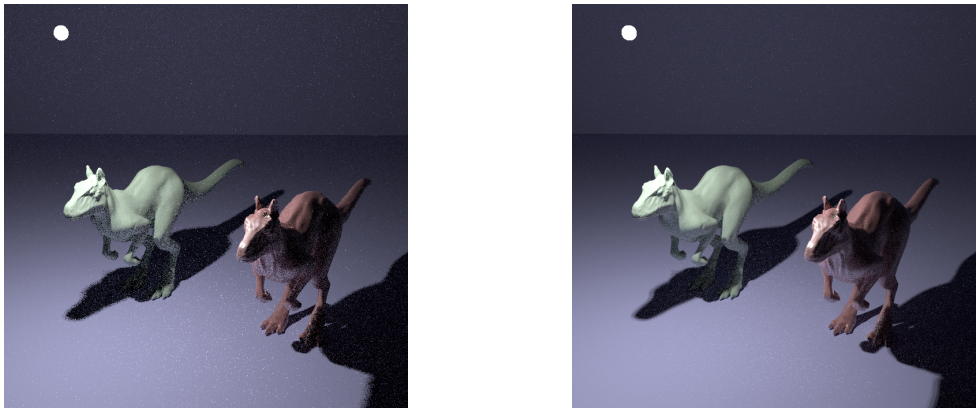
November 17, 2019

In this tutorial we are going to look at how you can easily extend the functionality of pbrt by implementing a new sampler. We base the new sampler on the Stratified sampler. In the original sampler, when we use the jittered mode we use a uniform distribution to sample the offset from the center [1]. As such we are going to look at the steps necesary to add the new functionality to pbrt. All the changed files described below, you can find in the code folder of the tutorial.

First, we need to define the new samplers as part of the pbrt API, which can be done in `src/core/api.cpp`. We include the header file of our new sampler, `gaussjitter.h`, and define the new sampler in the `MakeSampler`.

Next, we define the header file and its implementation in folder where the rest of the samplers are implemented, `src/samplers/gaussjitter.h` and `src/samplers/gaussjitter.cpp`. If you examine `stratified.cpp` you will see the external calls, such as `StratifiedSample1D`, `StratifiedSample2D`, etc., where the actual sampling is implemented. As such our new sampler follows almost the exact same structure as the stratified sampler.

Most of the changes are implemented `src/core/sampling.cpp`. First we implemented `sampleGauss` which samples according to a normal distribution $N(\mu, \sigma^2)$, where $\mu = 0.5$ (the center of the sampling region) and a σ specified when defining the scene. We then substitute the uniform sampling with the just defined normal distribution sampling.



(a) (b)

Figure 1: Example killeroo scene with (a) random and (b) gaussian stratified sampler defined here.

And that's it! The just defined sampler can be used as the other samplers, by specifying it in a `.pbrt` file. You have been provided with all the code to allow you to make that simple modification

to the stratified sampler. Make sure that you can apply the described changes and are able to compile the modified pbrt.

As mentioned in previous tutorials to compile pbrt you can follow the documentation here: https://github.com/mmp/pbrt-v3. For dice machines follow the section Makefile builds (Linux, other Unixes, and Mac), with the only difference being that you need to run `cmake3`, rather than `cmake`. To recompile the project after you have made a change is enough to run the `make -j` from the build folder again.

General tips:

1. While it is good to have a good knowledge of the different components of pbrt, when it comes to extending a certain module think about the input-output requirements of that module. What is been provided to that module; what do I need to return?

2. Make use of the existing documentation and examples. The pbrt book describes in a lot of details how the different components good. Moreover, there are multiple examples of different samplers, materials, integrators, etc. - different parts that you might want to extend. It is often beneficial to find the example that it is closely related to what you want to do, and extend it.

References

[1] Stratified sampling. `http://www.pbr-book.org/3ed-2018/Sampling_and_Reconstruction/Stratified_Sampling.html`.