Practice 2

MLCG 2021-2022

Ricardo Marques

ricardo.marques@ub.edu

Universitat de Barcelona – Master in Artificial Intelligence (MAI)

Introduction

The goal of the practice 2 is to put in practice the Monte Carlo-related concepts seen in the theory class. This lab is divided in two different parts, and will run for two weeks. In Part I, you will work in a controlled testing environment named appWorkbench (more details are given below). Then, in Part II, and once your understanding of Monte Carlo methods is consolidated, you will apply this understanding to solve the illumination integral using the appRenderer, that you used in Practice 1.

Part I – Monte Carlo in the appWorkbench

The appWorkbench

The appWorkbench is a script for you to prototype, evaluate and analyze a particular Monte Carlo method for hemispherical integration. Throughout this course, you will resort to the appWorkbench to develop, validate and compare different methods such as, classic Monte Carlo (MC), Bayesian Monte Carlo (BMC), and variance reduction techniques such as Importance Sampling (IS).

To use the appWorkbench, the **first step** is to ***set-up the function we wish to integrate.*** To this end, you should resort to the *Function* abstract base class [defined in PyRT_Common.py]. The function to be integrated (i.e., the integrand) can be a function with a constant value over the hemisphere, a cosine lobe to a given power ($\cos^{\mathrm{n}}\theta_i$) or a function given by the Arch environment map (i.e., experimentally measured). These functions are specified through the classes Constant, CosineLobe and ArchEnvMap in the file PyRT_Common.py, and they all inherit from the abstract base class Function. To use them, you just have to instantiate an object of the class you wish to use.

The **second step** to use the appWorkbench consists of **setting up the probability density function** (pdf) used to sample the integrand. To set-up the pdf used to generate random directions on the hemisphere, you should use the PDF abstract base class [see PyRT_Common.py]. The pdf can be of type UniformPDF or CosinePDF, which are both derived from the base class PDF. The classes which inherit from the PDF abstract base class offer two methods:

- $generate\_dir(u_i, u_2)$ which, given two random numbers u_1 and u_2 uniformly distributed in $[0,1[$, returns a spherical direction
- $get\_val(\omega_i)$, which returns the probability of generating the direction ω_i

In this practice you will only use the UniformPDF, the CosinePDF being useful for later practices where importance sampling techniques will be explored.

Finally, the **third step** for using the appWorkBench is to **define the experimental set-up**. The experimental set-up is captured by a set of variables:

- *ns_max* (integer), which defines the minimum number of samples to be used in the monte carlo estimate
- *ns_max* (integer), which defines the maximum number of samples to be used in the monte carlo estimate
- *nEstimates* (integer), which defines the number of estimates to be performed for each particular number of samples

Based on the above-defined variables, a 2D matrix called *results* is then initialized with zeros. The role of this matrix is to store the (average) estimate error (absolute value) for each method and for each number of used samples. During the main loop of the script, the values of this matrix must be filled appropriately. Finally, once the error associated with each method and each number of samples is computed, the *results* matrix content is displayed. Fig. 1 shows an example of the final result produced by the appWorkbench when evaluating the performance of 4 different estimation methods (MC, MC IS, BMC and BMC IS) for estimating the value of particular integral. Note that, in this practice, you will only be able to visualize results for the MC method.

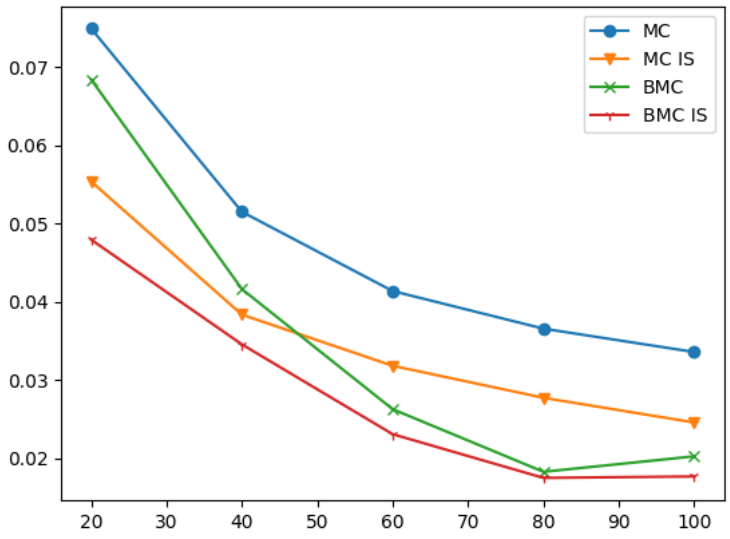

Figure 1. Average error of estimates of the value of the integral $I = \int_{\Omega_{2\pi}} L_i(\omega_i) \cos^4 \mathrm{d}\omega_i$. using different Monte Carlo-based techniques as a function of the number of samples. For each sample count (ranging from 20 to 100), a total of 2500 estimates have been performed and averaged. Note the significant error reduction obtained when using the machine learning-based approach of Bayesian Monte Carlo (BMC).

Assignment 2.1: Estimate the value of a hemispherical integral

Your task in this assignment is to develop a Monte Carlo estimator with uniform sampling of the unit hemisphere. Then, use it to estimate the value of the integral over the hemisphere defined as:

$$I = \int_{\Omega} \cos\theta_i \, \mathrm{d}\omega_i$$

Once you have your Monte Carlo estimator ready, use it to compute a plot with estimate the error as a function of the number of samples. You will probably notice that the error plot is highly irregular and changes at each run (see two left-most examples in Figure 2). This is because of the variance inherent to the MC estimator. To alleviate the problem, you should average the estimate error over a number of different estimates for each sample count, by adapting the main loop of the script. An example of the final result is shown in Fig. 2 (right). In case your implementation is correct, you should observe the error converging to zero as the number of samples used in the MC estimate increases. This experiment should be repeated using different integrands that you might want to test, as well as varying number of minimum and maximum samples.

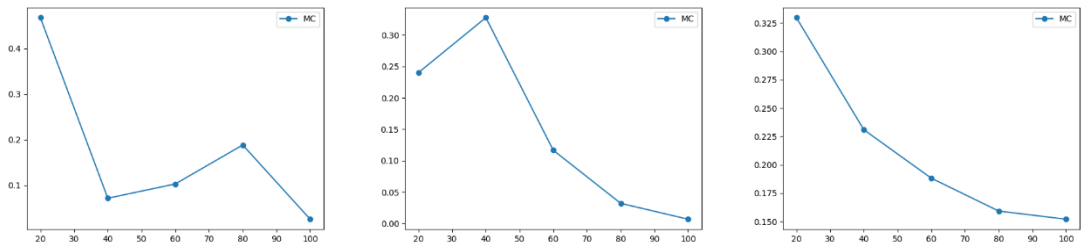

Figure 2. *Left and center: Examples of different executions when measuring the estimate error as a function of the number of samples, using Monte Carlo. Right: Result when averaging over 1000 estimates.*

Part II – Monte Carlo for rendering (appRenderer)

Assignment 2.2: Estimate the value of the illumination integral

With your MC estimator validated, you should work on its application to the computation of real images, in the context of the *appRenderer* script. To this end, you should develop the *compute_color()* method of the *CMCIntegrator* class so that it uses a Monte Carlo estimator to approximate the value of the illumination integral. In general terms, this can be achieved by implementing the *compute_color()* method such that, among other things, it contains the following steps:

```
(…)
Generate a sample set S of samples over the hemisphere

For each sample ω_j ∈ S:
        Center the sample around the surface normal, yielding ω_j'
        Create a secondary ray r with direction ω_j'
        Shoot r by calling the method scene.closest_hit()

        If r hits the scene geometry, then:
                L_i(ω_j) = object_hit.emission;
        Else:
                If the scene has an environment map
                        L_i(ω_j) = scene.env_map.getValue(ω_j);
                End If
        End If
        (…)
End For
(…)
Return result;
```

The first step is to generate a set of samples distributed over the hemisphere according to some probability density function. To this end, you must resort to the function *sample_set_hemisphere()*, which can be found in *PyRT_Common.py*. However, the resulting sample set is generated around the world normal vector $(0, 1, 0)$. Therefore, you must use the function *center_around_normal()* to rotate it, using the surface normal of the current scene point where you want to compute the illumination integral. This operation yields $\omega_j{}'$, i.e., the direction for which we want to sample the incident radiance value $L_i(\omega_j')$. Follows the construction and shooting of a ray r with direction $\omega_j{}'$. Then, depending on whether the ray r hits or not the scene geometry, we compute the value of the sample $L_i(\omega_j)$. Finally, based on the information collected by sampling, you should compute a Monte Carlo estimate of the illumination integral. Fig. 3 shows an example of the final result.

Figure 3: Image rendered using the Monte Carlo integrator. The result was generated using 40 samples per each estimate of the illumination integral, and using the sphere_test_scene with areaLS=False and use_env_map=True. The rendering time was approximately 175s.