UNIVERSITÄT DES SAARLANDES
Lehrstuhl für Computer Graphik
Prof. Dr. Philipp Slusallek,
Prof. Dr. Hab. Inż. Karol Myszkowski (MPII), and
Dr. Gurprit Singh (MPII)
Tutors: Duarte David and
Pascal Grittmann

25 June 2020

Realistic Image Synthesis (SS 2020)
Assignment 6

Submission deadline for the exercises: 9 July 2020

6.1 Introduction

In this assignment, you are going to extend the framework from the previous assignment by implementing progressive photon mapping.
You are going to implement the algorithm itself inside `src/algorithms/render_ppm.cpp`. We again provided you with the basic structure. Use the following commands to render the provided test scenes with progressive photon mapping:

```
arty -a 2 cornell_box.yml --width=512 --height=512
arty -a 2 cornell_box_specular.yml --width=512 --height=412
arty -a 2 cornell_box_water.yml --width=512 --height=412
arty -a 2 cornell_box_glossy.yml --width=512 --height=512
```

6.2 Photon Shooting (40)

The first step for photon mapping is tracing particle paths from the light sources through the scene. The particles are stored in a photon map and used for the density estimation later on. Implement the trace_photons() function that does this.

You need to make random walks and store all paths (or hits) until the photon is absorbed based on a random decision (Russian Roulette). This part of the exercise is not much different from a path tracer. Every time you bounce a photon from a surface you need to re-scale its energy to take into account the surface reflectivity and your sampling strategy.

To initialize a photon path, you have to randomly choose one light source and sample a position and direction form that light. The Light interface provides the function sample_emission() for this. Make sure that you account for the PDFs of all random decisions that you made.

6.3 Density Estimation (30)

We provided you with code that builds an acceleration data structure (hash grid) over the photons. You can use this to efficiently perform density estimation.
The function eye_trace is supposed to shoot a given camera ray into the scene and perform photon density estimation at the hit point. You can use the query() method of the PhotonMap to execute a lambda function for every photon within the radius.
Use the Epanechnikov filter as your density estimation kernel:

$$K(x_q) = \frac{2\sum_{i=1}^{N}\left(1-\left(\frac{x_i-x_q}{R}\right)^2\right)\Delta\Phi_i}{\pi R^2} \tag{1}$$

with x_q being the query point, $x_1, \ldots, x_N$ the positions of the found photons, N the number of found photons, $\Delta\Phi_i$ the power of the i-th photon, and R the search radius.

Keep in mind that performing a density estimation on a specular surface makes no sense, as the probability of a photon having the reflected direction as its incoming direction is zero. Also, you have to handle hitting emissive surfaces, i.e. lights, correctly as well.
If you implement the density estimation correctly, your results should look like the images in Figure 1.

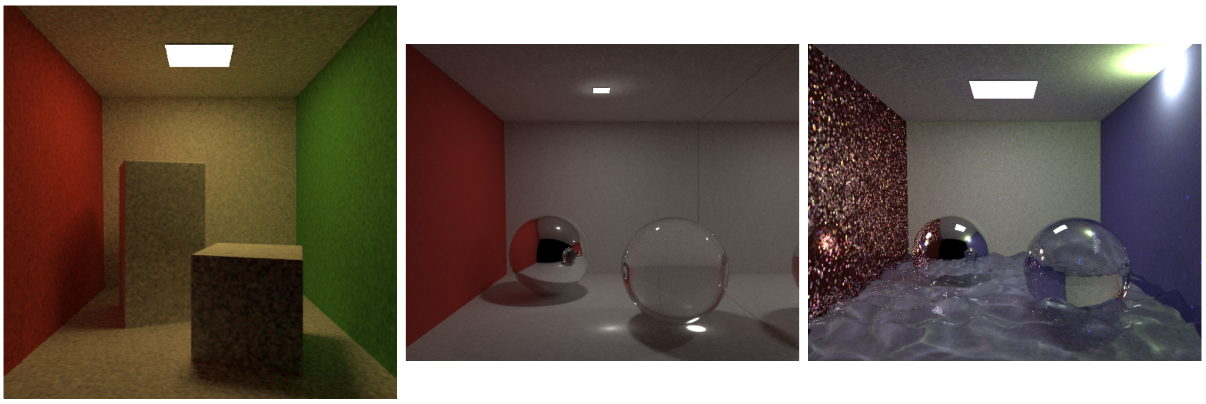

Figure 1: Images for the three Cornell Box scenes rendered with PPM. From left to right: diffuse Cornell Box with 10 iterations (samples per pixel), specular version (10 spp), and filled with water (10 spp).

Hints The trace_photons() function traces the photons of one light path. The photons created by one such path can be seen as a Monte Carlo estimate of the distribution of light in the scene. Because you are tracing multiple paths, you will have to take the number of light paths into account in your density estimation.

For those of you unfamiliar with C++, here is what using the query() method looks like.

```
photon_map.query(surf.point, [&] (const Photon& p, float d) {
    // Do something with the photon p and its distance from the hit point d
});
```

The `[&]` creates an anonymous function that captures all variables in the scope by reference. That is, you can read and modify all variables from inside the function.

6.4 Comparison to Path Tracing (10)

Take a look at the images from progressive photon mapping (Figure 1) and path tracing (Figure 2). Each of the two techniques handles some effects significantly better than the other. Find at least two of those effects and explain why (progressive) photon mapping is better/worse in those cases.

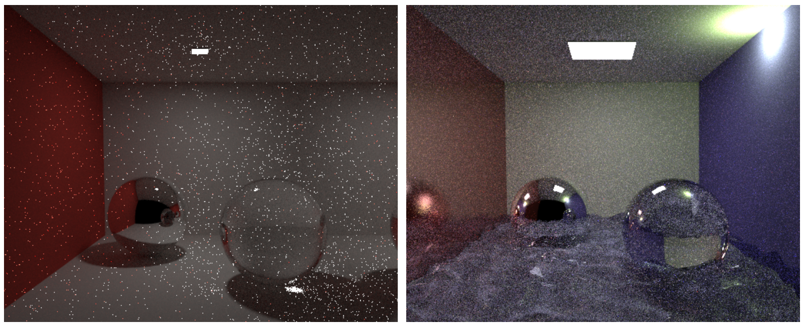

Figure 2: Comparison images from the path tracer, for the same scene as in Figure 1. The number of samples per pixel from left to right are: 100, 500.

6.5 A Simple Improvement for Glossy Materials (30)

A simple trick to improve efficiency of progressive photon mapping in the presence of glossy materials is to not do density estimation on glossy materials either.

Instead, compute the direct illumination at a glossy hit point, and bounce, as you would do in path tracing. Keep in mind that you will have to either ignore hitting the light source after a glossy bounce, or use MIS to combine direct illumination with randomly hitting the light.

Extend your photon mapper with this trick. If you did this correctly, your images will now look similar to the images in Figure 3.

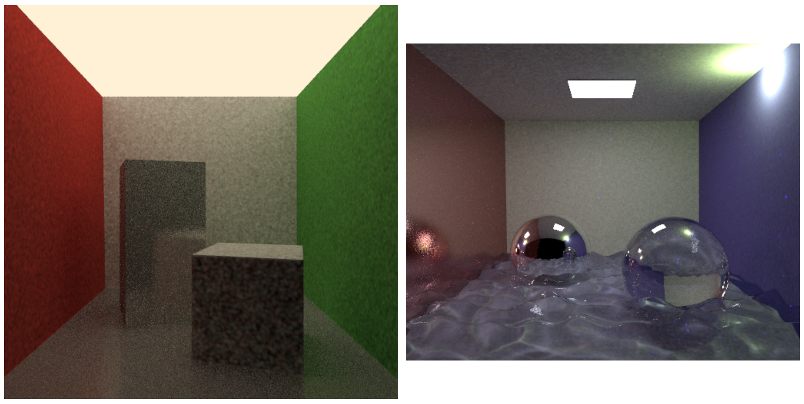

Figure 3: Results for the two scenes with glossy materials, after the optimization from the last exercise. Using 10 (left) and 100 (right) iterations.

Procedure of Submitting

Please submit your solutions in .pdf (or .zip for source code) format via email to the tutor: s8dudavi@stud.uni-saarland.de
The submission deadline is midnight, i.e., 9 July 2020 , 23:59. Late submissions within 24 hours will receive at most 75% of the points. Late submissions after more than 24 hours will not be accepted.