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


4 June 2020

# Realistic Image Synthesis (SS 2020)
## Assignment 4

**Submission deadline for the exercises: 11 June 2020**

### 4.1 Using the Framework

In this assignment, you will have to implement a simple path tracer. You will be provided with three simple Cornell Box scenes. The materials of the scenes will be modeled using Phong's reflection model (look in the Global Illumination Compendium, p. 32), and perfect specular reflection and refraction. Download the framework from the lecture's homepage, study the code, and make yourself familiar with the conventions. Please do not share the source code or make a public repository out of it, as doing so will be considered cheating. The use of a *private* repository is however allowed. Information on how to build and run the renderer can be found in the readme files. If you run the renderer right now, it will visualize the scene with eye-light shading (see `src/algorithms/render_debug.cpp`). The rendering algorithm can be specified via command line, or changed while rendering by pressing 'R'. Running the renderer with the following commands will render the test scenes from within the build folder:

```
src/arty --algo=pt ../test/cornell_box.yml --width=512 --height=512
src/arty --algo=pt ../test/cornell_box_glossy.yml --width=512 --height=512
```

The code that you will have to write will be in `src/algorithms/render_pt.cpp`. We provided you with a simple skeleton. In this assignment, you will write a basic path tracer.

### 4.2 A Basic Path Tracer 5 + 20 + 5

We are going to start with a basic path tracer. The function path_trace() gets a primary ray (sampled from the camera), the scene, and a random number generator. It is supposed to trace a path, starting with this primary ray. Once a light is hit, the radiance transported along the path is returned.

**a.)** Start by computing the contribution from a light source that is hit. If you run your code now, you should get a black image with only the light source visible.

**b.)** Next, continue the path by sampling the BRDF. The Bsdf classes offer a method sample() to achieve this (see `src/materials.h`). To get those working, you will have to implement the functions

sample_cosine_hemisphere(), and cosine_hemisphere_pdf() in `src/random.h`

for the diffuse part, and the corresponding ... _cosine_power_ ... methods for the glossy part.

**c.)** Running the code now might take very long (or even forever), because paths are only terminated when they leave the scene or hit the light. You are now going to implement Russian Roulette to change this. The function russian_roulette in `src/random.h` computes the survival probability of a path based on the luminance of its current throughput.

Your images should match those in Figure 1 and Figure 2.

**Hints:** Make sure that whenever you make a random decision, you take its probability into account. In other words, be careful to use unbiased estimators in your implementation.

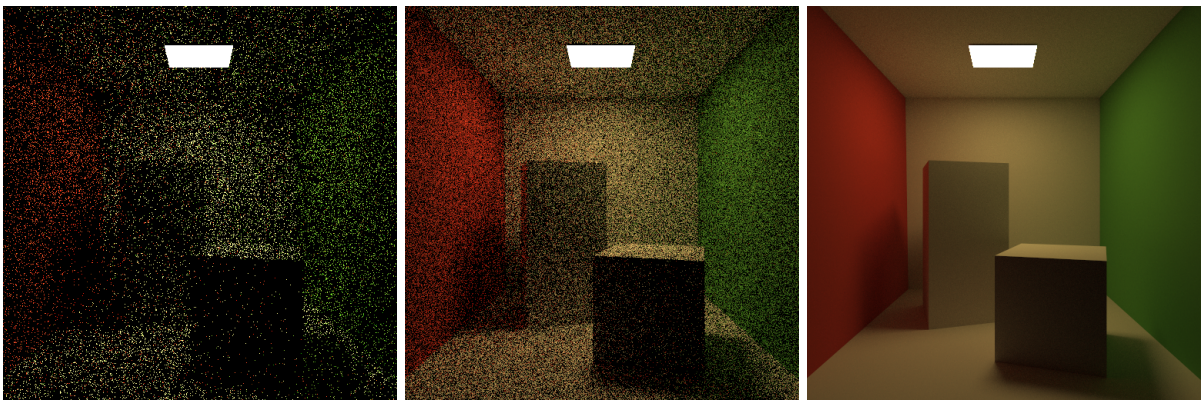

Figure 1: Images from the basic path tracer for the simple Cornell Box scene. From left to right: 10 samples per pixel, 100 samples per pixel, and the reference image.

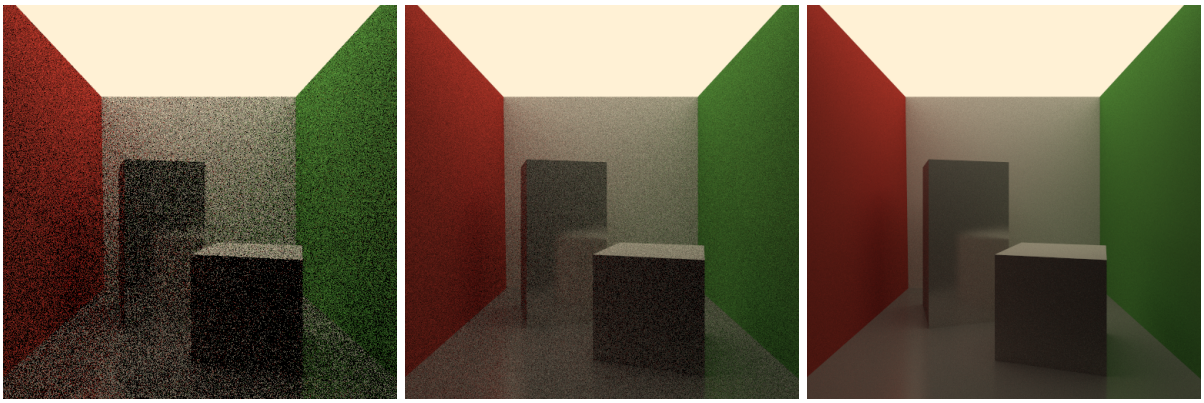

Figure 2: Images from the basic path tracer for the glossy variation of the Cornell Box scene. From left to right: 10 samples per pixel, 100 samples per pixel, and the reference image.

## 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., 11 June 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.