Advanced Topics in Computer Graphics I

Universität Bonn
Institut für Informatik II
June 3, 2025

Summer term 2025
Prof. Dr. Reinhard Klein
Domenic Zingsheim

Sheet R08 - Photon Mapping

Hand in your solutions via eCampus by Tue, 17.06.2025, **12:00 p.m.**. Compile your solution to the theoretical part into a single printable PDF file. For the practical part, hand in a single ZIP file containing only the exercise* folder within the src/ directory. Please refrain from sending the entire framework.

Assignment 1) Photon Mapping *(7 Pts)*

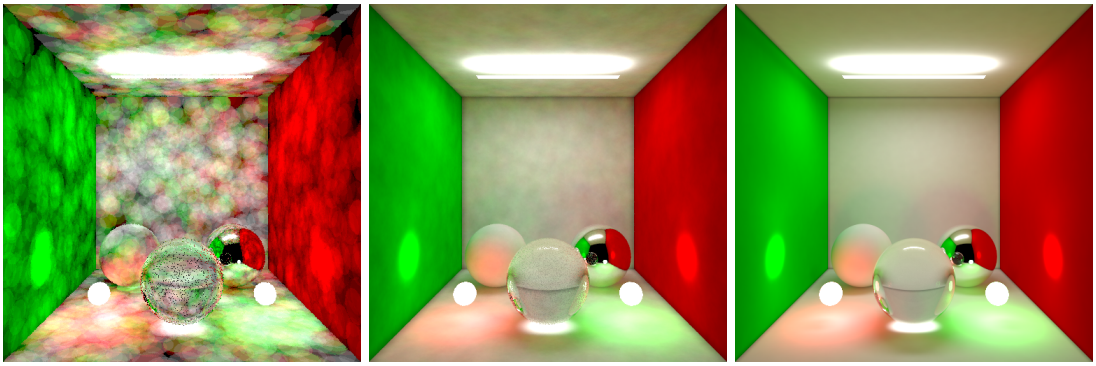

Figure 1: Cornell Box scene containing an elongated rectangular area light and two sphere lights rendered with photon mapping using approximately 1×2^{13}, 10×2^{17} and 1000×2^{17} photons in the photon map, respectively. To obtain these renderings, run `./bin/exercise08_PhotonMapping -s data/exercise08_PhotonMapping/cornell_box_spheres_area_lights.xml`. The number of photons in the photon map and other parameters can be adjusted in the scene description file.

In this exercise, you complete a photon mapper. Instead of simply tracing rays through the camera and accumulating radiance, the photon mapping ray generator is more complex. First rays are traced from the camera, accumulating radiance as usual, but also recording a photon gathering request for each pixel in a separate buffer. Then a photon map is generated by tracing rays from the light sources, depositing photons in the photon map whenever the ray lands on a diffuse surface. These two steps can happen in any order. The photons are then arranged in a KD-tree such that we can traverse the photon map more efficiently. Finally, a CUDA kernel is launched that performs the gathering requests from the camera rays in the photon map and the result is added to the final image.

In the raytracing shaders the main difference is that there are two ray generation methods (`__raygen__traceLights()` and `__raygen__traceCamera()`). The emitters now have a method for spawning new photons `samplePhoton()`, and the BSDFs have a method to simply evaluate the diffuse albedo `evalDiffuseAlbedo()` instead of the whole BSDF to compute the correct weight of the gathering requests without knowing the light direction.

Tasks:

a) Gather photons in `__raygen__traceCamera()` when a ray hits a diffuse surface, instead of computing the illumination by tracing new rays in `photonmappingraygenerator.cu`. You can use the `cornell_box_{spheres,cubes}.xml` scenes for testing.

b) Implement the photon generation in the `__direct_callable__*_samplePhoton()` methods for sphere and mesh light sources in `lightsources.cu`.

c) Implement the traversal of the photon map KD-tree and collect new photons for each photon gathering request in the `gatherPhotonsKernel()` method in `photonmapkernels.cu`. Note: the photon map is currently built on the host and not GPU in `buildPhotonMapKDTree()` in `photonmappingraygenerator.cpp`.

d) Implement progressive photon mapping [1] by successively adjusting the gathering radius in the `gatherPhotonsKernel()` method in `photonmapkernels.cu`. Progressive photon mapping shrinks the gathering radius depending on a user-specified parameter and how many photons have been gathered.

Theoretical Assignments

Assignment 2) Photon mapping *(3 + 2 BonusPts)*

- What does it mean, that photon mapping is a biased technique? How does this compare to path tracing?
- How can we decrease the systematic error introduced by photon mapping? To which value does the systematic error converge? How do we call an estimator with this property?
- How are the photons stored? Which operation on the stored photons has to be implemented very efficiently for fast rendering?

Bonus:

- When changing the camera position, do we have to regenerate the photon map? Why or why not? When is generating a new photon map necessary?
- What is the benefit of splitting up the photons into a global map and a caustics map? Which map does typically need to store more photons? Why is this separation performed and how is it done?

References

[1] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. In *ACM SIGGRAPH Asia 2008 papers*, pages 1–8. 2008.

Good luck!