Parallel Progressive Photon Mapping on GPUs

Toshiya Hachisuka Henrik Wann Jensen
UC San Diego

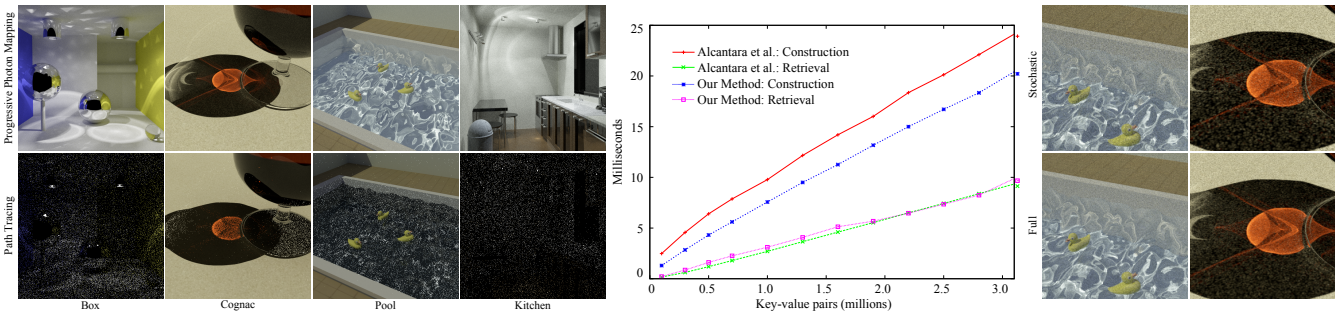

Figure 1: *Left: rendered images using path tracing and progressive photon mapping using the same ray tracing core in the same rendering time (10 min). The box scene and the kitchen scene are entirely illuminated by caustics from the light bulbs, which is common in the real world. A typical implementation of path tracing on GPUs cannot handle such illumination efficiently, whereas progressive photon mapping is robust under such common illumination settings. Middle: performance comparison with the hashing scheme by Alcantara et al. [2009] using random points. The measurement for the method by Alcantara et al. is directly taken from their paper, which is measured on GeForce GTX 280 SSC. Our measurement is done on Radeon HD 4850 which is in the same performance range as GeForce GTX 280. Our method achieves similar performance as their method with a significantly simpler algorithm. Right: effect of the stochastic selection on rendered images with 100M emitted photons using 512^2 photons per pass. Extra noise due to the stochastic selection is visually subtle.*

Introduction Accurate global illumination rendering using GPUs is gaining attention because of the highly parallel nature of global illumination algorithms. For example, computing the radiance of each pixel using path tracing is embarrassingly parallel. Some major commercial rendering software also started adopting global illumination on GPUs.

Although path tracing may be suitable for GPUs, it is not necessarily the most robust rendering algorithm under complex, but common lighting configurations. Progressive photon mapping [Hachisuka and Jensen 2009] is a new global illumination algorithm that has been demonstrated to be more robust than existing methods. In particular, if one wants robustness to various types of light paths with specular-diffuse-specular light transport, progressive photon mapping is currently the only choice. However, an efficient implementation of progressive photon mapping on GPUs poses some challenges. In particular, we need a fast construction of photon maps and efficient range query of photons on GPUs since the algorithm repeatedly constructs and uses a photon map.

We present a data-parallel implementation of progressive photon mapping on GPUs. The key contribution is a new stochastic spatial hashing scheme that achieves a data-parallel construction of a photon map and an efficient range query of a photon map on GPUs. Our method provides a very simple way to map computation of progressive photon mapping into data parallel computation. The new hashing scheme achieves about the same performance as the state of the art method in our implementation (the graph in Figure 1), but with a significantly simpler algorithm and finer granularity of parallelism. The images in Figure 1 show that our implementation of progressive photon mapping can render less noisy images in some common lighting configurations compared to path tracing which is often implemented in an off-line rendering system on GPUs.

Method Our implementation uses spatial hashing for photon maps. Standard spatial hashing has some properties that are not desirable for a GPU processing as were also mentioned by Alcantara et al [2009]. In a typical CPU implementation of spatial hashing, each hash entry keeps a list of elements when a hash collision occurs. Creating a list is not suitable for GPUs because this is a dependent and serial process. Since the number of elements per hash entry typically varies, work distribution at the time of data query varies as well, which may result in underutilization of GPU units.

Our solution is to stochastically store a single photon instead of storing a list of photons at each hash entry. More precisely, given n photons that would be mapped to the same hash entry, we select one photon according to the uniform probability $p(x) = \frac{1}{n}$. To keep the result consistent, stored photon flux is now divided by the probability $p(x) = \frac{1}{n}$ (i.e., multiplied by n). Assuming photon tracing processes are statistically independent among threads, this is easily done by performing random writes of photons into the hash table without any thread synchronization. In other words, we can ignore hash collisions. The n value can be efficiently computed by additive blending of 1s or atomic increments using the same random writes, which is done among hash entries in parallel.

This key idea is very simple and easy to implement, yet resolves all of the above mentioned issues of standard spatial hashing methods. Since each hash entry now keeps only one photon, we no longer need to perform chaining of photons into a list. The amount of work for data query is exactly one data fetch regardless of the number of collisions, which results in completely uniform work distribution. Alcantara et al. [2009] also provide a solution to those issues, but with a more complex algorithm than our algorithm. We can also perform range query of photons just by looking at grid cells that intersect with a sphere defined by query position and query radius. The size of grid cell is automatically chosen to be equal to the maximum query radius to simplify the range query. The extent of spatial hashing is decided by the bounding box of visible points. The stochastic selection adds subtle noise (Figure 1, right) but no extra bias since it is equivalent to Russian roulette. The number of collisions is on average 3 to 4 in our tests.

Note that our method needs fundamentally less work than any of previous methods based on tree data structure (e.g., [Zhou et al. 2008; Fabianowski and Dingliana 2009]). For example, the number of data fetches for tree traversal is fundamentally larger than what is required in our method. Even if a tree is perfectly balanced, 256K ($= 2^{18}$) photons will result in 18 data fetches, whereas our method requires exactly one data fetch regardless of the number of photons. In practice, due to an unbalanced tree, data retrieval using a tree results in quite irregular work distribution and poor utilization of GPU units. Indeed, the range query is several times faster than Fabianowski and Dingliana [2009]. The hash table construction is also faster than a tree construction. For example, Zhou et al. [2008] reported the construction time of 9ms for 200K points, whereas our hash table construction of 200K points is around 2.5ms.

References

ALCANTARA, D. A., SHARF, A., ABBASINEJAD, F., SENGUPTA, S., MITZENMACHER, M., OWENS, J. D., AND AMENTA, N. 2009. Real-time parallel hashing on the gpu. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, 1–9.

FABIANOWSKI, B., AND DINGLIANA, J. 2009. Interactive global photon mapping. *Computer Graphics Forum 28*, 4, 1151–1159.

HACHISUKA, T., AND JENSEN, H. W. 2009. Stochastic progressive photon mapping. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, 1–8.

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5, 1–11.