Eurographics Symposium on Rendering 2009
Hendrik P. A. Lensch and Peter-Pike Sloan
(Guest Editors)

Volume 28 (2009), Number 4

Interactive Global Photon Mapping

B. Fabianowski and J. Dingliana

GV2 Group, Trinity College, Dublin, Ireland

Abstract

We present a photon mapping technique capable of computing high quality global illumination at interactive frame rates. By extending the concept of photon differentials to efficiently handle diffuse reflections, we generate footprints at all photon hit points. These enable illumination reconstruction by density estimation with variable kernel bandwidths without having to locate the k nearest photon hits first. Adapting an efficient BVH construction process for ray tracing acceleration, we build photon maps that enable the fast retrieval of all hits relevant to a shading point. We present a heuristic that automatically tunes the BVH build's termination criterion to the scene and illumination conditions. As all stages of the algorithm are highly parallelizable, we demonstrate an implementation using NVidia's CUDA manycore architecture running at interactive rates on a single GPU. Both light source and camera may be freely moved with global illumination fully recalculated in each frame.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

1. Introduction

Photon mapping [Jen96] is a rendering algorithm reproducing the full range of global illumination effects. Photons are Monte Carlo traced from the light sources in a preprocessing pass and their hit points recorded in a photon map. Illumination may then be reconstructed at any point in the scene by density estimation on nearby photon hits. The bandwidth of the density estimation kernels is adapted to local hit density by looking up the distance to the *k*th-nearest hit.

Its high computational cost initially made photon mapping strictly an offline rendering technique. By concentrating on specular reflections and the resulting caustics, several authors have since achieved real-time performance, making approximations to the exact algorithm and using a cluster of computers [GWS04] or a GPU [SKUP*09, ZHWG08] for the required computational performance.

Having to locate the *k* nearest photon hits before any illumination can be computed is an important bottleneck. Splatting techniques avoid this issue by assigning each photon hit a fixed footprint instead. From heuristics operating on a per-surface basis [SB97], the methods for calculating footprints have progressed to using a photon's path probability [HHK*07] or differential vectors traced with it [SFES07].

We build on the latter work, obtaining footprints after both specular and diffuse reflections. Knowing the footprints of all hits allows us to construct a hierarchy of tight bounding volumes around them. From this, the photon hits relevant to any point in the scene can quickly be retrieved with their contributions calculated as the hits around found. In contrast to image-space splatting, illumination is also reconstructed at surfaces seen only after specular reflection from the eye.

The final image is computed by ray tracing with indirect illumination retrieved from the photon map. Photon and ray tracing are inherently parallel. By combining these with a highly parallel BVH build and density estimation based on photon footprints, the entire algorithm can make efficient use of the CUDA manycore architecture [NVI09]. We demonstrate this by calculating two bounces of indirect lighting for scenes with specular and diffuse surfaces at interactive frame rates with both the light source and camera freely movable.

2. Related work

The original photon mapping algorithm [Jen96] has attracted extensive follow-up work. We focus on the aspect most relevant to our own work here, accelerating density estimation. Hashed photon maps [MM02] enable faster photon hit re-

trieval but only approximate the k nearest hits. Gathering photon hits from within a fixed radius [WKB*02] provides no automatic bandwidth adaptation. To accelerate photon hit retrieval without compromising quality, photon map kd-trees built according to the voxel volume heuristic [WGS04] may be used. A recent implementation of real-time specular photon mapping in CUDA [ZHWG08] employs this heuristic. Specular photon mapping at real-time speeds is also demonstrated on a cluster of computers [GWS04].

An alternative to kth-nearest-neighbor density estimation is to assign each photon hit an individual footprint. These may be derived from the number of hits and area of each surface, then used to splat contributions directly to the surfaces [SB97]. Clipping the footprints against scene geometry and iteratively adjusting their sizes reduces bias [LP03]. Another method [LP02] uses least squares cross-validation to compute optimal footprints per surface. Footprints may also be based on the photons' path probabilities [HHK*07]. The entire paths are stored and their contributions splat to sample points generated by ray tracing from the eye.

Igehy [Ige99] computes footprints for the purpose of texture look-ups with a solid basis in differential geometry. These are obtained by taking the derivatives of a ray's position and direction in space with respect to the parameters that govern its emission. Applied to photon mapping [SFES07], the technique is shown to yield better bandwidth adaptation than kth-nearest-neighbor searches but allows perfectly specular reflections only. Footprints after diffuse and glossy reflections may be computed with the extension of ray differentials to path differentials [SW01] at the expense of higher computational cost.

While our work is concerned with high quality global illumination, its interactive nature warrants a review of some prominent approximate techniques in that domain. Instant radiosity [Kel97] traces a small number of photons and places secondary light sources at the hit points, approximating indirect lighting from diffuse and moderately glossy surfaces. Follow-up work replaces the original rasterization with ray tracing on a cluster and adds approximated specular photon mapping to simulate caustics [WKB*02].

Dachsbacher et al. [DS06] present a GPU implementation of a similar algorithm. Secondary lights are positioned by sampling a deep shadow map and their contributions approximated in a shader. Focusing specifically on specular interactions and caustics, a number of approximate real-time GPU algorithms have been developed [SKUP*09].

The introduction of programmable GPUs has sparked interest in accelerating more sophisticated rendering algorithms on their parallel processing units. Ray-triangle intersection implemented on the GPU [CHH02] was followed soon by a full GPU ray tracer then extended to photon mapping [PDC*03], albeit yielding low performance.

A key operation for both ray and photon tracing is the traversal of acceleration data structures. Due to the inefficiencies in supporting stacks, early GPUs encourage stackless kd-tree traversal approaches [FS05]. By adding a short stack and ray packetization [HSHH07], this yields real-time ray tracing performance. Similar speeds result from using roped kd-trees for stackless traversal [PGSS07].

More direct access to a modern NVidia GPU's computational units is provided by the recent introduction of the CUDA programming interface [NVI09]. Highly efficient implementations of key parallel processing paradigms are available for CUDA, such as parallel scan [HSO07] and fast sorting [SHG09]. Ray tracing with packetized, stackless BVH traversal executes in real time [GPSS07]. Recently, the highest kd-tree traversal performance has been shown to result from using a traversal stack and no explicit packetization [BAGJ08, ZHWG08]. CUDA's computational power also allows acceleration structures to be built in real time, whether they be kd-trees [ZHWG08] or BVHs [LGS*09].

3. Photon mapping

In the photon mapping algorithm, photons are emitted by the light sources and Monte Carlo traced through the scene. On each surface hit, the position, flux and incident direction of the photon are recorded. Illumination may then be retrieved from the resulting photon map by density estimation as

$$L_o(\vec{x},\omega_o) \approx \sum_{i=1}^{n} \frac{1}{r_i^2} K\left(\frac{\|\vec{x}-\vec{x}_i\|^2}{r_i^2}\right) f_r(\vec{x},\omega_i,\omega_o)\,\Phi_i. \quad (1)$$

Given a query point $\vec{x}$, nearby photon hits are retrieved and a kernel K aligned with the surface at $\vec{x}$ is placed around each. Weighted by the kernel's footprint area and its value at $\vec{x}$, every photon's flux Φ_i then yields an irradiance contribution. These are reflected by the surface's BRDF f_r from their incident directions ω_i to the desired outgoing direction ω_o. As this estimate is prone to noise and bias, direct illumination is typically computed separately and photon mapping used for the difficult indirect lighting only.

3.1. k-nearest-neighbor density estimation

To obtain a kernel bandwidth that adapts to local illumination conditions, photon mapping retrieves a fixed number k of nearest photon hits and then sets the bandwidth to the radius $r(\vec{x})$ of their minimal bounding sphere around $\vec{x}$. This yields the desired adaptability but also introduces a bottleneck: The bandwidth initially is only known to be bounded by a global r_{max}, requiring a temporary set of k candidate hits to be maintained and updated as closer ones are found. Their contributions can only fully be evaluated when the search has completed and the bandwidth $r(\vec{x})$ is known.

3.2. Local kernel density estimation

An alternative approach is local kernel density estimation: For every photon hit, a kernel bandwidth r_i is chosen before

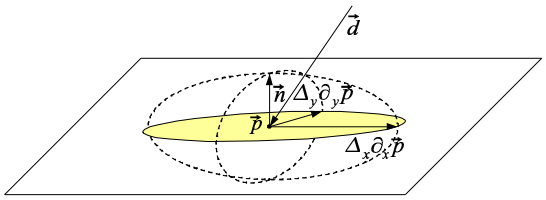

Figure 1: *A photon traveling in direction $\vec{d}$ hits a surface at $\vec{p}$. Its positional differentials $\partial_x\vec{p}$ and $\partial_y\vec{p}$ along with multipliers Δ_x and Δ_y define a skewed elliptical footprint. Adding the surface normal, a skewed footprint ellipsoid is produced.*

image generation. As each hit has a predetermined kernel footprint, their contributions are decoupled from each other. All photon hits whose footprints overlap the query point $\vec{x}$ may be located and their outgoing radiance contributions calculated independently. This directly corresponds to equation 1 with n the number of photon hits whose footprints overlap the query point. A disadvantage of this method is that the adaptation of the kernel bandwidth to the hit density is lost. For high image quality, adaptive bandwidths must be computed in another way. Photon differentials, described in the next section, are an elegant solution to this problem.

4. Photon differentials

The concept of photon differentials [SFES07] is an application of ray differentials [Ige99] to photon tracing. Each photon is equipped with differential vectors $\partial_x\vec{p}$ and $\partial_x\vec{d}$, the partial derivatives of its position $\vec{p}$ and direction $\vec{d}$ with respect to every parameter x that had influence on its path through the scene. Upon hitting a surface, a footprint is obtained by first-order Taylor expansion, multiplying $\partial_x\vec{p}$ by the distance Δ_x to the next sample taken of x. If the photon has differentials with respect to exactly two parameters, two vectors on the hit surface result. These can be interpreted as the semiminor and semimajor axes of a skewed elliptical footprint (fig. 1). The skew stems from the fact that the two differentials need not be orthogonal to each other.

As the query points for which illumination is to be reconstructed may lie on other surfaces than the photon hit, a skewed ellipsoid is actually stored. The footprint's two axes and the normal of the surface provide a coordinate system in which the ellipsoid is defined. During density estimation, a footprint for any other surface may then be obtained by cutting along it through the ellipsoid.

Equations for updating differentials after transmission through the scene and perfectly specular interaction are provided by Igehy [Ige99]. Since neither requires additional parameters to be sampled, only differentials with respect to the parameters governing initial emission need to be maintained. The resulting footprints adapt to local illumination conditions: Transmission through space lets footprints expand as is the case for beams of light. Specular reflection leads to expansion or contraction, depending on whether the surface hit is concave or convex. The resulting kernel bandwidths are shown to more accurately adapt to local photon hit density than in kth-nearest-neighbor density estimation [SFES07].

4.1. Emission

We stochastically emit photons from an isotropic point light source. An emission direction is chosen for each photon by uniformly sampling $\cos\theta \in [-1,1]$ and $\varphi \in [0,2\pi[$ to yield

$$\vec{d} = \begin{pmatrix} \cos\varphi\sin\theta \\ \sin\varphi\sin\theta \\ \cos\theta \end{pmatrix}. \tag{2}$$

The initial positional differentials are zero. Initial directional differentials are computed as the partial derivatives of $\vec{d}$, resulting in vectors orthogonal to $\vec{d}$ and to each other

$$\partial_{\cos\theta}\vec{d} = \begin{pmatrix} -\cos\varphi\cot\theta \\ -\sin\varphi\cot\theta \\ 1 \end{pmatrix} \text{ and } \partial_\varphi\vec{d} = \begin{pmatrix} -\sin\varphi\sin\theta \\ \cos\varphi\sin\theta \\ 0 \end{pmatrix} \tag{3}$$

In order to generate footprints, we must further determine the spacings $\Delta_{\cos\theta}$ and Δ_φ between the current photon and its nearest neighbors in the parameter plane. Because we use stochastic sampling, actual spacings cannot a priori be computed and expected values are calculated instead. These follow from an analysis of emission directions:

Using the initial differentials of the current photon, the emission directions of its neighbors can be approximated by first-order Taylor expansion as $\vec{d} + \Delta_{\cos\theta}\partial_{\cos\theta}\vec{d}$ and $\vec{d} + \Delta_\varphi\partial_\varphi$. Due to the light source's isotropic emission characteristics, the expected orthogonal distances between $\vec{d}$ and both of these directions are equal,

$$\|\Delta_{\cos\theta}\partial_{\cos\theta}\vec{d}\| = \|\Delta_\varphi\partial_\varphi\|. \tag{4}$$

When a total of n photons are emitted into a solid angle of 4π, the expected distances can be further quantified as

$$\|\Delta_{\cos\theta}\partial_{\cos\theta}\vec{d}\| = \|\Delta_\varphi\partial_\varphi\| = \sqrt{\frac{4\pi}{n}}. \tag{5}$$

The lengths of the initial differentials in equation 3 are

$$\|\partial_{\cos\theta}\vec{d}\| = \frac{1}{\sin\theta} \quad \text{and} \quad \|\partial_\varphi\vec{d}\| = \sin\theta. \tag{6}$$

Combining equations 5 and 6, we obtain the desired expected spacings between neighboring photons,

$$\Delta_{\cos\theta} = 2\sqrt{\pi/n}\sin\theta \quad \text{and} \quad \Delta_\varphi = 2\sqrt{\pi/n}\frac{1}{\sin\theta}. \tag{7}$$

For each photon, we must therefore keep track of two positional and two directional differentials as well as the two spacings calculated on its emission. As the photon traverses the scene, its differentials change but the spacings remain constant. We note the opportunity for an optimization here:

When photon differentials are updated during propagation, any scaling previously applied to them is preserved. If the differentials are multiplied by a scaling factor and the

corresponding spacing by its inverse, the resulting footprints are unaffected. We exploit this fact by scaling $\partial_{\cos\theta}\vec{d}$ with $\sin\theta$ and $\partial_{\varphi}\vec{d}$ with $1/\sin\theta$ directly on emission. The initial differentials then become

$$\partial_{\cos\theta}\vec{d} = \begin{pmatrix} -\cos\varphi\cos\theta \\ -\sin\varphi\cos\theta \\ \sin\theta \end{pmatrix} \quad \text{and} \quad \partial_{\varphi}\vec{d} = \begin{pmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{pmatrix}. \quad (8)$$

The corresponding offsets are scaled by the inverses of these factors, yielding

$$\Delta_{\cos\theta} = \Delta_{\varphi} = 2\sqrt{\pi/n}. \quad (9)$$

With the spacings now equal for all photons, we need to only keep track of the differentials per photon, using one global spacing $\Delta = 2\sqrt{\pi/n}$ to generate their footprints.

4.2. Russian roulette

Photon mapping makes a random decision between absorption and reflection whenever a photon encounters a surface. As this Russian roulette is a stochastic sampling, it introduces differentials with respect to an additional parameter per surface interaction. Keeping track of all such differentials is possible and yields path differentials [SW01]. To reduce computational cost, we apply a result of Herzog et al. [HHK*07] instead: It is shown there that footprints related to the inverse probability of a photon's path adapt well to the local hit density. Rather than adding new differentials for each Russian roulette sampling, we therefore rescale the existing differentials by $1/\sqrt{p}$, increasing footprint areas by a factor of $1/p$, the inverse of the reflection probability.

4.3. Diffuse reflection

Reflection by a diffuse surface requires an outgoing direction to randomly be chosen. This introduces two more parameters, $\cos\theta' \in [0,1]$ and $\varphi' \in [0, 2\pi[$, along with new differentials. The number of differentials is kept constant by the following procedure: When a photon hits a diffuse surface, this is interpreted as an absorption and a reemission. The photon is thus considered to originate at the diffuse surface with $\cos\theta'$ and φ' as its only parameters.

The photon's outgoing direction is computed from $\cos\theta'$ and φ' as in equation 2. Its directional differentials then analogously follow equation 3, while the positional differentials are both zero. Before we discuss the computation of spacings $\Delta_{\cos\theta'}$ and $\Delta_{\varphi'}$, we note that by completely discarding the photon's previous differentials, any adaptation to the illumination conditions at its new origin would be lost.

In order to retain this information, we virtually offset the photon's origin along the reverse of its outgoing direction so that where the photon passes the surface, its old and new differentials correspond to footprints of equal area (fig. 2).

Assuming the photon's initial differentials have been scaled to follow equation 8 upon emission, its footprint at the diffuse surface has semiaxes $\Delta\partial_{\cos\theta}\vec{p}$ and $\Delta\partial_{\varphi}\vec{p}$. The potentially skewed elliptical footprint's area then is

$$A = \pi\|\Delta\partial_{\cos\theta}\vec{p} \times \Delta\partial_{\varphi}\vec{p}\|. \quad (10)$$

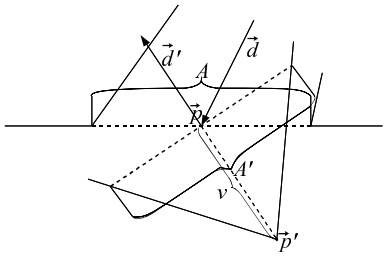

Figure 2: *A photon traveling in direction $\vec{d}$ hits a diffuse surface at $\vec{p}$. On reemission in direction $\vec{d}'$, its origin is offset backwards to $\vec{p}'$ so that the footprint areas A and A' match.*

Offsetting the photon's new origin to $\vec{p}'$, it will have traveled a distance v (fig. 2) as it reaches $\vec{p}$ again. Its positional differentials will therefore have expanded from their initial values of zero to

$$\partial_{\cos\theta}\vec{p} = v\partial_{\cos'\theta}\vec{d}' \quad \text{and} \quad \partial_{\varphi}\vec{p} = v\partial_{\varphi'}\vec{d}'. \quad (11)$$

The photon's new footprint area is thus

$$A' = \pi\|\Delta_{\cos\theta'} v\partial_{\cos\theta'}\vec{d}' \times \Delta_{\varphi'} v\partial_{\varphi'}\vec{d}'\|. \quad (12)$$

The required offset distance is computed by equating A and A', then solving for v to obtain

$$v = \sqrt{\frac{\|\Delta\partial_{\cos\theta}\vec{p} \times \Delta\partial_{\varphi}\vec{p}\|}{\|\Delta_{\cos\theta'}\partial_{\cos\theta'}\vec{d}' \times \Delta_{\varphi'}\partial_{\varphi'}\vec{d}'\|}}. \quad (13)$$

In calculating the values $\Delta_{\cos\theta'}$ and $\Delta_{\varphi'}$, we adopt the concept of global deltas by Suykens and Willems [SW01]. When tracing a large number of independent paths, they suggest the spacings be based on the number of paths reaching each bounce depth and the number of sampling decisions this requires. Treating differential reflection as reemission, each photon path contains exactly two sampling decisions.

The number of paths decreases at higher bounce depths due to Russian roulette. Rather than introducing the parallelization bottleneck of having to count photons at each depth before proceeding with reemission, we expand the surviving photons' footprints to account for Russian roulette (section 4.2) and in sampling, treat their number as remaining approximately constant. We therefore always compute $\Delta_{\cos\theta'}$ and $\Delta_{\varphi'}$ based on the sampling of two parameters by n photons. All considerations of section 4.1 then hold, allowing us to initialize the new directional differentials by equation 8 and retaining a single Δ for footprint computations.

Our method of computing photon differentials after diffuse reflection meets three aims. First, the number of differentials is kept constant, leading to fast computation. Second, the adaptation to illumination conditions at the reemission point is preserved. Third, the photon's footprint continues to adapt as it is traced onward through the scene.

In the context of efficient rendering, we find that full adaptation to illumination conditions is not always desirable. Photon mapping with kth-nearest-neighbor density estimation computes a single bandwidth for all photon hits near a shading point, without distinguishing between those that sparsely represent dim lighting and denser ones corresponding to brigher illumination. By calculating individual footprints for each photon, we achieve better adaptation as on the same surface, photon hits representing different illumination can have largely varying footprints. While this allows illumination to be reconstructed with constant variance, large photon footprints are detrimental to rendering speeds.

We thus prefer footprint sizes that shift the tradeoff toward less bias and more variance as photon hits get more sparse. After photon tracing has completed, a nonlinear scaling is therefore applied to the footprints of all diffusely reflected photons. Each footprint semiaxis is rescaled by a factor $\|\partial_x \vec{p}\|^{-\frac{3}{4}}$ with $\partial_x \vec{p}$ the positional differential that was used to generate it. This compresses the range of footprint radii, reducing adaptation as hits become more sparse. All semiaxes are furthermore uniformly scaled by a smoothing factor s that gives the user control over the overall tradeoff between variance and bias in the scene.

5. BVH photon map

After photon tracing, a hierarchical data structure is built that accelerates the retrieval of photon hits relevant to the query points. Top-down construction begins with the set of all hits, recursively determining *where* to split the set and *whether* to split at all or build a leaf node instead.

A kd-tree of hit positions [Jen96] is typically used. For kth-nearest-neighbor density estimation, the precise volume of space within which a photon hit contributes flux is not a priori known. This makes a data structure that organizes point data the logical choice. Because we know the actual footprint ellipsoids, a BVH, a hierarchy of tight axis-aligned bounding boxes, is a more natural choice. To reconstruct illumination at a query point $\vec{x}$, the BVH is traversed, descending into all nodes that contain $\vec{x}$ and accumulating the contributions of hit points found in the visited leaves. We compare two methods for building such BVHs and show that linear BVH construction [LGS*09] is a good option, combining high retrieval performance with low construction cost.

5.1. Voxel volume heuristic

The original choice of balanced kd-trees yields suboptimal acceleration. Faster photon hit retrieval is achieved by building kd-trees according to the volume heuristic (VVH) [WGS04], recursively minimizing the cost metric

$$C = V_L N_L + V_R N_R. \tag{14}$$

The cost of a node with children L and R is approximated as the sum of costs incurred by visiting them, weighted by the probabilities of having to do so. A child's volume serves as an estimate of the probability while the cost of visiting is assumed to be proportional to the number of hits contained in the node. As the actual regions of contribution are unknown, V_L and V_R are conservative estimates.

This heuristic is directly applicable to our BVH: Assuming a uniform distribution of query points in the scene, the probability of having to visit a node is proportional to its volume. We retain the assumption that the cost of doing so is linear in the number of photon hits the node contains and arrive at equation 14 again. While there are 2^N ways in which N photon hits could be distributed among two children, we follow current practice in ray tracing acceleration [WBS07] and only consider the $3N$ partitionings obtained by sweeping a plane through the photon hits along one of the axes.

5.2. Linear BVH

Linear BVH construction [LGS*09] is a faster alternative to the use of a cost metric. All photon hits are first sorted according to their Morton codes, the positions along a space-filling Morton curve. Because these are computed to a limited bit length, closeby hits may be assigned the same code. BVH construction then logically proceeds by splitting the set of all photon hits according to their Morton codes, recursing from most to least significant bit. As the photon hits are sorted by these codes, each split simply partitions an interval of their sequence into two subintervals.

The construction process is highly parallel. Morton codes can be computed from the overall bounding volume and photon hit position alone, a fast radix sort may be used due to the limited bit length of the codes and all inner nodes can be located in parallel [LGS*09]. In section 6.2, we furthermore describe the fast computation of all node bounding volumes.

5.3. Termination criterion

Both voxel volume heuristic and linear BVH construction address the question *where* to split but leave open the question *whether* to split at all. With the VVH, a natural criterion is to stop when the expected cost of the best split is larger than that of a leaf at the parent node P. Taking into account the cost C_{BV} of testing the query point against two child bounding volumes, this yields the termination criterion $V_P C_{BV} + V_L N_L + V_R N_R > V_P (N_L + N_R)$.

Adjusting the value of C_{BV} lets the splitting terminate at different points. However, we find that all values result in BVHs with poorer query performance than continuing to split until each leaf contains just one hit. We therefore do not use this criterion and instead simply stop splitting when the number of hits reaches a threshold. Table 1 lists frame rates for a number of benchmark scenes rendered with BVHs built using the VVH and linear methods with different leaf size thresholds. The optimal threshold is marked in each case

Scene	Sponza	Scene 6		Conference		Sibenik	Wall
Photons	292k	161k	321k	98k	391k	414k	70k
Voxel volume heuristic							
1	6.45	3.33	0.83	5.78	1.21	4.61	4.12
2	7.57	3.95	0.97	7.02	1.48	5.56	4.51
4	8.08	4.28	1.09	7.36	1.68	6.19	4.72
8	8.25	4.33	1.15	7.26	1.79	6.48	4.92
16	7.98	4.23	1.18	6.94	1.81	6.48	4.98
32	7.40	3.96	1.19	6.45	1.78	6.22	4.99
64	6.75	3.67	1.17	5.86	1.66	5.73	4.80
128	5.99	3.32	1.16	5.24	1.51	5.13	4.39
256	5.11	2.95	1.11	4.55	1.38	4.36	3.93
512	4.24	2.49	1.02	3.78	1.22	3.64	3.36
T_{auto}	5	19	56	7	22	5	42
$\frac{FPS_{auto}}{FPS_{max}}$	100 %	96.3%	98.3%	99.5%	100%	97.7%	99.8%
Linear BVH build							
1	6.40	3.11	0.76	5.79	1.20	4.24	4.00
2	7.19	3.49	0.86	6.49	1.34	4.74	4.23
4	7.55	3.78	0.95	6.77	1.53	5.48	4.42
8	7.60	3.86	1.01	6.80	1.64	5.81	4.64
16	7.44	3.83	1.06	6.52	1.70	5.79	4.70
32	6.82	3.66	1.11	6.04	1.66	5.58	4.59
64	6.20	3.43	1.12	5.52	1.57	5.29	4.23
128	5.51	3.12	1.14	4.88	1.43	4.73	3.72
256	4.69	2.80	1.09	4.25	1.27	4.15	3.20
512	3.99	2.46	1.01	3.50	1.12	3.56	2.56
T_{auto}	5	19	56	7	22	5	42
$\frac{FPS_{auto}}{FPS_{max}}$	100%	98.4%	99.1%	100%	99.4%	97.6%	97.0%

Table 1: *Rendering speeds in FPS using BVHs built using the two methods and different leaf size thresholds; automatic thresholds and the fraction of the maximal speed these yield.*

and can be seen to depend on the scene but also on illumination conditions, indicated by the number of photon hits.

Rather than manually tuning the threshold for each scene or choosing a single compromise value, we propose a heuristic that automatically adjusts the threshold to scene and illumination conditions: T_{auto} is proportional to the ratio of the cumulative areas of all photon hit footprints and all scene surfaces. Intuitively, this corresponds to the number of photon hits that would contribute at each query point $\vec{x}$ if both the query points and photon hits were uniformly distributed on the surfaces. When more photon hits are expected to contribute, larger leaves are built.

The values of T_{auto} and the rendering speeds these yield are also given in the tables. We find that the heuristic reaches an average of 98.80% of the best manually tuned speed for the VVH and similarly, 98.79%, for the linear build. Importantly, the threshold is automatically adjusted when the light source is moved and illumination conditions change.

6. Manycore implementation

We have implemented the algorithms described in this paper using NVidia's CUDA [NVI09]. This is an example of a modern manycore architecture, reaching high computational performance through massive parallelization. Threads running a common CUDA kernel are launched by the host CPU and executed on a GPU's parallel processing cores. In our work, a GeForce GTX280 with 240 cores is used.

16 threads always execute in wide SIMD fashion, incurring a performance penalty if their code paths diverge. These are further grouped into blocks of up to 512 threads among which explicit synchronization and data exchange via a small pool of fast shared memory are possible. Beyond this, accesses to the GPU's global memory feature high latency, limited bandwidth and no caching. High performance is attained by spawning thousands of threads so that when stalls occur, CUDA's hardware multithreading can switch to other waiting threads, hiding memory latency.

6.1. Photon and ray tracing

Recent results [BAGJ08, ZHWG08] show that in CUDA, stack-based traversal of a scene acceleration data structure with threads corresponding to individual rays and no grouping into packets leads to the fastest ray tracing. Latencies incurred by maintaining traversal stacks in global memory are well hidden by CUDA's multithreading. Wide SIMD coherence is automatically exploited where threads follow identical code paths, regardless of whether they are accessing the same part of the scene or not.

We follow these findings in our implementation. As only static geometry is currently used, a high quality kd-tree is prebuilt on the CPU. For rendering, one thread per eye ray is spawned. During photon tracing, 2^{16} threads are used with each responsible for multiple photons. We use a physically plausible modified Phong BRDF [LW94], adding a term for perfectly specular reflection.

6.2. Photon map construction

Lauterbach et al. [LGS*09] find that linear BVH construction is very efficient in CUDA but produces BVHs of significantly lesser quality than cost metric minimization. Table 1 shows that this result from ray tracing acceleration does not apply to BVH photon maps: The linear build with automatically tuned leaf size threshold produces photon maps that on average yield 92.58% of the rendering speed achieved with BVHs built using the voxel volume heuristic. As we rebuild the BVH for each frame and linear construction combines fast build times with good rendering performance, we therefore choose this method.

After photon tracing, all hits are counted and their overall bounding volume as well as the cumulative footprint area are computed using a CUDA parallel scan [HSO07]. We then largely follow the original linear BVH construction algorithm, calculating 30-bit Morton codes, sorting the hits according to these and comparing neighboring codes to produce a *split list* that indicates where the sorted hits are to be split on each hierarchy level.

The BVH should be built by splitting top-down until the leaf size threshold is reached, but we can reverse this process: First, a segmented scan computes bounding volumes for subsets of hits sharing the same Morton code. These subsets cannot be split further and are our candidate leaves. After this operation, we sort the split list by hierarchy level.

The list is then processed bottom-up, for each hierarchy level spawning one thread per split. The thread retrieves the hit counts and bounding volumes of its two children, both of which have been built at deeper hierarchy levels. If their added hit counts are above the threshold, an inner node is built. Otherwise, the two children must be leaves and are fused into one larger leaf. This way, the termination criterion is honored as splits below the threshold leaf size are undone.

As a simplification, we omit the segmented scan in our current implementation and instead loop over each subset of hits sharing a Morton code. We find that because these subsets are small, this does not affect performance.

6.3. Photon hit retrieval

To reconstruct indirect illumination, we accumulate the contributions of all photon hits whose footprint ellipsoids overlap the surface point being shaded. The BVH photon map is recursively traversed, reusing the traversal stack allocated for ray tracing. We compactly store the child bounding boxes with each inner node, reducing the number of global memory reads required. Because in a linear BVH, each leaf corresponds to an interval in the sorted photon hit sequence, a leaf is encoded simply as the index of its first photon hit and a flag bit on the last hit marking its end. All photon hit data is read via the GPU's texturing units as these provide some minimal caching and thus slightly increase performance.

7. Results

We present the results of running our algorithm on an NVidia GTX280 GPU. Benchmark scenes are rendered at 512×512 resolution with ray tracing for direct illumination and photon mapping for two bounces of indirect lighting. One level of recursion is used where eye rays hit reflective objects. Screenshots comparing ray tracing to our technique are shown in figure 3. Note the indirect lighting in otherwise dark areas and the color bleeding between surfaces.

The benchmark scenes are chosen to evaluate a wide range of application scenarios. Sponza and Sibenik are very large, requiring high numbers of photons. Scene 6 and Wall exhibit strong color bleeding. Ring demonstrates that we can reproduce both diffuse and specular illumination. Conference is a medium-sized scene with detailed geometry. Many surfaces are partially specular, such as a wall-sized mirror in Scene 6 and a reflective floor in Sibenik. Specifics of the scenes, benchmark setups and results are given in table 2.

We emit as many photons as possible while maintaining 3–4 frames per second, maximizing image quality. The smoothing factor controlling the tradeoff between variance and bias is adjusted for optimal image quality in each scene. For photons that have undergone only specular reflections, a smoothing factor of 5 is used, ensuring very sharp caustics. We report the average number of photon hits contributing to a shading point, k_{PD}, and the average frame rate, FPS_{PD}. Both are measured over animations in which camera and light move through the scene. This way, different parts of the scenes and varying illumination conditions are considered.

Scene	Sponza	Scene 6	Conf.	Sibenik	Wall	Ring
Tris	76k	804	283k	77k	30	138
Emits	256k	128k	128k	512k	128k	128k
Hits	299k	164k	83k	418k	69k	76k
s	40	17.5	22.5	60	15	10
r_{max}	0.7	0.7	0.7	0.7	0.7	0.7
k_{PD}	175	344	234	224	480	182
FPS_{PD}	3.98	3.50	4.50	3.43	4.22	6.22
k_{PM}	155	319	241	197	511	151
FPS_{PM}	0.60	0.17	0.55	0.46	0.25	0.52

Table 2: *Number of triangles, photon emissions and hits; smoothing factor and maximal footprint radius; average number of photon hits in density estimate and frame rate. For comparison, number of photon hits in density estimate and frame rate using traditional photon mapping.*

As a reference, we have rendered the same scenes using traditional photon mapping with kth-nearest-neighbor density estimation. We find that quality equal to our algorithm is achieved by setting $k_{PM} \approx k_{PD}$. Because we have not implemented a per-frame rebuild of the balanced kd-tree photon map, it is constructed once and the light source then left static throughout the animation. Despite this, rendering with kth-nearest-neighbor density estimation, even though also implemented in CUDA, yields uncompetitive frame rates.

In traditional photon mapping, all photon hits within a predefined maximal radius must initially be taken into consideration. Only when the first k_{PM} hits have been found can the search radius be reduced. This, however, requires a min-heap of the retrieved hits to be built and then continuously updated to determine which ones are nearest. The data structure is too large to fit into CUDA's fast shared memory and residing in slow global memory, degrades performance.

We do not compare our work to approximate techniques as the visual impact of approximations is difficult to quantify and weigh against the frame rate. In comparison to splatting-based approaches, our method has the advantage of also reconstructing indirect lighting at surfaces that can only be seen after specular reflection of the eye ray.

Table 3 shows the distribution of footprint radii computed by our algorithm for each scene which can be seen to vary over a wide range. To avoid frame rate degradation by statistical outliers, the radii are clamped to $r_{max} = 0.7$ before rendering. This value was chosen as it affects only a small number of footprints. An exception is Sibenik where many foot-

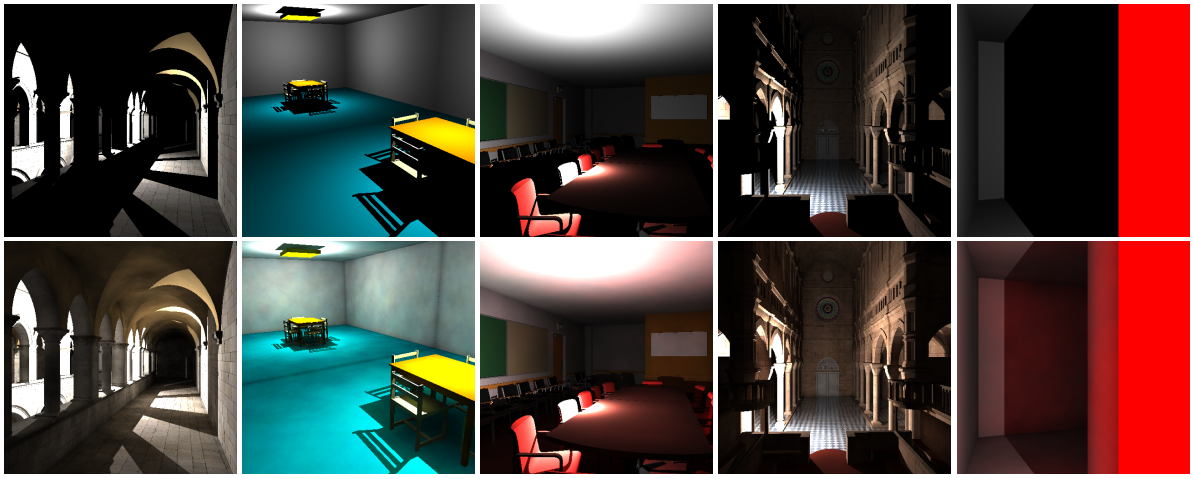

Figure 3: *Screen shots of ray tracing (direct illumination only) and interactive photon mapping (two bounces of indirect lighting) in our benchmark scenes. From left to right: Sponza, Scene 6, Conference, Sibenik and Wall.*

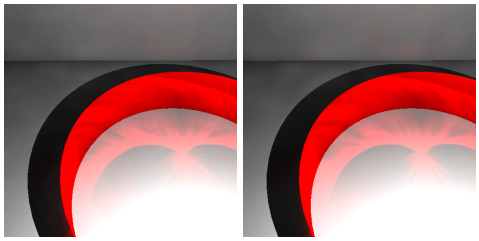

Figure 4: *Diffuse and caustic illumination with constant (left) and Epanechnikov (right) kernel.*

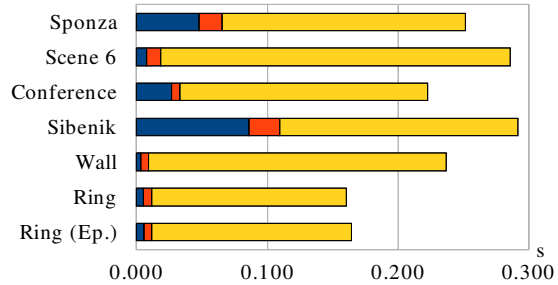

Figure 5: *Average times for photon tracing (blue), BVH construction (red) and rendering (yellow) per frame.*

prints have radii in the range of 0.7. This is because Sibenik is a very large scene approaching the maximum of what our method can currently handle at interactive rates.

In the Ring scene, specular reflection generates the familiar cardioid caustic. Photons are emitted isotropically and specular photons are thus interspersed with large numbers of diffusely reflected global photons. Figure 4 illustrates that our algorithm can faithfully reconstruct both types of illumination from a single BVH photon map. In traditional photon mapping, separate global and diffuse photon maps would have to be used [Jen96]. We also evaluate the impact of using an Epanechnikov kernel instead of a simpler uniform kernel here, finding that image quality is slightly improved while the frame rate drops from 6.24 to 6.12.

Figure 5 shows a breakdown of the average times spent tracing photons, building the BVH and rendering, including illumination reconstruction. The rendering phase clearly dominates, indicating that even better BHVs reducing query times further hold the most promise for future speed-ups.

Radius	Sponza	Scene 6	Conf.	Sibenik	Wall	Ring
0.00 - 0.05	0	0	0	0	0	2263
0.05 - 0.10	0	67	0	0	0	2599
0.10 - 0.15	0	1312	59	4	5239	77738
0.15 - 0.20	0	4924	1508	122	113176	91040
0.20 - 0.25	0	22669	4561	169	42587	6818
0.25 - 0.30	0	141992	9556	152	4338	820
0.30 - 0.35	203	115835	42702	403	800	176
0.35 - 0.40	2996	29015	60651	1064	206	45
0.40 - 0.45	9780	11434	31223	1338	70	14
0.45 - 0.50	35825	4171	13529	2399	21	7
0.50 - 0.55	106333	1767	5652	13039	12	1
0.55 - 0.60	160218	1147	2282	67409	6	1
0.60 - 0.65	145408	838	990	148809	2	1
0.65 - 0.70	83393	669	492	183957	1	0
0.70 - 0.75	37485	558	247	155829	1	0
0.75 - 0.80	17311	464	132	119112	0	0
0.80 - 0.85	8895	399	78	72895	0	0
0.85 - 0.90	4929	348	43	40362	0	0
0.90 - 0.95	2845	304	25	22173	0	0
0.95 - 1.00	1732	266	16	12636	0	0
> 1.00	3108	2406	30	26417	0	0

Table 3: *Distribution of photon footprint radii as computed from photon differentials in each scene.*

8. Conclusions and future work

We have presented a photon mapping algorithm capable of computing diffuse and specular indirect lighting at interac-

tive frame rates, taking advantage of the processing power available on the manycore CUDA architecture. Geometry is static in our current implementation but light source and camera can freely be moved. Contrary to existing CUDA photon mapping implementations, we reconstruct not just a small, focused caustic but compute accurate illumination on all visible surfaces.

Many extensions could further improve our method. Footprints are currently only defined for emission by point light sources, diffuse and perfectly specular reflection. Highly glossy surfaces could be treated approximately as perfectly specular, but a more accurate computation is desirable. Efficient footprint calculation for area light sources will also be a valuable addition.

Our use of random numbers makes the noise in the images incoherent between frames. By using a quasi-random sequence instead, we can improve coherence and eliminate flickering artifacts. Due to its periodicity, the quasi-random Halton sequence can further be used to explore the scene with a small number of pilot photos and then follow up with many similar paths where necessary [GWS04].

To improve speed, we will concentrate on the BVH construction, looking for a more potent heuristic that produces BVHs with better query times. As our technique is highly parallel, we want to experiment with distributing it further across the cores of multiple GPUs. Finally, we want to add support for deformable geometry by applying a fast rebuilding technique to the scene's acceleration data structure.

9. Acknowledgments

We thank the anonymous reviewers for their many valuable comments and suggestions. This work was supported by IRCSET under the Embark Initiative. Models used are courtesy of M. Dabrovic, A. Grynberg, P. Shirley and G. Ward.

References

[BAGJ08] BUDGE B., ANDERSON J., GARTH C., JOY K.: A straightforward CUDA implementation for interactive ray-tracing. In *RT* (2008), p. 178. 2, 6

[CHH02] CARR N., HALL J., HART J.: The ray engine. In *EGGH* (2002), pp. 37–46. 2

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *I3D* (2006), pp. 93–100. 2

[FS05] FOLEY T., SUGERMAN J.: Kd-tree acceleration structures for a GPU raytracer. In *EGGH* (2005), pp. 15–22. 2

[GPSS07] GÜNTHER J., POPOV S., SEIDEL H.-P., SLUSALLEK P.: Realtime ray tracing on GPU with BVH-based packet traversal. In *RT* (2007), pp. 113–118. 2

[GWS04] GÜNTHER J., WALD I., SLUSALLEK P.: Realtime caustics using distributed photon mapping. In *EGSR* (2004), pp. 111–121. 1, 2, 9

[HHK*07] HERZOG R., HAVRAN V., KINUWAKI S., MYSZKOWSKI K., SEIDEL H.-P.: Global illumination using photon ray splatting. In *Eurographics* (2007), pp. 503–513. 1, 2, 4

[HSHH07] HORN D., SUGERMAN J., HOUSTON M., HANRAHAN P.: Interactive k-d tree GPU raytracing. In *I3D* (2007), pp. 167–174. 2

[HSO07] HARRIS M., SENGUPTA S., OWENS J.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*. Addison-Wesley Professional, 2007, pp. 851–876. 2, 6

[Ige99] IGEHY H.: Tracing ray differentials. In *ACM SIGGRAPH* (1999), pp. 179–186. 2, 3

[Jen96] JENSEN H.: *The Photon Map in Global Illumination*. PhD thesis, Technical University of Denmark, Lyngby, Denmark, 1996. 1, 5, 8

[Kel97] KELLER A.: Instant radiosity. In *ACM SIGGRAPH* (1997), pp. 49–56. 2

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast BVH construction on GPUs. In *Eurographics* (2009), pp. 375–384. 2, 5, 6

[LP02] LAVIGNOTTE F., PAULIN M.: A new approach of density estimation for global illumination. In *WSCG* (2002), pp. 263–270. 2

[LP03] LAVIGNOTTE F., PAULIN M.: Scalable photon splatting for global illumination. In *GRAPHITE* (2003), pp. 203–210. 2

[LW94] LAFORTUNE E., WILLEMS Y.: *Using the Modified Phong Reflectance Model for Physically Based Rendering*. Tech. Rep. CW 197, K.U. Leuven, 1994. 6

[MM02] MA V., MCCOOL M.: Low latency photon mapping using block hashing. In *EGGH* (2002), pp. 1–11. 1

[NVI09] NVIDIA CORPORATION: NVIDIA CUDA programming guide version 2.2, 2009. 1, 2, 6

[PDC*03] PURCELL T., DONNER C., CAMMARANO M., JENSEN H., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *EGGH* (2003), pp. 41–50. 2

[PGSS07] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P.: Stackless kd-tree traversal for high performance GPU ray tracing. In *Eurographics* (2007), pp. 415–424. 2

[SB97] STÜRZLINGER W., BASTOS R.: Interactive rendering of globally illuminated glossy scenes. In *EGWR* (1997), pp. 93–102. 1, 2

[SFES07] SCHJØTH L., FRISVAD J., ERLEBEN K., SPORRING J.: Photon differentials. In *GRAPHITE* (2007), pp. 179–186. 1, 2, 3

[SHG09] SATISH N., HARRIS M., GARLAND M.: Designing efficient sorting algorithms for manycore GPUs. In *IPDPS* (2009). 2

[SKUP*09] SZIRMAY-KALOS L., UMENHOFFER T., PATOW G., SZÉCSI L., SBERT M.: Specular effects on the GPU: State of the art. accepted for publication in Computer Graphics Forum, 2009. 1, 2

[SW01] SUYKENS F., WILLEMS Y.: Path differentials and applications. In *EGWR* (2001), pp. 257–268. 2, 4

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM TOG 26*, 1 (2007), 6:1–6:18. 5

[WGS04] WALD I., GÜNTHER J., SLUSALLEK P.: Balancing considered harmful – faster photon mapping using the voxel volume heuristic –. In *Eurographics* (2004), pp. 595–603. 2, 5

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *EGWR* (2002), pp. 15–24. 2

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. In *SIGGRAPH Asia* (2008), pp. 126:1–126:11. 1, 2, 6