# Massively Parallel Path Space Filtering


**Nikolaus Binder, Sascha Fricke, and Alexander Keller**

**Abstract** Restricting path tracing to a small number of paths per pixel in order to render images faster rarely achieves a satisfactory image quality for scenes of interest. However, path space filtering may dramatically improve the visual quality by sharing information across vertices of paths classified as proximate. Unlike screen space approaches, these paths neither need to be present on the screen, nor is filtering restricted to the first intersection with the scene. While searching proximate vertices had been more expensive than filtering in screen space, we greatly improve over this performance penalty by storing, updating, and looking up the required information in a hash table. The keys are constructed from jittered and quantized information, such that only a single query very likely replaces costly neighborhood searches. A massively parallel implementation of the algorithm is demonstrated on a graphics processing unit (GPU).

**Keywords** Integral equations · Real-time light transport simulation · Variance reduction · Hashing · Monte Carlo integration · Massively parallel algorithms

## 1 Introduction

Realistic image synthesis consists of high-dimensional numerical integration of functions with potentially high variance. Restricting the number of samples therefore often results in visible noise, which efficiently can be reduced by path space filtering [19] as shown in Fig. 1.

N. Binder · A. Keller (✉)
NVIDIA, Fasanenstr. 81, 10623 Berlin, Germany
e-mail: akeller@nvidia.com

N. Binder
e-mail: nbinder@nvidia.com

S. Fricke
University of Braunschweig, 38106 Braunschweig, Germany



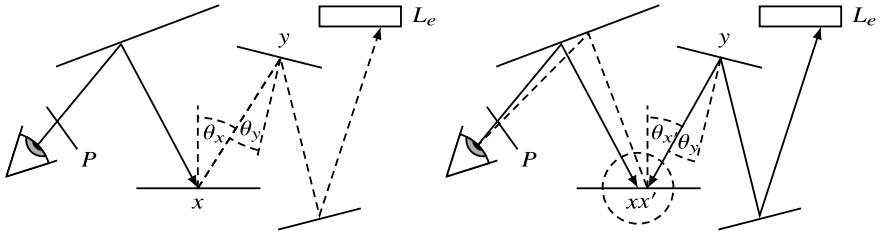
A. Keller (ed.), *Monte Carlo and Quasi-Monte Carlo Methods*, Springer Proceedings in Mathematics & Statistics 387, https://doi.org/10.1007/978-3-030-98319-2_7

**Fig. 1** Path tracing at one path per pixel (top) in combination with hashed path space filtering (middle) very closely approximates the reference solution using 1024 paths per pixel (bottom) and does so with an overhead of about 1.5 ms in HD resolution. Scene courtesy of Epic Games

We improve the performance of path space filtering by replacing costly neighborhood search with averages of clusters in voxels resulting from quantization. Our new algorithm is suitable for interactive and even real-time rendering and it enables many applications trading a controllable bias for a dramatic speedup and noise reduction.

## 2 Light Transport Simulation

As illustrated in Fig. 2, light transport is simulated by tracing rays to create paths that connect the light sources and the camera sensor through a three-dimensional scene that is represented by surfaces and scattering properties of the materials and volumes




(a) forward path tracing, subpath connections, and next event estimation

(b) density estimation and path space filtering

**Fig. 2** Geometry of light transport simulation by path tracing. Starting from the eye on the left through the image plane $P$, a light transport path segment ends in $x$. (**a**) Forward path tracing continues the path until it terminates on the surface of a light source. A path can also be completed by directly connecting to a point $y$ on the surface of a light source (next event estimation) or to a vertex $y$ of the same or a different path (subpath connection). (**b**) Radiance in point $x$ can directly be evaluated by accumulating radiance in vertices $x'$ in a local neighborhood either for density estimation or path space filtering

of participating media. Light transport is ruled by an integral equation: the incident radiance

$$L_i(x,\omega) = L_e(x,\omega) + L_r(x,\omega) \tag{1}$$

is the sum of the emitted radiance $L_e$ and the reflected radiance $L_r$ in direction $\omega$. The following equations show four equivalent ways to formulate the reflected radiance $L_r$ in a point $x$ in direction $\omega_r$ and Fig. 2 illustrates the corresponding sampling techniques:

$$L_r(x,\omega_r)$$

$$= \int_{\mathcal{S}^2_-(x)} L_i(x,\omega) f_r(\omega_r,x,\omega)\cos\theta_x d\omega \tag{2}$$

$$= \int_{\partial V} L_i(x,\omega) f_r(\omega_r,x,\omega)\cos\theta_x \frac{\cos\theta_y}{|x-y|^2} V(x,y) dy \tag{3}$$

$$= \lim_{r(x)\to 0} \int_{\partial V} \int_{\mathcal{S}^2_-(y)} \frac{1_{B(r(x))}\big(x,h(y,\omega)\big)}{\pi r(x)^2} L_i\big(h(y,\omega),\omega\big)\cdot$$

$$\cdot\, f_r\big(\omega_r,h(y,\omega),\omega\big)\cos\theta_y d\omega dy \tag{4}$$

$$= \lim_{r(x)\to 0} \int_{\mathcal{S}^2_-(x)} \frac{\int_{\partial V} 1_{B(r(x))}\big(x,x'\big) w(x,x') L_i(x',\omega) f_r(\omega_r,x,\omega)\cos\theta_{x'} dx'}{\int_{\partial V} 1_{B(r(x))}\big(x,x',r(x)\big) w(x,x') dx'} d\omega \tag{5}$$

Here, the ray tracing function $h(y, \omega)$ determines the closest point of intersection of the scene surface and a ray starting in the point $y$ traced into direction $\omega$. The characteristic function

$$1_{B(r)}(c, x) := \begin{cases} 1 & \|c - x\|_2 < r \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

determines whether the point $x$ is inside the ball $B$ with radius $r$ centered at $c$ or, in other words, whether the points $c$ and $x$ are no further apart than the distance $r$. The sampling techniques are:

**Forward Path Tracing**: While in reality paths of photons start on emissive surfaces, and the camera sensor measures the ones terminating on its surface, simulations often construct paths backwards. Somewhat counter-intuitively, such a simulation in reverse photon direction is called "forward path tracing"—forward in view direction. Equation (2) integrates radiance over the upper hemisphere $S^2_-$ by multiplying the incident radiance $L_i$ from the angle $\omega$ in the point $x$ with the spatio-directional reflectivity $f_r$ for the two angles in the point $x$ and the cosine between the normal of the surface and the incident ray to account for the change of area of the projected solid angle. Inserting the equation into Eq. (1) and the result back into Eq. (2) allows for subsequently prolonging paths and is known as *(recursive) forward path tracing*.

**Next Event Estimation and Subpath Connection**: Equation (3) changes the integration domain to the scene surface $\partial V$ so that a path can be constructed in which the point $x$ connects to any point $y$ on the scene surfaces. The integrand is then extended by the mutual visibility $V$ of the two points $x$ and $y$. The fraction of the cosine of the second angle and the squared distance of the two points accounts for the change of measure. One often refers to this fraction as the *geometric term*. Equation (3) is especially useful since it allows to directly connect to the surface of a light source (*next event estimation*) or any other vertex of any path (*subpath connection*).

**Density Estimation**: Equation (4) again integrates over the scene surface. However, it realizes density estimation by restricting to a local neighborhood in a sphere with radius $r(x)$ using the characteristic function of the ball $1_B$. The density is then obtained by dividing by the area of the circle that stems from the intersection of the ball and the flat surface. For a radius going to zero, the formulation is equivalent to the previous formulations. Density estimation tracing photons from the light sources is referred to as *photon mapping* [17].

**Path Space Filtering:** Similarly, Eq. (5) integrates over a local neighborhood of $x$. In contrast to density estimation, a local weighted average is calculated. This local average is normalized by the integral of all weights in the neighborhood instead of the area of a circle. For a finite non-zero radius, the method trades a certain bias for variance reduction. Due to the filtering of the local average this technique has been introduced as *path space filtering* [19].

While implementations of equations (2)–(5) each individually come with their own strengths and weaknesses and are therefore often combined for robustness in offline rendering applications [12, 15, 21, 35], time constraints of real-time image synthesis as well as advances in *path guiding* [6, 26, 27] lead to the vast majority of implementations only employing equations (2) and (3). Our work aims at a substantial acceleration of path space filtering, resulting in a considerable variance reduction in real-time applications at the cost of a controllable amount of bias.

## *2.1 Previous Work*

Filtering results of light transport simulation is gaining more and more attention in real-time, interactive, and even offline rendering. The surveys by Zwicker et al. [37] and Sen et al. [33] present an overview of recent developments. The fastest approaches use only information available at primary intersections and perform filtering in screen space. Further recent work is based on deep neural networks [1, 4], hierarchical filtering with weights based on estimated variance in screen space [31], or on improving performance by simplifying the overall procedure [24].

Fast filtering is also possible in texture space [28], which requires a bijection between the scene surface and texture space. While this may be tricky already, issues may arise along discontinuities of a parametrization in addition. Furthermore, filtering is restricted to locations on surfaces when operating in texture space, and thus volumetric effects must be filtered separately.

Path space filtering [19], on the other hand, averages contributions of light transport paths in path space, which allows for filtering at non-primary intersections and for a more efficient handling of dis-occlusions during temporal filtering. Multiple Importance Sampling weights, for example those for path space filtering, can be further optimized [36]. However, querying the contributions in path space so far had been significantly more expensive than filtering the contributions of neighboring pixels in screen space. Neglecting the fact that locations that are close in path space are not necessarily adjacent in screen space enables interactive filtering in screen space [11]. As a consequence, filtering in screen space is almost only efficient for primary rays or reflections from sufficiently smooth and flat surfaces. In fact, such filtering algorithms are a variant of a bilateral filtering using path space proximity to determine weights. Sharing information across pixels according to a similarity measure dates back as early as the 1990s [18]. Since then, several variants have been introduced, for example by re-using paths in nearby pixels [2], for filtering by anisotropic diffusion [25] or using edge-avoiding Á-Trous wavelets [7].

Kontkanen et al. explore irradiance filtering, a subset of path space filtering [20]. Spatial caching of shading results in a hash table for walkthroughs of static scenes [8] uses similar methods to the ones presented in this work for the lookup of these results. Again, the method can be seen as a subset of path space filtering: It is restricted to caching diffuse illumination and neither includes filtering nor spatial and temporal integration in an arbitrary number of vertices of a light path.

Hachisuka et al. also use a hash table in a light transport simulation on the GPU [14]. The approach is fundamentally different in two aspects: First, it traces photons and stores them in the hash table for density estimation, while our method averages radiance in vertices from arbitrary light paths. Second, their method implements simple sampling without replacement in voxels, culling all but one photon per voxel. Our method does the exact opposite: It collects radiance from all paths whose vertices coincide in a voxel.

Mara et al. summarize and evaluate a number of methods for photon mapping on the GPU [23]. Their evaluation also includes work from Ma and McCool using hash tables with lists of photons in per voxel [22]. While all examined methods may be used for path space filtering instead of photon mapping, their performance is at least limited by the maintenance of lists.

Havran et al. use two trees for final gathering with photon mapping [16]. The overhead of tree construction and traversal as well as iterating through lists of vertices severely limit the performance in our intended real-time use case.

## 3 Algorithm

Like path space filtering [19], the algorithm receives a set of vertices in which radiance should be filtered to reduce variance. For each vertex complementary information such as the surface normal, the attenuation from the camera along the light transport path up to the vertex, and incident radiance is provided.

A first phase generates the aforementioned data from light transport paths, for example, by path tracing. The second phase averages the radiance of vertices in voxels, as described in Sect. 3.1, and stores and looks up the averages in a hash table, see Sect. 3.2. Finally, for each vertex its associated average is multiplied by its attenuation and accumulated in its respective pixel. Techniques described in Sect. 3.3 reduce the variance of voxels with a small number of vertices. Sect. 3.4 discusses filtering over time for interactive light transport simulation.

### *3.1 Averaging in Voxels*

Rather than averaging the radiance of vertices in a three-dimensional ball, we partition the space of vertex information into voxels and compute one average per voxel. We therefore introduce the concept of a key $k$ of a vertex $x$. The key is a subset of the data stored for a vertex and at least contains the 3-dimensional position of the vertex $x$ along with possibly other vertex data, for example, the surface normal (see Sect. 3.1.1). The voxels result from quantizing the scaled components of a key vector to integers. The scale defines the size of the voxels and is determined by the resolution selection function $s(k)$, which itself may depend on the key (see

Sect. 3.1.2). It balances bias and variance, while remaining quantization artifacts are taken care of by stochastic interpolation (see Sect. 3.1.3).

We define the characteristic function

$$1_V(k,k') := \begin{cases} 1 & \lfloor s(k) \odot k \rfloor = \lfloor s(k') \odot k' \rfloor \wedge s(k) = s(k') \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

telling us whether two key vectors $k$ and $k'$ of vertices $x$ and $x'$, respectively, share the same voxel. Note that sharing the same voxel requires identical scale vectors, too. Here $\odot$ denotes component-wise multiplication and the floor function $\lfloor \cdot \rfloor$ is applied per component. Depending on the number of components selected for a key, voxels are not necessarily three-dimensional, and their extent may vary between components.

Replacing the characteristic function $1_B$ by $1_V$ in path space filtering as given by equation (5), selecting the weight $w(x,x') \equiv 1$, and considering an approximation rather than the limit, we obtain

$$L_r(x,\omega_r) \approx \int_{\mathcal{S}^2_\perp(x)} \frac{\int_{\partial V} 1_V(k,k') L_i(x',\omega) f_r(\omega_r,x,\omega) \cos\theta_{x'} dx'}{\int_{\partial V} 1_V(k,k') dx'} d\omega. \tag{8}$$

If $f_r(\omega_r,x,\omega)$ is separable into $f_r(\omega_r,x) \cdot f_i(x,\omega)$, and $f_i(x,\omega)$ is—at least approximately—constant within the voxel, we will be able to rewrite Eq. (8) as

$$L_r(x,\omega_r) \approx f_r(\omega_r,x) \int_{\mathcal{S}^2_\perp(x)} \frac{\int_{\partial V} 1_V(k,k') L_i(x',\omega) f_i(x',\omega) \cos\theta_{x'} dx'}{\int_{\partial V} 1_V(k,k') dx'} d\omega. \tag{9}$$

In the following, we call the product $L_i(x',\omega) \cdot f_i(x',\omega) \cdot \cos\theta_{x'}$ the *contribution* of the vertex in $x'$. Since all terms of the integrand except for $1_V(k,k')$ are independent of $x$ and $\omega_r$, it is now possible to calculate the integral in Eq. (9) only once for all vertices sharing a voxel. Thereby, the integral calculated per voxel is independent of those components excluded from the key.

### 3.1.1 Construction of Keys

The key $k$ of a vertex is a vector that contains a subset of the information stored with a vertex $x$. This vector incorporates all information required to cluster proximate vertices. The selection of components is critically important for defining the tradeoff between bias and variance reduction: including additional components of the vertex information in the key may reduce the bias, while excluding components allows for the inclusion of more vertices in the integral in Eq. (9), therefore reducing variance. In the following we will give an overview of components (see Fig. 3) that one would typically include in the key.

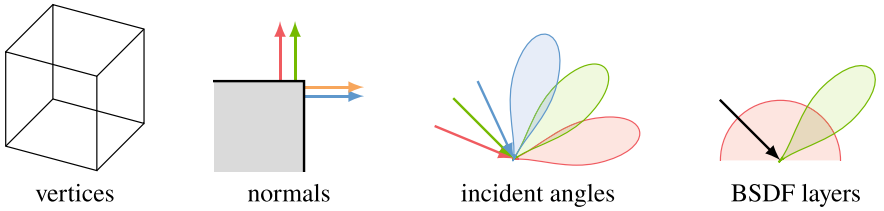



**Fig. 3** The components of the key vector $k$ of a vertex usually consist of the coordinates of the point $x$ of the vertex and optionally may include the normal in $x$, the angle of incidence of radiance, and an identifier of a layer of a bidirectional scattering distribution function (BSDF)

The quality of the approximation in Eq. (9) highly depends on the deviation of $L_i$ in $x'$ from the one in $x$. First and foremost, it is therefore recommended to restrict the world space extent of the voxel by including the position $x$ of the vertex in the key $k$. While $L_i$ is not continuous in practice—for example along shadow edges—the visible error of the approximation decreases with the world space extent of a voxel.

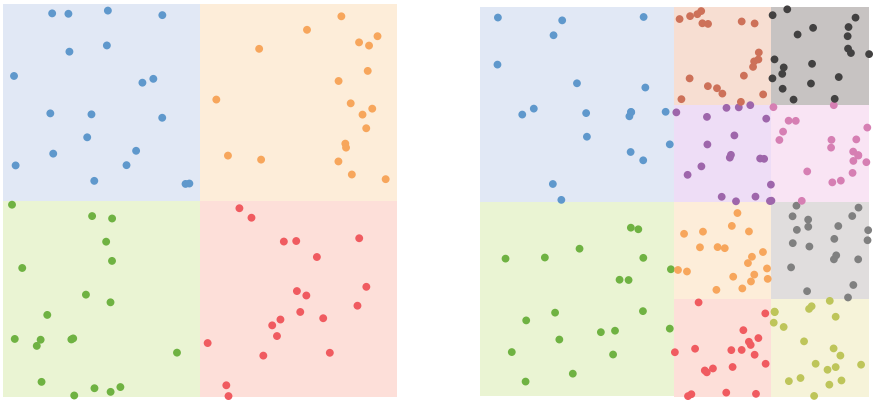
In practice, one can furthermore not guarantee that $\lim_{x' \to x} \cos \theta_{x'} = \cos \theta_x$ due to different surface orientations in the two locations, for example along edges of objects. Including the normal of the surface in the point $x$ in the key avoids a potential "smearing" around edges and "flattening" of surfaces. Representations of unit vectors are surveyed in [5].

Splitting $f_r(\omega_r, x, \omega)$ into $f_r(\omega_r, x) \cdot f_i(x, \omega)$ is not always possible. On highly reflective surfaces, $f_r$ is defined as a Dirac delta function, and filtering is pointless. Therefore, vertices on such surfaces should not be selected in the first place. On glossy surfaces, however, filtering may reduce variance efficiently, again at the cost of a certain bias. While $f_r$ cannot be split on these surfaces without unpleasantly and undesirably changing the visual appearance, partitioning the domain of incident angles and computing separate averages for each interval may be a viable tradeoff. Therefore, for vertices on such surfaces one can append the incident angle to the key in order to identify vertices with similar incident angle.

Materials are often composed of different layers with different properties. Filtering the layers independently offers the opportunity to use different voxel resolutions as well as constructing keys with different components for the different layers. For example, a material consisting of a glossy layer on top of a diffuse layer could only include the angle $\omega_r$ in the key used for filtering the glossy layer since the attenuation of the diffuse layer is independent of it. In turn, the integral for the diffuse layer benefits from including more samples. Appending an identifier of the layer to the key partitions the average into several individual ones, which can be combined later.

### 3.1.2 Adaptive Resolution

In the simplest case, the resolution selection function $s(k)$ is a constant. In practice, it is often advisable to increase the world space extent of a voxel with its distance to the



(a) $s(k) \propto 1$ (b) $s(k) \propto 2^{\lfloor \log_2 x_0 \rfloor}$

**Fig. 4** The characteristic function of a voxel $1_V(k, k')$ is 1 for all $k, k'$ inside the same voxel, here depicted by the same color. A constant resolution selection function $s(k)$ yields uniformly sized voxels (**a**), while increasing $s(k)$ along the $x_0$-axis from left to right shrinks voxels according to distance (**b**) reminiscent of a quad-tree structure

camera sensor along the light transport path. Then, one can filter more aggressively in distant voxels, and increasing the size counteracts the decrease of the density of vertices from paths directly coming from the camera sensor with increasing distance. Our implementation parameterizes $s(k)$ by defining an area on the screen, and then calculates the projected size of the area on the screen using the projection theorem. Fig. 4 illustrates the principle for sets of two-dimensional keys defined by this characteristic function. In practice, keys are at least three-dimensional, i.e. defined by the world space position of the vertex.

The choice of the resolution selection function $s(k)$ is crucial for finding a good tradeoff between visible bias and reduction of variance. Shrinking the voxels by increasing $s(k)$ reduces bias by averaging over a smaller neighborhood, but increases variance. In theory, $s(k)$ should be large in areas with a lot of high frequency detail in $L_i$. In practice, those areas are almost always unknown since $L_i$ is unknown. Fig. 5 shows how a sharp shadow is blurred due to averaging radiance in a large voxel. One would therefore like to adaptively chose a finer resolution along its boundary.

Finite spatial differences may be used in heuristics for adaptation. While their computation either introduces a certain overhead or reduces the number of independent samples, cost may be amortized over frames in environments changing only slowly over time. Note that finite differences only estimate spatial variations of the averages, and one must therefore carefully both choose and adjust such heuristics as well as determine the number samples used for finite spatial differences.

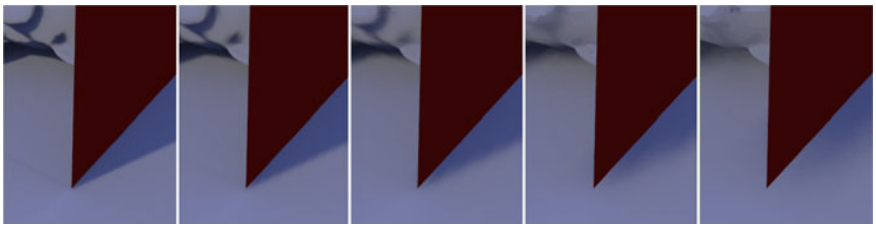


**Fig. 5** Increasing the filter size by lowering the resolution $s(k)$ (left to right) increasingly blurs shadows and also increases the amount of light and shadow leaking. The leftmost image is representative of the reference solution

### 3.1.3 Filter Kernel Approximation by Jittering

The discontinuities of quantization are removed by jittering components of the key, which in fact amounts to approximating a filter kernel by sampling. Jittering depends on the kind of component of the key, for example, positions are jittered in the tangent plane of an intersection, see Algorithm 1. The resulting noise is clearly preferable over the visible discretization artifacts resulting from quantization, as illustrated and shown in Figs. 6 and 7. In contrast to discretization artifacts, noise from jitter is less perceptually pronounced and simple to filter. Note that jittering is not limited to the position; in fact jittering is advantageous for all continuous components of the key that may suffer discontinuities introduced by quantization, such as the normal.

Jittering can be performed either before accumulation or before lookup, leading to similar results. In practice, we suggest using the same jittered key for both accumulation and lookup. Then, the contribution is guaranteed to be included in the average. Furthermore, this allows us to determine the index in the hash table only once for accumulation, and re-use the index later for the lookup of the average without having to calculate the two hash functions and solve potential collisions with linear probing redundantly.

**Fig. 6** Jittering trades quantization artifacts for noise. Left: Note that the resolution $s(k)$ at the jittered location (red) may differ from the one of the original location (green). Spatial jittering hides otherwise visible quantization artifacts (middle): The resulting noise (right) is more amenable to the eye and much simpler to remove by a secondary filter


**Fig. 7** Two-dimensional example for averages in voxels: The noisy input (**a**) is filtered in each vertex by path space filtering (**b**) yielding a splotchy result that is difficult to filter. Instead, the new method filters in each voxel, resulting in block artifacts (**c**). Additionally jittering before accumulation and lookup resolves the artifacts in noise that is simple to filter (**d**)

## *3.2 Accumulation and Lookup in a Hash Table*

The averages in each voxel can be calculated in two different ways: First, each voxel can gather radiance of all included vertices. This process may run in parallel over all voxels and does not require any synchronization. On the other hand, a list of voxels as well as a list of vertices per voxel must be maintained. The second way to calculate the average radiance in a voxel runs in parallel over all vertices: Each vertex atomically adds its contribution $L_i(x', \omega) f_i(x', \omega) \cos\theta_{x'}$ to a running sum of the voxel and increments the counter of the voxel. Dividing the sum by the counter yields the average. While the latter approach requires atomic operations, it does not involve the maintenance of any lists. Furthermore, the summation can be parallelized over the paths or over the vertices, matching the parallelization scheme of typical light transport simulations. Finally, parallelization per path or per vertex exposes more parallelism, and therefore the second approach significantly outperforms the first one on modern *graphics processing units* (GPUs).

Accumulation with the latter approach needs a mapping from the key of a vertex to the voxel with its running sum and counter. Typically, the set of voxels is sparse since vertices are mostly on two-dimensional surfaces in three-dimensional space. Additional components of the key increase sparsity even further.

Hash tables provide such a mapping in constant time for typical sets of keys: First, a hash of the key is calculated using a fast hash function. A modulo operator then wraps this hash into the index range of the table cells. Since both the hash function as well as the modulo operator are not bijective, different keys may be mapped to the same index. Therefore, an additional check for equality of keys is required, and keys must also be stored in the table. Sect. 3.2.1 details a cheaper alternative for long keys.

Upon index collision with a different key, linear probing subsequently increments the index, checking if the table cell at the updated index is empty or occupied by an entry with same key. There exist various other collision resolution methods that improve upon several aspects of linear probing and have proven to be more efficient in certain use cases, especially for hash tables with high occupancy. On the other hand, we do not primarily aim to minimize the size of the hash table, and our experiments show that linear probing comes with a negligible overhead if the table is sufficiently

large. We restrict the number of steps taken for linear probing to avoid performance penalties of extreme outliers, and resort to the unfiltered contribution of the vertex if the number of steps exceeds this limit. So far, our choices for the table size and number of steps so extremely rarely resulted in such failures that further improvement has been deemed unnecessary. Sect. 3.2.2 broadens the application of linear probing from only collision resolution to an additional search for similar voxels.

### 3.2.1 Fingerprinting

Instead of storing and comparing the rather long keys, we calculate a shorter fingerprint [30, 34] from a second, different hash function of the same key and use it for this purpose, see Algorithm 1. Using a sentinel value that cannot be a fingerprint, we can furthermore mark empty cells.

Using fingerprints instead of the full keys is a tradeoff between correctness and performance: In theory, fingerprints of different keys may coincide. In practice, our choice of 32bit fingerprints never caused any collision in our evaluation of several test scenes and numerous simulations. Still, there is a certain probability of failure, and we deliberately favor the tiny probability of a failure over the performance penalty of storing and comparing full length keys.

**Algorithm 1:** Computation of the two hashes used for lookup. Note that the arguments of a hash function, which form the key, may be extended to refine clustering (denoted by "...", see Sect. 3.1.2).

**Input**: Location $x$ of the vertex, the normal $n$, the position of the camera $p_{\text{cam}}$, and the scale $s$ (see Sect. 3.1.1).
**Output**: Hash $i$ to determine the position in the hash table and hash $f$ for fingerprinting.

$l \leftarrow \texttt{level\_of\_detail}(|p_{cam} - x|)$
$x' \leftarrow x + \texttt{jitter}(n) \cdot s \cdot 2^l$
$l' \leftarrow \texttt{level\_of\_detail}(|p_{cam} - x'|)$
$\tilde{x} \leftarrow \left\lfloor \frac{x'}{s \cdot 2^{l'}} \right\rfloor$
$i \leftarrow \texttt{hash}(\tilde{x}, \ldots)$
$f \leftarrow \texttt{hash2}(\tilde{x}, n, \ldots)$

### 3.2.2 Searching by Linear Probing

As shown in Fig. 8, linear probing may be used to differentiate attributes of the light transport path at a finer resolution: For example, normal information may be included in the key handed to the fingerprinting hash function instead of already including it in the main key. This allows one to search for similar normals by linear probing.

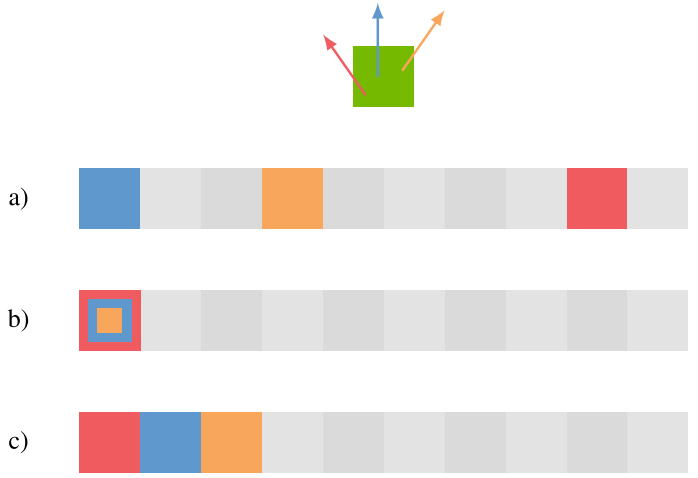



**Fig. 8** Instead of including normals in the key (**a**) to differentiate contributions whose vertices fall into the same voxel (**b**), the fingerprint of a key may include normal information. This allows one to differentiate normal information by linear probing as shown in (**c**)

Note that due to completely unrelated voxels also possibly occupying neighboring cells in the hash table, searching with linear probing must go beyond those with mismatching fingerprints. Therefore, the method works best if both the number of additional contributions as well as the occupancy of the hash table are low.

## *3.3 Handling Voxels with a low Number of Vertices*

Often, there exists a tiny number of voxels that contain very few vertices. Examples of such voxels include those that are only slightly overlapped by objects. Sects. 3.3.1 and 3.3.2 present two approaches that reduce variance in such voxels at an almost negligible cost.

### 3.3.1 Neighborhood Search

Accumulation in voxels by using quantized keys and a hash table requires one `atomicAdd` operation for each component of the radiance of each vertex as well as one `atomicAdd` or `atomicInc` operation for updating the counter of the voxel. The final average is computed with one additional non-atomic read operation per component for the sum and one for the counter. So, as long as access to the hash table happens in constant time, the calculation of the average also takes constant time. This is in sharp contrast to existing methods that compute sums or averages in

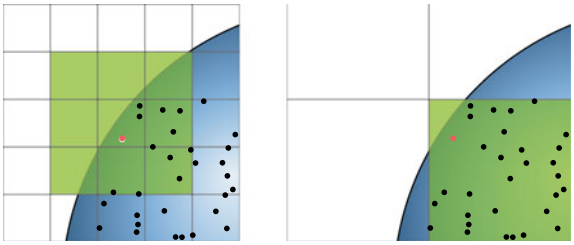


**Fig. 9** Instead of searching the $\mathbf{n}^{\mathbf{d}}$ neighborhood in $d$ dimensions (left), we utilize clustering resulting from quantization at a lower resolution to accumulate contributions, which allows for a single look up (right). We only resort to an additional neighborhood search in the rare case that the number of vertices in a voxel falls below a certain threshold

a spatial neighborhood which for each vertex takes linear time for a search within a given radius or (typically) logarithmic time for a fixed number of neighboring vertices.

Even if primarily only one average per voxel is computed and looked up, searching for neighboring voxels can still be valuable: In theory, increasing the resolution $s(k)$ and additionally searching for neighboring voxels may result in variance reduction similar to the one at a lower resolution, however then with a lower bias. Yet, the number of neighbors grows exponentially with the number of components of the key. Hence, such an approach is ruled out by the time constraints of real-time applications. Fig. 9 shows a comparison between neighborhood search and clustering by selecting a coarser resolution.

Neighborhood search is still very valuable as a fallback: If the number of vertices in a voxel falls below a certain threshold, we allow for an additional search. We observe that given an appropriate threshold, the number of such voxels is so low that the overhead is negligible while the perceptual improvement is clearly visible.

### 3.3.2 Multiresolution Accumulation

A special treatment of voxels with averages from only very few vertices is important for visual fidelity: Even if the number of voxels with a high variance is very low, they may be very visible, especially since their appearance is so different from the rest. Such voxels are very often found on the silhouette of objects. While one may not be able to identify them in still images, they become especially visible across frames. Besides searching in a local neighborhood to reduce variance in these cases (see Sect. 3.3.1), selecting a coarser resolution also effectively increases the number of vertices in the local average—at the price of an increased bias. Using more than one resolution at a time avoids the chicken-and-egg problem that arises from first selecting an appropriate resolution, and then, after accumulation according to this resolution, determining that it has been set too high or too low.

For simulations in interactive scenarios, one may also select the resolution based on information from previous frames, see Sect. 3.4.1.

## *3.4 Accumulation over Time*

Reusing contributions over time dramatically increases efficiency. Attention should be paid to pitfalls and aspects of efficiency: Sect. 3.4.1 details the differences and similarities between filtering and integration across frames, Sect. 3.4.2 explains the handling of resolution changes across frames, and Sect. 3.4.3 is concerned with the amount of information stored over time to avoid running out of memory in the hash table.

### 3.4.1 Temporal Filtering and Temporal Integration

For static scenes, the averages will converge with an increasing number of frames. For dynamic environments, maintaining two sets of averaged contributions and combining them with an exponential moving average $c = \alpha \cdot c_{\text{old}} + (1 - \alpha) \cdot c_{\text{new}}$ is a common tradeoff between convergence and temporal adaptivity.

However, combining the averages $c_{\text{old}}$ and $c_{\text{new}}$ by an exponential moving average is not equivalent to temporal integration. Especially averages in voxels with relatively few samples do not converge. In fact, denoting $N_{\text{old}}$ and $N_{\text{new}}$ the number of vertices in the voxel in the previous and current frame, and setting $\alpha := \frac{N_{\text{old}}}{N_{\text{old}}+N_{\text{new}}}$ correctly integrates across frames. On the other hand, temporal integration is only possible if the underlying setting, including lighting conditions and object positions, remains unchanged across frames.

A first, simple heuristic is to accumulate samples over time up to a certain degree. This may be implemented using a fixed threshold for the number of samples and accumulating samples across frames until reaching it. Note that this heuristic is completely unaware of changes in the scene.

A second, more expensive heuristic builds upon temporal finite differences: A number of paths is re-evaluated with the same parameters, and the difference of their contribution to the original ones allows one to detect changes that affect the current voxel. Similar to the spatial finite differences in Sect. 3.1.2, the additional cost may be amortized across frames, and the number of samples used for finite differences as well as their influence on the balance between temporal adaptation and temporal integration must be carefully optimized. Note that averaging in voxels can be used for the samples used for finite differences, too. Figure 10 shows a comparison of temporal filtering, temporal integration and a hybrid that blends both based on temporal finite differences. A similar approach for screen space filtering has been explored in detail by Schied et al. [32].

**(a)** temporal filtering **(b)** temporal integration **(c)** hybrid

**Fig. 10** In a scene with a moving light source, temporal filtering using an exponential moving average blurs shadow boundaries (**a**), and temporal integration averages out the entire shadow (**b**). Adaptively blending $\alpha$ between zero (for large temporal differences) and $\alpha := \frac{N_{\text{old}}}{N_{\text{old}}+N_{\text{new}}}$ (for no temporal differences) combines both and preserves sharp shadow boundaries (**c**). Note that jittering has deliberately been disabled here to enable simple distinction

### 3.4.2 Changing Resolution Across Frames

In many simulations, the camera is dynamic, and therefore the resolution of a voxel may change across frames if it depends on the position of the camera. Then accumulated contributions in a voxel at one resolution must be copied to a voxel at either a higher or lower resolution.

If the resolution in the new frame decreases, one can simply add up the contributions in voxels of higher resolution. Since we store sums and counters, both only need to be added individually.

If the resolution in the new frame increases, the contributions in voxels of lower resolution must be distributed to voxels at a higher resolution. Due to the lack of resolution, this case is much nuanced: On the one hand, using already collected contributions lowers variance, but on the other hand, the coarser resolution may become unpleasantly visible. One therefore needs to find a good compromise between the two, and set $\alpha$ in the exponential moving average accordingly.

Finite spatial or temporal differences can also be filtered across frames in a similar way.

### 3.4.3 Voxel Eviction Strategy

Evicting contributions of voxels which have not been queried for a certain period of time is necessary for larger scenes and changing camera. Besides the least recently used (LRU) eviction strategy, heuristics based on longer term observations are efficient.

A very simple method relies on replacing the most significant bits of the fingerprinting hash by a priority composed of for example the number of vertices in the voxel and last access time during temporal filtering. Thus the pseudo-randomly hashed least significant bits guarantee eviction to be uniformly distributed across the

scene, while the most significant bits ensure that contributions are evicted according to priority. This allows collision handling and eviction to be realized by a single `atomicMin` operation.

## 4 Results and Discussion

While filtering contributions at primary intersections with the proposed algorithm is quite fast, it only removes some artifacts of filtering in screen space. However, hashed path space filtering has been designed to target real-time light transport simulation: It is the only efficient option when screen space filtering fails or is not available, for example, when filtering after the first diffuse bounce (Fig. 11).

Filtering on non-diffuse surfaces requires to include additional parameters in the key and heuristics such as increasing the quantization in areas with non-diffuse materials to reduce the visible artifacts.

Filtering, and especially accumulating contributions, is always prone to light and shadow leaking (see Fig. 5), which is the price paid for performance. Some artifacts may be ameliorated by employing suitable heuristics as reviewed in [19, Sect. 2.1] and in Sect. 3.1.2.

The new algorithm filters incoherent intersections at HD resolution ($1920 \times 1080$ pixels) in about 3ms on an NVIDIA Titan V GPU. Filtering primary intersections doubles the performance due to the more coherent memory access patterns.

The image quality is determined by the filter size, which balances noise versus blur as shown in Fig. 5. Both the number of collisions in the hash table and hence the performance of filtering depend on the size of the voxels, too. We found specifying the voxel size by $s_0$-times the projected size of a pixel most convenient. Since $s_0$ specifies the tradeoff between bias and variance reduction, its value highly depends on the scene and variance. Values between 4 and 16 may serve as a good starting point.

Note that maximum performance does not necessarily coincide with best image quality. The hash table size is chosen proportional to the number of pixels at target resolution such that potentially one vertex could be stored per pixel. In practice, filtering requires multiple vertices to coincide in a voxel, and therefore the occupancy of the hash table is rather low. Such a small occupancy improves the performance as it lowers the number of collisions and time spent for collision resolution.

While path space filtering dramatically reduces the noise at low sampling rates (see Fig. 1), some noise is added back by spatial jittering. Instead of selecting the first sufficiently diffuse vertex along a path from the camera, path space filtering can be applied at any vertex. For example, filtering at the second sufficiently diffuse vertex as shown in Fig. 11 resembles final gathering or local passes [17]. Furthermore, it is possible to filter in several vertices along the path at the same time. In fact, path space filtering trades variance reduction for controlled bias and is orthogonal to other filtering techniques. We therefore abstain from comparisons with these:

**Fig. 11** Indirect illumination by hashed path space filtering only at the second bounce: At 16 paths per pixel (left), the variance of the integrand is dramatically reduced (right)

temporal anti-aliasing and complimentary noise filters in screen space are appropriate to further reduce noise [24]. A local smoothing filter [31] can even help reduce the error inherent in the approximation.

## 5 Conclusion

In combination with hardware accelerated ray tracing, our variance reduction technique enables visual fidelity of light transport simulation in real-time. Relying on only a few synchronizations during accumulation, path space filtering based on hashing scales on massively parallel hardware. Both accumulation as well as queries run in constant time per vertex. Neither the traversal nor the construction of a hierarchical spatial acceleration data structure is required. At the same time, the simplistic algorithm overcomes many restrictions of screen space filtering, does not require motion vectors, and enables variance reduction beyond the first intersection of a light transport path, including non-diffuse surfaces.

The hashing scheme still bears potential for improvement. For example, important hashes could be excluded from eviction by reducing the resolution that is accumulating their contributions at a coarser level. Other than selecting the resolution by the length of the path, path differentials and variance may be used to determine the appropriate resolution.

Besides the classic applications of path space filtering [19, Sect. 3] like multi-view rendering, spectral rendering, participating media, and decoupling anti-aliasing from shading, the adaptive hashing scheme can be applied to photon mapping [13, 17] and

irradiance probes in reinforcement learned importance sampling [6] in combination with final gathering. Since the first publication of this work as a technical report evolutions of the presented method have improved the efficiency of real-time ambient occlusion in massive scenes [9, 10], reinforcement learned importance sampling [29], and reservoir-based importance resampling [3] in light transport simulation.

**Acknowledgements** The authors thank Petrik Clarberg for profound discussions and comments.

## References

1. Bako, S., Vogels, T., McWilliams, B., Meyer, M., Novák, J., Harvill, A., Sen, P., Derose, T., Rousselle, F.: Kernel-predicting convolutional networks for denoising Monte Carlo renderings. ACM Trans. Graph. **36**(4), 97:1–97:14 (2017). https://doi.org/10.1145/3072959.3073708
2. Bekaert, P., Sbert, M., Halton, J.: Accelerating path tracing by re-using paths. In: Debevec, P., Gibson, S. (eds.) Eurographics Workshop on Rendering. The Eurographics Association (2002). https://doi.org/10.2312/EGWR/EGWR02/125-134
3. Boissé, G.: World-space spatiotemporal reservoir reuse for ray-traced global illumination. ACM Trans. Graph. **40**(6) (2021)
4. Chaitanya, C., Kaplanyan, A., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., Aila, T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. ACM Trans. Graph. **36**(4), 98:1–98:12 (2017). https://doi.org/10.1145/3072959.3073601
5. Cigolle, Z., Donow, S., Evangelakos, D., Mara, M., McGuire, M., Meyer, Q.: A survey of efficient representations for independent unit vectors. J. Comput. Graph. Tech. (JCGT) **3**(2), 1–30 (2014). http://jcgt.org/published/0003/02/01/
6. Dahm, K., Keller, A.: Learning light transport the reinforced way. In: Owen, A., Glynn, P. (eds.) Monte Carlo and Quasi-Monte Carlo Methods. MCQMC 2016. Proceedings in Mathematics & Statistics, vol. 241, pp. 181–195. Springer, Berlin (2018)
7. Dammertz, H.: Acceleration methods for ray tracing based global illumination. Ph.D. thesis, Universität Ulm (2011)
8. Dietrich, A., Slusallek, P.: Adaptive spatial sample caching. In: 2007 IEEE Symposium on Interactive Ray Tracing, pp. 141–147 (2007). https://doi.org/10.1109/RT.2007.4342602
9. Gautron, P.: Real-time ray-traced ambient occlusion of complex scenes using spatial hashing. In: Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks, SIGGRAPH '20. Association for Computing Machinery, New York, USA (2020)
10. Gautron, P.: Practical spatial hash map updates. In: Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR. Vulkan, and OptiX, pp. 659–671. Apress, Berkeley, CA (2021)
11. Gautron, P., Droske, M., Wächter, C., Kettner, L., Keller, A., Binder, N., Dahm, K.: Path space similarity determined by Fourier histogram descriptors. In: ACM SIGGRAPH 2014 Talks, SIGGRAPH '14, pp. 39:1–39:1. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2614106.2614117
12. Georgiev, I., Křivánek, J., Davidovič, T., Slusallek, P.: Light transport simulation with vertex connection and merging. ACM Trans. Graph. **31**(6), 192:1–192:10 (2012)
13. Hachisuka, T., Jensen, H.: Stochastic progressive photon mapping. In: SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers, pp. 1–8. ACM (2009)
14. Hachisuka, T., Jensen, H.: Parallel progressive photon mapping on GPUs. SIGGRAPH Sketches (2010). https://doi.org/10.1145/1899950.1900004
15. Hachisuka, T., Pantaleoni, J., Jensen, H.W.: A path space extension for robust light transport simulation. ACM Trans. Graph. **31**(6) (2012). https://doi.org/10.1145/2366145.2366210

16. Havran, V., Herzog, R., Seidel, H.P.: Fast final gathering via reverse photon mapping. Comput. Graph. Forum **24**(3), 323–332 (2005)
17. Jensen, H.: Realistic Image Synthesis Using Photon Mapping. AK Peters (2001)
18. Keller, A.: Quasi-Monte Carlo Methods for Photorealistic Image Synthesis. Ph.D. thesis, University of Kaiserslautern, Germany (1998)
19. Keller, A., Dahm, K., Binder, N.: Path space filtering. In: Cools, R., Nuyens, D. (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2014, pp. 423–436. Springer, Berlin (2016)
20. Kontkanen, J., Räsänen, J., Keller, A.: Irradiance filtering for Monte Carlo ray tracing. In: Talay, D., Niederreiter, H. (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2004, pp. 259–272. Springer, Berlin (2004)
21. Lafortune, E., Willems, Y.: Bi-directional path tracing. In: Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics' 93) (1998)
22. Ma, V., McCool, M.: Low latency photon mapping using block hashing. In: Ertl, T., Heidrich, W., Doggett, M. (eds.) SIGGRAPH/Eurographics Workshop on Graphics Hardware. The Eurographics Association (2002). https://doi.org/10.2312/EGGH/EGGH02/089-098
23. Mara, M., Luebke, D., McGuire, M.: Toward practical real-time photon mapping: efficient GPU density estimation. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'13) (2013). https://casual-effects.com/research/Mara2013Photon/index.html
24. Mara, M., McGuire, M., Bitterli, B., Jarosz, W.: An efficient denoising algorithm for global illumination. In: ACM SIGGRAPH/Eurographics High Performance Graphics, p. 7 (2017). http://casual-effects.com/research/Mara2017Denoise/index.html
25. McCool, M.: Anisotropic diffusion for Monte Carlo noise reduction. ACM Trans. Graph. **18** (2002). https://doi.org/10.1145/318009.318015
26. Müller, R., McWilliams, B., Rousselle, F., Gross, M., Novák, J.: Neural importance sampling. ACM Trans. Graph. **38**(5), 145:1–145:19 (2019)
27. Müller, T., Gross, M., Novák, J.: Practical path guiding for efficient light-transport simulation. Comput. Graph. Forum **36**(4), 91–100 (2017)
28. Munkberg, J., Hasselgren, J., Clarberg, P., Andersson, M., Akenine-Möller, T.: Texture space caching and reconstruction for ray tracing. ACM Trans. Graph. **35**(6), 249:1–249:13 (2016). https://doi.org/10.1145/2980179.2982407
29. Pantaleoni, J.: Online path sampling control with progressive spatio-temporal filtering (2020)
30. Rabin, M.: Fingerprinting By Random Polynomials. Center for Research in Computing Technology, Harvard University, Technical report (1981)
31. Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A., Salvi, M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In: Proceedings of High Performance Graphics, HPG '17, pp. 2:1–2:12. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3105762.3105770
32. Schied, C., Peters, C., Dachsbacher, C.: Gradient estimation for real-time adaptive temporal filtering. Proc. ACM Comput. Graph. Interact. Tech. **1**(2) (2018)
33. Sen, P., Zwicker, M., Rousselle, F., Yoon, S.E., Kalantari, N.: Denoising your Monte Carlo renders: recent advances in image-space adaptive sampling and reconstruction. In: ACM SIGGRAPH 2015 Courses, SIGGRAPH '15, pp. 11:1–11:255. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2776880.2792740
34. Slaney, M., Casey, M.: Locality-sensitive hashing for finding nearest neighbors [lecture notes]. IEEE Signal Process. Mag. **25**(2), 128–131 (2008). https://doi.org/10.1109/MSP.2007.914237
35. Veach, E.: Robust Monte Carlo Methods for Light Transport Simulation. Ph.D. thesis, Stanford University (1997)
36. West, R., Georgiev, I., Gruson, A., Hachisuka, T.: Continuous multiple importance sampling. ACM Trans. Graph. (Proceedings of SIGGRAPH) **39**(4) (2020)
37. Zwicker, M., Jarosz, W., Lehtinen, J., Moon, B., Ramamoorthi, R., Rousselle, F., Sen, P., Soler, C., Yoon, S.E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. Comput. Graph. Forum **34**(2), 667–681 (2015)