EG UK Computer Graphics & Visual Computing (2019)
G. K. L. Tam and J. C. Roberts (Editors)

Hash-based Hierarchical Caching for Interactive Previews in Global Illumination Rendering

T. Roth[1,2], M. Weier[1,3], P. Bauszat[4], A. Hinkenjann[1], Y. Li[2]

[1]Hochschule Bonn-Rhein-Sieg University of Applied Sciences
[2]Brunel University London, [3]Saarland University, [4]TU Delft

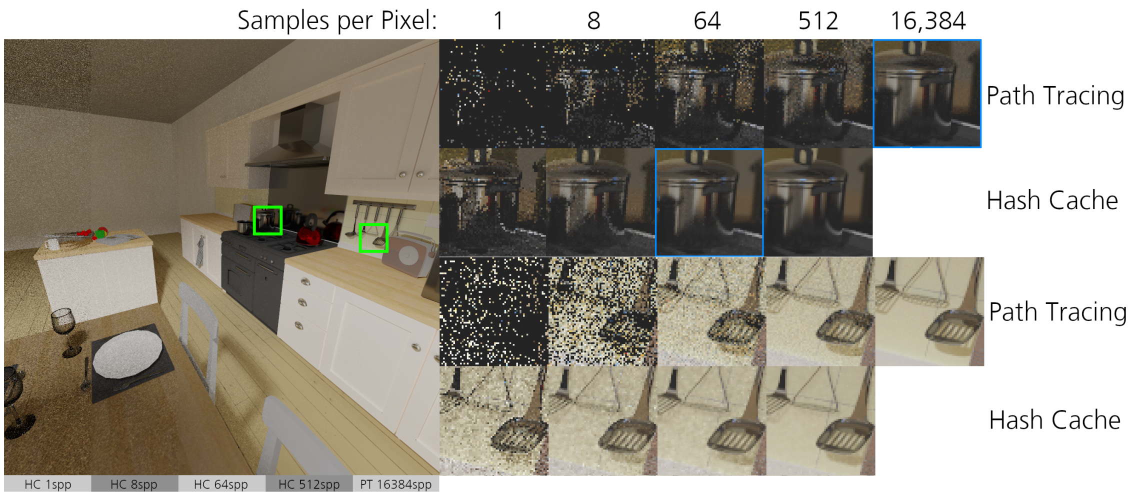

Figure 1: *Comparison of images generated with pure path tracing and with the support of HashCache.* ***Left:*** *The full image subdivided horizontally into areas rendered with HashCache at 1spp, 8spp, 64spp, 512spp, and Path Tracing at 16,384 spp. Insets in green are magnified on the right for the various settings.* ***Right:*** *Zoomed-in rendering for comparison, rendered with Path Tracing and HashCache (with three more frames in the sequence rendered before as a warm-up phase). For the upper inset, the HashCache system already yields a quality at 64spp (512spp for individual rendering) that is at least on par with the path traced image at 16,384spp.*

Abstract

Modern Monte-Carlo-based rendering systems still suffer from the computational complexity involved in the generation of noise-free images, making it challenging to synthesize interactive previews. We present a framework suited for rendering such previews of static scenes using a caching technique that builds upon a linkless octree. Our approach allows for memory-efficient storage and constant-time lookup to cache diffuse illumination at multiple hitpoints along the traced paths. Non-diffuse surfaces are dealt with in a hybrid way in order to reconstruct view-dependent illumination while maintaining interactive frame rates. By evaluating the visual fidelity against ground truth sequences and by benchmarking, we show that our approach compares well to low-noise path traced results, but with a greatly reduced computational complexity allowing for interactive frame rates. This way, our caching technique provides a useful tool for global illumination previews and multi-view rendering.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Ray tracing;*** *Image-based rendering;*

1. Introduction

Global Illumination (GI) rendering based on Monte Carlo (MC) methods allows for generating images of astonishing realism that can often hardly be distinguished from real photographs. Even though these methods have been around for a long time, their computational complexity remains a major challenge. Ray-based approaches like *path tracing* may require a considerable number of rays to be traced through a scene to determine an approximate solution for the Rendering Equation [Kaj86]. This process can take anywhere from mere seconds to hours until a noise-free result emerges. In recent years, numerous methods have been introduced for filtering the noise from images rendered with low sample counts. Al-

DOI: 10.2312/cgvc.20191261

though these methods often result in visually pleasing, noise-free images, rendering the GI of a scene involves computing the light transport at many surfaces that are not directly visible in image space. Reusing this information for the computation of successive frames can increase visual quality and shorten rendering times at the same time. Therefore, we introduce the *HashCache*, a hierarchical world-space caching method for GI rendering of static scenes, based on a linkless octree [CJC*09]. Using a hash-based approach makes it possible to perform the reconstruction of cached illumination in constant time, depending only on the actual screen resolution (assuming that the visible geometry is known). Despite only caching diffuse illumination, our system explicitly supports non-diffuse materials through a hybrid reconstruction scheme. Having similarities to photon mapping [Jen96, SJ09], it can be interpreted as a kind of approximate final gathering step before the actual reconstruction. Compared to precomputed radiance transfer, our preprocessing time is much shorter as we only need to determine geometric cell occupations. In order to compute noise-free results without visible quantization artifacts, we employ a spatial jittering method inspired by [BFK18] as well as a basic cross-bilateral reconstruction method. Finally, we present image quality comparisons, performance benchmarks, and an analysis of memory requirements, showing the practicability of our approach.

2. Related Work

Caching samples is a proven tool for several computer graphics applications [SYM*12]. Currently, most methods try to exploit the temporal coherence in image-space. However, caching in world-space has the advantage to prolong the validity of samples in case of view-dependent (dis-)occlusions and surfaces that are not directly visible. This is especially beneficial for methods that handle indirect GI. In the following section, we provide an overview of the relevant research related to our system, including the fields of world-space sample caching and interactive GI.

World-space Sample Caching Early work by Ward et al. [WRC88] uses an octree to cache irradiance values in world-space. This approach is easy to implement when rays are cast sequentially. However, updating a data structure is challenging when data is accessed in a parallel fashion. In work by Christensen et al. [CB04], a sparse octree is suggested as a 3D Mipmap to store irradiance values. Radiance Caching [KGPB08] is a method for accelerating GI computation in scenes with low-frequency glossy BRDFs based on spherical harmonics. Higher-frequency content is supported in work by Omidvar et al. [ORC*15], using Equivalent Area Light Sources. However, all the methods presented so far are offline processes for non-interactive systems.

The *Render Cache* [WDP99] is an interactive caching and reprojection technique with adaptive sampling. In order to be efficient, only samples within the view frustum are reprojected from one frame to the next. GI computations on surfaces outside the current frame's frustum are not cached at all. Dietrich et al. [DSS06] propose a cache that employs a hash map as the spatial index structure to store shading and illumination without the need for a preprocessing step. While the presented work shares many similarities to this approach, the hashing mechanism by Dietrich et al. cannot be easily ported to highly parallel systems such as the GPU. Moreover, it does not provide a level-of-detail mechanism. Hachisuka and Jensen [HJ10] describe how to use spatial hashing for constructing photon maps on the GPU. Their method stores a single photon stochastically instead of storing lists or aggregations. This allows the approach to ignore hash collisions but limits the sample set size and expressiveness. Caching samples in world space is either computationally demanding or only worked for a limited set of samples. Hence, there is a need for fast world-space sample caching techniques that allow updating aggregated samples in parallel.

Interactive Global Illumination Ritschel et al. [RDGK12] present a comprehensive summary of the major challenges in interactive GI. Their work includes the underlying theoretic aspects, phenomena, and methods for the actual rendering task. A real-time approach for approximating GI is presented by Thiedemann et al. [THGM11]. While diffuse near-field GI is rendered at high visual fidelity, voxel-based visibility computation causes glossy reflections not to be handled well. Crassin et al. [CNS*11] provide a technique for real-time GI, based on approximate cone tracing using a sparse voxel octree. However, the appearance greatly differs between renderings generated with their method and unbiased results. Mara et al. [MLM13] present a method for efficient density estimation for photon mapping on the GPU. While their work also contains information on using a hash map as their data structure, it is strictly limited to photon mapping. Our approach, however, can be used with several GI methods. A direct-to-indirect transport technique is described in [SL17]. While the achieved visual quality is convincing, this method requires a relatively long precomputation time. Our method works magnitudes faster to preprocess the scene's geometry and to build the hashing structure.

In addition to the methods presented so far, image-space filtering techniques have been introduced to allow for real-time rendering with extremely low sampling rates (< 16 spp) whilst maintaining acceptable image quality. Such techniques make use of fast approximations of joint bilateral filtering [DSHL10], Guided Image Filtering [BEMA11], or adaptive manifolds [GO12]. Schied et al. [SKW*17, SPD18] suggest methods for generating temporally stable image sequences from GI at one sample per pixel. The effective sample count available to their approach is increased by accumulating samples temporally. All of the methods above greatly benefit from exploiting temporal coherence in image-space. We argue that a world-space sampling caching technique can further improve the image quality of such filtering methods, especially in complex scenes with many occluding surfaces and arbitrary views.

3. Method

In this section, we give an overview of the employed caching structure and describe how it can be used to cache the data generated by stochastic rendering methods. Subsequently, we give more details on how samples are generated during the rendering process in order to reuse recursively generated hitpoints. It is also shown how the actual cache updates are performed. Eventually, we provide information about the reconstruction process, including the support of non-diffuse materials.

3.1. Cache Structure

MC-based rendering methods provide the means to solve the Rendering Equation [Kaj86] numerically. In our implementation, a straightforward path tracer with next event estimation and multiple importance sampling is used for computing the illumination data. The path tracing process generates millions of randomly and sparsely distributed hitpoints located on the scene geometry in each iteration. Consequently, a data structure that allows for efficient caching of such data must allow for querying large amounts of randomly distributed keys at a high performance. The core of our HashCache system is a linkless octree [CJC*09], consisting of a number of hash maps implemented with Cuckoo Hashing [Alc11], where the latter allows for a worst-case constant lookup time. Being a hierarchical data structure, the HashCache allows for choosing the level of the hierarchy whose resolution most closely resembles the projected pixel size in object space, hiding subsampling artifacts effectively.

While the hash-based octree representation is a compact structure, there still is a trade-off between memory consumption and access time. In order to construct the compact hash map, all cells occupied with geometry have to be marked at the highest resolution available in the octree. This information is determined by testing all grid cells within each triangle's bounding box for an intersection with the triangle, resembling typical grid construction algorithms such as the work by Perard-Gayot et al. [PGKS17]. Because of the large number of grid cells at high resolutions, we choose to represent each cell by a single bit in a field of 32 bit types. Each 32 bit chunk forms a block, which is subdivided spatially at a resolution of $(4\times4\times2)bits = 32bits$. The implementation uses CUDA's atomic operations on the respective chunks, effectively yielding the number of occupied cells. During the hash map initialization, this number is used in combination with a space-usage factor to limit the actual memory requirements. We choose an initial space-usage factor of $f_0 = 1.1$. If the hash map construction fails, another attempt is made with $f_n = 1.01 f_{n-1}$ until construction succeeds. This construction process is performed for each octree level, with cell indices being adapted to match the coarser representation. As the utilized hash map implementation is bound to 32 bit keys, we represent higher resolutions by splitting space into multiple hash maps per octree level.

The values stored in the octree's underlying hash maps are actual indices to global data arrays. Note that the presented implementation relies on caching only the outgoing diffuse illumination without any directional information other than the *front* and *back* of each cache cell. While it would be possible to store information for more directions this would negatively influence storage requirements and performance. However, storing at least two directions is necessary since infinitesimally thin geometric primitives may be illuminated differently from both sides. We construct the arrays to contain diffuse illumination for the front and back of each cell as six half values. Additional stored data includes cell normals compressed as one 32bit value per cell, the current accumulated number of samples per cell and the frame index each cell has last been reset, aggregated in one 32bit value per cell. The reset information is required to rebuild the cache when illumination changes occur. Note that the diffuse illumination is not attenuated by the diffuse material colour (albedo) at this point, which is accounted for during reconstruction. This allows for a higher-quality representation of spatial variation in the appearance of diffuse surfaces. The front normal for each cell is determined by performing an atomic compare-and-swap if no normal is stored in the according cell. Afterwards, the generated samples can be assigned to the front or back by comparing their stored normals with the newly set front normal. The total amount of memory required for the data of one cell is $(12+4+4+4)Bytes = 24Bytes$. There are no specific constraints for the number of triangles per octree cell (or, vice versa, the number of octree cells per triangle), as the required resolution largely depends on the lighting situation and the actual camera settings and position – for quick previews during modeling of individual objects, lower resolutions such as 256^3 or 512^3 may already yield satisfactory results.

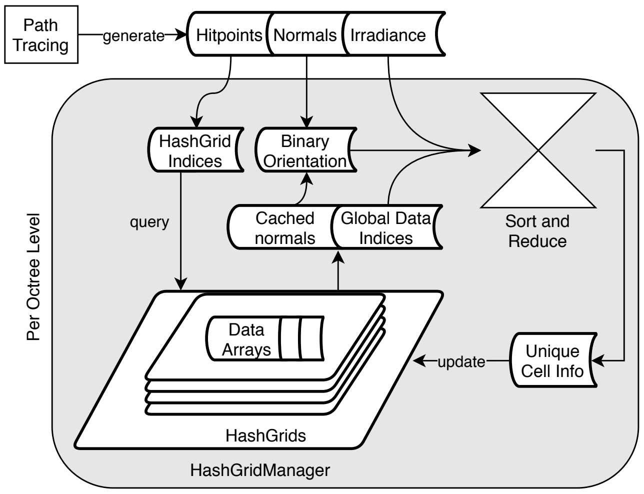

Figure 2: *An overview of the actual caching process. Hitpoints, normals and irradiance samples are generated by the path tracing process. The hitpoints' coordinates are then used to determine the according HashGrid indices for each hitpoint. Using these indices, the HashGridManager is queried for the global indices to the data arrays and the cached normals. A binary orientation for each irradiance sample, the actual irradiance sample and the global data indices are now used to collate the data in a sort-and-reduce operation. The resulting unique information per grid cell is then merged with the current cache data.*

3.2. Caching

An overview of the process described in this section is given in Figure 2. During the caching process, rays are shot into the scene from the current camera view and traced along randomly generated paths $\bar{\mathbf{x}} = \mathbf{x}_0\mathbf{x}_1 \dots \mathbf{x}_k$, with $\mathbf{x}_i$ being that path's individual vertices located on scene surfaces, and ℓ_{rec} being the maximal recursion depth. As we want to cache data not only for the first hitpoint (which would effectively only represent directly visible geometry), we compute illumination along subpaths with a maximum length of ℓ_{len} and store these for the first ℓ_{store} hitpoints. Thus, since all vertices of a path should account for the energy transported along the same num-

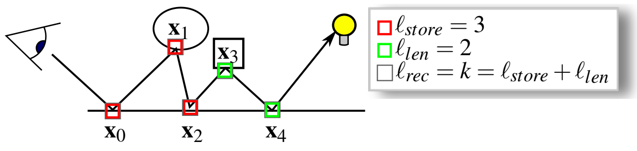

Figure 3: *Parameters for a single path. The parameter ℓ_{store} determines the maximum depth to which values are stored in the cache. After that depth, the illumination along subpaths is computed up to a maximum length of ℓ_{len}. The length of both determines the maximal recursion depth ℓ_{rec}.*

ber of consecutive vertices in order to provide consistent data, the maximum path length is $\ell_{store} + \ell_{len}$ and the indirect illumination contributed to each vertex $\mathbf{x}_i$ along the path has to be limited to the subpath vertices $\mathbf{x}_{i+1}, \ldots \mathbf{x}_{i+\ell_{len}}$, $i + \ell_{len} \leq \ell_{rec}$. This is illustrated in Figure 3. As soon as the local illumination and the reflected direction ω_i for the current vertex $\mathbf{x}_j$ have been computed, the energy transported along the current path is updated by computing the throughput $\mathbf{T}_i = f_r(i)/\text{prob}(x_j)(\omega_i \cdot \vec{n})$ according to the locally evaluated BRDF. The first vertex along the current path that should still account for energy originating at the current vertex is at index $p = \max\{0, j - \ell_{len}\}$. In order to take into account the accumulated throughput for the current subpath from vertex $\mathbf{x}_j$ back to vertex $\mathbf{x}_{j'}$, each preceding vertex $\mathbf{x}_{j'<j}$ is updated with the reflected local energy $\mathbf{L}_j$ by computing the component-wise multiplication

$$\mathbf{E}_{j'} = \mathbf{L}_j \odot \frac{\mathbf{T}_{j'}}{\mathbf{C}_{j'}} \odot \prod_{m=j'+1}^{j} \mathbf{T}_m. \quad (1)$$

Here, the diffuse material colour $\mathbf{C}_{j'}$ is not accounted for in vertex $\mathbf{x}_{j'}$. It is instead taken into account after reconstruction in order to avoid loss of spatial variation in the appearance of diffuse materials. All vertices from each path that belong to a Lambertian material are stored in the respective arrays indexed by the hash map. This includes diffuse illumination values $\mathbf{E}_j$, compressed normal vectors $\vec{n}_j$, the linear map index $\mathcal{H}$ (only required if the hash map's resolution $R > 1024^3$) and the linear cell index $\mathcal{C}$, where $\mathcal{H}$ and $\mathcal{C}$ are necessary to store the data in our data structure correctly.

Now, the respective cells of the HashCache are updated with the newly computed light transport data. When updating the individual octree levels, the collected data is pre-accumulated before performing an update on the global data arrays in order to avoid synchronization issues. Pre-accumulation is implemented using a radix sort approach with the global data index as the primary key and the binary orientation information (*front* or *back*) as the secondary key. The individual samples' normal vectors can be replaced with a scalar: 1, if the sample lies within the front-facing hemisphere, -1 otherwise. However, if storage is not an issue more directions can be represented, which may also allow for caching glossy materials. Afterwards, the data is coarsened for the preceding octree level and the process is repeated until all levels have been updated. The full octree update is in $O(n \log n)$, with n being the number of updated cache cells.

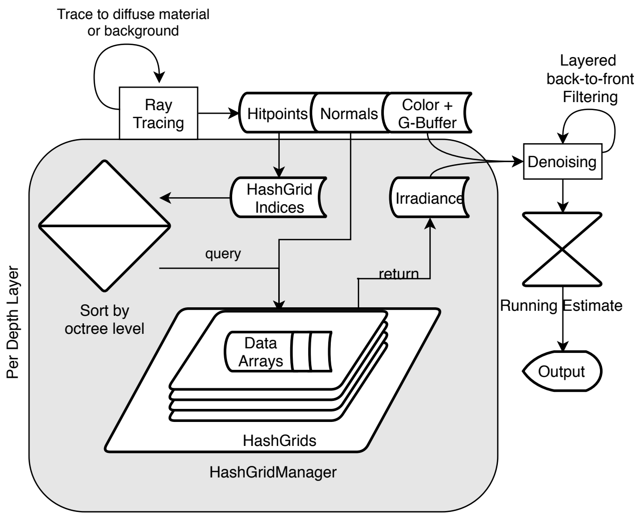

Figure 4: *An overview of the reconstruction process. A ray tracing step is performed to find the actual hitpoints that have to be reconstructed from the cache. In order to support non-lambertian materials, this step includes tracing rays until a diffuse surface is found, the background is hit or a user-definable maximum recursion depth is reached. The HashGrid indices computed from the hitpoints are then sorted by the appropriate octree-level and the cache is queried for the actual data indices. Together with hitpoint-wise texture and geometry information, the acquired irradiance is filtered in a denoising step in a layered back-to-front way. This means that each recursion level is filtered individually and then combined with the next (nearer) level, until the primary hitpoints are reached. The result is then combined in a running estimate to achieve higher image quality when the camera is not moving.*

3.3. Reconstruction

An overview of the process described in this section is given in Figure 4.

As rendering scenes with Lambertian materials exclusively may cause them to appear visually dull and unrealistic, our system provides the means for handling materials with glossy or specular properties. For the reconstruction step, primary geometry hitpoints are determined for each individual pixel, with the exception of glossy and specular materials, where the specific rays are traced further until they eventually arrive at maximum depth ℓ_{rec}, a diffuse material, or hit the background. For each path, the accumulated throughput is stored for the first ℓ_{p-1} vertices as $\mathbf{T}_{acc} = \prod_{i<\ell_p - 1} \mathbf{T}_i$ together with the diffuse material colour, the local normal, and the appropriate octree level (selected by projecting the pixel area in object space). The reconstruction is executed per-level and accumulated in the image by selecting the correct orientation from each cell and multiplying the retrieved diffuse illumination value with $\mathbf{T}_{acc} \cdot \mathbf{C}_{\ell_p - 1}$.

In order to reduce the blocky appearance caused by low cache resolutions, we employ a spatial jittering method. Finally, a basic edge-aware cross-bilateral denoiser filters remaining noise for each depth layer individually. Figure 1 shows a quality comparison of

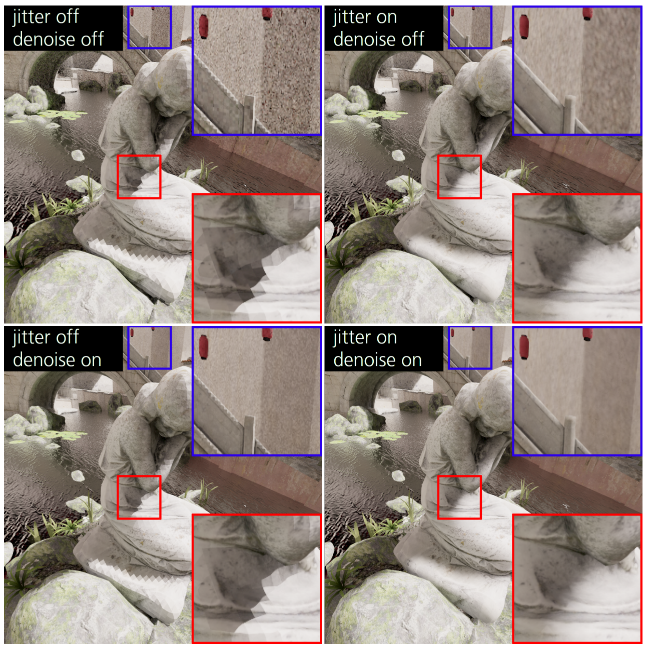

Figure 5: *Comparison of the same viewpoint rendered with all combinations of jittering and denoising.* ***Top left:*** *HashCache-based reconstruction without any spatial jittering or denoising applied.* ***Top right:*** *While spatial jittering cannot get rid of the high-frequency noise visible in the upper inset, it works well for hiding quantization artifacts.* ***Bottom left:*** *Denoising alone does work well for fine-grained noise, but cannot remove quantization artifacts very well.* ***Bottom right:*** *Combining jittering and denoising works well for both kinds of artifacts.*

the HashCache compared to path tracing. Figure 5 shows the effect of jittering and denoising in two areas: While the wall in the back shows more high-frequency noise, the statue in the front reveals quantization artifacts due to great differences between neighboring cache values. However, such artifacts are efficiently removed by the denoiser.

Note that spatial jittering may result in slight artifacts when cells are processed which do not have geometry in all neighboring cells that lie on the respective tangent plane. This is mainly caused by the fact that our data structure does not support enhanced sparsity encoding, but rather relies on constrained access [LH06] to avoid further memory consumption. Two cases may appear:

1. The hash key for the neighboring grid cell may belong to another cell that belongs to the scene's geometry. In such a case, visual artifacts may occur.
2. The hash key for the neighboring grid cell may yield an empty entry in the hash map. In this case, the irradiance value is set to the average of the pixel's neighbors, i.e., invalid or unsampled pixels resulting from spatial jittering are filled in.

However, during our evaluation, we did not observe any major artifacts resulting from this. Thus, we decided not to include any way of querying a cell for its grid coordinates.

4. Results and Evaluation

In this section, we evaluate the visual quality, performance and GPU memory requirements of the system. All measurements in this section were performed on a Linux system equipped with a GeForce GTX Titan X (Maxwell), a Core i7-7700 and 32GiB of main memory. Images were rendered at a resolution of 1024×1024 pixels.

Visual Quality To determine the visual error, we rendered the scenes *Country Kitchen* (CK) and *Streets of Asia* (SoA) using a camera fly-through of 500 frames with different configurations of the HashCache and regular path tracing for comparison. On the one hand, standard path tracing was rendered for 1, 8, 64, 512, 4,096, and 16,384 samples per pixel (spp) for both scenes. The outputs with the highest sample counts are used as reference images for error computation. On the other hand, results using the HashCache system were generated with 1, 8, 64, and 512 spp with the presented reconstruction technique. The hash map resolution was chosen to be 4096^3 for CK and 2048^3 for SoA. All rendered images underwent identical tone mapping processes. The results illustrated in Figure 6 show a comparison of the visual quality measured as (top) the relative mean-square error (relMSE) and (bottom) the multi-scale structural similarity (MS-SSIM) [WSB03] for varying sample counts. MS-SSIM takes perceptual phenomena into account and is an important tool to judge image quality with the human visual system taken into consideration. The reuse of already computed information in combination with a basic cross-bilateral filtering method already leads to the HashCache at 64spp yielding a quality similar to path tracing at 4096spp for the scene CK. For SoA, the HashCache does not show an improvement in relMSE between 64spp and 512spp. At 64spp the relMSE is similar to path tracing at 512spp. Note that the scene SoA was rendered using a HashCache with a lower maximal resolution of only 2048^3. As the spatial extent of the scene is high and the camera gets close to certain objects in the scene during the fly through, quantization artifacts are likely to occur and cause a larger difference between the reference solution and the cached irradiance. We expect even better quality at low sample densities when more advanced state-of-the-art filtering methods are integrated into our layered filtering framework (see Section 5).

Performance Figure 7 shows rendering times and the average number of samples cached per frame for two different cache settings: $(\ell_{store} = 3, \ell_{len} = 3)$ and $(\ell_{store} = 8, \ell_{len} = 8)$. The caching resolution factor (CRF) serves to adjust the number of rays cast in the caching phase. If set to 1, the number of primary rays will be the same as the chosen image resolution in the reconstruction step. While the cache construction times behave quadratic with regard to the CRF (because resolution is multiplied with the CRF both horizontally and vertically), reconstruction times do not depend on this setting. The average filtering time measured for the bilateral filter is 22ms, while the cache query took 4.5ms and the ray tracing phase for determining the visible geometry took around 31ms. Thus, at a frame rate of around 17fps, the reconstruction itself is well-suited for interactive previews. The additional time taken by the caching procedure depends largely on the rendering and caching parameters. Setting low values for ℓ_{store} and ℓ_{len} causes the ray tracing step not only to work with shorter but also more coherent paths,

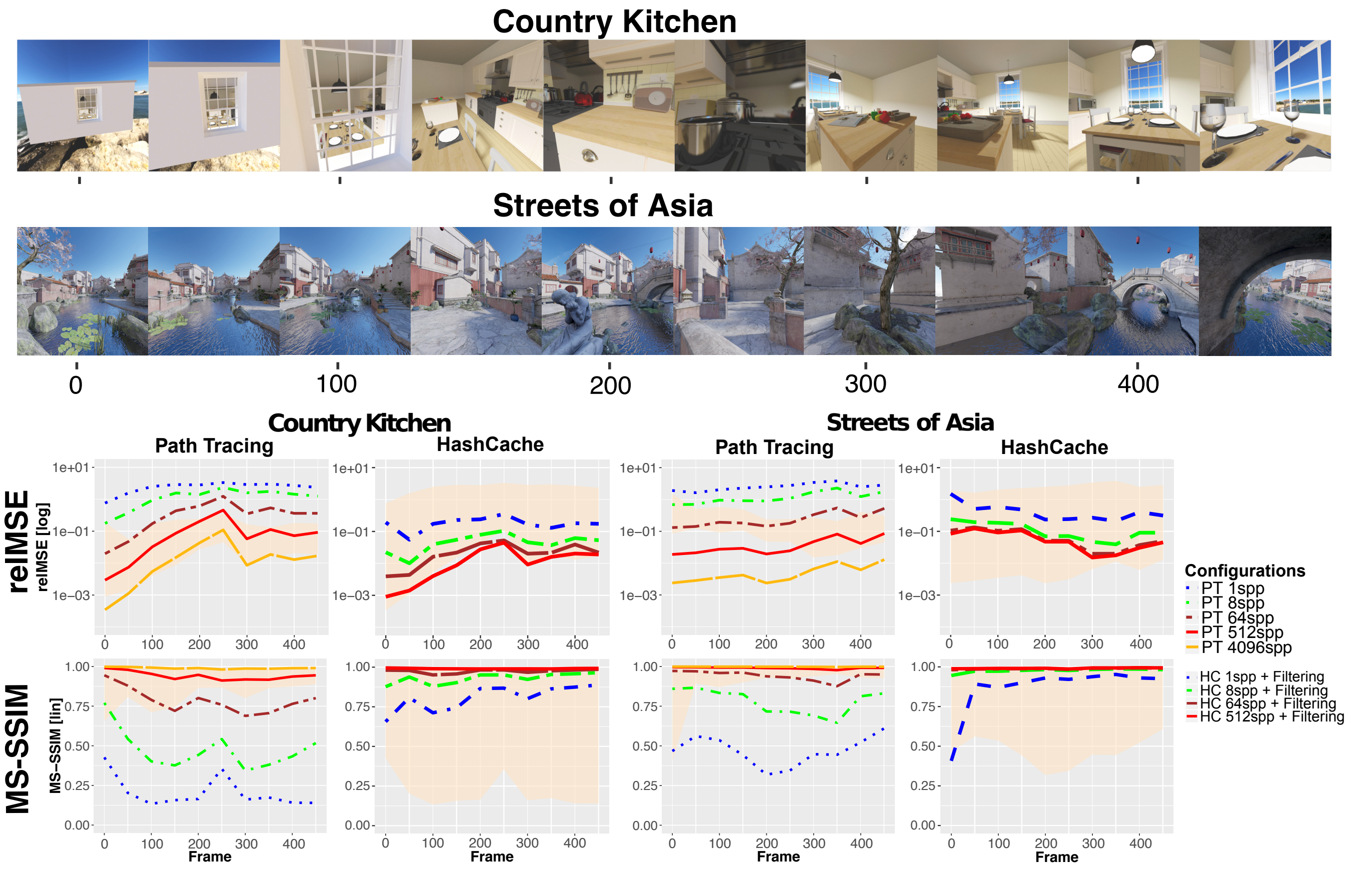

Figure 6: *Relative mean square error (relMSE, lower is better) and Multi-scale Structural Similarity (MS-SSIM, higher is better) for the scene* Country Kitchen *(maximal build resolution* 4096^3*) and* Streets of Asia *(maximal build resolution* 2048^3*) at various sampling densities using path tracing and the HashCache system. The shaded area in the background outlines the respective relMSE and MS-SSIM ranges for the other rendering method for better comparison.*

which is beneficial for its performance. However, despite the top configuration delivering the fastest caching times, the number of updated cache cells on the right also shows that cache convergence should be expected to be relatively slow. Only storing the first vertex for each path also causes issues with glossy, specular or transparent materials where the positions of subsequent vertices have not yet been seen directly by the user. Imagine a white wall reflected inside a mirror. It will only show a correctly rendered reflection if it has already been viewed before. However, this issue can be easily avoided by choosing a higher setting for ℓ_{store}, which is desirable anyway in order to cache illumination for scene parts that are not directly visible. The relatively slow cache update times for high CRF settings and high recursion and storage settings visible in Figure 7 are not a real issue for interactive exploration; in order to keep the process interactive, we modified our exemplary implementation to adjust the CRF to maintain a certain framerate while the camera is moving and only perform caching with the full resolution in the absence of movement. As shown in Figure 8, the actual reconstruction time is low enough to allow for fully interactive camera movements when the CRF is lowered. While the initial construction of the cache may take a couple of seconds depending on the set resolution, it is faster than Radiance Caching approaches [KGPB08, ORC*15]. Yet, it means that the HashCache is too slow to support fully dynamic scenes. We hope to omit the necessity of an explicit hash map construction step by using non-perfect hashing schemes in our future work.

Memory Requirements The amount of required GPU memory for both scenes *Country Kitchen* and *Streets of Asia* is shown in Table 1. While data density decreases by a factor of roughly 0.5 from level i to $i+1$, memory requirements still increase by a factor of roughly 4 to 5. In total, the 4096^3 representation of *Country Kitchen* required an amount of 2.17GiB for the data arrays and 406.92MiB for the hash maps, while *Streets of Asia* required 691.37 MiB for the data arrays and 126.75 MiB for the hash maps at a total resolution of 2048^3. One possible approach to handle the memory requirements resulting from higher resolutions would be the integration of an out-of-core component into our system, dynamically loading currently required data from host memory to the GPU.

Comparison to the State-of-the-Art Methods such as the work by Schied et al. [SKW*17, SPD18] have lower filtering run times (e.g. 4-5ms on Titan X vs. 22ms for the unoptimized HashCache

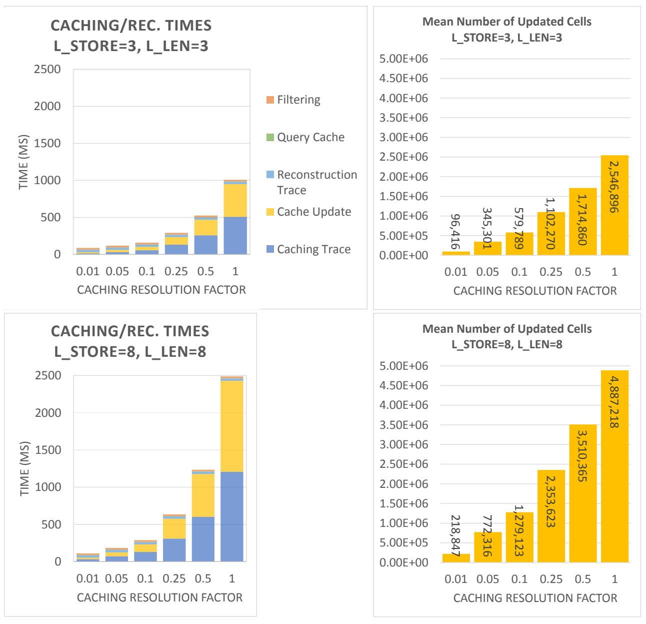

Figure 7: *Statistics for the scene Country Kitchen with different cache settings at a resolution of* 2048^3. ***Left:*** *Total rendering times, split into different steps.* ***Right:*** *Avg. number of cells updated per frame of the 500 frame sequence. These numbers include the update of all octree levels. Note that the resolution factor can be adjusted dynamically to the current situation, e.g., when the user is moving.*

filter) and produce a comparably high visual quality. However, these techniques rely on a temporal coherence between subsequent views. In contrast, the HashCache allows for integrating GI data from arbitrary spatial locations. Thus we are certain that all of these techniques can benefit from the knowledge from world-space caches. With the HashCache's hybrid reconstruction method, specular materials can be supported with ease (see Figure 9). Notwithstanding querying the world-space cache is slower than a cache in image space (4.5ms for HashCache vs. ∼0.5-1ms for Schied et al. [SPD18]). Yet in contrast to techniques such as NVIDIA's machine learning solution [CKS*17] that needs specific training data or might produce inconsistent results, the HashCache can be filled at run time. Admittedly though the caching itself is a very costly operation (see Figure 8), but the CRF can be freely adapted. The figure also shows how the actual perspective influences caching and rendering times. For *Country Kitchen*, rendering at the beginning takes only little time because the camera is still outside the room, which means that a lot of rays actually hit the background. As the camera gets nearer to the room, recursion depth increases because the rays are reflected between surfaces a lot more often. For the reconstruction, there is no vast difference caused by such a scenario. The only increase in recursion depth is caused by non-diffuse materials.

Once caches are filled to a certain extent, it is possible to limit the CRF largely and thus significantly reduce the cache update times. Moreover, the run time and caching behaviour can be adapted dynamically to the user's needs to stay within certain frame rate limits.

Country Kitchen

Level	Res	Density	Data Mem.	Hash Mem.
0	1	1	24 B	4.4 B
1	2	0.5	96 B	17.6 B
2	4	0.375	576 B	105.6 B
3	8	0.193	2.32 kiB	435.6 B
4	16	0.124	11.93 kiB	2.19 kiB
5	32	0.070	54.07 kiB	9.91 kiB
6	64	0.038	234.47 kiB	42.99 kiB
7	128	0.023	1.11 MiB	209.23 kiB
8	256	0.012	4.66 MiB	874.83 kiB
9	512	0.006	19.43 MiB	3.56 MiB
10	1024	0.003	81.60 MiB	14.96 MiB
11	2048	0.002	362.09 MiB	66.38 MiB
12	4096	0.001	1.71 GiB	320.90 MiB
		Sum	2.17 GiB	406.92 MiB

Streets of Asia

Level	Res	Density	Data Mem.	Hash Mem.
0	1	1	24 B	4.4 B
1	2	0.5	96 B	17.6 B
2	4	0.266	408 B	74.8 B
3	8	0.197	2.37 kiB	444.4 B
4	16	0.135	12.91 kiB	2.37 kiB
5	32	0.074	57.14 kiB	10.48 kiB
6	64	0.043	265.92 kiB	48.75 kiB
7	128	0.025	1.21 MiB	226.35 kiB
8	256	0.014	5.50 MiB	1.01 MiB
9	512	0.008	25.68 MiB	4.71 MiB
10	1024	0.004	114.77 MiB	21.04 MiB
11	2048	0.003	543.88 MiB	99.70 MiB
		Sum	691.37 MiB	126.75 MiB

Table 1: *Data densities and memory requirements. The data density is the quotient of occupied cells and the actual number of cells for a full grid of resolution* n^3. *Data memory is the amount of memory required to store the full data arrays. Hash Memory is the amount of memory reserved for the hash map representation for the respective level.*

5. Conclusion and Outlook

In this paper, we presented a method for caching and reconstructing diffuse global illumination in a hash-based linkless octree to store illumination data for subsequent views. While the introduced reconstruction technique already produces images at a remarkable quality, it is well possible to use the HashCache for enhancing the visual quality of other more elaborate image-space filtering techniques. We are certain that the HashCache can show its potential, especially in conjunction with methods that rely upon temporal integration in image-space such as the work by Schied et al. [SKW*17, SPD18] or machine learning techniques such as the work by Chaitanya [CKS*17]. However, the latter will probably require a new training set due to the more stable output produced by the HashCache.

While we left out the rendering of specular-to-diffuse transport, using our system with a bidirectional path tracer would also allow for such paths to be handled more effectively. Though, care has to be taken of the high-frequency content such as caustics, as this may quickly exceed the cache's spatial resolution. One option would be

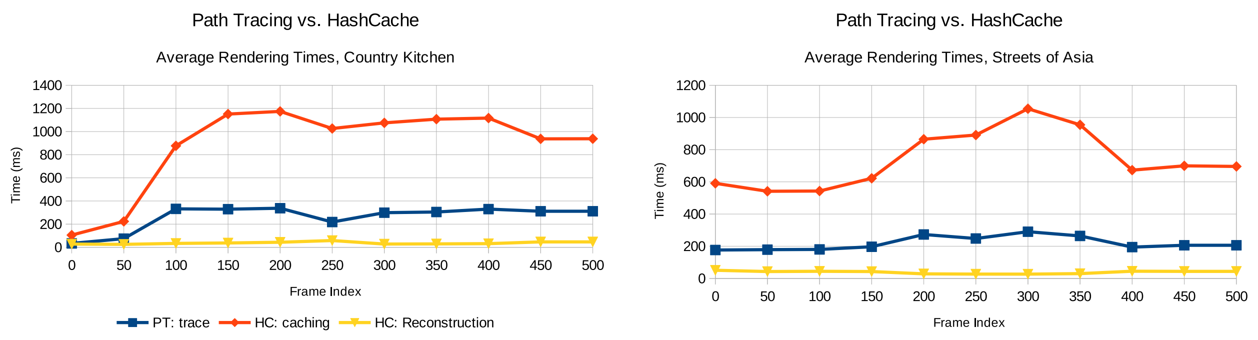

Figure 8: *Path Tracing and HashCache rendering times for Country Kitchen and Streets of Asia flythroughs. Rendering time is given for 1spp.* PT: trace *is the time for pure path tracing at a maximum recursion depth of 8,* HC: caching *is the caching part of HashCache computations which includes path tracing and cache updates,* HC: Reconstruction *is the reconstruction part of HashCache computations, which includes ray tracing, fetching the respective data from the cache and reconstructing an image from the computed data (including filtering). While caching times are significantly higher than pure path tracing times, data is reused between frames so that the actual caching resolution factor (CRF) can be reduced either permanently or deactivated during user input, allowing for smooth interaction.*

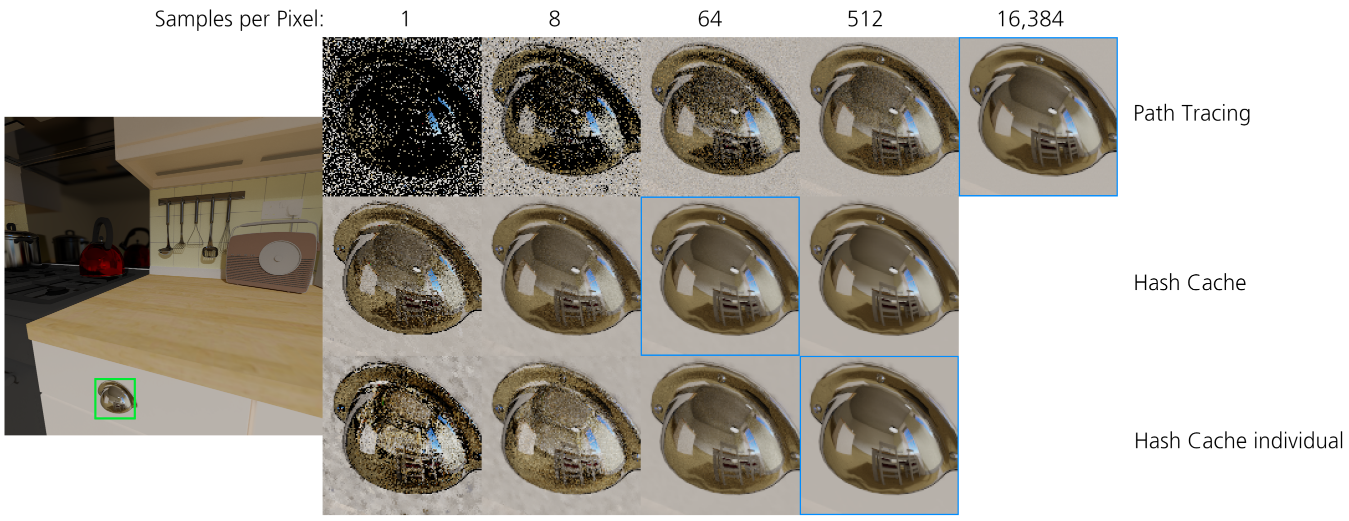

Figure 9: *Comparison of images generated with pure path tracing and with the support of HashCache.* ***Left:*** *The full image rendered at the reference sample count of* 2^{15}*. The green inset is magnified on the right for the various rendering settings.* ***Right:*** *Zoomed-in rendering for comparison, rendered with Path Tracing, HashCache (with 4 more frames in the sequence rendered before as a warm-up phase), and HashCache individual (without filling the cache in preceding frames). For the upper inset, the HashCache system already yields a quality at 64spp (512spp for individual rendering) that's at least on par with the path traced image at 16,384spp.*

to provide a dynamic local caching mechanism that works at higher resolutions and adjusts to a scene's light distribution. Also, instead of storing each level of the octree in a separate hash map, we can store all levels in one hash map with occupation information. This approach avoids separate hash map queries per octree level. This way the correct data array indices can be computed from the occupation information. However, this increases memory requirements.

Ultimately, we want to modify the caching method and employ a different hashing algorithm that allows for dynamic allocation on GPUs. This would be interesting in order to avoid the necessity of data structure rebuilds when geometry changes occur in a scene.

Also, we firmly believe that the HashCache can be a great tool to improve multi-view VR systems. Extending the system to support updates from multiple GPUs or even multiple machines, data quality throughout the scene could be improved in comparison to only caching illumination for the currently used perspective.

6. Acknowledgements

The work is supported by the German Federal Ministry for Economic Affairs and Energy (BMWi) funding the MoVISO ZIM-project under Grant No.: ZF4120902

References

[Alc11] ALCANTARA D. A. F.: *Efficient Hash Tables on the GPU*. Dissertation, University of California Davis, 2011. 3

[BEMA11] BAUSZAT P., EISEMANN M., MAGNOR M., AHMED N.: Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering EGSR) 30*, 4 (Jun 2011), 1361–1368. 2

[BFK18] BINDER N., FRICKE S., KELLER A.: Fast path space filtering by jittered spatial hashing. In *ACM SIGGRAPH 2018 Talks* (New York, NY, USA, 2018), SIGGRAPH '18, ACM, pp. 71:1–71:2. 2

[CB04] CHRISTENSEN P. H., BATALI D.: An irradiance atlas for global illumination in complex production scenes. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2004), EGSR'04, Eurographics Association, pp. 133–141. 2

[CJC*09] CHOI M. G., JU E., CHANG J.-W., LEE J., KIM Y. J.: Linkless octree using multi-level perfect hashing. *Computer Graphics Forum 28* (2009), 1773–1780. 2, 3

[CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., 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 (July 2017), 98:1–98:12. 7

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel-based cone tracing: An insight. In *ACM SIGGRAPH 2011 Talks* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 20:1–20:1. 2

[DSHL10] DAMMERTZ H., SEWTZ D., HANIKA J., LENSCH H.: Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In *Proc. High Performance Graphics 2010* (2010), pp. 67–75. 2

[DSS06] DIETRICH A., SCHMITTLER J., SLUSALLEK P.: *World-space sample caching for efficient ray tracing of highly complex scenes*. Tech. rep., Citeseer, 2006. 2

[GO12] GASTAL E. S. L., OLIVEIRA M. M.: Adaptive manifolds for real-time high-dimensional filtering. *ACM TOG 31*, 4 (2012), 33:1–33:13. Proceedings of SIGGRAPH 2012. 2

[HJ10] HACHISUKA T., JENSEN H. W.: Parallel progressive photon mapping on gpus. In *ACM SIGGRAPH ASIA 2010 Sketches* (New York, NY, USA, 2010), SA '10, ACM, pp. 54:1–54:1. 2

[Jen96] JENSEN H. W.: Global illumination using photon maps. In *Rendering TechniquesÃćâĈňâĎć 96*. Springer, 1996, pp. 21–30. 2

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM SIGGRAPH Computer Graphics* (1986), vol. 20, pp. 143–150. 1, 3

[KGPB08] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. In *ACM SIGGRAPH 2008 Classes* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 75:1–75:19. 2, 6

[LH06] LEFEBVRE S., HOPPE H.: Perfect spatial hashing. *ACM Trans. Graph. 25*, 3 (July 2006), 579–588. 5

[MLM13] 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* (New York, NY, USA, 2013), I3D '13, ACM, pp. 71–78. 2

[ORC*15] OMIDVAR M., RIBARDIÈRE M., CARRÉ S., MÉNEVEAUX D., BOUATOUCH K.: A radiance cache method for highly glossy surfaces. *The Visual Computer* (Oct. 2015), 1–12. 2, 6

[PGKS17] PÉRARD-GAYOT A., KALOJANOV J., SLUSALLEK P.: Gpu ray tracing using irregular grids. *Comput. Graph. Forum 36*, 2 (May 2017), 477–486. 3

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. *Comput. Graph. Forum 31*, 1 (Feb. 2012), 160–188. 2

[SJ09] SPENCER B., JONES M. W.: Hierarchical Photon Mapping. *IEEE Transactions on Visualization and Computer Graphics 15*, 1 (Jan. 2009), 49–61. 2

[SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., 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* (New York, NY, USA, 2017), HPG '17, ACM, pp. 2:1–2:12. 2, 6, 7

[SL17] SILVENNOINEN A., LEHTINEN J.: Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Transactions on Graphics 36*, 6 (Nov. 2017), 1–13. 2

[SPD18] SCHIED C., PETERS C., DACHSBACHER C.: Gradient estimation for real-time adaptive temporal filtering. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 2 (Aug. 2018), 24:1–24:16. 2, 6, 7

[SYM*12] SCHERZER D., YANG L., MATTAUSCH O., NEHAB D., SANDER P. V., WIMMER M., EISEMANN E.: Temporal coherence methods in real-time rendering. *Computer Graphics Forum 31*, 8 (2012), 2378–2408. 2

[THGM11] THIEDEMANN S., HENRICH N., GROSCH T., MÜLLER S.: Voxel-based global illumination. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 103–110. 2

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive Rendering using the Render Cache. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (June 1999), Lischinski D., Larson G. W., (Eds.), vol. 10, Springer-Verlag, pp. 235–246. 2

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 85–92. 2

[WSB03] WANG Z., SIMONCELLI E. P., BOVIK A. C.: Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003* (Nov. 2003), vol. 2, pp. 1398–1402 Vol.2. 5