Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction

MATIAS KOSKELA, KALLE IMMONEN, MARKKU MÄKITALO, ALESSANDRO FOI, TIMO VIITANEN, PEKKA JÄÄSKELÄINEN, HEIKKI KULTALA, and JARMO TAKALA, Tampere University, Finland

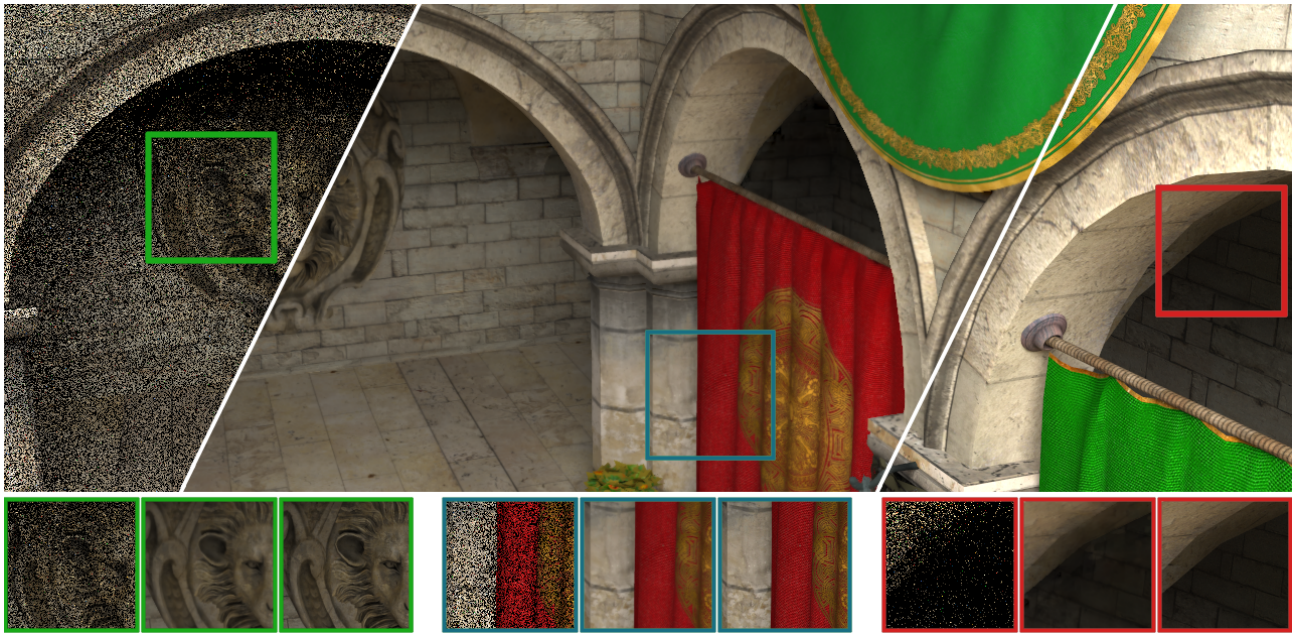


Fig. 1. In all image sets, left: 1 sample per pixel path-traced input, center: result of the proposed post-processing denoising/reconstruction pipeline, and right: 4096 samples per pixel reference. Leftmost highlights: the lion is barely visible in the input, but the proposed pipeline is able to produce realistic illumination results without blurring the edges and high-frequency albedo details. Center highlights: the best case for the pipeline is geometry with sufficient light in the input. Rightmost highlights: the worst case for the pipeline is the one with occlusions and almost no light, resulting in blurry artifacts.

Path tracing produces realistic results including global illumination using a unified simple rendering pipeline. Reducing the amount of noise to imperceptible levels without post-processing requires thousands of *samples per pixel* (spp), while currently it is only possible to render extremely noisy 1 spp frames in real time with desktop GPUs. However, post-processing can utilize feature buffers, which contain noise-free auxiliary data available in the rendering pipeline. Previously, regression-based noise filtering methods have only been used in offline rendering due to their high computational cost. In this paper we propose a novel regression-based reconstruction pipeline, called *Blockwise Multi-Order Feature Regression* (BMFR), tailored for path-traced 1 spp inputs that runs in real time. The high speed is achieved with a fast implementation of augmented QR factorization and by using stochastic regularization to address rank-deficient feature data. The proposed algorithm is 1.8× faster than the previous state-of-the-art real-time path tracing reconstruction method while producing better quality frame sequences.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; *Rendering*; *Image processing*;

Additional Key Words and Phrases: path tracing, reconstruction, regression, real-time

ACM Reference Format:
Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction. *ACM Trans. Graph.* X, Y, Article Z (May 2019), 14 pages. https://doi.org/0000001.0000001

Authors' address: Matias Koskela, matias.koskela@tuni.fi; Kalle Immonen, kalle.immonen@aspekt.fi; Markku Mäkitalo, markku.makitalo@tuni.fi; Alessandro Foi, alessandro.foi@tuni.fi; Timo Viitanen, viitanet@gmail.com; Pekka Jääskeläinen, pekka.jaaskelainen@tuni.fi; Heikki Kultala, heikki.kultala@tuni.fi; Jarmo Takala, jarmo.takala@tuni.fi, Tampere University, Tampere, 33720, Finland.

1 INTRODUCTION

Real-time path tracing has been a long-standing goal of graphics rendering research due to its ability to produce natural soft shadows, reflections, refractions, and global illumination effects using a conceptually simple unified drawing method. However, its computational complexity is a major challenge; contemporary ray tracing frameworks [AMD 2017; Parker et al. 2010; Wald et al. 2014]

are able to produce only around one path tracing *sample per pixel* (spp) at real-time frame rates on desktop-class hardware. It is expected that the real-time performance will increase in the near future as new generations of high-end GPUs will integrate hardware acceleration for ray tracing [Patel 2018]. Nevertheless, a linear improvement in rendering quality requires a quadratic increase in computational complexity: to halve the signal-to-noise ratio in path tracing, the number of samples per pixel has to be quadrupled [Pharr and Humphreys 2010]. Consequently, reducing the amount of noise to imperceptible levels without post-processing requires thousands of samples per pixel and, therefore, denoising filters are used even in offline path-traced movie rendering [Goddard 2014].

The trend of rising resolutions and refresh rates, driven especially by the needs of virtual reality immersion, increases the amount of required computations at the same rate as the computing hardware is improved. As a consequence, it is unrealistic to expect the computing hardware performance to improve fast enough to support real-time path tracing at high frame rates. It seems that the achievable real-time path tracing sample rates will remain around 1 spp with the consumer devices of the near future [Alla Chaitanya et al. 2017; Schied et al. 2017; Viitanen et al. 2018]. Therefore, there is an urgent need for novel real-time post-processing denoising methods that are targeted for 1 spp path-traced inputs.

Constructing high quality results from a 1 spp starting point is hard even when done offline without strict real-time constraints. The input has an extreme amount of noise, much more than conventional image denoising algorithms can handle. However, the reconstruction results can be improved by utilizing *feature buffers*, which contain noise-free auxiliary data available from the path tracer. The buffers can include useful information such as surface normals and texture albedo colors. As is essential for the real-time goal, this information can be extracted from a path tracer with little performance overhead. Utilizing feature buffers allows reconstruction filters to, e.g., avoid blurring samples across geometry edges, which is a very disturbing artifact for the human eye, or it can reduce smearing the details in the textures.

Moreover, fast path tracers can reproject and accumulate samples from multiple previous frames to reduce temporal noise that varies between successive frames. Flickering artifacts are especially noticeable by the end users. Real-time denoising algorithms must specifically account for the temporal noise as there is no option of simply adding more samples per pixel and the denoising needs be fast enough to fit in the time slot left over from the rendering.

In this article we propose a new regression-based reconstruction pipeline optimized for 1 spp input images that runs in real time on desktop GPUs. The proposed method is 1.8× faster and has better objective quality than the previous state-of-the-art real-time path tracing reconstruction method. The article presents the following contributions:

- A novel *Blockwise Multi-Order Feature Regression* (BMFR) algorithm, where multiple versions of the feature buffers of different orders are used for fitting.
- A fast GPU-based implementation of the BMFR algorithm.
- Proposal to use stochastic regularization to address the possible rank-deficiency of the blockwise features, avoiding numerical instabilities without the extra complexity of pivoting.

In other words, the proposed algorithm combines a completely novel concept (multi-order feature buffers) with a few established concepts (feature regression, QR factorization). Regression-based methods have typical had execution times in order of seconds [Moon et al. 2016] and have been considered to be applicable only in offline context [Alla Chaitanya et al. 2017; Schied et al. 2017]. However, we do regression in an unusual way (blockwise processing, augmented factorization with stochastic regularization) and, therefore, the proposed method is the first regression-based method to achieve real-time performance.

2 RELATED WORK

Path tracing reconstruction methods are covered in a recent comprehensive survey article [Zwicker et al. 2015]. In general, the methods can be divided into three categories based on their complexity: offline methods, interactive methods, and real-time methods. Real-time methods are closest to the context of this article, but we also draw ideas from and compare to methods from the other categories.

Naturally, the best reconstruction quality for path tracing can be achieved with offline methods. Since there is no strict time budget, offline methods can use complicated and slow algorithms. Furthermore, as they are not constrained by real-time deadlines, their execution time can vary heavily based on the input data. Typically, offline methods target inputs that have more than 1 spp, because it is not a problem to generate more path tracing samples if the filtering itself takes a comparatively long time. In offline methods it is also possible for the filtering to guide the sample generation process in path tracing so that more samples are generated at problematic areas in screen space [Li et al. 2012]. Offline reconstruction can be implemented, for example, with general edge-preserving image filters like guided image filtering [He et al. 2013; Liu et al. 2017] or bilateral filtering [Tomasi and Manduchi 1998], which are guided with feature buffers. Another approach is to use a neural network [Kalantari et al. 2015], which can be trained even with a complete set of frames from a feature-length movie [Bako et al. 2017]. A third approach is to fit the noise-free feature buffers to the noisy image data [Bitterli et al. 2016; Moon et al. 2014, 2015].

Neural networks can also be used at interactive frame rates as shown recently by Alla Chaitanya et al. [2017]. Since the quality of the interactive methods is not as good as in offline methods, extra care needs to be taken to address temporal stability of the results. One way to address temporal noise is to use recurrent connections in each neural network layer [Alla Chaitanya et al. 2017]. Sheared filtering is another approach to achieve interactive frame rates [Yan et al. 2015]. In contrast to the neural network approach, sheared filtering also supports effects that produce noise to the feature buffers, such as motion blur [Egan et al. 2009].

Reconstruction based on the guided image filter is the closest method in the literature to the proposed one which can also reach interactive frame rates [Bauszat et al. 2011]. However, it is not an appealing approach for real-time implementation on modern GPUs,

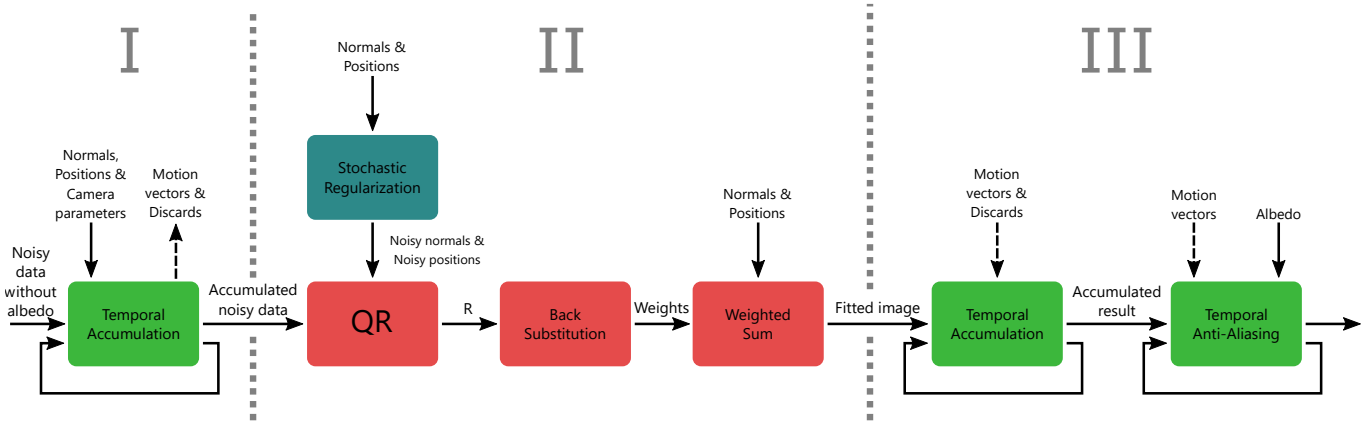


Fig. 2. Overview of the proposed reconstruction pipeline. The pipeline inputs a noisy 1 spp path-traced frame and the corresponding normal and world-space position buffers. It outputs a noise-free image with a good approximation of global illumination. Without the stochastic regularization, the back substitution block produces NaNs and Infs due to rank deficiency.

since it requires either dozens of moving window operations or generating as many summed-area tables. Moving window operations involve several orders of magnitude less parallel work than a modern GPU can process concurrently, whereas generating summed-area tables requires an expensive parallel scan pattern and higher precision values stored in the buffers.

There is recent research interest on algorithms that can perform path tracing reconstruction in real time. A way to achieve required execution speed is to use approximations or variants of the bilateral filter, such as a sparse bilateral filter [Mara et al. 2017], or a hierarchical filter with multiple iterations [Burt 1981] expanded with customized edge-stopping functions [Dammertz et al. 2010; Schied et al. 2017].

Real-time methods are typically targeted for 1 spp inputs because the motivation for attempting to perform the reconstruction in real time is low if the input frames must be computed offline anyway. In case of 1 spp inputs and fast lower quality reconstruction, even higher degree of variation is expected in the results, making temporal stability an even bigger problem with real-time methods.

Temporal stability can be improved by accumulating projected frames [Yang et al. 2009], which produces a greater effective spp and more static noise in world coordinate locations. A similar idea can also be used for dealing with ambient occlusions [Jiménez et al. 2016]. However, in these reprojection-based techniques some of the rendered pixels cannot utilize the accumulated data because they were occluded in the previous frame. Such *disocclusion events* can be recognized, for example, based on inconsistencies in the world-space position or normal data in the feature buffers for the subsequent frames. Interestingly, the reprojection method can also support, for example, rigid body animations if there is a way to find out where the current pixel was in the previous frame [Rosado 2007]. Temporal stability can be further improved, e.g., with simple *Temporal Anti-Aliasing* (TAA) [Karis 2014], which uses colors from the neighborhood of the pixel in the current frame to adjust the data sampled from the previous frame. The idea of using temporal data in anti-aliasing was introduced in *Enhanced Subpixel Morphological Antialiasing* (SMAA) [Jimenez et al. 2012].

As in previous work, the proposed reconstruction algorithm also utilizes TAA, and also reprojects and accumulates noisy data from previous frames. However, we dynamically change the weight of the new frame so that first samples after an occlusion do not get overweighted. Moreover, we add an additional step of data accumulation after filtering to increase temporal stability and to avoid artifacts. Moreover, instead of using the typical approximations of the bilateral filter we use regression-based reconstruction, which has been previously considered too slow for real-time use cases [Alla Chaitanya et al. 2017; Schied et al. 2017]. By means of applying augmented QR factorization and stochastic regularization we made the regression fast enough for real-time use. Finally, we introduce BMFR, where multiple versions of the feature buffers of different orders are used for fitting, improving the chances for the fitting to succeed.

3 RECONSTRUCTION PIPELINE

The proposed reconstruction pipeline can be divided into three main phases: preprocessing, feature fitting and post-processing. The phases, marked with roman numerals, are illustrated in Fig. 2 and explained in subsections below. The proposed algorithm does not need to guide the path tracing process in any way.

3.1 Input

The input for the real-time reconstruction filter is a 1 spp path-traced frame and its accompanying feature buffers. The 1 spp frames are generated by using a rasterizer for producing the primary rays and feature buffers. We use mipmapped textures in albedo. Next, we do so-called next event estimation: we trace one shadow ray towards a random point in one random light source and then continue path tracing by sending one secondary ray to a random direction. Namely, we use multiple importance sampling [Veach and Guibas 1995]. The direction of the secondary ray is decided based on importance sampling. We also trace a second shadow ray from the intersection point of the secondary ray. Consequently, the 1 spp

pixel input has one rasterized primary ray, one ray-traced secondary ray and two ray-traced shadow rays. The random numbers in the path tracer were generated with Wang hash [Wang et al. 2008]. The ray configuration was chosen because it can be path traced in real time and is able to reproduce effects like realistic global illumination, soft shadows, and reflections. Every time we refer to *1 spp data* in this article, we refer to this ray configuration.

Before inputting the 1 spp input into our post-processing pipeline, we remove first bounce surface albedo from it. Reconstructing without albedo is a common practice [Alla Chaitanya et al. 2017; Bako et al. 2017; Mara et al. 2017; Schied et al. 2017] because it ensures that high-frequency details in first-bounce textures are not blurred by the filter. The other commonly used idea is to decompose the lighting contribution to a direct and indirect component [Bauszat et al. 2011; Mara et al. 2017]. However, we do not do the separation, because it typically assumes that the direct lighting component is noise-free. Instead, we have 1 spp path-traced soft shadows in the direct component and we filter both components at once. Filtering two noisy components separately would require running the pipeline twice, which does not double the runtime since heaviest parts of the work can be shared. However, we did not find significant quality increase and the slowdown is unacceptable in our real-time context.

If the scene contains multilayer materials, the pipeline has to be run separately for every material's illumination component. However, all illumination components can be combined and filtered at once if the albedo is the same for every layer. An optimization opportunity for multilayer materials is to compute a weighted sum of different albedos and illuminations and filter all illuminations at once [Schied et al. 2017]. Even though combining the illuminations before filtering does not produce a physically correct result, this approach can be used as a fast approximation.

3.2 Preprocessing

The preprocessing phase (I) consists of temporal accumulation of the noisy 1 spp data, which reprojects the previous accumulated data to the new camera frame. In the reprojection process, world-space positions and shading normals are used to test whether we can accumulate previous data or have to fall back to the current frame's 1 spp path-traced result. Because of accumulation, in most of the pixels the effective spp can be greater than 1 even though the individual frame inputs are 1 spp. In addition, accumulation improves temporal stability of the noise.

Following a previous work [Schied et al. 2017; Yang et al. 2009], we compute an *exponential moving average* and mix 80% of the history data with 20% of the current frame data. However, we apply one significant modification compared to the previous work: we start by computing a *cumulative moving average* of the samples, and use the exponential moving average only after the cumulative moving average weight of the new sample would be less than 20%. The use of regular average on the first frames and after occlusions makes sure that the first samples do not get an excessively high weight, and limiting the weight to a minimum of 20% makes sure that the aged data fades away.

Computing the cumulative moving average requires that we store and update the sample count of every pixel. Since we are interested in the sample count only if the count is small, the values can be stored in just a few bits. Loading and storing, for example, 8-bit integers is insignificant compared to other memory accesses of the temporal accumulation.

We use bilinear sampling of the history data and do a discard test for each pixel separately. The final color is normalized by the sum of the accepted sample weights only, thus the discarded pixels do not affect the brightness of the sample. Also the sample count data is sampled using the same custom bilinear sampling and the result is rounded to the closest integer value.

3.3 Blockwise Multi-Order Feature Regression (BMFR)

The feature fitting phase (II) is based on the following feature regression operated on non-overlapping image blocks, covering the entire single frame.

Let $F = \left[F_1, \ldots, F_N\right]$ denote a set of available noise-free feature buffers, such as the world-space positions and shading normals. Typically these buffers are created as a side product of the path tracing. However, they could contain artificially created data like gradients. Every buffer in F has the same resolution as the noisy frame. We consider an *extended* set T of M feature buffers:

$$T = \left[F_{n_1}^{\gamma_1}, \ldots, F_{n_m}^{\gamma_m}, \ldots, F_{n_M}^{\gamma_M}\right], \tag{1}$$

where $n_m \in \{1, \ldots, N\}$, $m = 1, \ldots, M$, $\gamma_1 = 0$, and $\gamma_m > 0$, $m = 2, \ldots, M$, are positive exponents. The first buffer in T is a constant buffer $F_{n_1}^0 = \mathbf{1}$. Note that γ_m, $m > 1$, need not be an integer and can be larger as well as smaller than 1.

Denoting by $\Omega_{i,j}$ the set of absolute coordinates of the pixels within an image block located at position (i, j), the *Blockwise Multi-Order Feature Regression* (BMFR) problem can be formulated like a standard least-squares expression with respect to the multi-order features T as

$$\hat{\alpha}^{(c)} = \underset{\alpha^{(c)} \in \mathbb{R}^M}{\operatorname{argmin}} \sum_{(p,q) \in \Omega_{i,j}} \left(Z^{(c)}(p,q) - \sum_{m=1}^{M} \alpha_m^{(c)} F_{n_m}^{\gamma_m}(p,q) \right)^2, \tag{2}$$

where $Z^{(c)}$ is the c channel of the noisy path-traced input which can be temporally accumulated (e.g., c may be red, green, blue, or any relevant luminance or chrominance component). The estimate of the noise-free scene Y for channel c and block $\Omega_{i,j}$ is thus

$$\hat{Y}^{(c)}(p,q) = \sum_{m=1}^{M} \hat{\alpha}_m^{(c)} F_{n_m}^{\gamma_m}(p,q)\,. \tag{3}$$

While being a simple linear solution, it is non-linear w.r.t. the features F, which makes it more flexible and capable of better fit to the data than established methods based on linear regression on F.

3.4 Feature Fitting with Stochastic Regularization

We solve the least-squares problem (2) by the Householder QR factorization [Heath 1997]. Specifically, and using matrix-vector notation, let us reshape the M blockwise feature buffers $F_{n_m}^{\gamma_m}(p,q)$, $(p,q) \in \Omega_{i,j}$, $m = 1, \ldots, M$, as column vectors of length W, where W is the number of pixels in the block $\Omega_{i,j}$, and let $\mathbf{T}$ be the $W \times M$

(a) 1 spp (b) World-space positions (c) Normals (d) World positions [2] (e) Just constant buffer in fitting (f) + world positions (g) + normals (h) + world positions [2] (i) Reference

Fig. 3. Different buffers and results with a single 64×64 block of BMFR. Notice how adding world-space positions squared allows the fitting to generate a more realistic soft shadow under the edge. In the fast implementation we use 32×32 blocks, but the larger blocks visualize the benefit in a single block more clearly. The results get closer to the reference when temporal accumulation averages multiple blocks from different displacements.

matrix obtained by horizontal concatenation of such column vectors. Further, let

$$\widetilde{\mathbf{T}}^{(c)} = \left[\mathbf{T}, \mathbf{z}^{(c)}\right] \tag{4}$$

be the $W \times (M+1)$ matrix obtained by augmenting $\mathbf{T}$ with $\mathbf{z}^{(c)}$, which is $Z^{(c)}(p,q)$, reshaped into a column vector of length W. We expect $W \gg M$, meaning that each block has much more pixels than there are feature buffers.

Assuming that $\widetilde{\mathbf{T}}^{(c)}$ is full rank, the Householder QR factorization yields an $(M+1) \times (M+1)$ upper triangular matrix $\widetilde{\mathbf{R}}^{(c)}$ such that $\widetilde{\mathbf{T}}^{(c)} = \widetilde{\mathbf{Q}}^{(c)}\widetilde{\mathbf{R}}^{(c)}$, where $\widetilde{\mathbf{Q}}^{(c)}$ is a $W \times (M+1)$ matrix with orthonormal columns. Given $\widetilde{\mathbf{R}}^{(c)}$, there is no need to compute $\widetilde{\mathbf{Q}}^{(c)}$ for solving the linear least squares problem; instead, we can solve the transformed system contained in $\widetilde{\mathbf{R}}^{(c)}$ [Heath 1997, pp. 92-93]. By dealing just with the smaller matrix $\widetilde{\mathbf{R}}^{(c)}$ we get a significant performance improvement.

Specifically, if we denote by $\mathbf{R}$ and by $\mathbf{r}^{(c)}$, respectively, the top-left $M \times M$ sub-matrix and the top-right $M \times 1$ sub-column of $\widetilde{\mathbf{R}}^{(c)}$, the solution $\hat{\alpha}^{(c)}$ of (2) is given as

$$\mathbf{R}\hat{\boldsymbol{\alpha}}^{(c)} = \mathbf{r}^{(c)}, \tag{5}$$

which can be solved, for example, via back substitution, which is simple and fast. Hence, $\hat{Y}^{(c)}(p,q)$, $(p,q) \in \Omega_{i,j}$, (3) is obtained as

$$\hat{\mathbf{y}}^{(c)} = \mathbf{T}\hat{\boldsymbol{\alpha}}^{(c)}, \tag{6}$$

where $\hat{\mathbf{y}}^{(c)}$ is $\hat{Y}^{(c)}$ reshaped into a column vector of length W. Observe that $\mathbf{R}$ (and its inverse) does not depend on $\mathbf{z}^{(c)}$, and that $\mathbf{r}^{(c)}$ can be computed for different channels without recalculating $\mathbf{R}$, which allows to process multiple channels with minimal extra cost.

In practice, $\widetilde{\mathbf{T}}^{(c)}$ may be rank-deficient, leading to numerical instabilities that break the factorization. While the rank-deficiency is typically managed by pivoting, we employ *stochastic regularization*. That is, we add noise to the input buffers, which makes them linearly independent, i.e., (4) becomes

$$\widetilde{\mathbf{T}}^{(c)} = \left[\mathbf{T} + \mathbf{N}, \mathbf{z}^{(c)}\right], \tag{7}$$

where $\mathbf{N}$ is a $W \times M$ matrix of zero-mean independent and identically distributed noise. T within every block is scaled to be in range $[-1, 1]$, before this addition. Since the average of the noise is zero, we can expect that this regularization does not increase the fitting bias. The synthesis (6) always uses the noise-free buffers $\mathbf{T}$, so the noise itself is not visible in the reconstructed estimate. In our implementation, we use zero-mean uniformly distributed noise over an interval $[-\varepsilon, \varepsilon]$, thus having variance $\varepsilon^2/3$. The value of ε that worked with all our tested scenes was 0.01. Much stronger noise ($\varepsilon \approx 1.0$) caused visibly too bright and dark constant blocks, whereas much weaker noise ($\varepsilon \approx 0.0001$) failed to regularize, leading to divisions by zero in the factorization.

3.5 Post-processing

The purpose of the post-processing phase (III) is to increase temporal stability and the perceived visual quality.

First, the fitted frame is temporally accumulated, which reduces blocky artifacts caused by operating the BMFR algorithm on non-overlapping blocks and improves temporal stability. Importantly, small fitting errors caused by the stochastic regularization can be expected to cancel out when multiple frames are accumulated because the injected noise has a zero mean. To aid the reduction of blockiness, BMFR processes each frame over a grid of non-overlapping blocks which is displaced with random offsets. These offsets prevent the artifacts that would arise from reusing same block positions on a static scene with a static camera.

This post-processing phase is essentially the same process that was done in the preprocessing step to increase the effective spp. However, the process is faster because bandwidth can be saved by reusing the motion vectors and discard decisions from the preprocessing phase. By loading for every pixel just 2 floats and 4 booleans, we avoid loading all 5 world-space positions and shading normals again, all containing three channels (1 from current frame, and 4 for bilinear sampling of the previous frame).

In this second temporal accumulation we use 10% of new data and 90% old data because these values hide the block place variations. Similarly to the first temporal accumulation we use the cumulative moving average until the weight of the new sample has reached the chosen 10%. Using the cumulative moving average in this second temporal accumulation is crucial since the first block fitted after an occlusion is more likely to contain outlier data and with the cumulative moving average it is mixed with subsequent frames more quickly. For example, if we used the exponential moving average, after three frames the weight of the very first fitted data would still be more than half. With cumulative moving average the weight is the same as in a regular average: one third.

As a last step of the pipeline, TAA [Karis 2014] is used. While in many of the test scenes TAA decreases the quality measured by the

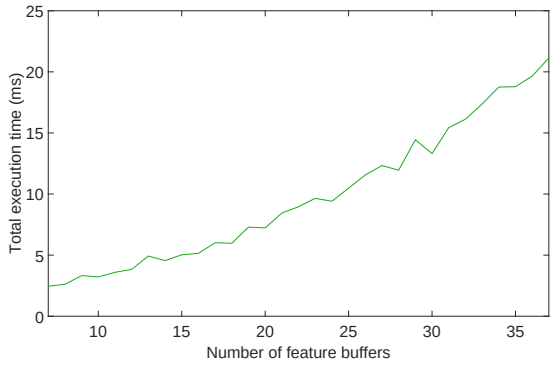


Fig. 4. The execution time of the whole pipeline with different counts of feature buffers. The QR block size used in this measurement was 32×32. In the rest of the runtime results we use 10 feature buffers. All of the test scenes have a similar runtime since the runtime varies only in the stages that access previous frame data from pixels stated by the motion vectors.

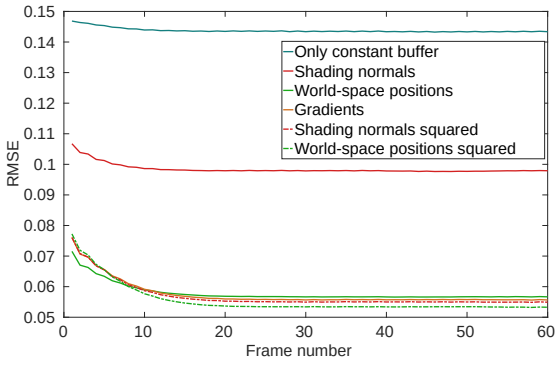


Fig. 5. The effect on denoising quality as more sets of buffers are added cumulatively, measured by *Root Mean Square Error* (RMSE) (lower is better) for the Sponza test scene with a static camera. The buffers are greedily added in the order specified in the legend from top to bottom.

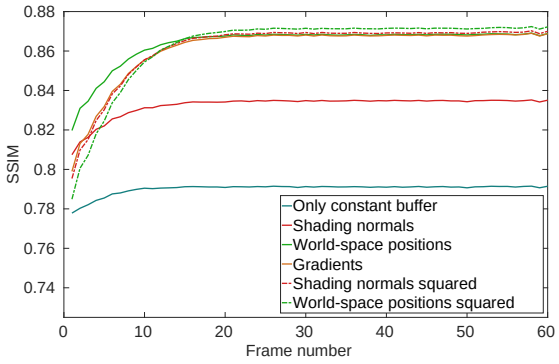


Fig. 6. The effect on denoising quality as more sets of buffers are added cumulatively, measured by *Structural SIMilarity* (SSIM) [Wang et al. 2004] (higher is better) for the Sponza test scene with a static camera. The buffers are greedily added in the order specified in the legend from top to bottom.

objective quality metrics, in our experience it gives more visually pleasing results.

4 COMPLEXITY ANALYSIS

Phases I and II in the proposed pipeline can be implemented using the parallel map and parallel stencil patterns. Thus, the execution time of these phases is linearly dependent on the number of pixels in the input image. In these phases adding more feature buffers only increases the amount of data stored in first accumulation stage. In other words, the processing can be parallelized easily because the result pixels are independent of each other. However, adding more computing hardware is likely to quickly reach its limits because all the stages are mostly memory bound.

The most important stage in the pipeline regarding the complexity analysis is the QR stage. When the number of pixels in the input image is increased, the number of QR blocks grows linearly. The blocks do not affect each other in any way, so all of them can be loaded and processed in parallel, and therefore performance scales linearly. In contrast, if one feature buffer is added, it must be transformed by all of the previous feature buffers. The transform requirement comes from the Householder reflections method: the number of required transforms is $O(M(M+1)/2) = O(M^2)$, where M is the number of feature buffers. However, the work per each feature buffer in the proposed method is quite small, which can be seen in Fig. 4. With a reasonable number of feature buffers, the execution time increase is almost linear. For comparison, guided filter's [Bauszat et al. 2011] requirement is to generate $O(M^2)$ summed-area tables. Therefore, we can include more feature buffers in the same execution time to produce results that have a higher visual quality.

5 FEATURE BUFFER SELECTION

The choice of which feature buffers to include in the filtering is crucial. Including additional feature buffers increases the computational complexity by $O(M^2)$, but the resulting quality improvement varies dramatically based on the buffer type. It is thus essential in real-time filtering to include only the most beneficial feature buffers.

To this end, we measured the effects of different buffer types by greedily adding all available set of buffers to find the ones that helped the most. Greedy addition means that we tested every available buffer and added the one that improved the objective quality the most. After each addition we started the same process again with the rest of the available buffers. Fig. 5 and Fig. 6 show the obtained results for the Sponza test scene with a static camera; the corresponding results for our other test scenes yield similar conclusions.

We also experimented by adding horizontal and vertical gradient buffers consisting of a horizontal or a vertical gradient from 0 to 1 for each block, respectively. The idea was to provide more freedom for the feature regression (2). However, the gradient buffers yielded only minor quality improvements, as Fig. 5 and Fig. 6 also show.

Only minor quality improvements make sense because typically there is always gradient-like data available in the world position buffers.

Every channel of each feature buffer was added at once even though some channels might have not contributed much to the result, because otherwise the feature selection would have suffered from overfitting to camera orientations.

Based on the aforementioned measurement, we adopted the following multi-order set of feature buffers:

$$T = \left[\mathbf{1}, \mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z, \mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z, \mathbf{w}_x^2, \mathbf{w}_y^2, \mathbf{w}_z^2\right], \tag{8}$$

where $\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z$ are the three channels of shading normals, and $\mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z$ are the three channels of world-space positions. This set of buffers was selected because, as can be seen in the figures, the error is decreased the most by adding the normals and the world-space positions. The benefit of adding further buffers appears to get negligible compared to increased execution time.

However, the computational error metrics do not reveal small problematic areas in the result, and therefore, after visual examination, we decided to add the second-order world-space positions. Fig. 3 illustrates the reason for this choice; world-space positions are particularly useful for getting more convincing soft shadows.

In the proposed method, the specular highlight is generated from a feature buffer that happens to have data similar to the highlight. If the highlight is not well presented by the available feature data, the result improves when multiple block locations from successive frames are accumulated. Adding material roughness to the set of feature buffers allows illumination to vary between regions inside a block, which only helps if there are materials that have a roughness texture with fine details. However, the constant feature buffer generates the same result if regions of the input larger than block size have uniform roughness.

6 TEST SETUP

We measured the visual quality and execution speed of the proposed algorithm while rendering animations. To provide the algorithm with a realistic amount of accumulated frame data, which is also hindered by occlusions, all except two of the test inputs had continuously moving cameras. Each frame of these animations can be found in the supplementary material of this article. One frame consists of 1 spp input data, the corresponding feature buffers, and a 4096 spp reference rendering.

In the following we describe our test setup, which includes an example implementation of the proposed algorithm and a set of compared algorithms.

6.1 GPU Implementation

To measure the performance of the proposed algorithm with realistic hardware, we implemented the algorithm using OpenCL and optimized it for a contemporary high-end desktop GPU, AMD Radeon Vega Frontier Edition. The code we wrote for the measurements is available as supplementary material of the article. The primary implementation choices that affect performance as it pertains to our target hardware are described next.

The block size was chosen to be 32×32 because even though we found that the best quality is achieved with a 64×64 block, 32×32 block gives us four times more parallel work to improve the processing element utilization. Moreover, we need to synchronize within the block, and synchronization can be done in groups of 256 parallel work items in the targeted hardware. Consequently, already with the 32×32 block, the code needs to be unrolled four times between the synchronization points.

Table 1. The number of memory accesses in parallel reduction. *Level* means combining two values into one value. *Iteration* is one code block without synchronization between parallel workers.

Levels per iteration	1	2	3	4	5
Elements per iteration	2	4	8	16	32
Memory accesses	3	5	9	17	33
Accesses per level	3	2.5	3	4.25	6.6
Accesses for 1024 elements	3069	1705	1317	1161	1089

For the displacement, we used a static sequence of 16 random offsets, uniformly distributed over the whole set of possible offsets. The displacement is done both horizontally and vertically. This number gives enough variety of displacements with the chosen blendings of history data and the new frame in temporal accumulations.

After avoiding the heavy matrix multiplications by just computing $\widetilde{\mathbf{R}}$, the computation on the targeted hardware was limited by the speed of accessing the data and performing reduction in local memory, i.e., computing the sum of all concurrently processed elements in local memory[1], which is the fastest memory space visible to the whole block.

The reduction calculations are needed for the sum calculations of the dot products and vector norms, both of which are computed multiple times in the Householder algorithm. Reduction is also used in every block to find out the local minimum and maximum of every feature, which are used to scale the values to be in the same range in the fitting. We implemented the reduction with parallel reduction, where all parallel processing elements process more than two elements on every iteration. The number of memory accesses per iteration for different counts of elements processed at once can be seen in Table 1. The fastest alternative for reduction of $32 \times 32 = 1024$ elements on our target hardware was experimentally determined to consist of summing 4 elements on the first two iterations and 8 elements on the last two iterations. This approach appears to be a good compromise between the parallelism available and the total amount of memory accesses. Fewer levels per iteration gives more parallel work. In contrast, more levels per iteration results in less memory accesses in total.

The largest implemented kernel was fitting, which contains almost all the stages of phase II. In contrast to what was found by Laine et al. [2013], in this case a single "megakernel" which included the heaviest stages of phase II was the fastest because the intermediate data could be passed through fast local memory and registers.

For faster data access we use half-precision floating-point numbers as the temporal storage type and order the pixels such that every 32×32 block is at consecutive addresses in memory. Thanks

[1] We use OpenCL terminology and call this memory space *local memory*. The corresponding CUDA term is *shared memory*.

to the memory layout, the hardware can load and store the data with faster vector accesses. It is also possible that the path tracer stores the data directly in this format because many path tracers render square blocks of pixels in one work group since then there is more cache locality in the primary rays [Aila and Karras 2010].

6.2 Compared Algorithms

We compared the proposed algorithm to five state-of-the-art algorithms: 1) The neural network denoiser which is freely available in the OptiX 5.0 SDK. In this article we refer to it as the *OptiX Neural Network Denoiser* (ONND). 2) A recent state-of-the-art real-time Monte Carlo reconstruction algorithm, *Spatiotemporal Variance-Guided Filtering* (SVGF) [Schied et al. 2017]. 3) *Guided Filtering* (GF) [Bauszat et al. 2011], which we consider the algorithm-wise ancestor of the proposed work. 4) An off-line reconstruction algorithm called *Nonlinearly Weighted First-order Regression* (NFOR) [Bitterli et al. 2016]. 5) Another real-time reconstruction method, namely *An Efficient Denoising Algorithm for Global Illumination* (EDAGI) [Mara et al. 2017], which is separately compared in Subsection 7.3.

The ONND implementation is based on the interactive reconstruction from the article by Alla Chaitanya et al. [2017], but differs in a few ways. Most importantly, every frame is denoised individually, which causes low temporal stability. The OptiX implementation also does not separate albedo from the input before filtering. Moreover, it uses a different set of feature buffers than the original article. We attempted to use the filter with temporally accumulated noisy data similar to our method but found that with the default training set the filter is not able to discriminate between detail and noisy data due to changes in noise characteristics caused by accumulation. Consequently, we had to use a 1 spp input with this denoiser. Furthermore, ONND requires that the input is tone-mapped and gamma-encoded.

The authors of SVGF did not provide an implementation for accurately reproducing the results of their article. Therefore we used a freely available implementation of the algorithm in the quality assessments.[2] We modified the implementation to follow the original article's algorithm more closely by running it separately for direct and indirect lighting and by removing albedo before filtering. We also changed it to use the same TAA [Karis 2014] as in the SVGF article.

We used our own code for the Guided Filter implementation. Our implementation is based on the MATLAB code provided by the authors of the original article on guided filter [He et al. 2013] but has been extended to allow an arbitrary number of feature buffers. As in the article by Bauszat et al. [2011], we used a 4-dimensional guidance image consisting of three normal channels and depth. In the article, only indirect illumination is filtered. For the indirect component, we used radius 24 and epsilon 0.01 as suggested in the article. Because in our dataset also the direct illumination component is noisy, we filtered it as well with guided filter. We used a smaller filter size (radius 12) to cause less blurring, and therefore to improve the results. The epsilon used for direct illumination was the same as for indirect illumination. Finally, we extended the method with albedo removal and accumulation of noisy data.

For NFOR we used the freely avaible code released by the original authors [Bitterli et al. 2016]. For comparison fairness, instead of using 1 spp inputs, we used the reprojected and accumulated inputs and reprojected running variances, which improved the quality significantly. We also applied TAA to the results because it improved subjective quality in all test scenes and objective quality in approximately half of the tests scenes.

7 RESULTS

This section reports the performance of the algorithm in terms of the visual quality of the result and the execution speed with the test setup described in the previous section.

7.1 Objective Quality

We used four different metrics to measure the objective quality of our method compared to the other methods: *Root Mean Square Error* (RMSE), *Structural SIMilarity* (SSIM) [Wang et al. 2004], temporal error [Schied et al. 2017], and *Video Multi-Method Assessment Fusion* (VMAF)[3] [Aaron et al. 2015; Li et al. 2016]. The results of our measurements are presented in Table 2 and Table 3, and comparison images of all the methods are shown in Fig. 7. The known limitations of the proposed method are further discussed in Section 8.

As expected, the offline comparison method NFOR is able to obtain best results in most of the scenes with most of the metrics. However, the results of the proposed method are close to the NFOR results with more than ten thousand times faster runtime. NFOR is not originally designed for 1 spp inputs, but when we give it reprojected inputs, the effective spp count gets close to the counts used in the original paper.

In the majority of the test scenes, our method outperforms the previous real-time methods in terms of RMSE, SSIM and VMAF. In the remaining scenes our results are still generally comparable to the other real-time methods, with only marginal differences at the top. In the few cases where our results are average in terms of one metric, such as for RMSE in the moving light Sponza, another metric still ranks us at the top, in this case VMAF. Hence, in such cases the performance difference can be at least partially attributed to inherent limitations in the simple metrics, as they disagree with each other to some extent; therefore, we provide the results for several metrics. Moreover, our results could be improved if only optimizing these metrics by skipping TAA in phase III, since it introduces some blur in the results and thus affects RMSE, SSIM and VMAF negatively. Nevertheless, we chose to apply it due to it producing visually more pleasing results to our eyes.

In terms of temporal error [Schied et al. 2017], our results are overall similar to those of the guided filter and ONND. SVGF yields the lowest temporal error in all of the scenes, with our method being on par with it in the static scene. However, the used temporal error metric is rather simple, as it only considers the average per-pixel luminance differences between adjacent frames, so its correlation with subjectively perceived temporal quality variance is not immediately evident. This observation is further corroborated by the fact

[2]The SVGF version which was used as a base of our modifications can be found at https://github.com/ruba/RadeonProRender-Baikal, Git commit hash ed2a7e2d929653551f8a93ada5b164d2f9f624e7.

[3]The VMAF model used in the comparison was v0.6.1.

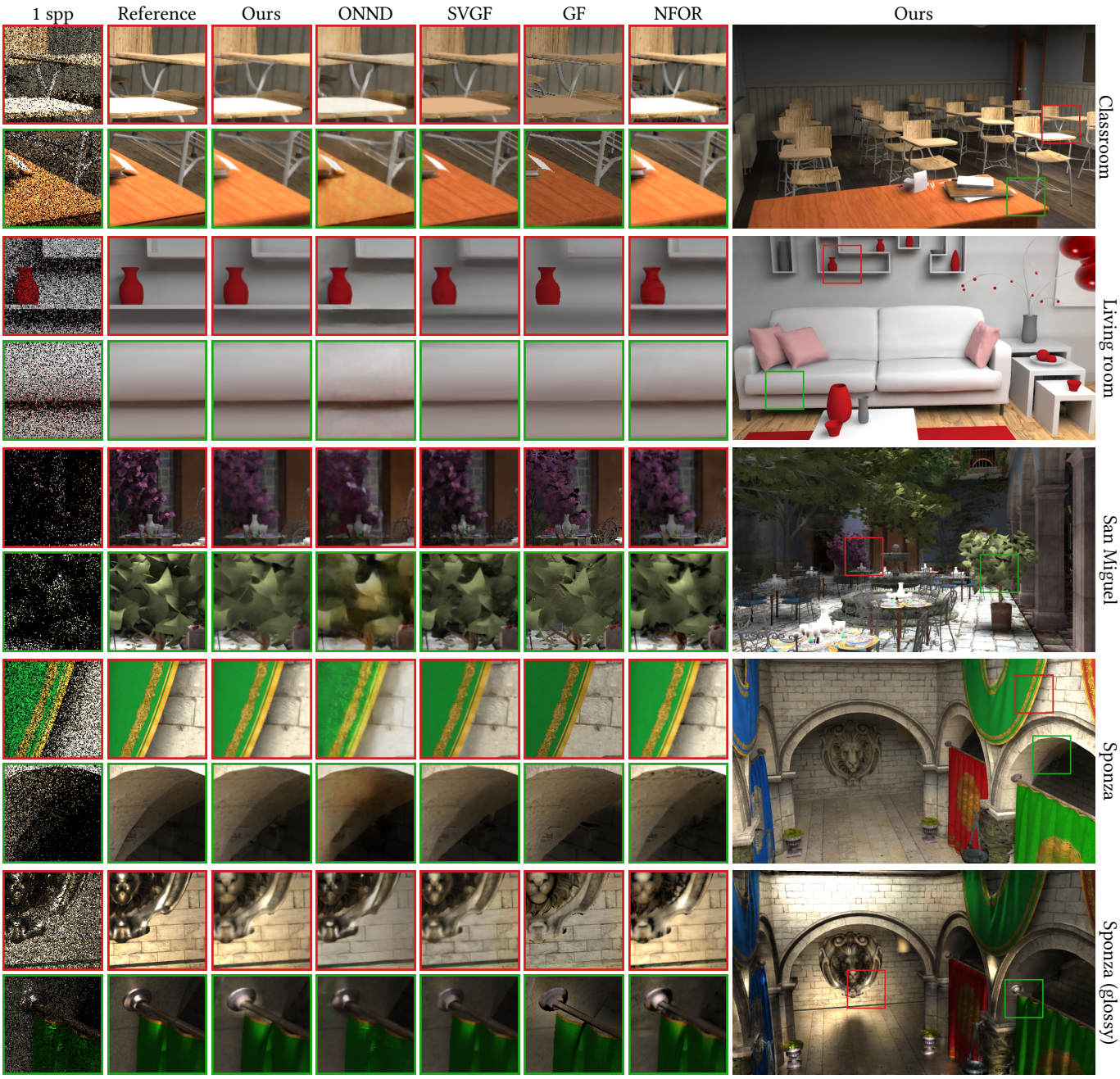


Fig. 7. Closeups highlighting the quality differences between the proposed pipeline and the comparison methods taken from animated sequences after 30 frames. Detailed description of the insets is in Subsection 7.2. Reference is 4096 spp and the comparison methods are *OptiX Neural Network Denoiser* (ONND) which is based on Alla Chaitanya et al. [2017], *Spatiotemporal Variance-Guided Filtering*, (SVGF) [Schied et al. 2017], *Guided Filtering* (GF) which is based on Bauszat et al. [2011], and *Nonlinearly Weighted First-order Regression* (NFOR) [Bitterli et al. 2016].

Table 2. Objective quality measurements for the results measured with RMSE (lower is better) and SSIM (higher is better), each averaged over all 60 frames. For brevity, Sponza with a static camera is referred to as "Sponza (static)", and Sponza with a moving light as "Sponza (light)".

	Average RMSE					Average SSIM				
	Proposed	ONND	SVGF	GF	NFOR	Proposed	ONND	SVGF	GF	NFOR
Classroom	0.0356	0.0431	0.0561	0.0586	**0.0321**	0.960	0.938	0.957	0.937	**0.961**
Living room	0.0315	0.0526	0.0434	0.0608	**0.0272**	0.953	0.927	0.936	0.918	**0.957**
San Miguel	0.0895	0.0982	0.1160	0.1011	**0.0812**	0.753	0.669	0.745	0.742	**0.757**
Sponza	**0.0282**	0.0591	0.0661	0.0509	0.0306	**0.965**	0.901	0.943	0.961	0.957
Sponza (glossy)	0.0564	0.0671	0.0900	0.0736	**0.0503**	**0.906**	0.847	0.893	0.885	0.899
Sponza (static camera)	**0.0317**	0.0756	0.1159	0.0939	0.0394	**0.973**	0.876	0.906	0.932	0.948
Sponza (moving light)	0.1450	**0.0773**	0.1418	0.0936	0.0811	0.835	0.846	0.855	**0.913**	0.892

Table 3. Objective quality measurements for the results obtained with various algorithms, measured with a simple temporal error as in [Schied et al. 2017], averaged over all 60 frames. The quality is also measured with VMAF (higher is better).

	Average temporal error						VMAF				
	Ref.	Proposed	ONND	SVGF	GF	NFOR	Proposed	ONND	SVGF	GF	NFOR
Classroom	0.043	0.037	0.040	0.035	0.039	0.040	53.14	52.78	41.51	24.90	**61.11**
Living room	0.024	0.021	0.027	0.019	0.019	0.022	57.85	61.53	46.78	21.58	**64.61**
San Miguel	0.082	0.060	0.061	0.056	0.073	0.067	21.68	21.63	10.77	25.73	**28.03**
Sponza	0.059	0.051	0.048	0.045	0.056	0.052	**74.81**	49.66	50.58	58.04	70.54
Sponza (glossy)	0.058	0.050	0.043	0.044	0.053	0.054	38.50	32.87	22.73	24.22	**44.87**
Sponza (static camera)	0.004	0.001	0.019	0.001	0.003	0.001	**70.46**	32.71	30.53	24.97	65.52
Sponza (moving light)	0.011	0.007	0.022	0.006	0.010	0.009	32.19	32.46	19.09	26.40	**49.24**

that, as Table 3 shows, the temporal error of the reference itself is typically higher than that of the reconstructed result. Hence, instead of merely focusing on the absolute error, it may also be useful to consider how close the error of the reconstructed result is to the error of the reference. However, similar temporal error readings can be caused by completely different changes in the consecutive frames. On the other hand, VMAF demonstrably correlates well with subjective quality [Li et al. 2016], and in most cases our method yields significantly higher VMAF results than the other real-time methods.

7.2 Subjective Quality

Subjective quality of the proposed method can be evaluated with Fig. 7. Moreover, all full resolution frames and a video are available in the supplementary material of this article.

In Fig. 7 the insets of the Living room and Classroom scenes represent cases where our algorithm is able to outperform the comparison methods. ONND sometimes starts to generate details that are not present in the reference at all. Due to its À-Trous nature, SVGF generates sometimes light artifacts that are typical to À-Trous based methods. These are visible for example in the red inset of the Living room scene. On the other hand, GF often overblurs the illumination, which might be due to poor parameter selection. We used the best parameters according to the original authors.

Insets of the San Miguel scene show different foliage cases. Our method produces results which are visually pleasing and believable, though somewhat overblurred.

One of the main motivations of our work is visible in the red insets of the Sponza scene. The proposed method can produce in real time dynamic soft shadows that are very close to the reference. The green insets of the same scene represent a case where there is just a small amount of light and our algorithm must rely on 1 spp data due to occlusions (camera is moving back and rightwards). In this case the result contains some blurred artifacts.

The roughness in the Sponza (glossy) scene is 0.1 for every material. As can be seen in the red insets of the Sponza (glossy) scene in Fig. 7, our algorithm can perform well with even quite complex specular highlights. On the other hand, the green insets of the same scene represent a hard case where all of the methods fail and it is up to the viewer to decide which type of imperfection is the least disturbing. More discussion on the limitations of the specular highlights can be found in Section 8.

7.3 Comparison to Noise-Free Direct Lighting

In this subsection we report a separate comparison with EDAGI [Mara et al. 2017]. This method is treated separately because it assumes a rasterized noise-free direct lighting component. Thus, it is incompatible with the stochastic noisy direct lighting in our input dataset, preventing an objective comparison to the fully path-traced reference like that in Tables 2 and 3.

Fig. 8 presents some of the test scenes from the original EDAGI work, as reconstructed by the proposed method from a fully stochastic path-traced lighting. When comparing these images to those in their online supplementary material, it is visible how realistic

Fig. 8. Some of the scenes from [Mara et al. 2017] reconstructed with the proposed work.

Table 4. Average execution times of different stages in the proposed pipeline on AMD Radeon Vega Frontier Edition. The division to OpenCL kernels is different than the stages in the Fig. 2. The division was chosen for performance and code readability reasons. The total runtime is measured from the beginning of the first kernel to the end of last kernel. All of the scenes and camera paths yield similar timings, because only the runtime of accumulation and TAA kernels is affected by the input data.

Phase	Kernel	Runtime
I	Temporal accumulation	0.44 ms
II	QR & back substitution	1.55 ms
	Weighted sum	0.12 ms
III	Temporal accumulation	0.23 ms
	TAA	0.16 ms
Total		2.54 ms

Table 5. GPU runtimes of different comparison methods for 720p frames as reported in the original articles. NVIDIA GeForce Titan X (Pascal) runtime was measured with the OpenCL implementation, which was developed on AMD Radeon Vega Frontier Edition. Consequently, there is likely room for optimizations on NVIDIA platforms.

Method	Runtime	Hardware
Proposed (OpenCL)	2.5 ms	AMD Radeon Vega FE
	2.4 ms	NVIDIA Titan X (Pascal)
[Alla Chaitanya et al. 2017]	55 ms	NVIDIA Titan X (Pascal)
SVGF [Schied et al. 2017]	4.4 ms	NVIDIA Titan X (Pascal)
GF [Bauszat et al. 2011]	94 ms	NVIDIA GTX 285
EDAGI [Mara et al. 2017]	9.2 ms	NVIDIA Titan X (Pascal)

soft shadows produced by the stochastic direct lighting make the proposed kind of rendering compelling.

7.4 Execution Speed

The average execution times of different parts of the proposed pipeline can be seen in Table 4. In the measurements we assumed that the path traced 1 spp input and feature buffers are in GPU buffers when we start the timer and that the result can be left to another GPU buffer. That is, we model a scenario where a GPU-based path tracer has left its data to GPU buffers and at the end, the results are written to the frame buffer.

All of the runtimes reported in this section are with 1280×720 frames. We also confirmed with measurements that, as analyzed in Section 4, the runtime scales linearly relative to the number of pixels.

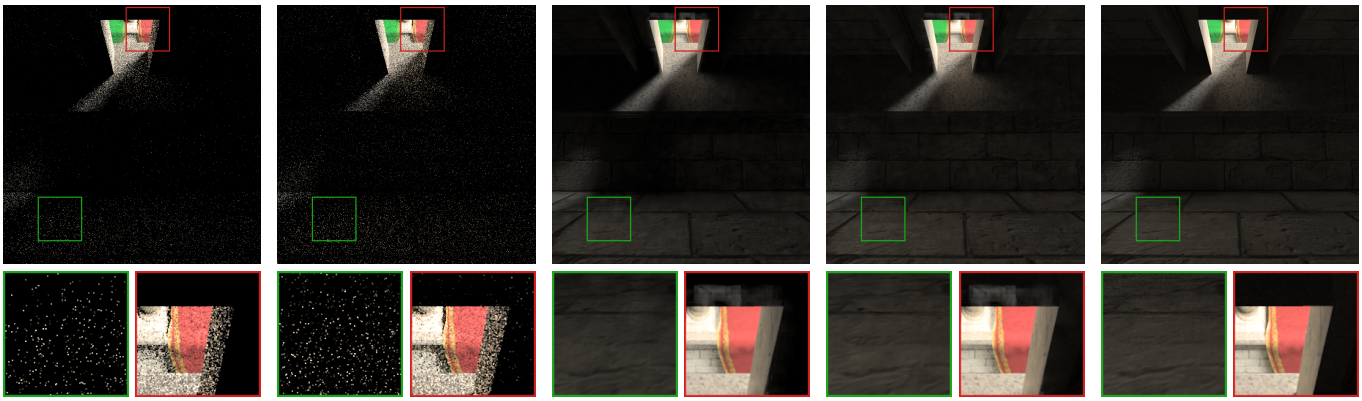
The execution time of the proposed pipeline was stable on AMD Vega Frontier Edition (variation approximately ± 0.04 ms) across different scenes and animation frames. The only pipeline stages where runtimes are affected by the input data are the ones with temporal accumulation. The runtime variation is due to cache misses of dispersed loads and early exits, e.g., in case of projected pixels that are detected to fall outside the new frame.

The proposed pipeline clearly outperforms the other algorithms (listed in Table 5) in terms of execution speed. NFOR runtime is left out from the table because it is in order of minutes rather than milliseconds. SVGF, the previous state-of-the-art real-time method, reports average execution times of 4.4 ms on NVIDIA Titan X (Pascal). Our 2.4 ms execution time is thus 1.8× faster. Moreover, SVGF's execution time depends more on the input data because they fall back to a slower method with harder inputs. The other real-time method [Mara et al. 2017] has an average execution time of 9 ms on NVIDIA Titan X (Pascal). However, they expect noise-free direct lighting which makes the comparison difficult. Alla Chaitanya et al. [2017] report runtimes of 55 ms on NVIDIA Titan X (Pascal), which means the proposed pipeline is 22× faster. Guided filter [Bauszat et al. 2011] execution time linearly scaled to 720p frame is 94 ms on NVIDIA 285 GTX, and even for that number, noise-free direct lighting is required. However, the article where the number was reported is already a bit old and uses a previous generation GPU, thus there could be room for improvement if the algorithm was optimized for a modern GPU.

8 LIMITATIONS

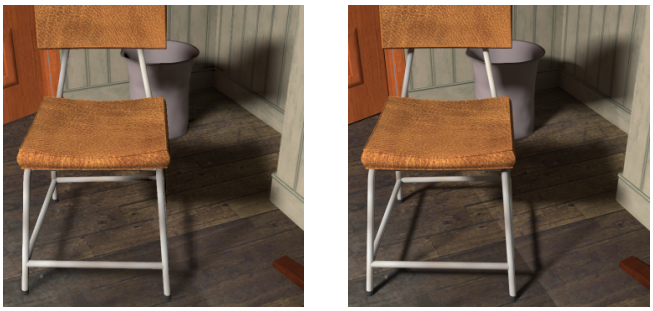
We have observed three different categories of imperfections in the results of the proposed method, which we plan to address in our future work:

1) Because of the fixed sizes of the blocks, the algorithm can sometimes have difficulty constructing illumination that is not visible in the feature data and is smaller than the block size. Example of small soft shadows can be seen in Fig. 10. Another example of this is specular highlights. High values in a small area are typically blurred as can be seen in the last row of Fig. 7. However, different order versions of the feature buffers and block place variation reduces the problem significantly. Moreover, the quality can be improved by using feature buffers containing noise-free data related to the cause of the problematic illumination. The effect of adding this kind of a buffer can be seen in Fig. 12.



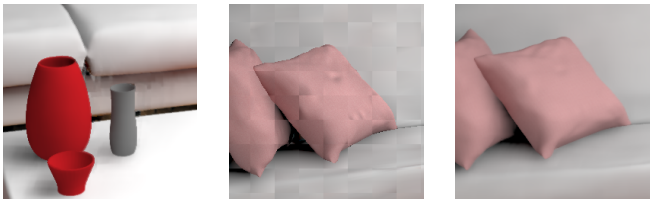
(a) 1-bounce 1 spp (b) 9-bounces 1 spp (c) 1-bounce 1 spp BMFR (d) 9-bounce 1 spp BMFR (e) 9-bounce reference

Fig. 9. An example of how the proposed method handles inputs which have more than one bounce. In this mostly indirect illumination case the 1-bounce 1 spp BMFR is slightly too dark in the green inset since it is very unlikely that the only secondary ray finds its way out from the opening. The 9 bounce 1 spp BMFR is already close to the reference. However, in the red inset the fireflies on the dark wall next to the opening cause more brightness to bleed to the wrong side of the corner. In these figures the gamma correction was modified so that the problems standout more clearly.



(a) Proposed algorithm (b) Reference

Fig. 10. If a shadow smaller than the block size is not represented in the feature buffers, the resulting shadow can be too soft. However, bigger shadows like the contact shadow of the trash can follow the reference quite closely.

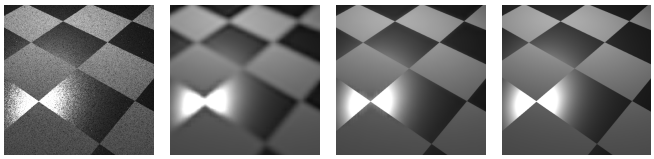


(a) Artifacts can be seen on the occluded areas. This is the worst case since the camera is moving to the right with a high speed.

(b) First frame without any accumulated data (effective 1 spp input) shows the blockwise nature of the algorithm.

(c) Accumulation and block place variation removes the blocky look from the same pillows as in Fig. 11b

Fig. 11. Different artifacts from the proposed pipeline. The lack of detailed texturing in the scene makes the artifacts stand out more than usual.



(a) 1 spp (b) The proposed BMFR (c) BMFR with material ID feature buffer (d) Reference

Fig. 12. In this example the only difference in the flat surface is its roughness. Also, albedo is constant for the whole surface but the black background makes the smoother surface seem darker. The only feature data which is not constant are the two world position axes and BMFR has to construct the final image from them. In Fig. 12c we add the material ID feature buffer, which allows BMFR to differentiate between the two materials and, therefore, improves the results significantly.

2) The proposed algorithm is affected by the same problems as the previous works that use reprojected temporal data. Since the reprojection is done to the first bounce intersection world-space position, e.g., reflections and specular highlights get overblurred. However, if the material is a completely reflecting mirror, the problem can be fixed by using a virtual world-space position, but if there are both a reflecting and a non-reflecting component in the material, we would have to store and reproject those separately [Zimmer et al. 2015]. Occlusions with the reprojected data also cause the input to have different amounts of effective spp in different screen space areas. Different effective spp causes the quality of the output of our algorithm to be decreased in the occluded areas as can be seen in Fig. 11a. The visibility of these artifacts on a still frame does not correspond to their visibility on a moving scene, due to how perception works. The artifacts are stronger in case of fast camera movement causing larger disocclusions, but those cases are also the ones where

(a) Frame 15 of the proposed algorithm. (b) Frame 15 of the reference. (c) Frame 25 of the proposed algorithm.

Fig. 13. A scene with a light moving towards the camera shows the temporal lag caused by the temporal accumulation. With the proposed temporal accumulation parameters, it takes approximately 10 frames for the proposed method to produce the most similar lighting.

artifacts get harder to be noticed by the user's perception [Reibman and Poole 2007].

Reprojected temporal data also causes lag in the lighting changes caused by animations. Fig. 13 shows a scene with a moving light. With the proposed setup (where 20% is from the newest path tracing samples and 10% is from the newest fitted frame), it takes approximately 10 frames for the image to converge to a similar appearance as the reference. However, in a real use case where the 4096 spp reference is not available, the lag is hard to notice since 10 frames is not a long time with the frame rates the proposed algorithm is able to generate. One solution to the temporal lag problem was provided in a concurrent work [Schied et al. 2018]. However, the same algorithm cannot be directly used with the proposed work because it would generate blocking artifacts to areas where illumination changes drastically.

3) The blockwise nature of the algorithm causes blocking artifacts visible in Fig. 11b. These can be seen on the first frame when there is no accumulated data in the input and no block displacement in BMFR. On the first frame, the problem could be fixed by running the fitting phase (II) twice with two different grid locations and smoothly blending the overlapping pixels from one block to another. Moreover, during the first frames in a completely new camera location it is hard for the human visual system to perceive artifacts [Reibman and Poole 2007]. However, the issue can be adequately resolved by using a fade-in effect over a few frames when the camera is "teleported" to a completely new location.

We have also tested the proposed method with more than one bounce of path tracing. An example of this is shown in Fig. 9. The only limiting factor is that the radiance of fireflies is only propagated within a single block area, which is defined in screen space. This limitation is not visible in typical scenes, but it can be a problem in dedicated test scenes where a path to the light is very unlikely to be found. However, temporal accumulation after the fitting phase robustly removes temporal artifacts caused by the fireflies. If the fireflies are very rare and there is a need for some illumination in the results, it might be possible to use path space regularization techniques [Kaplanyan and Dachsbacher 2013].

During prototyping the algorithm, we noticed that using multiple iterations of BMFR with multiple orders of features, different block locations, and different block sizes on each iteration, can reduce the artifacts discussed in this section. However, having a single iteration with fixed-sized blocks was best suited for our real-time implementation. Akin to multivariate monomials, the extended set of feature buffers in Eq. 1 may also include generic products of the form $F_{n_j}^{\gamma_j} F_{n_k}^{\gamma_k}$, however this opportunity has not been investigated for this work.

One more limitation of our algorithm is that noise in the feature buffers, due to, e.g., motion blur or depth of field, is visible in the results. These kinds of effects would require denoising the feature buffers first. However, in both examples we can compute how much the data in the feature buffer should be blurred to follow the physical phenomenon. We plan to address the problem of noisy feature buffers in future work.

9 CONCLUSIONS

In this article, we introduced *Blockwise Multi-Order Feature Regression* (BMFR). In BMFR, different powers of the feature buffers are used for blockwise regression in path-tracing reconstruction. We show that a real time GPU-based implementation of BMFR is possible; the evaluated example implementation processes a 720p frame in 2.4 ms on a modern GPU, making it 1.8× faster than the previous state-of-the-art real-time path tracing reconstruction algorithm with better quality in almost all the used metrics. The code and the data to reproduce our results is available in the supplementary material of this article.

The high execution speed of the proposed algorithm is achieved by augmented QR factorization and the use of stochastic regularization, which addresses rank-deficiencies and avoids numerical instabilities without the extra complexity of pivoting. Like in previous work, our algorithm relies on reprojecting and accumulating previous frames, which increases the effective samples-per-pixelcount in our input. Instead of using exponential moving average for the data accumulation all the time, on the first frames and after an occlusion we use a cumulative moving average of the samples. Cumulative moving average does not give an excessive weight to the very first samples and, therefore, reduces artifacts. In our algorithm we use similar accumulation also after the regression to increase the temporal stability and to decrease the amount of artifacts in the results.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers, Petrus Kivi, and Atro Lotvonen for their fruitful comments. We are also grateful to Aleksandr Diment for letting us use his workstation for the NVIDIA Titan X (Pascal) measurements, Dmytro Rubalskyi for his open-source SVGF implementation used in the comparison, and to the model makers: Frank Meinl for Sponza (CC BY 3.0, [McGuire 2017]), Guillermo M. Leal Llaguno for San Miguel (CC BY 3.0, [McGuire 2017]), Christophe Seux for Classroom (CC0), and Wig42 for Living room (CC BY 3.0, [Bitterli 2016]). Finally, this work was supported by the funding from TUT Graduate School, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Nokia Foundation, Business Finland (funding decision 40142/14, FiDiPro-StreamPro), Academy of Finland (funding decisions 297548, 310411) and ECSEL JU project FitOptiVis (project number 783162).

REFERENCES

Anne Aaron, Zhi Li, Megha Manohara, Joe Yuchieh Lin, Eddy Chi-Hao Wu, and C.-C Jay Kuo. 2015. Challenges in Cloud Based Ingest and Encoding for High Quality Streaming Media. In *Proceedings of the Image Processing*.

Timo Aila and Tero Karras. 2010. Architecture Considerations for Tracing Incoherent Rays. In *Proceedings of the High Performance Graphics*.

Chakravarty Alla Chaitanya, Anton Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *Transactions on Graphics* 36, 4 (2017).

AMD. 2017. RadeonRays SDK. Online. (2017). Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK, Referenced: January 23 2018.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *Transactions on Graphics* 36, 4 (2017).

Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Guided Image Filtering for Interactive High-quality Global Illumination. *Computer Graphics Forum* 30, 4 (2011).

Benedikt Bitterli. 2016. Rendering resources. (2016). https://benedikt-bitterli.me/resources/.

Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35, 4 (2016).

Peter Burt. 1981. Fast Filter Transform for Image Processing. *Computer Graphics and Image Processing* 16, 1 (1981).

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. 2010. Edge-avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of the High Performance Graphics*.

Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *Transactions on Graphics* 28, 3 (2009), 93.

Luke Goddard. 2014. Silencing the Noise on Elysium. In *ACM SIGGRAPH 2014 Talks*.

Kaiming He, Jian Sun, and Xiaoou Tang. 2013. Guided Image Filtering. *Transactions on Pattern Analysis and Machine Intelligence* 35, 6 (2013).

Michael Heath. 1997. *Scientific Computing*. McGraw-Hill.

Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. 2012. SMAA: Enhanced Morphological Antialiasing. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 31, 2 (2012).

Jorge Jiménez, X Wu, A Pesce, and A Jarabo. 2016. Practical real-time strategies for accurate indirect occlusion. *SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice* (2016).

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *Transactions on Graphics* 34, 4 (2015).

Anton Kaplanyan and Carsten Dachsbacher. 2013. Path space regularization for holistic and robust light transport. *Computer Graphics Forum* 32, 2pt1 (2013).

Brian Karis. 2014. High-quality Temporal Supersampling. In *ACM SIGGRAPH 2014, Advances in Real-Time Rendering in Games*.

Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. In *Proceedings of the High Performance Graphics*.

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *Transactions on Graphics* 31, 6 (2012).

Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward a Practical Perceptual Video Quality Metric. Online. (2016). Available: https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652, Referenced: January 23 2018.

Yu Liu, Changwen Zheng, Quan Zheng, and Hongliang Yuan. 2017. Removing Monte Carlo Noise Using a Sobel Operator and a Guided Image Filter. *The Visual Computer* 34, 4 (2017).

Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of the High Performance Graphics*. http://casual-effects.com/research/Mara2017Denoise/

Morgan McGuire. 2017. Computer Graphics Archive. (2017). https://casual-effects.com/data.

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *Transactions on Graphics* 33, 5 (2014).

Bochang Moon, Jose A Iglesias-Guitian, Sung-Eui Yoon, and Kenny Mitchell. 2015. Adaptive Rendering with Linear Predictions. *Transactions on Graphics* 34, 4 (2015).

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *Transactions on Graphics* 35, 4 (2016).

Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. 2010. Optix: a General Purpose Ray Tracing Engine. *Transactions on Graphics* 29, 4 (2010).

Amar Patel. 2018. *D3D12 Raytracing Functional Spec, v0.09*. Microsoft. Available: http://forums.directxtech.com/index.php?topic=5860.0, Referenced: March 23 2018.

Matt Pharr and Greg Humphreys. 2010. *Physically Based Rendering: From Theory to Implementation* (2nd ed.). Morgan Kaufmann.

Amy R Reibman and David Poole. 2007. Predicting packet-loss visibility using scene characteristics. In *Proceedings of the Packet Video*.

Gilberto Rosado. 2007. *GPU gems 3*. Addison-Wesley Professional, Chapter 27. Motion Blur as a Post-Processing Effect.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-guided Filtering: Real-time Reconstruction for Path-traced Global Illumination. In *Proceedings of the High Performance Graphics*.

Christoph Schied, Christoph Peters, and Carsten Dachsbacher. 2018. Gradient Estimation for Real-Time Adaptive Temporal Filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 2 (2018), 24.

Carlo Tomasi and Roberto Manduchi. 1998. Bilateral Filtering for Gray and Color Images. In *Proceedings of the Computer Vision*.

Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the Computer graphics and interactive techniques*.

Timo Viitanen, Matias Koskela, Kalle Immonen, Markku Mäkitalo, Pekka Jääskeläinen, and Jarmo Takala. 2018. Sparse Sampling for Real-time Ray Tracing. In *Proceedings of the GRAPP*.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *Transactions on Graphics* 33, 4 (2014).

Yong Wang, Xiaofeng Liao, Di Xiao, and Kwok-Wo Wong. 2008. One-way hash function construction based on 2D coupled map lattices. *Information Sciences* 178, 5 (2008).

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image Quality Assessment: from Error Visibility to Structural Similarity. *Transactions on Image Processing* 13, 4 (2004).

Ling-Qi Yan, Soham Uday Mehta, Ravi Ramamoorthi, and Fredo Durand. 2015. Fast 4D sheared filtering for interactive rendering of distribution effects. *Transactions on Graphics* 35, 1 (2015), 7.

Lei Yang, Diego Nehab, Pedro V Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. 2009. Amortized Supersampling. *Transactions on Graphics* 28, 5 (2009).

Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. 34, 4 (2015).

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum* 34, 2 (2015).