Spatiotemporal Variance-Guided Filtering for Motion Blur

MAX OBERBERGER, AMD, Germany and Technical University of Munich, Germany
MATTHÄUS G. CHAJDAS, AMD, Germany
RÜDIGER WESTERMANN, Technical University of Munich, Germany

Fig. 1. We reconstruct motion blur from a 1 sample per pixel ray traced input (left). We compare our results (center) against brute force Monte Carlo motion blur (right) with 4096 samples per pixel.

Adding motion blur to a scene can help to convey the feeling of speed even at low frame rates. Monte Carlo ray tracing can compute accurate motion blur, but requires a large number of samples per pixel to converge. In comparison, rasterization, in combination with a post-processing filter, can generate fast, but not accurate motion blur from a single sample per pixel.

We build upon a recent path tracing denoiser and propose its variant to simulate ray-traced motion blur, enabling fast and high-quality motion blur from a single sample per pixel. Our approach creates temporally coherent renderings by estimating the motion direction and variance locally, and using these estimates to guide wavelet filters at different scales.

We compare image quality against brute force Monte Carlo methods and current post-processing motion blur. Our approach achieves real-time frame rates, requiring less than 4ms for full-screen motion blur at a resolution of 1920×1080 on recent graphics cards.

CCS Concepts: • **Computing methodologies → Ray tracing**.

Additional Key Words and Phrases: motion blur, ray tracing, reconstruction, real-time rendering

Authors' addresses: Max Oberberger, AMD, Munich, Germany and Technical University of Munich, Munich, Germany, max.oberberger@tum.de; Matthäus G. Chajdas, AMD, Munich, Germany, chajdas@tum.de; Rüdiger Westermann, Technical University of Munich, Munich, Germany, westermann@tum.de.

2577-6193/2022/7-ART
https://doi.org/10.1145/3543871

ACM Reference Format:
Max Oberberger, Matthäus G. Chajdas, and Rüdiger Westermann. 2022. Spatiotemporal Variance-Guided Filtering for Motion Blur. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 3 (July 2022), 13 pages. https://doi.org/10.1145/3543871

1 INTRODUCTION

When capturing a still image, objects that move during the exposure time appear blurred. The same effect also applies to videos and even the human eye. Video games often try to emulate motion blur to convey the sense of motion, e.g. in racing games. Most modern games and game engines generate motion blur in a post-process effect [Epic Games 2022; Rosado 2007; Unity Technologies 2021]. Post-process motion blur can produce convincing results, but quality and accuracy is limited, as disocclusion can not be solved by traditional rasterization-based rendering. Distributed ray tracing [Cook et al. 1984] in combination with brute force Monte Carlo methods can generate photo-realistic motion blur at the expense of significantly increased rendering time, thereby exceeding time constraints for real-time rendering. The introduction of hardware accelerated ray tracing [Wyman et al. 2018] allowed real-time ray tracing with a limited number of rays per pixel. The overall increased capabilities of modern graphics cards lead to many contributions in the field of reconstructing low sample count path traced images.

Based on the works of Schied et al. [Schied et al. 2017], we introduce a novel reconstruction filter for ray traced motion blur. The filter generates a temporally stable image sequence from a one ray per pixel input. Rays are only distributed over time and no additional stochastic ray tracing effects are handled by the filter. Temporal accumulation is used to increase the effective sample count. The filter follows a hierarchical structure with separate reconstruction of moving object and static objects/background. Fullscreen motion blur is typically computed in less than four milliseconds, thus allowing ray traced motion blur to be used in realtime computer graphics.

2 RELATED WORK

Post-Process Motion Blur. To convey the sense of speed, a post-process effect is used in most games to simulate motion blur. This effect works through blurring the current framebuffer by accumulating multiple samples along a line [Ritchie et al. 2010; Rosado 2007]. Orientation and length of this line are given by a per-pixel motion vector. This causes the blur to only be applied to the visible area of the moving object, i.e. blurring the object inwards. Some post-process motion blur implementations [McGuire et al. 2012; Shimizu et al. 2003] extends the blurred object's visible area to create a more realistic effect. As all of the above mentioned effects rely on a rasterized framebuffer as an input, none of them can retrieve information behind the moving object. This disocclusion leads to artifacts, especially if multiple moving object are overlapping.

Stochastic Motion Blur. Disocclusion can be solved via distributed ray tracing [Cook et al. 1984]. A random timestamp within the frame's shutter is assigned to each ray. Ray origin and direction as well as the scene geometry is adjusted to match this timestamp. Accumulating multiple samples yields a converged image. Stochastic rasterization [Akenine-Möller et al. 2007; Enderton and McGuire 2012] can achieve similar results, whilst using standard rasterization hardware. Triangle geometry is converted to time-continuous triangles and visibility can be altered by discarding specific fragments to mimic stochastic ray tracing. Due to performance limitations of the time-continuous triangles, stochastic rasterization can be seen as superseded by hardware accelerated ray tracing.

Motion Blur Reconstruction. A reconstruction filter generates a noise-free image from a low sample count Monte Carlo input. As ray tracing is a very time consuming process, minimizing the required samples can greatly decrease render times. Different ways of reconstructing stochastic motion blur have already been explored. Egan et al. analyse motion blur in the frequency domain and arrive

at an approximation for the spectrum of moving scenes, which can be used to compute adaptive sampling rates [Egan et al. 2009]. They also introduce a sheared filter, which is aligned with the motion direction and allows for an input sample rate of typically four to eight samples per pixel. Munkberg et al. [Munkberg et al. 2014] present a layered approach for simultaneous reconstructing of motion blur and defocus blur. Hasselgren, Munkberg, and Vaidyanathan [Hasselgren et al. 2015] add significant performance improvements to this reconstruction approach by leveraging anisotropic sampling present on modern GPUs. All of the above mentioned contributions are aimed at accelerating ray traced motion blur by reducing the required sample count to 4-8 samples per pixel. On current hardware, this still exceeds performance targets for real-time graphics. Lehtinen et al. [Lehtinen et al. 2011] present a general reconstruction technique, that can also be applied to motion blur. Their approach is designed with multiple input samples in mind, but can reconstruct motion blur from a single sample per pixel. With a filter runtime of multiple seconds, this approach can not be adopted for real-time graphics.

Modern Reconstruction Techniques. The increased interest in real-time ray tracing has lead to significant contributions in the field of reconstructing realistic light transport from a limited number of ray traced input samples. Mara et al. present a hybrid approach, which uses rasterization for direct and two ray traced samples for indirect lighting. Dammertz et al. [Dammertz et al. 2010] present an edge-avoiding filter using a hierarchical à-trous wavelet transform to reconstruct one sample per pixel global illumination. The wavelet transform is iteratively applied on the output of the previous iteration. The filter kernel has a constant number of non-zero coefficient samples, with a per-iteration increasing number of zero coefficients in between. Schied et al. [Schied et al. 2017] extend this concept by adding a continuously updated variance estimate, which guides the reconstruction filter. The resulting filter reconstructs global illumination from a single path traced sample per pixel. Our paper adapts this filtering technique to stochastic motion blur.

Neural Reconstruction Techniques. In recent years, neural reconstruction approaches have become a viable alternative to classic filtering denoisers. Chaitanya et al. [Chaitanya et al. 2017] present an autoencoder based denoising filter for global illumination at interactive framerates. Hasselgren et al. [Hasselgren et al. 2020] use temporal data and guides to generate a neural sample map estimate for adaptive sampling and subsequent reconstruction. Both methods only focus on reconstructing global illumination and rely on locally consistent visibility. Thus neither method is suitable or adaptable to stochastic motion blur reconstruction.

3 RECONSTRUCTION PIPELINE

In the following section, we provide an overview of our reconstruction pipeline, its components and the data flow between them.

Ray Tracing. The ray tracing implementation used to generate input data for the reconstruction filter closely follows distributed ray tracing, as described by Cook et al. [Cook et al. 1984]. A pseudorandom number generator [Blackman and Vigna 2021] is used to generate a timestamp t within the interval $[0; 1]$ for each ray. The timestamp describes the ray's temporal position inside the frame's shutter, whereby $t = 0$ and $t = 1$ refer to beginning and end of the frame respectively. Linear interpolation of camera and object transforms is used to represent the scene at the given timestamp.

Our ray tracing implementation is based on a DirectX raytracing sample for real-time ray-traced ambient occlusion provided by Microsoft [Microsoft 2019]. Hardware accelerated ray tracing [Microsoft 2021; Wyman et al. 2018] is used to achieve interactive framerates.

The ray tracer generates a one sample per pixel color output. An auxiliary framebuffer also provides the screen-space motion vector, depth and ray timestamp to the subsequent reconstruction filter. The data in the auxiliary buffer is stored as packed 16-bit floating-point numbers to save

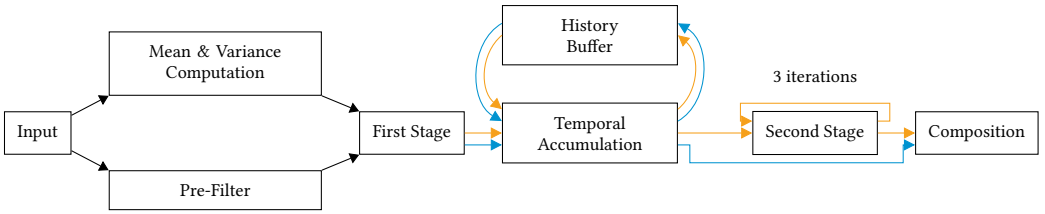


Fig. 2. Data flow in the reconstruction pipeline. Pre-processing computes the mean motion direction and variance, which is used to guide the filter. The filtering section separates moving (orange) from non-moving (blue) samples and reconstructs both independently. Temporal accumulation is used to increase effective sample counts. Composition combines moving and non-moving output of the filtering section.

memory bandwidth. Further frame data, such as normals or mesh ids are not required. No secondary rays (shadows, reflections) are traced.

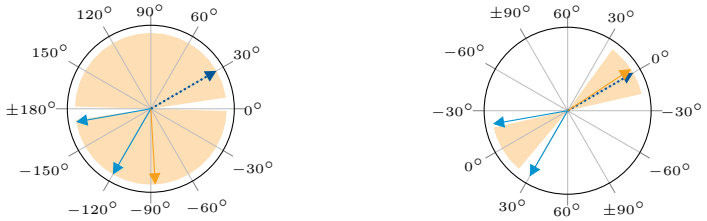
Reconstruction. Figure 2 visualizes data flow inside the reconstruction filter. We first use the per-pixel motion vector to compute a local variance estimate, which is later used to control the filtering process. Next, the first iteration of the wavelet transform filter separates moving pixel from the static background. We refer to this step as the *first stage*. The filtered results are stored in separate textures, referred to as *moving* and *non-moving* texture/channels. This separation is critical, as the à-trous wavelet transform is applied in multiple, subsequent iterations. Schied et al. [Schied et al. 2017] demodulate surface albedo from lighting reconstruction for the same reason. Filtering and storing moving and non-moving pixel together would thus either result in blurring non-moving samples or insufficient blurring of moving samples. Moving pixels are processed by an additional pre-filter, which increases the effective sample count. After this initial filtering step, the history buffer is re-projected onto the current frame. Re-projected samples are integrated into both channels. The moving texture is processed by three more iterations of the à-trous wavelet transform, each with increasing filter kernel footprint. We refer to this as the *second stage*. Lastly, the moving texture is composited on top of the non-moving texture. A blending factor, which is computed as a bi-product of the variance estimate, is used to correctly layer both textures.

4 SPATIOTEMPORAL FILTER

4.1 Variance Estimate

In their seminal work, Schied et al. [Schied et al. 2017] introduce the idea of using a variance estimate to control or guide the filtering process. Temporal variance of the color luminance, accumulated over multiple frames, is used as a proxy for detecting noise. For motion blur, temporal accumulation is challenging, as geometric consistency can not be ensured for raw stochastic inputs. In addition, a per pixel temporal variance estimate does not yield correct results if a moving object passes over a static pixel and no samples from the moving object have been recorded yet. Variance in the input signal is directly caused by motion. Thus the motion vector, or rather its length, can be used as a variance estimate instead.

We filter a 15×15 pixel area of the motion vector in the full-resolution input and compute the mean motion vector length. To increase performance, the filtering is done in 2×2 tiles, i.e. at half resolution. Non-moving pixels are ignored and a depth threshold is used to retain sharp edges of occluding geometry. In order to align the subsequent reconstruction filter with the motion vector, the mean motion direction is also computed as the angle between the motion vector and the x-axis. Discontinuity between $0°$ and $360°$ negatively impacts both mean and variance computation. Thus a local approach, with angles relative to the first sample (center of the filter region) was chosen



(a) $\mu = -86.6°, \sigma = 84.9$ (b) $\mu = 3.3°, \sigma = 20.5$

Fig. 3. Computing mean motion angle as angle between motion vector and x axis can lead to undesired results and high variance (*left*). A rotated, wrap-around coordinate system (*right*) is used to compute the mean motion angle (orange) and variance relative to the first motion vector (dotted, 30°).

instead. Back-facing directions with relative angles outside the $[-90°; +90°]$ interval are flipped by 180°. The filter does not differentiate between motion in a single direction and motion in opposite direction. Note that the direction (i.e. sign) of the angle is important and thus computing the relative angles via the dot product in not possible. Figure 3 compares the computation using relative angles to a naive approach. The angle variance σ_θ^2 is normalized to the interval $[0; 1]$.

Mean motion vector length and angle define scale and rotation of the à-trous wavelet transform. Additionally, the local variance of both motion vector length and motion vector angle is used to adjust reconstruction filter parameters. As the variance estimate is not dependent on the color input, updating the estimate during filtering is not needed.

4.2 Pre-Filter

Our implementation of the hierarchical à-trous wavelet transform, as presented by Dammertz et al. [Dammertz et al. 2010], deviates from the fixed step sizes for each level. Instead the kernel footprint is adjusted to match the motion vector magnitude. This way not every pixel is guaranteed to be directly included in the hierarchical filter. We apply a pre-filter to collect up to 18 random contributions along the current pixel's motion vector. This filter mimics the effect of rendering the image with multiple samples per pixel, by accumulating select samples onto the current one.

For each pixel p with motion vector v_p and timestamp t_p, a random sample

$$s = p + v_p(t' - t_p) + j$$

with random value $t' \in [0; 1]$ and jitter offset $j \in [-1.5; 1.5]^2$ is generated. Two samples are taken for each pixel. Neighboring samples are shared across a 3×3 area via local data share memory. We evaluate the distance $d(p, s)$ between p and the motion vector line segment of s (see eq. 1). Samples below the threshold of 2.5 are accumulated. The pre-filter is skipped for pixels with a motion vector magnitude less than 3. Figure 4 shows the processed output of the pre-filter as well as the final filter output with and without the pre-filter enabled. The pre-filter helps reduce input variance and filter outputs more closely match ground truth data, especially along object edges.

4.3 Spatial Filter

Spatial filtering is done in two stages. The first stage separates moving and non-moving pixels and applies the first iteration of the wavelet transform separately to both channels. The output of the first stage is enhanced by temporal accumulation (see sec. 4.4). The second stage applies three iterations of the wavelet transform to the moving channel. The filter is only applied in region with a mean motion vector length greater than zero.

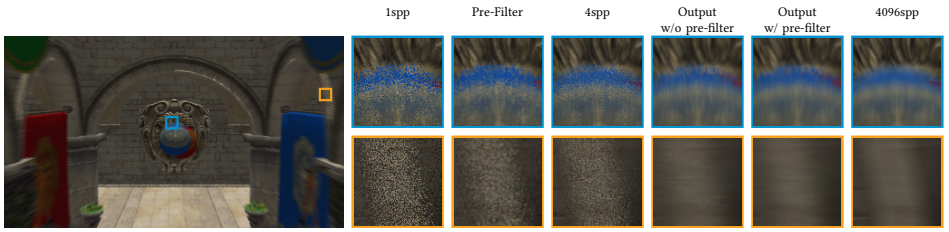


Fig. 4. The pre-filter increases the effective sample rate by collecting contributions from distant samples. Filtered images resemble inputs with 4 – 8 samples per pixel. Crytek Sponza by Frank Meinl [McGuire 2017], CC BY 3.0

À-trous wavelet transform. We use a 5×5 cross-bilateral filter kernel with a single weight function $w(p, q)$ between kernel center p and sample q. The weight function determines, whether the sample q could be found at position p with a different timestamp. Given the sample position q, motion vector v_q and timestamp t_q, we can compute the line segment $q_0 q_1$, along which q moves during the frame as follows:

$$q_0 = q - v_q t_q$$
$$q_1 = q + v_q (1 - t_q)$$

The weight function is based on the distance $d(p, q)$ between p and this line segment.

$$l = \frac{(p - q_0) \cdot (q_1 - q_0)}{||q_1 - q_0||^2}$$
$$d(p, q) = ||p - (q_0 + l(q_1 - q_0))|| \tag{1}$$

If l exceeds the interval $[0; 1]$, the sample is assigned a zero-weight.

The weight function is inversely proportional to $d(p, q)$ and is computed via linear mapping. The mapped interval is defined by the local motion vector length variance $\sigma^2_{||v||}$ and normalized motion angle variance σ^2_θ. Parameters β_l and β_a adjust influence of each variance respectively.

$$c_a = 1 + \min\{1, \beta_a \sigma^2_\theta\}$$
$$c_l = 1 + \min\{1, \beta_l \sigma^2_{||v||}\}$$
$$w(p, q) = 1 - d(p, q) \frac{1}{c_a c_l}$$

In testing $\beta_l = 10$ and $\beta_a = 1$ were found to yield good results. Figure 5 visualizes our weight function as a distance field.

First Stage. The first stage applies the à-trous wavelet transform with the aforementioned weight function to all moving pixels. The wavelet transform filters both color and motion vector values. No scaling or rotation is applied to the filter kernel. Non-moving pixels are filtered using the same 5×5 cross-bilateral filter kernel, but without any weight functions. A depth threshold is used to not filter any moving samples, that are occluded by non-moving geometry.

Second Stage. The second stage only filters the moving channel. The filter kernel is rotated to align with the local mean motion angle (see sec. 4.1). Scaling factor s_x controls filter kernel step width along the motion vector and s_y orthogonally. We deviate form the fixed kernel size and exponential 2^{i-1} increase in zero-weight contribution, as presented by [Dammertz et al. 2010] and scale the kernel step by the local mean motion vector length l. s_x for iteration i is calculated as

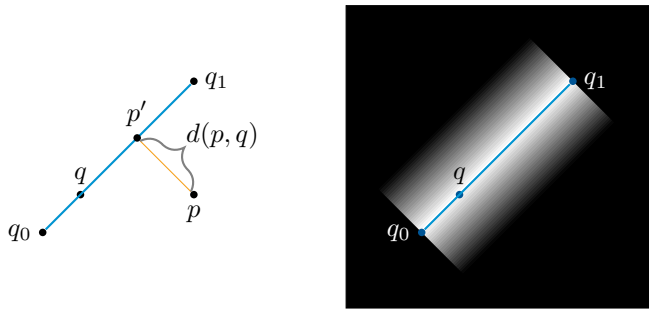


Fig. 5. Sample q is weighted proportional to the shortest distance $d(p,q)$ between the motion line segment of q and the kernel center p (*left*). If the projection p' is not on the segment q_0q_1, q receives a zero-weight. The resulting weight function is shown as a distance field on the right.

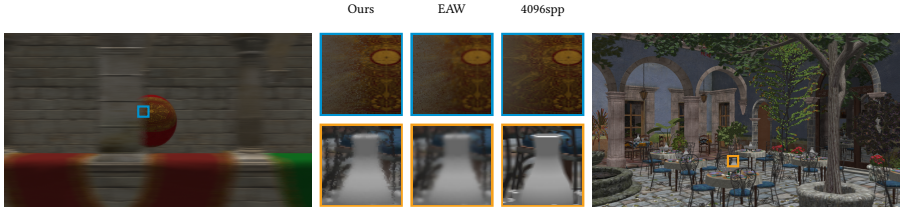


Fig. 6. Fast (Sponza) and slow (San Miguel) sideways camera movement. Comparison of our kernel step scaling against the original edge-avoiding à-trous wavelet transform (EAW) [Dammertz et al. 2010]. The fixed size kernel does not cover the entire motion vector length of fast moving objects. For slow movement, EAW does not collect enough close-by samples, resulting in remaining variance, while our filter slightly overblurs the image. San Miguel by Guillermo M. Leal Llaguno [McGuire 2017] CC BY 3.0

follows:

$$s_x = \frac{1}{5}\frac{l}{2^{3-i}}$$

The constant $\frac{1}{5}$ converts the filter kernel footprint scale to kernel step scale for a 5×5 filter kernel. The scaling factor $\frac{1}{2^{3-i}}$ mimics the exponential increase of 2^{i-1} zero weights in the original à-trous filter implementation. To account for contributions with different motion direction, the normalized angle variance σ_θ^2 is used to determined the orthogonal kernel step, by means of linear interpolation between 1 and s_x. Scaling and rotation are computed as floating points and rounded to whole pixels. This modified kernel scaling mostly affects areas with high motion vector magnitude, as distant samples would not be reached by a fixed size kernel (see fig. 6).

4.4 Temporal Filter

To increase the effective sample count used in the reconstruction, we use temporal accumulation to reuse samples from previous frames. We store partially filtered results from the previous frame in a history buffer an re-project it to fit the current frame. Traditional re-projection methods use the motion vector to find candidate samples in the history buffer and validate them using consistency checks based on depth and normals [Karis 2014; Schied et al. 2017]. For motion blur, primary visibility is not consistency between frames, thus these consistency checks may not yield correct

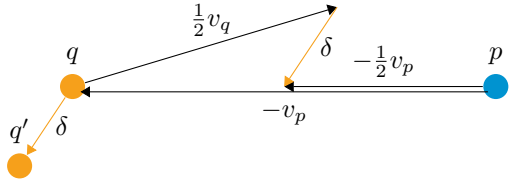
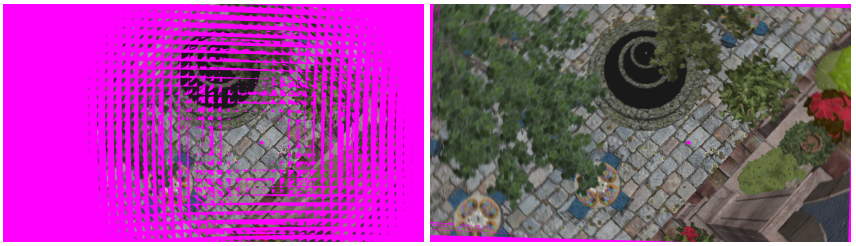


Fig. 7. Temporal re-projection uses the difference δ between back-projecting the current sample p and forward-projecting the sample candidate q to generate the next sample candidate q'. Updates are applied iteratively until $||\delta||$ is below a set threshold.



(a) 1 Iteration (b) 2 Iterations

Fig. 8. Temporal re-projection of camera rotation. Pixel colored in pink indicate failed re-projection. The iterative back- and forwards-projection (right) can re-project challenging scenarios, like camera rotation. Simple back-projection (i.e. 1 iteration; left) can only re-project a smaller subset of samples in the same situation.

results. Instead we implement an iterative search algorithm to find and evaluate candidate samples in the history buffer.

Given the current pixel p, we use its motion vector v_p to find an initial guess q. If q is a valid sample, using v_q to forward-project it should yield p. We use the filtered motion vector output of the first stage for the initial guess and refinement, as raw samples from the ray tracer contain too much variance. As filtered timestamps converge towards 0.5, we compute the difference δ between this backward- and forward projection at the half-way point (using $\frac{1}{2}v_p$ and $\frac{1}{2}v_q$ instead). We use a lower threshold to accept valid samples and an upper threshold to cancel the re-projection process. If neither threshold is met, δ is used to update the sample position.

$$
\begin{aligned}
q &= p - v_p \\
\delta &= \left(p - \frac{1}{2}v_p\right) - \left(q + \frac{1}{2}v_q\right) \\
q' &= q + \delta
\end{aligned}
$$

Figure 7 visualizes this process. In testing we found an iteration limit of 2 to be sufficient for most scenarios.

The final q' is expected to better connect with p, i.e. forward-projection of q' using the actual timestamp would result in a samples that moves through or close to the current sample. In most situations, one iteration, i.e. no update step, is sufficient for re-projection. Difficult to re-project scenarios, such as rotation are handled by at most two iterations (see fig 8).

To integrate the re-projected samples into reconstruction pipeline, we recreate the value sum and weight sum for both moving and non-moving channels from the output of first stage. The same

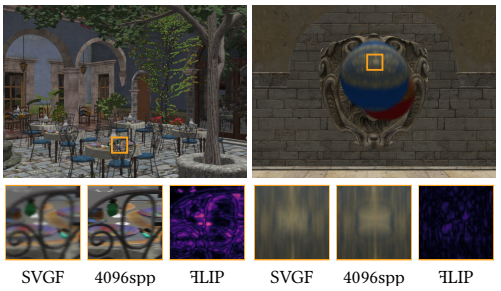


Fig. 9. Motion blur reconstruction introduces a slight overall blur to the image (*left*). Fine details can not be fully reconstructed, even with temporal accumulation (*right*).

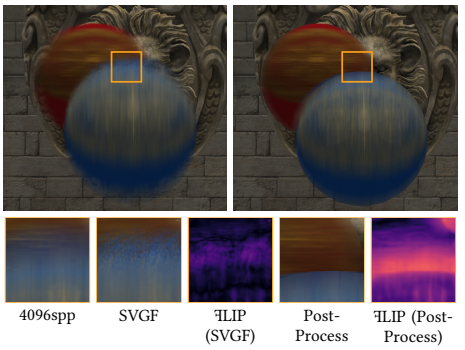


Fig. 10. Two spheres moving in different directions. Disocclusion occurs at the edge between the spheres.

non-rotated, non-scaled filter kernel from the first stage is used and samples are directly added to the output of the prior filter. The filtered output is copied into the history buffer.

5 RESULTS

We evaluate our reconstruction filter under the following metrics: image quality, temporal stability and performance. Image quality is evaluated in comparison to 4096 samples per pixel ray traced motion blur. All tests were conducted on an AMD Radeon RX 6800 XT. Frame rates were locked at 60 frames per second.

5.1 Image Quality

We evaluate the quality of our reconstruction filter by comparing filtered images to brute force Monte Carlo images with 4096 samples per pixel. We use the difference evaluator ꟻLIP [Andersson et al. 2020], as it is specifically designed for such comparisons. Our test scenes include a mixture of camera and object movements.

As no state-of-the-art reconstruction algorithm, which generates motion blur from a one sample per pixel input within the same time budget (i.e. real-time) is known to us, we chose to only compare our implementation against ground truth data. In addition, we draw comparisons to a simple post-process motion blur implementation based on [Ritchie et al. 2010; Rosado 2007] and the state-of-the-art post-process motion-blur in Epic Games' Unreal Engine 4 [Epic Games 2022], in order to demonstrate core limitations of post-process techniques.

In general, our filter reduces variance in the input signal to an indistinguishable amount, when viewed as a continuous image sequence. In still images, some remaining noise can be seen mostly along edges of objects. These regions do not contain enough information for complete reconstruction and are also challenging for our temporal re-project algorithm. The filter introduces a slight overall blurring to all moving areas, even slow moving ones (see fig. 9). This can be traced back to slight errors in temporal re-projection and the distance threshold in the filter weight function.

Ray traced motion blur solved the disocclusion problem typically associated with post-process motion blur. Figure 10 compares disocclusion along the edge of two object moving in different directions. Our filter closely matches ground truth results, whilst the post-process effects fails to generate the smooth blending of both objects. Similar artifacts are also visible in Epic Games' Unreal Engine [Epic Games 2022] (see fig. 11).

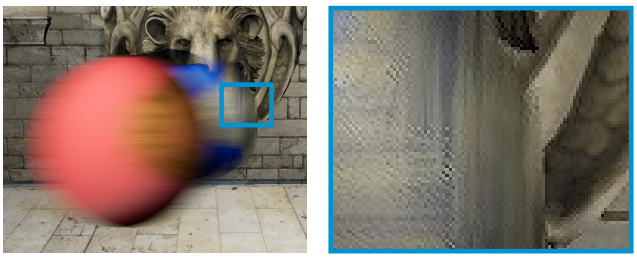


Fig. 11. Post-process motion blur rendered in Unreal Engine 4.27. Spheres moving in horizontal (red) and vertical (blue) direction. The post-process motion blur effect can not resolve disocclusion artifacts.

5.2 Temporal Stability

Evaluating temporal stability of moving images is difficult. Traditional difference evaluators, such as ꟻLIP, only compare one frame against a ground truth reference. Differences between frames can not be used, as the content changes due to motion. The ground truth data is assumed to be temporally stable. Individual frames are compared to their respective ground truth and the mean ꟻLIP error is measured over time. This error is used as a proxy for determining temporal stability. Figure 12 shows the mean and weighted median error over time. A consistent low error rate is maintained and visual inspection concluded with no temporal inconsistencies.

5.3 Performance

Our reconstruction filter is aimed at reconstructing ray traced motion blur with interactive framerates. Figure 13 shows filter execution time in different scenarios. Runtime consistently remains below 4 milliseconds in all test cases. Performance is directly bound by the screen-space area with visible motion (see figure 13). Motion vector magnitude and orientation have a negligible effect on performance. In scenarios without any visible motion, the filter runtime is $1 - 1.5$ms. This can mostly be attributed to the variance estimate, which filters the entire framebuffer. In general, filter performance is limited by the available memory bandwidth.

Average runtime of the post-process motion blur filters is 0.4ms in the case of the simple reference implementation and 1.5 milliseconds in Unreal Engine 4. The performance advantages of the post-process filters can be attributed to the reduced algorithmic complexity and reduced number of texture reads as compared to the presented filter.

6 LIMITATIONS AND FUTURE WORK

Our filter can reduce variance in a noisy, one sample per pixel input. Nevertheless, we identify the following limitations and opportunities for future contributions.

Shadows & Reflections. The current filter implementation does not account for secondary ray effects, such as reflections or shadows. Our ray tracer supports both shadows and glossy reflections. Reflections on moving objects mostly remain stationary, while shadows cast by moving objects introduce additional variance on non-moving surfaces (see figure 14). As these effects are directly part of the color information, the filter can not differentiate between primary and secondary rays. Additional variance introduced by stochastic secondary rays, such as global illumination or soft shadows is also not handled by our reconstruction filter. Passing separate color and motion vector information for each secondary ray and filtering them separately could solve this issue.

Lighting. Similar to shadows and reflections, lighting is baked into the color information. Thus lighting, or rather the movement of lights, can not be inferred from the input signal. By providing

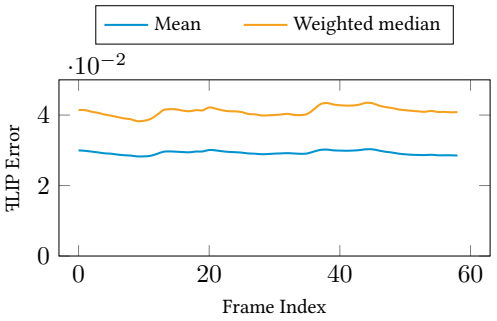


Fig. 12. Mean and weighted median FLIP error measured for 60 individual frames. We compare the filter output to a 4096 samples per pixel reference. Error rates remain low for the entire duration of the sequence. In combination with a visual inspection, the reconstructed output seen as a temporally stable image sequence.

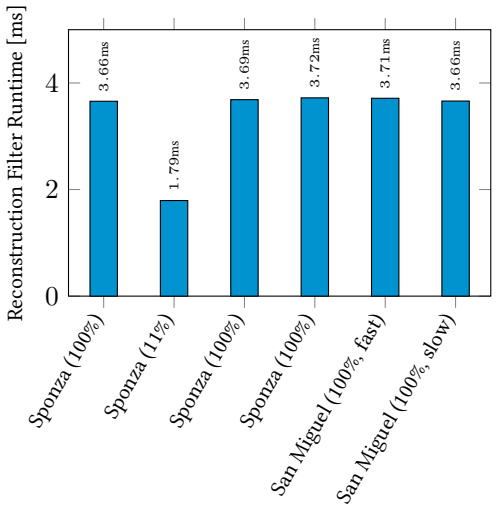
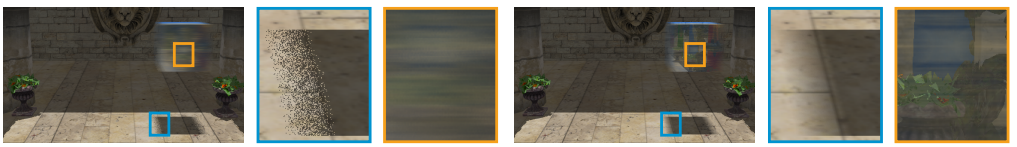


Fig. 13. Reconstruction filter times in different scenes and with different motion. Numbers in brackets indicate screen-space area with moving pixels. Measurements were taken at a resolution of 1920 × 1080 pixels on an AMD Radeon RX 6800 XT.



(a) Reconstruction Filter (b) 4096spp

Fig. 14. Reflections and shadows cast by moving object are currently not handled by the reconstruction filter. Variance in the shadows can not be filtered without blurring the static background. The reconstruction filter can not preserve stationary reflections on moving objects.

additional surface information, lighting could be demodulated from the input color and could be filtered separately. Motion between frames is usually rather small. One could argue, that stochastically moving lights would not yield significant improvement to photorealistic images and can therefore be omitted.

Non-Linear Motion. Currently motion is defined as linear interpolation between two transforms. Epic Games' Unreal Engine 4 includes a special version of their post-process motion blur filter, which also supports radial motion blur [Epic Games 2019]. Adding similar functionality to the reconstruction filter could improve results, especially in challenging scenarios, such as the spinning propellers of the Buster Drone. Significant changes to the ray tracer and reconstruction filter are needed to integrate non-linear motion.

Adaptive & Reservoir Sampling As explained in section 5.1, the one sample per pixel input provides sufficient information to reconstruct most scenarios. Areas with exceedingly high variance can not be fully reconstructed. Increasing the number of input samples in these areas would improve the

reconstruction. Finding high variance areas prior to the ray tracing step is challenging. Under the assumption that motion is mostly continuous, the variance estimate of the previous frame could be forward-projected onto the current frame. Alternatively a second ray tracing pass could be used after the current variance estimate is available. Hasselgren et al. [Hasselgren et al. 2020] present a neural sample map estimator based on temporal and auxiliary data. It may be possible to adapt this technique to use a non-stochastic, rasterized image, possibly at a lower resolution, to generate a sample map estimate for motion blur.

Reservoir sampling [Ouyang et al. 2021] collects and evaluates a set of samples with high contributions in reservoirs. Reservoir are filled with both neighboring and temporally accumulated samples. Unfortunately, we see limited applicability of this techniques for motion blur. Samples are distributed over time, resulting in a very limited sample space and every sample yielding the same contribution. Therefore no reservoir can be build or shared between neighboring pixels.

7 CONCLUSION

In this paper, we present a novel reconstruction filter for ray traced motion blur. Our filter generates a temporally stable image sequence of accurate motion blur from a one sample per pixel stochastic input. The required processing time consistently remains below four milliseconds, making the filter ready for real time applications, such as games.

We adapted spatiotemporal variance-guided filtering to stochastic motion blur. Our filter uses motion as a variance estimate. We also introduce an iterative algorithm for re-projecting a partially filtered, noisy history buffer.

Image quality of our filter does not fully match ray traced ground truth data. Variance in the input signal is reduce to an indistinguishable amount when viewed as an image sequence, but is visible in a still image. Compared to traditional post-process motion blur, our filter creates better consistency between frames, which leads to a more convincing effect.

Performance of our filter is suitable for real-time applications. At a framerate of 30 frames per second, filtering consumes less than 15% of the total frame time. This brings interactive applications, such as games, closer to fully ray traced rendering.

REFERENCES

Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. 2007. Stochastic Rasterization Using Time-Continuous Triangles. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware (GH '07)*. Eurographics Association, Goslar, DEU, 7–16.

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Aström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2 (2020). https://doi.org/10.1145/3406183

David Blackman and Sebastiano Vigna. 2021. Scrambled Linear Pseudorandom Number Generators. *ACM Trans. Math. Softw.* 47, 4 (2021). https://doi.org/10.1145/3460772

Chakravarty R. Alla Chaitanya, Anton S. 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. *ACM Trans. Graph.* 36, 4, Article 98 (jul 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

Robert L. Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed Ray Tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '84)*. Association for Computing Machinery, New York, NY, USA, 137–145. https://doi.org/10.1145/800031.808590

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. 2010. Edge-Avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of the Conference on High Performance Graphics (HPG '10)*. Eurographics Association, Goslar, DEU, 67–75.

Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Fredo Durand, and Ravi Ramamoorthi. 2009. Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur. *ACM Transactions on Graphics (SIGGRAPH 09)* 28, 3 (2009). http://graphics.cs.berkeley.edu/papers/Egan-FAS-2009-07/

Eric Enderton and Morgan McGuire. 2012. Stochastic Rasterization.

Epic Games. 2019. Radial Motion Blur. https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/HowTo/RadialMotionBlur/

Epic Games. 2022. Motion Blur. https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/PostProcessing/1_12/

Jon Hasselgren, Jacob Munkberg, Marco Salvi, Anjul Patney, and Aaron Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. *Computer Graphics Forum* (2020). https://doi.org/10.1111/cgf.13919

Jon Hasselgren, Jacob Munkberg, and Karthik Vaidyanathan. 2015. Practical Layered Reconstruction for Defocus and Motion Blur. *Journal of Computer Graphics Techniques (JCGT)* 4, 2 (2015), 45–58. http://jcgt.org/published/0004/02/04/

Brian Karis. 2014. High-Quality Temporal Supersampling. In *SIGGRAPH Courses: Advances in Real-Time Rendering in Games.*

Jaakko Lehtinen, Timo Aila, Jiawen Chen, Samuli Laine, and Frédo Durand. 2011. Temporal Light Field Reconstruction for Rendering Distribution Effects. *ACM Trans. Graph.* 30, 4 (2011).

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

Morgan McGuire, Padraic Hennessy, Michael Bukowski, and Brian Osman. 2012. A Reconstruction Filter for Plausible Motion Blur. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '12)*. Association for Computing Machinery, New York, NY, USA, 135–142. https://doi.org/10.1145/2159616.2159639

Microsoft. 2019. D3D12 Raytracing Real-Time Denoised Ambient Occlusion sample. https://github.com/microsoft/DirectX-Graphics-Samples/tree/master/Samples/Desktop/D3D12Raytracing/src/D3D12RaytracingRealTimeDenoisedAmbientOcclusion

Microsoft. 2021. DirectX Raytracing (DXR) Functional Spec. https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html

Jacob Munkberg, Karthik Vaidyanathan, Jon Hasselgren, Petrik Clarberg, and Tomas Akenine-Möller. 2014. Layered Reconstruction for Defocus and Motion Blur. *Computer Graphics Forum* 33, 4 (2014), 81–92. https://doi.org/10.1111/cgf.12415

Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. *Computer Graphics Forum* 40, 8 (2021), 17–29. https://doi.org/10.1111/cgf.14378

Matt Ritchie, Greg Modern, and Kenny Mitchell. 2010. Split Second Motion Blur. In *ACM SIGGRAPH 2010 Talks (SIGGRAPH '10)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1837026.1837048

Gilberto Rosado. 2007. Motion Blur as a Post-Processing Effect. In *Gpu Gems 3*, Hubert Nguyen (Ed.). Addison-Wesley Professional, 575–582. https://developer.nvidia.com/gpugems/gpugems3/part-iv-image-effects/chapter-27-motion-blur-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 High Performance Graphics (HPG '17)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3105762.3105770

Clement Shimizu, Amit Shesh, and Baoquan Chen. 2003. Hardware accelerated motion blur generation. In *EUROGRAPHICS*, Vol. 22. 2003.

Unity Technologies. 2021. Post Processing Effects: Motion Blur. https://learn.unity.com/tutorial/post-processing-effects-motion-blur-2019-3

Chris Wyman, Shawn Hargreaves, Peter Shirley, and Colin Barré-Brisebois. 2018. Introduction to DirectX Raytracing. In *ACM SIGGRAPH 2018 Courses (SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3214834.3231814