Supplemental Material: Improved Stochastic Texture Filtering Through Sample Reuse

BARTLOMIEJ WRONSKI, NVIDIA, USA
MATT PHARR, NVIDIA, USA
TOMAS AKENINE-MÖLLER, NVIDIA, Sweden

ACM Reference Format:
Bartlomiej Wronski, Matt Pharr, and Tomas Akenine-Möller. 2025. Supplemental Material: Improved Stochastic Texture Filtering Through Sample Reuse. *Proc. ACM Comput. Graph. Interact. Tech.* 37, 4, Article 111 (August 2025), 6 pages. https://doi.org/XXXXXXX.XXXXXXX

This supplemental material includes further discussion of Monte Carlo estimates with stochastic texture filtering (STF), a variance analysis of filtering after shading, further examples of texel sharing footprints as well as a description of our optimization algorithm for generating sparse footprints, an example implementation of the `GetFilterPMF` function used in the example implementation in Section 4.5, and some results and discussion about the use of our sample sharing techniques with volumetric ray marching.

S-1 STF and Monte Carlo Integral Estimates

Pharr et al.'s paper on stochastic texture filtering did not make a direct connection between integral Monte Carlo estimators and the stochastic texture filtering algorithms introduced there but rather derived STF algorithms by framing them as stochastic evaluation of sums of weighted texel values [S3]. Because our weighted STF estimator, Equation 10, is an integral estimator, here we make the straightforward connection between integral estimators and stochastic texture filtering for completeness.

With traditional STF, we are applying Monte Carlo integration to the integral texture filtering equation, 2:

$$t_{\mathrm{r}}(u,v) = \iint \left(\sum_i^n \delta(u'-u_i)\delta(v'-v_i)\,\mathbf{T}_{u_i,v_i}\right) f_{\mathrm{r}}(u-u',v-v')\,\mathrm{d}u'\,\mathrm{d}v' \tag{S1}$$

$$= \iint \sum_i^n \delta(u'-u_i)\delta(v'-v_i)\,\mathbf{T}_{u_i,v_i}\, f_{\mathrm{r}}(u_i-u',v_i-v')\,\mathrm{d}u'\,\mathrm{d}v'. \tag{S2}$$

We will define the PDF

$$p(u,v) = \sum_i^n \delta(u-u_i)\delta(v-v_i)\, f_{\mathrm{r}}(u_i-u,v_i-v). \tag{S3}$$

Authors' Contact Information: Bartlomiej Wronski, NVIDIA, Brooklyn, USA, bwronski@nvidia.com; Matt Pharr, NVIDIA, San Francisco, USA, matt@pharr.org; Tomas Akenine-Möller, NVIDIA, Lund, Sweden.

ACM 2577-6193/2025/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

Under the assumption that the texture reconstruction filter f_r is normalized, it is easy to see that this is a valid PDF.

Samples from the PDF can be taken by selecting a term i of the sum with probability proportional to $f_r(u_i - u, v_i - v)$. In turn, we have a sample point (u_i, v_i).

If we apply the importance sampling Monte Carlo estimator, we have

$$t_r(u, v) \approx \frac{\cancel{\delta(u - u_i)}\,\cancel{\delta(v - v_i)}\,\mathbf{T}_{u_i,v_i}\,\cancel{f_r(u_i - u, v_i - v)}}{\cancel{\delta(u - u_i)}\,\cancel{\delta(v - v_i)}\,\cancel{f_r(u_i - u, v_i - v)}} = \mathbf{T}_{u_i,v_i}, \tag{S4}$$

giving the one-tap STF estimator.

S-2 Variance and Bias with Filtering After Shading

For cases where filtering before shading is the desired result, it is useful to be able to characterize the error from STF and filtering after shading in order to evaluate and design STF estimators. This is challenging in general, as a wide variety of nonlinearities are present in shading functions. We therefore propose a simple approach based on statistical analysis of nonlinear transformations of random variables. For a more complete treatment of the statistical analysis of error resulting from nonlinearities applied to random variables, we refer the reader to the statistical and control theory literature and extended Kalman filters [S1, S4] as well as recent advances in unbiasing rendering algorithms using telescoping Taylor series [S2].

Consider a shading function f where the true filtered texture value is μ: with filtering before shading, we filter the texture using Equation 3 to compute μ and then return $f(\mu)$. With filtering after shading and one-tap STF [S3], the texture is represented by a random variable X corresponding to a single texel that is sampled according to the texture filter, $X \sim f_r$. The estimator is $f(X)$. We would like to understand how the expected value of filtering after shading, $\mathbb{E}[f(X)]$, relates to $f(\mu)$.

To approximate this error, we can use the Taylor expansion of f around μ. For example, consider the second-order expansion:[1]

$$\mathbb{E}[f(X)] = \mathbb{E}[f(\mu + (X - \mu))] \tag{S5}$$

$$\approx \mathbb{E}\left[f(\mu) + f'(\mu)(X - \mu) + \frac{1}{2}f''(\mu)(X - \mu)^2\right] \tag{S6}$$

$$= f(\mu) + f'(\mu)\,\mathbb{E}[X - \mu] + \frac{1}{2}f''(\mu)\,\mathbb{E}\left[(X - \mu)^2\right]. \tag{S7}$$

Because one-tap STF gives an unbiased estimate of μ, $\mathbb{E}[X - \mu] = 0$ and $\mathbb{E}\left[(X - \mu)^2\right]$ is X's variance, which we will denote by σ_X. Dropping $f(\mu)$, the result of filtering before shading, we are left with

$$\frac{f''(\mu)}{2}\sigma_X^2 \tag{S8}$$

as the error due to filtering after shading. In other words, the error depends on the second derivative of the shading function and the squared variance of X (and the higher-order terms we have neglected); we see how the variance of X directly contributes to the final error and the resulting bias.

This analysis fits with our earlier error analysis in Section 3: when the variation in the filtered texel values is small, STF yields a small error and filtering after shading gives results that are close

[1] In practice, many functions used in rendering, such as specular shading, are nonlinear and have slowly decaying higher-order derivatives, so a second order expansion is insufficient. However, we can apply this expansion to multi-variate functions and include higher-order derivatives and thus, higher-order statistical moments.

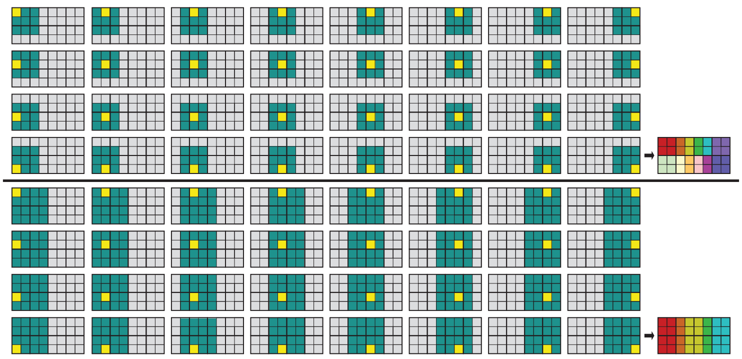

Fig. S1. A set of possible 3×3 footprints (top) and 4×4 footprints (bottom). As in Figure 6, for each footprint, yellow signifies the lane in the wave that the footprint is associated with and green signifies the other lanes that it draws texel values from. The colored illustrations on the right shows which groups of lanes all end up with the same texel values to filter after sharing.

to filtering before shading. With constant signals with zero variance, the error is zero and the two approaches are equivalent.

Unlike one-tap STF, our method uses an estimator that has a small bias (Section 4.2). Thus, $\mathbb{E}\left[X-\mu\right] \neq 0$ and $\mathbb{E}\left[(X-\mu)^2\right]$ includes both bias and variance. The error is approximated as:

$$f'(\mu)\,\mathbb{E}\left[X-\mu\right] + \frac{1}{2}f''(\mu)\,\mathbb{E}\left[(X-\mu)^2\right]. \tag{S9}$$

The error includes terms that depend on both the first and second derivatives of the shading function f, though in practice the overall error is lower than with one-tap STF since X is closer to μ with our approach.

From these results, we can see why the requirement of not introducing any variation in regions of constant texture values is so important—even a small amount of error may introduce a large error in the shaded result. Furthermore, estimators like standard importance sampling that may have unbounded weights (recall Section 4.1) result in not only much higher variance, but also high further statistical error moments.

While Taylor expansion formally does not apply to every function used in rendering (for example, a step comparison operator used in shadow mapping is not differentiable), this analysis still provides an insight and intuition for evaluating different estimators: that the better the variance reduction of an estimator (or, generalizing to higher-order moments, smaller amounts of noise and tighter distributions), the closer the result is to filtering before shading.

S-3 Illustration of Larger Footprints

Figure S1 illustrates the footprint for deterministic 3×3 and 4×4 square footprints in a 32-lane wave. Note that as the footprints become larger, more lanes in the wave all filter the same set of texel values; these texel sharing sets are illustrated in the lower right. However, since the filters are larger, we find that the error is also lower in general, as discussed in Section 5.2.

S-4 Wave Intrinsics in Other Shader Stages

While the specific wave mapping for other shading stages, such as ray tracing shaders, is undefined, we have successfully tested our method with general wave intrinsics in those stages. We have verified that our method works with shadow rays that sample alpha masks from textures and is

)

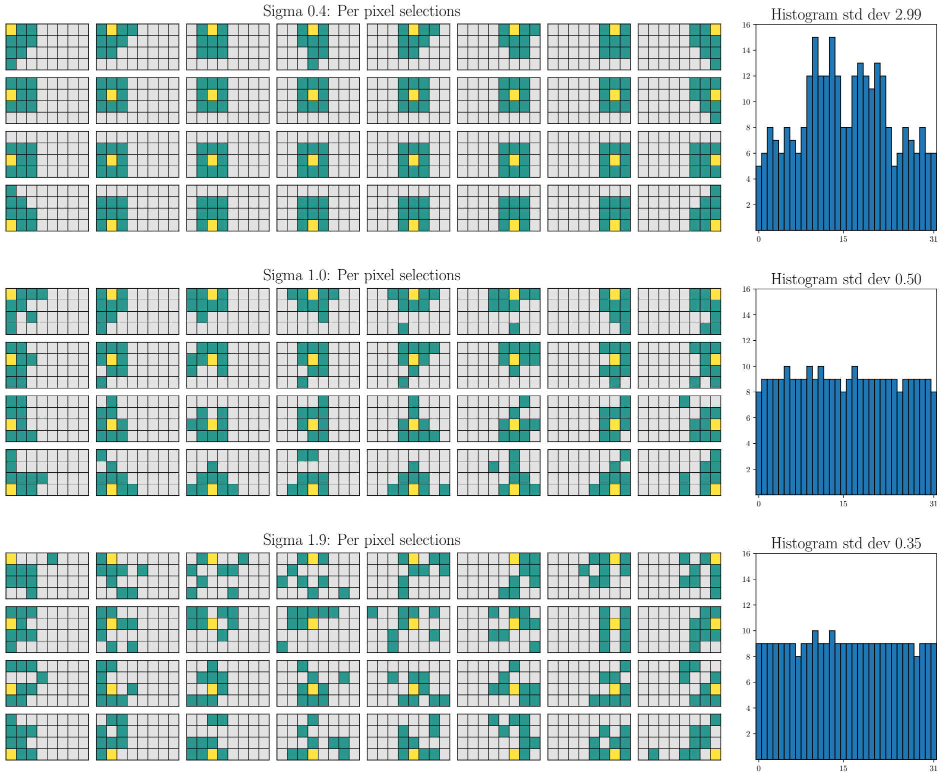

Fig. S2. Three sets of sample sharing footprints, with increasing values of the σ parameter. **Top**: with a small σ, most of the patterns are the same as the deterministic 3×3 footprint; as shown by the histogram on the right, some lanes (the ones in the center of the wave) are used for sharing much more than others (the ones along the edges and the corners.) **Middle**: increasing the σ gives more irregular patterns and a more uniform histogram. **Bottom**: a further increase of σ gives a histogram with slightly lower standard deviation but with irregular patterns that may sample far-away lanes. In general, the farther away the lanes used for sharing are, the less effective reuse will be, since shared texels may be outside of the current lane's texture filter footprint.

able to achieve some visual quality improvements over the original STF technique. However, we note that lanes can map to arbitrary and distant rays and thus the resulting filtering quality is not guaranteed. We also advise additional caution and checking whether other lanes are active, as behavior of `WaveReadLaneAt(value, laneId)` is undefined for inactive lanes.

S-5 Pseudorandom Sparse Footprint Generation Algorithm

To generate the sparse wave sharing footprints introduced in Section 4.3.2, we use a simple three-stage global optimization algorithm.

In the first stage, for each lane in a wave, we generate a set of 16–32 candidate configurations. Candidate elements to share are generated by sampling from a normal distribution with a fixed standard deviation σ. Close to the wave borders and corners, we relax the σ to allow considering farther-away candidates. We continue taking samples, discarding repeated lanes, until the number of candidates equals the desired wave sharing element count. Increasing values of σ reduce the footprints' locality but make it easier to have each lane be used for sharing the same number of times and have irregular shapes that do not lead to visible structure in images (Figure 7). Figure S2 shows results for three different σ values; the bottom part of Figure 8 was generated with a σ of 1.4.

Having generated candidates for each pixel, we proceed to the second stage of the algorithm. We randomly select a candidate for each pixel, count how many times each wave lane has been selected for sharing, and then find the standard deviation of these counts. We use the standard deviation as a score, where lower standard deviations are better, corresponding to greater uniformity in how often each lane is used. We repeat this process 10,000 times and select the configuration with the best score.

In the third stage, we proceed with a variation of the *coordinate descent* algorithm, where we attempt to improve the configuration selected after the second stage. We exhaustively iterate through all the lanes and check if using any of the other candidates from the first stage for the lane would improve the configuration's overall score. We continue this process until no better candidate is found for any of the lanes.

Finally, because this approach does not guarantee convergence to a global minimum, we repeat it from scratch 30 times and retain the pattern with the best score.

Although our optimization algorithm is brute-force, the search space of waves of 32–64 elements is relatively small and our algorithm still runs quickly. Our naive Python implementation running on a single CPU core can generate a complete set of sharing footprints in less than 40 seconds of CPU time. This has allowed for quick iterations and experiments as well as generating different patterns for different frames to break up temporal artifacts. For the results in the paper, we run ten times as many iterations of the first two stages, with a corresponding increase in pattern generation time by a factor of ~10. We find this time to be acceptable for a preprocess; if higher performance was necessary, the iterations of the first two stages could run in parallel and a higher-performance language like C++ could be used for the implementation. We include our implementation in the supplementary material.

S-6 Filter PMF Implementation

Each time through the loop in the code listing in Section 4.5, we consider a texel sampled by one of the lanes in the sharing footprint for the current lane. To evaluate its contribution to the weighted importance sampling estimator, Equation 10, we need to compute the probability that the current lane would have sampled that texel; this is handled by the call to `GetFilterPMF`.

Below is an example implementation of this function for a bilinear filter. Because the filter is normalized, the PMF for a given texel is simply its bilinear filter weight, which is easily computed in a few lines of code. We note that depending on the rendering technique used, this function might need to aditionally verify if other lanes sample from the same texture set and return a zero PMF on any mismatch.

```
float GetFilterPMF(float2 texelFloatCoords, int2 texelIntCoords)
{
    float2 texelDistance = texelFloatCoords - texelIntCoords;
    float2 filterPdf = clamp(1 - abs(texelDistance), 0, 1);
    return filterPdf.x * filterPdf.y;
}
```

S-7 Tricubic Reconstruction for Volumetric Rendering

We have also investigated the effect of sample sharing for STF with ray marched volumetric rendering; our results are summarized in Figure S3. As shown in the plot, texel sharing significantly reduces the numeric error compared to one-tap STF for a given number of texel lookups. We also note that one-tap STF has significantly lower error than full tricubic filtering given an equal number

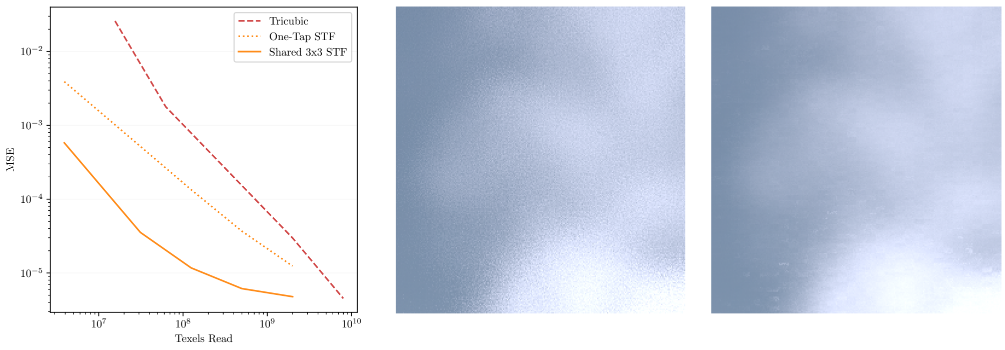

Fig. S3. Sample reuse for volumetric ray marching of a cloud data set. **Left**: Plot comparing mean squared error for full tricubic filtering, one-tap STF, and STF with 3×3 sparse sharing footprints. Sharing significantly reduces MSE, roughly in proportion to the number of successfully shared texel lookups. **Middle**: with 64 ray marching steps and one-tap STF, error manifests as high-frequency noise. **Right**: with 64 steps with texel sharing, numeric error is much lower and noise is reduced, but block artifacts appear.

of samples; we attribute this to the benefit of importance sampling—full tricubic filtering accesses all the 64 texels under the filter, many of which make a relatively small contribution to the final result.

However, as shown by the images, texel sharing leads to block-structured artifacts in the image with sharing and 64 ray marching steps. We believe that the high-frequency noise from one-tap STF is likely to be preferable in this case as it would be easier to remove with post-rendering filtering like TAA or DLSS. (These artifacts do disappear at higher sampling rates, i.e., more steps along each ray.) We attribute these artifacts to the extent of the tricubic filter: with 3×3 sharing in a wave, even if all neighboring pixels provide unique useful texels, less than 15% of the texels under the filter will be available. Further, nearby pixels will generally filter similar incomplete sets of texels, leading to correlation between pixels in each wave.

References

[S1] Brian D. O. Anderson and John B. Moore. 1979. *Optimal Filtering.* Prentice-Hall.

[S2] Zackary Misso, Benedikt Bitterli, Iliyan Georgiev, and Wojciech Jarosz. 2022. Unbiased and Consistent Rendering using Biased Estimators. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 41, 4 (July 2022). https://doi.org/10/gqjn66

[S3] Matt Pharr, Bartlomiej Wronski, Marco Salvi, and Marcos Fajardo. 2024. Filtering After Shading with Stochastic Texture Filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7, 1 (2024), 14:1–20.

[S4] Gerald L Smith, Stanley F Schmidt, and Leonard A McGee. 1962. *Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board a Circumlunar Vehicle.* Vol. 135. National Aeronautics and Space Administration.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009