Karlsruhe Institute of Technology

Institute for Visualization and Data Analysis (IVD)

Percentile-based Adaptive Spatiotemporal Variance-Guided Filtering for GI and Volumes

Lucas Domingo Alber

We present an improved implementation of SVGF that aims to achieve good denoising quality for combined high- and low-frequency content—for example, combined direct and indirect illumination—in highly dynamic scenes. Unlike A-SVGF, our approach avoids the added complexity of gradient computation, coherent sample spaces, and forward projection while maintaining robustness. Central to our method is a percentile-based outlier detection scheme that adaptively modulates the accumulation factor and also suppresses firefly artifacts in the input signal. Additionally, we present several modifications that significantly reduce artifacts, particularly in scenarios involving disocclusions and rapid camera motion and our approach for reprojecting volumes for denoising.

Percentile-based Adaptive Accumulation

To quickly react to lighting changes and reduce ghosting, the accumulation factor should adaptively be reduced in areas where the temporal information is outdated, i.e., the luminance in the temporal buffer differs a lot from the current input image. For that, we compute user-defined luminance percentiles in 8x8 pixel-blocks. This can be implemented efficiently using an odd-even sort in shared memory. We then compute a target accumulation factor using the formula

```
α = (1 - s · lin(up, up + IPR, lum))
  · (1 - s · (1 - lin(low - IPR, low, lum))),
```

where `lin, s, low, up, lum` are the linear step function, a user-defined strength, the lower and upper percentiles of the current block, and the luminance from the accumulation buffer. The inter-percentile range `IPR` is computed as the difference of the lower and upper percentiles and is multiplied with a user-defined factor. The resulting accumulation factor α is not used directly; instead we reduce the history length variable according to `history = min(1 / (1 - α) - 1, history)`. This increases the accumulation of new information in the following frames as well.

Percentile-based Firefly Filtering

To reduce flickering, we use a similar percentile-based approach to filter fireflies in the input image. According to

```
max_lum = bias + up + IPR
```

we compute a maximum luminance, which is used to clamp the samples in the input. We add a small bias to help in situations with sparse sampling.

Variance Estimation

The original implementation employs a 3×3 pixel Gaussian blur to improve variance estimation. However, we observed that this introduces flickering and still fails to eliminate black artifacts caused by zero-variance estimates. To address this, we estimate variance over a larger 5×5 pixel neighborhood and adaptively reduce the spatial influence based on the temporal history length.

Reducing Artifacts

We found that artifacts caused by missing information after disocclusions were particularly distracting during rapid gameplay. To mitigate this, we reuse information from nearby pixels, which may introduce minor artifacts but are significantly less perceptually disruptive. For motion vectors pointing outside the image plane, we reuse the information at the border by intersecting the motion vector with the plane, resulting in more pleasing artifacts compared to clamping. If the reprojected pixel is discarded due to normal or depth differences, we extend the search radius to nearby pixels. To increase reuse in the filtering step, we modified the edge-avoiding functions to decrease their influence depending on the depth value of the current center pixel.

Volume Denoising

To improve the reprojection of volumetric effects—such as light shafts—and better account for parallax, we replace surface-based motion vectors with those derived from virtual points of high contribution within the volume, which are forward-projected into the current frame to compute motion vectors. While the points can be selected stochastically using a weighted mean based on transmittance sampling, we instead leverage data from our real-time distance guiding. Beyond this modification, we retain the base SVGF implementation but disable normal-based rejection and reduce the influence of depth values.

Performance and Memory Usage

Performance was measured on an AMD Radeon RX 7900 XTX at 1080p resolution. We run two instances of SVGF: one for surfaces and one for volumes. Each instance runs in around 1.05 ms in total for 5 filter iterations and requires around 68 MB of temporary memory. In addition, the implementation needs access to the results and the G-Buffer from the previous frame, which amounts to around 117 MB.

Code:
github.com/LDAP/merian