Beyond Spatiotemporal Variance-Guided Filtering: Temporally Stable Filtering of Path-Traced Reflections in Real-Time*

Victor Voorhuis
Utrecht University
Utrecht, The Netherlands
victorvoorhuis@gmail.com

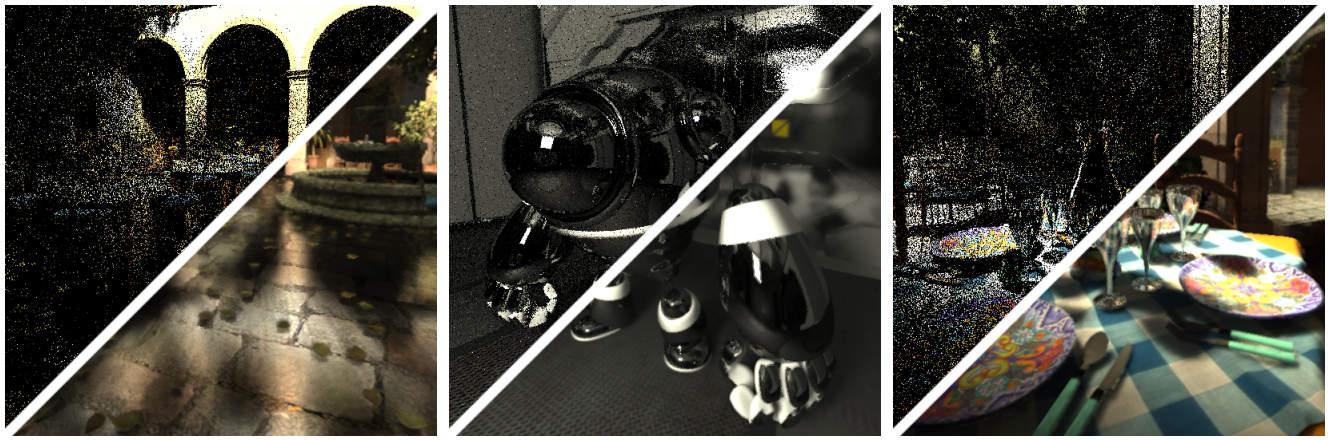


Figure 1: Various scenes, shown before and after filtering with our extended and modified version of SVGF by Schied et al. [21]. Our extended SVGF technique is able to reconstruct temporally stable path-traced reflections in real-time using a one sample per pixel render and a history buffer, which SVGF is unable to.

ABSTRACT

The path tracing rendering algorithm has long been considered to be unsuitable for real-time rendering, since a large amount of samples is required to produce noise-free renders. Many filtering methods have been proposed, which denoise renders by trading variance for bias. The recently introduced Spatiotemporal Variance-Guided Filter (SVGF) [21] achieves real-time denoising of path-traced renders, requiring only one sample per pixel.

SVGF employs a reprojection step to increase the amount of samples in the filter input. This leads to increased temporal stability. Reprojection is also employed to estimate the per-pixel variance, which is used to locally adapt the filter bandwidth to the signal. SVGF however only reprojects the primary hit, and is therefore not able to reproject geometry visible in reflections. We extend SVGF to reproject and filter geometry visible in both pure and glossy specular reflections. To prevent this reprojection from introducing ghosting artifacts, we apply a form of neighborhood clipping which is tailored to SVGF. With our modifications, SVGF can produce temporally stable filtered reflections in real-time.

We also extend SVGF to support supersampling and introduce several modifications to improve the robustness of the algorithm when the probability of paths that return energy is low. Our work makes SVGF usable in a wider variety of scenes and improves reconstruction quality in several scenarios, while retaining real-time performance on consumer hardware.

*Master thesis, Utrecht University, ICA-4146034

1 INTRODUCTION

The path tracing rendering algorithm [10] offers numerous advantages over traditional rasterization based approaches. By evaluating multidimensional integrals using Monte Carlo integration, physically accurate light transport can be modeled and evaluated. Numerous visual effects, such as indirect diffuse illumination, fit elegantly into the path tracing framework. The advantages of path tracing have led to its wide adoption in the movie industry [12]. For an introduction to path tracing, we refer to Appendix B.

The Monte Carlo integration which lies at the core of the path tracing technique unfortunately also introduces the technique's main drawback: the pixel estimates contain variance, which shows up in the image as noise. This variance decreases as the number of collected samples increases, but reducing the visible noise to an acceptable level can take a prohibitively large amount of time. This has led to the development of a wide variety of filtering approaches [28], which introduce some bias to reduce visible noise. Even with these filters, path tracing has long been considered to be too costly for real-time rendering applications.

Recently, several filtering approaches have been proposed [2, 4, 6, 14, 21] that aim to make the path tracing algorithm applicable to real-time rendering. One of the current state-of-the-art approaches is the Spatiotemporal Variance-Guided Filter (SVGF) [21], which is able to reconstruct temporally stable sequences of path-traced renders in real-time. SVGF makes use of reprojection, re-using samples taken in the past where possible for increased temporal stability and an increased effective sample count. Filtered frames

are produced with only one sample per pixel and a history buffer. However, the reprojection technique used by SVGF is not aware of reflected geometry, which results in ghosting reflections.

In this work, we introduce several modifications and extensions to the SVGF approach, making the technique both more effective and applicable to a wider variety of materials. Specifically, we extend SVGF to reproject pure and glossy specular reflections accurately, which allows for temporally stable filtering of reflections without ghosting artifacts. Our contributions include:

- We introduce a fast motion vector estimation technique for geometry visible in pure specular reflections, based on diamond search [27]. This allows us to accurately reproject and filter geometry visible in pure specular reflections.
- We develop a heuristic that allows us to apply our motion vector estimation technique to glossy specular reflections, enabling reprojection and filtering of geometry visible in glossy specular reflections.
- We extend SVGF with a neighborhood clipping step that is specifically tailored to SVGF. This helps to reduce ghosting artifacts for reprojected reflections.
- We extend SVGF to support supersampling using a modified guide rays approach. This supersampling allows for sharp anti-aliasing and increased filtering robustness.
- We apply several modifications to the SVGF algorithm, improving its robustness in situations where the energy in an area is concentrated in a few bright samples.

2 RELATED WORK

The visually disturbing noise in images rendered by path tracing arises due to variance present in the per-pixel color estimates. As we gather more samples, the variance of our estimates decreases, eventually leading to the noise disappearing. To speed up the rendering process, many denoising filters for path tracing have been proposed. The idea of these filters is to combine pixel estimators that should converge to similar values, trading variance for bias. We will briefly discuss some of these filtering approaches. For a more in-depth discussion, we refer to the survey by Zwicker et al. [28] and our own in-depth literature study in Appendix B.

Denoising Monte Carlo Renders To denoise Monte Carlo renders effectively, smoothing has to be performed while retaining high-frequency details such as edges. There are several ways to achieve this. One can for example employ a bank of filters with varying kernel sizes. Rousselle et al. [19] greedily choose from a bank of Gaussian filters, choosing the filter likely to have the smallest error for each pixel. Another approach is to use non-linear filters, such as the bilateral filter [24]. In the bilateral filter, weights between pixels are not only based on spatial proximity, but also on similarity in pixel color intensity. This makes the filter preserve edges, but also means that high-frequency noise in the input is preserved. Xu and Pattanaik [26] use the bilateral filter for Monte Carlo denoising, dealing with the preservation of high-frequency noise by prefiltering using a Gaussian filter.

Many recent approaches use auxiliary features of the pixels in the filtering process. Li et al. [13] use Stein's Unbiased Risk Estimator (SURE) to select from a bank of joint-bilateral filters with varying spatial support. The joint-bilateral filter also factors the similarity between feature values of pixels, such as the normal of the first hit, into the weight calculation. Rousselle et al. [20] also use SURE to select a filter from a filterbank, but instead use NL-means filters, which have a higher computational complexity than bilateral filters but are more robust against noise. The regression-based approaches [2, 3, 15, 16, 17] use auxiliary features in a radically different way: smoothing is performed by locally fitting fixed order regression models, which map from the auxiliary features to the noisy image as closely as possible. The final filtered image is formed using these fitted regression models.

Auxiliary features have proven to be effective in steering denoising filters. Unfortunately, several phenomena, such as depth of field and motion blur, introduce noise in these features. There are several ways of dealing with this. Moon et al. [15] perform a Truncated Singular Value Decomposition (TSVD) on their feature space, allowing them to operate on a smaller set of less noisy features. Sen and Darabi [22] measure how reliant the features are on the random parameters, diminishing the influence of more noisy features. Delbracio et al. [7] forego the usage of auxiliary features altogether, instead using color histograms to mix pixel color estimates that have similar color distributions together.

Denoising Monte Carlo Renders in Real-Time With the increasing capabilities of graphics hardware, path tracing in real-time has become a reality, albeit with low sample counts. This has led to an increased interest in denoising methods that execute in little time, while being able to work with low sample counts. Combined with a fast path tracer, these filters aim to make path tracing applicable to real-time rendering.

To achieve sufficient denoising at low sample counts, bilateral filters with large filter footprints would be required. The computational complexity of these filters is high. Thus, different approaches are required. Bauszat et al. [2] employ the guided image filter, which smooths by locally fitting first-order regression models. Later, Bauszat et al. [1] succesfully filtered path traced renders with depth of field by combining adaptive manifolds [8] and sweep blur [23].

Dammertz et al. achieve real-time filtering by employing the fast undecimated À-Trous wavelet transform to approximate a joint-bilateral filter with a large kernel. This comes down to repeatedly filtering the image with larger and larger kernels, which all contain the same amount of non-zero entries. A variety of edge-stopping functions, which use auxiliary feature values, is used in the weight calculation of these filtering steps to retain high-frequency content. The filter runs in real-time, but the method is not temporally stable, which results in distracting flickering. Feature buffers are assumed to be free of noise, making the technique incompatible with stochastic primary ray effects.

Recently, Chaitanya et al. [4] employed a recurrent denoising autoencoder to denoise Monte Carlo renders. The recurrent denoising autoencoder is a convolutional neural network that contains recurrent connections. These recurrent connections connect to the state in the previous frame, allowing a trained network to use information from the previous frame to create temporally coherent sequences of images. The approach produces high quality results without requiring guidance from the user, but unlike SVGF [21], is currently unable to run at real-time framerates on consumer hardware.

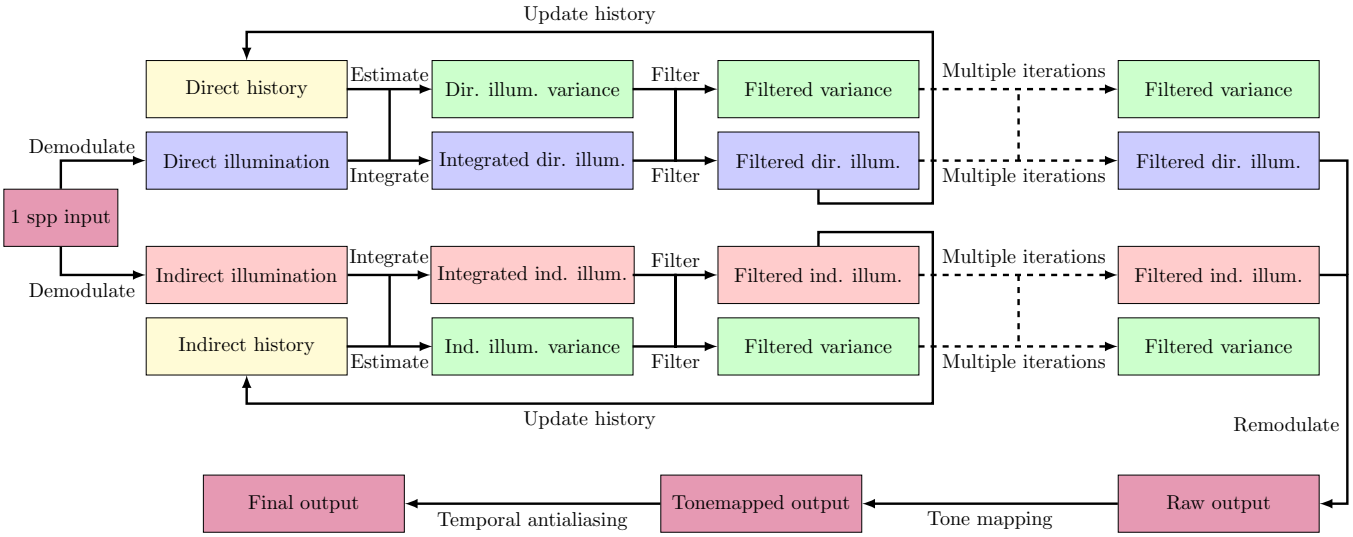


Figure 2: A diagram describing the working of the SVGF approach [21]. A one sample per pixel (spp) input is demodulated, split into direct and indirect illumination, temporally integrated using a history buffer and filtered using several À-Trous filter iterations. A variance estimate is also filtered in each iteration and used to control the illumination edge stopping.

3 SPATIOTEMPORAL VARIANCE-GUIDED FILTERING (SVGF)

Our work is an extension of the Spatiotemporal Variance-Guided Filter (SVGF), which was introduced by Schied et al. [21]. In this section, we will give a brief description of the filter. An overview is provided in Figure 2.

SVGF performs smoothing using the same À-Trous wavelet filter as was used by Dammertz et al. [6], although with alternative, scene-agnostic edge stopping functions. Indirect and direct illumination are filtered separately to increase reconstruction quality. To prevent the filter from having to retain high-frequency texture details, the authors demodulate the direct and indirect illumination with the surface albedo before filtering, and remodulate the illumination components after filtering. By filtering the separated, demodulated illumination, texture detail is preserved.

Schied et al. extend the work by Dammertz et al. by introducing a reprojection step. The technique uses screen-space motion vectors, derived from a rasterization pass for primary visibility, which describe what pixels in the previous frame correspond to the pixels in the current frame. Using these motion vectors, pixels from the previous frame are mixed with the unfiltered samples of the current frame using an exponential moving average. To prevent incorrect reprojections, there are several consistency tests based on the auxiliary features. The temporal reprojection leads to less noisy and more temporally stable sequences of filtered images.

Temporal reprojection is also used in SVGF to estimate the variance of the different pixel illumination values. When there are less than 4 successful reprojections, a spatial variance estimate is used instead. The bandwidth of the luminance edge stopping function is scaled using the variance estimate, meaning that noisy pixels are allowed to blend more with dissimilar pixels than less noisy pixels. After the temporal reprojection and the filtering passes, the filtered direct illumination and the filtered indirect illumination are combined and remodulated using the surface albedo. This output is tone mapped, and a temporal antialiasing [11] step is performed to reduce aliasing artifacts and further increase temporal stability.

The introduced temporal reprojection step reduces variance in the input to the À-Trous filter and improves temporal coherence, but can also increase bias. The temporal reprojection assumes that the illumination at a world position in the current frame is similar to the illumination of the corresponding world position in the previous frame. Unfortunately, when this assumption is false (for example when moving the camera around a specular surface), samples from the previous frame are mixed in incorrectly, introducing bias visible as ghosting artifacts.

Similar to the filter by Dammertz et al., SVGF assumes noise-free feature buffers and is thus incompatible with stochastic primary ray effects.

4 REPROJECTION AND FILTERING OF REFLECTIONS

SVGF reprojects pixels using a buffer of 2D screen space motion vectors, which are derived from a rasterization pass for primary visibility. These motion vectors allow for finding a pixel in the previous frame with a similar primary hit. Unfortunately, that does not mean that the same reflected geometry is visible in that pixel: different reflected geometry is visible at the same primary hit, depending on the viewpoint. Reprojecting using the primary motion vectors leads to ghosting reflected geometry.

For correctly filtered reflections, we require motion vectors describing the screen space motion of reflected geometry. These are much harder to determine, especially considering the fact that reflective surfaces can have arbitrary shapes. In this section, we

introduce a method to determine such "reflection motion vectors" in real-time, for both pure and glossy reflections.

4.1 Pure specular reflections

In order to filter pure specular reflections without blurring high-frequency details, we use auxiliary features of the first non-pure specular hit, instead of auxiliary features of the first hit. We also demodulate with the surface albedo of the primary hit multiplied with the surface albedo of the reflected geometry.

To prevent ghosting reflections, we reproject using motion vectors which describe the motion of the reflected geometry. To derive this motion vector for a specular pixel p, we search for the specular pixel q in the previous frame that has the closest world position feature to p. A naive way to do this is to examine all specular pixels in the previous frame, or to examine a large window of pixels in the previous frame. The computational cost of doing this for every specular pixel p is unfortunately prohibitively high.

Efficiently estimating motion vectors has been studied extensively for the purpose of video compression, where temporal redundancy between frames is exploited by estimating motion vectors for blocks of pixels. A popular algorithm for fast block-matching motion estimation is the diamond search algorithm [27], which finds matching blocks in different frames efficiently using a sequence of search iterations.

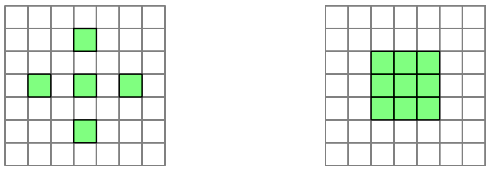
In each iteration, a pattern of taps centered on the current best match is evaluated. In the first phase, a Large Diamond Search Pattern (LDSP) is used. After each iteration, the search pattern is centered on the best found tap. The search continues until an iteration finds no improvements. The algorithm then moves on to the second phase, in which a Small Diamond Search Pattern (SDSP) is used for one final search step. The best found match determines the final motion vector. By starting with the LDSP, which has taps spread apart further than the SDSP, the algorithm can cover large distances quickly and is relatively resistant to getting stuck in local minima.

We employ a modified version of the diamond search algorithm to find a specular pixel q in the previous frame with a matching world position feature to p. During each search step, we choose the pixel q with the smallest distance to p:

$$\text{dist}(p, q) = ||\text{world}_p - \text{world}_q||. \tag{1}$$

Before starting the diamond search, we evaluate two starting positions and choose the best one. We evaluate the position of p and the matching position of p in the previous frame according to the primary hit motion vector, which in many scenarios is close to the best match. For increased efficiency, we use an LDSP with 5 taps instead of 9, as was also done by Cheng et al. [5] (see Figure 3). We modify the algorithm to start with a scaled up LDSP, and reduce the step size when no improved taps are found. The initial larger steps make the algorithm less prone to getting stuck in local minima and can speed up searches for large motion vectors. For a visualization of the working of the algorithm, see Figure 4. Pseudocode of the algorithm is given in Appendix A.

The described search algorithm produces motion vectors which lack subpixel accuracy. This can lead to unstable reprojection over multiple frames, visible as shaking reflections. To approximate the subpixel detail of the motion vectors, we score the 8 pixels around



(a) LDSP **(b) SDSP**

Figure 3: The search patterns used by our motion vector diamond search. The LDSP is the same as was used by Cheng et al. [5], but we scale up the distance of the taps from the center tap during early search steps.

the best matching pixel in the previous frame. We choose the block W of 2×2 pixels with the best combined score, and then obtain a subpixel position between the 4 pixels by weighting the coordinates of the 4 pixels using their scores. The weight of each pixel is the product of the distances of the other pixels, so that pixels with a smaller distance get a relatively higher weight:

$$w_p = \prod_{q \in (W/p)} \text{dist}(p, q), \tag{2}$$

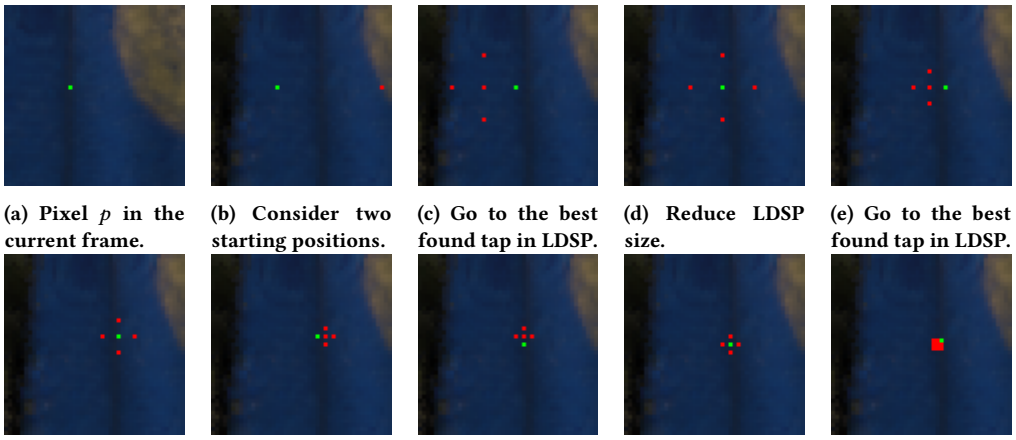
$$\text{coor}_{\text{match}} = \frac{\sum_{q \in W} (w_q * \text{coor}_q)}{\sum_{q \in W} w_q}. \tag{3}$$

4.2 Glossy specular reflections

We will now extend the reprojection approach proposed in the previous subsection to glossy specular reflections. We will discuss motion vector estimation, an adjusted À-Trous filter and a neighborhood clipping procedure.

4.2.1 Motion vector estimation for glossy speculars For glossy speculars with a roughness below a threshold, we derive motion vectors describing motion of reflected geometry. For materials that are more diffuse, we instead use primary hit motion vectors, since geometry becomes hard to recognize in the reflections. In our renderer, which uses a Beckmann normal distribution, we use roughness threshold 0.4. Deriving motion vectors for geometry visible in the blurry reflections is challenging, because pixels no longer have a single deterministic secondary hit. Instead, there is a *distribution of secondary hits*, which together determine the pixel color. For effective reprojection, a pixel needs to temporally accumulate with a pixel in the previous frame that has a similar distribution of secondary hit points, instead of with a pixel that happened to sample a similar hit point. However, gathering enough samples to characterize this distribution is expensive. Instead, we propose a cheap but effective approach based on a heuristic, that yields good results at low sample counts.

We perform the same modified diamond search algorithm as in the previous subsection, but use a different distance function. The distance between secondary hits no longer works well for glossy speculars: two pixels may have similar secondary hit distributions, but may have sampled two far apart positions. Instead, we base the distance function on the distance between the *secondary hit points with the highest probability*. Our heuristic makes the assumption that, if the secondary hits with the highest probability of two glossy



(a) Pixel p in the current frame. (b) Consider two starting positions. (c) Go to the best found tap in LDSP. (d) Reduce LDSP size. (e) Go to the best found tap in LDSP.

(f) Reduce LDSP size. (g) Go to the best found tap in LDSP. (h) Go to the best found tap in LDSP. (i) Switch to SDSP. (j) Best match in SDSP is match q.

Figure 4: A visualization of our motion vector search approach, based on diamond search, running for a pixel p. In (a), the green pixel is the current pixel p in the current frame. In a set of search iterations we search for a matching pixel q in the previous frame. In these images, the considered taps are marked as red, with the best considered tap marked as green.

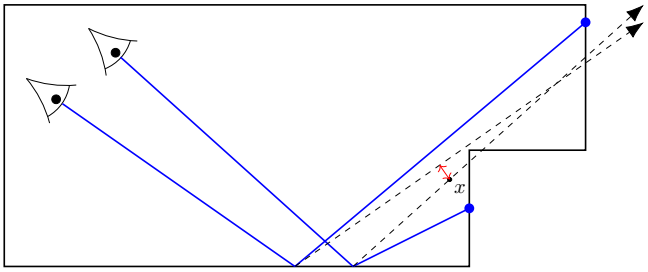


Figure 5: Visualization of our glossy reprojection heuristic. For paths with a primary hit on a glossy specular (shown in blue), we record a "representative ray" in the direction the reflection would have been in if the material was purely specular (see the dashed arrows). The distance function is calculated by taking the path with the smallest secondary depth, and taking the representative ray at that depth (point x). The shortest distance from that point to the other representative ray is the distance value (shown as the red arrow).

pixels are close together, the distributions of secondary hits of the two pixels are likely also similar.

To determine the secondary hits with the highest probability, we store for each pixel with a glossy specular primary hit the direction in which the ray would have been reflected if the material was purely specular. We call this direction, together with the primary hit as origin, the "representative ray" for the distribution of possible outgoing reflection rays. We assume that a BRDF is used in which the representative ray is the outgoing ray with the highest probability. The most likely secondary hit of a pixel is the representative ray's first intersection with the scene. To avoid an expensive raycast, we approximate the hit of the representative ray by extending it from its origin with the depth of the taken sample:

$$\text{approx hit}(r_p) = r_p.o + r_p.d * \text{depth}_p. \tag{4}$$

Using the distance between approximate representative ray hits of two pixels to steer the motion vector search algorithm already yields more stable reprojection and less noisy results than using the noisy world positions of the samples, but there is still an important issue. Consider a pixel p, whose samples partially hit an object in the foreground and partially hit the background. Pixel q in the previous frame has a similar distributions of secondary hits: p and q are suitable for temporal accumulation. However, if p's sample hits an object in the foreground (depth_p is small) and q's sample hits an objects in the background (depth_q is large), the distance between approx hit(r_p) and approx hit(r_q) will still be large, leading to no temporal accumulation between p and q.

To make sure p and q would temporally accumulate in this scenario, we do not directly use the distance between approximate representative ray hits. Instead, we use *the shortest distance from the closest approximate representative ray hit to the other representative ray*. Now, p and q will be considered as good candidates for temporally accumulation, because approx hit(r_p) is close to r_q (see Figure 5). We measure the distance from approx hit(r_{near}) to r_{far}, because we favor reprojecting near geometry over reprojecting far geometry: an incorrect foreground pixel color component tends to be more noticeable than an incorrect background pixel color component. This gives our final distance function:

$$\text{dist}(p, q) = ||r_{\text{far}}.\text{d} \times (\text{approx hit}(r_{\text{near}}) - r_{\text{far}}.\text{o})||. \tag{5}$$

4.2.2 Applying the À-Trous filter to glossy speculars Using the noisy features of geometry visible in the reflections to edge-stop in the À-Trous filter passes would lead to noisy results. Instead, we use the features of the glossy specular surface itself. There is no need for precise edge stopping on the blurry reflected geometry: previously, two nearby pixels could have completely distinct secondary hits

(a) Without ray differentials

(b) With ray differentials

Figure 6: A glossy specular orb from a long distance. When there is a large amount of geometry visible in a small amount of pixels on a glossy specular, overblur is prevalent. By scaling the spatial bandwidth using ray differentials, this overblur is reduced.

and completely different values. Combining the estimators would heavily bias the result. But now, two nearby pixels will likely have similar distributions of secondary hits, meaning that combining adjacent pixels is unlikely to lead to a large bias.

There is one caveat, however. If a large amount of geometry is visible in a small amount of pixels, it is still possible that two nearby pixels will have dissimilar distributions of secondary hits (if we look at a blurry reflection from far away, it will appear sharp again). The lack of edge stopping on features of reflected geometry and a fixed spatial bandwidth can now lead to overblur. To reduce this overblur, we calculate the world position ray differentials $\frac{\partial P}{\partial x}$ and $\frac{\partial P}{\partial y}$ at the secondary hit after a pure specular reflection (see [9]). We then use these to dynamically calculate a spatial bandwidth for a Gaussian kernel, which is then used in the À-Trous filter passes:

$$\sigma_{\text{spatial}} = \exp\left(\frac{-\left(\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y}\right)/2}{\omega}\right), \tag{6}$$

where ω is a scaling factor. We found a value of $\omega = 0.5$ to work well in our scenes. See Figure 6 for an example. We apply this to glossy specular surfaces with a roughness value under our roughness threshold, because on those surfaces this overblur can occur.

4.2.3 Reducing ghosting using neighborhood clipping In the SVGF technique, temporal accumulation only occurs if the features of samples are consistent, preventing ghosting. This approach does not work well for geometry visible in glossy reflections: a pixel might now correctly reproject to pixels with significantly different feature values. We disable the consistency checks for glossy reflections, but this can lead to ghosting artifacts. Besides this, our glossy reflections in which geometry is hard to recognize (the roughness is above our threshold of 0.4) is reprojected using primary hit motion vectors, which can also lead to ghosting.

In temporal antialiasing (TAA) [11], reprojection is also done, but to prevent ghosting, *neighborhood clipping* is performed instead of consistency checks with auxiliary features. Neighborhood clipping clips the history to the range of unreprojected colors in a small window (neighborhood) around the current pixel. This reduces ghosting drastically, because if a reprojected color does not occur in the local unreprojected neighborhood, it gets clipped away.

Neighborhood clipping cannot be applied to our SVGF reprojection as is. The current frame contains noise from the stochastic sampling, which means that the neighborhoods of nearby pixels can differ dramatically, leading to noisy results. To increase coherence of neighborhoods of nearby pixels, we apply a two-pass 7×7 Gaussian filter to the current unfiltered frame and take the neighborhood colors from the filtered image. To further reduce possible artifacts, we use a somewhat large neighborhood of 5×5 pixels. This neighborhood clipping step is applied to all pixels that have a primary hit with a roughness that we consider to not be perfectly diffuse (in our case, a roughness lower than 0.9).

5 SUPERSAMPLING FOR SVGF

Supersampling involves sampling each pixel with multiple samples for anti-aliased results and better sampling of subpixel details. This introduces a challenging issue: there are now multiple auxiliary feature values per pixel. To filter correctly, filtering would have to occur on a sample-level instead of on a pixel-level. This would increase memory usage and computational costs significantly. Averaging the features of the samples together to filter on a pixel-level is not viable, since it could lead to nonsensical feature values. Schied et al. [21] consider it to be unlikely that taking multiple samples per pixel will be feasible at real-time framerates for the foreseeable future. Instead of applying supersampling, they include a temporal anti-aliasing (TAA) step in SVGF for anti-aliased results.

We however find that taking multiple samples per pixel is possible for simple scenes. In these cases, supersampling might be preferable over using TAA, which tends to blur details under motion. Supersampling could also help to reconstruct areas where the probability of paths that return energy is low, which SVGF struggles with when there is no history available. Because of these reasons, we extend SVGF with support for supersampling.

Bauszat et al. [2] proposed an approach with "guide rays" to perform supersampling while filtering on the pixel-level. This involves always sending the ray of the first sample through the center of the pixel and filtering only the samples of these rays, known as guide rays. After filtering, one determines for each additional, non-guide ray sample to which of the four neighboring guide ray samples it is the most similar. The similarity is calculated using the depth and normal features. To produce the final render, the filtered color of each non-guide ray sample is set to the filtered color of the most similar guide ray (see Figure 7).

The guide rays approach produces anti-aliased results, but discards the energy values of non-guide ray samples. We modify the approach, making use of these extra samples to improve the quality of the filter input. Instead of determining the most similar guide ray sample of each non-guide ray sample after filtering, we already do this *before* the filtering. We then mix the unfiltered sample energy of each non-guide ray sample into the unfiltered sample energy of the most similar guide ray:

$$\hat{u}_i(p) = \frac{u_i(p) + \sum_{q \in A(p)} u_i(q)}{1 + |A(p)|}, \tag{7}$$

where $u_i(p)$ is the i'th color channel of a guide ray sample p and $A(p)$ is the set of non-guide ray samples whose most similar guide ray is p. To obtain anti-aliased results, we still set the colors of the non-guide ray samples to the sample of the most similar guide ray after filtering.

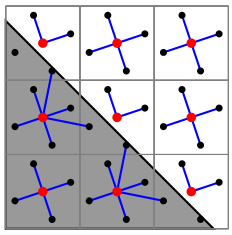


Figure 7: Visualization of the guide rays approach with 5 samples per pixel. The non-guide ray samples (black) of each pixel find the guide ray sample (red) with the most similar attributes (see the blue line). The final color of a pixel is formed by averaging its samples, where non-guide ray samples are set to the filtered color of their most similar guide ray. Two non-guide rays without a blue line are shown: these map to guide rays outside the image.

Mixing in non-guide ray samples before filtering leads to reduced variance in the filter input, but can also introduce additional bias, since non-guide ray samples will be filtered using the feature values of their most similar guide ray. However, the features of the most similar guide ray sample tend to be similar to the features of the non-guide ray sample, limiting introduced bias.

6 IMPROVING THE ROBUSTNESS OF SVGF

Areas where paths that return energy occur with a low probability are challenging to filter. At 1 sample per pixel, energy tends to be focused in a few bright samples, which have to be spread over a large area to produce good looking results. Schied et al. found that the variance in such areas is underestimated at low sample counts. For example, after a disocclusion, a pixel in such an area might be surrounded by exclusively black pixels, leading to a spatial variance estimate of 0. Insufficient smoothing occurs, resulting in black patches and bright dots. To counteract this, Schied et al. filtered their variance estimates with a small spatial filter. We observe that, even with the spatial filter, dimly lit areas are often still smoothed insufficiently after a disocclusion. To handle these areas more robustly, we introduce a number of modifications.

More reliable spatial variance estimate In SVGF, variance is estimated using a spatial window for the first three frames after a disocclusion. Instead of estimating the variance from the current luminance values in the spatial window, we estimate the variance using the *current temporally accumulated raw luminance moments* μ_1 *and* μ_2 of the pixels in the spatial window. The spatial variance estimate is now based on more samples when some history is available, improving its reliability. To ensure that μ_1 and μ_2 are accurate, we accumulate them using actual averages instead of exponential moving averages when there are 4 or less successful reprojections.

Increased luminance bandwidth at low sample counts The underestimation of variance is causing insufficient smoothing, because the estimated variance influences the strength of the luminance edge stopping. To counteract this, we scale the luminance bandwidth using the pixel's amount of successful reprojections h:

$$\sigma_l' = \sigma_l * (8 - \min(7, h)). \tag{8}$$

When a pixel has less than 7 successful reprojections, the luminance bandwidth gets increased. This prevents insufficient smoothing, but can, for a few frames after disocclusion, cause overblur of high-frequency content that is not preserved by auxiliary feature edge stopping alone. We prefer this over visually disturbing artifacts.

Firefly suppression It is possible for bright samples to not be smoothed sufficiently, which results in fireflies in the final render. To get rid of these, we propose a firefly suppression step. Fireflies are suppressed by applying a 3×3 median filter after the À-Trous filter has been applied. This does not influence the stored history for the next frame, but ensures that no fireflies appear in a frame presented to the user. To prevent specular highlights from being suppressed erroneously, we only apply this to pixels that have a diffuse primary hit.

Disabled indirect illumination variance filtering SVGF repeatedly filters the variance estimates, estimating the variance of the filtered illumination after every iteration of the À-Trous filter. The illumination values are based on more and more samples as the filter iterates, meaning that the variance estimates decrease after every iteration. By using these filtered variance values to control the strength of illumination edge stopping, overblurring is prevented.

However, in our experience, this overblurring prevention often leads to insufficient smoothing for indirect illumination. For this reason, we disable variance filtering for indirect illumination in our SVGF implementation. This may cause overblurring of indirect illumination, but we found the more smooth and more temporally stable indirect illumination to be worth the trade-off.

7 IMPLEMENTATION AND RESULTS

To achieve real-time rendering with path tracing and SVGF, we make use of an efficient custom GPU path tracer. The path tracer is implemented using the NVIDIA CUDA platform. We integrate SVGF, along with our extensions and modifications, as a set of CUDA kernels which run on the GPU.

We perform a variety of experiments to evaluate the produced results and performance of our proposed extensions and modifications, comparing results to our own base SVGF implementation. Quality of produced renders is evaluated using the Root-Mean Squared Error (RMSE) and the Structural Similarity Index (SSIM) [25] of renders to 4096 sample per pixel (spp) reference renders. The RMSE is based on per-pixel errors, and is calculated using tone-mapped images without gamma correction (in SVGF, tone-mapping is performed before the TAA step, so to evaluate the entire SVGF algorithm, tone-mapped images are required). SSIM is a perceptual metric, giving us an indication of perceived image quality by humans. We calculate SSIM using the final gamma-corrected, tone-mapped images. All experiments are conducted using an NVIDIA GTX 1070 at a 1280×720 resolution. We use SVGF parameters $\sigma_z = 1$, $\sigma_n = 64$ and $\sigma_l = 5$, which we found to produce good results in all tested scenes. Temporal integration uses $\alpha = 0.2$ for pixels with diffuse primary hits and $\alpha = 0.1$ for pixels with specular primary hits.

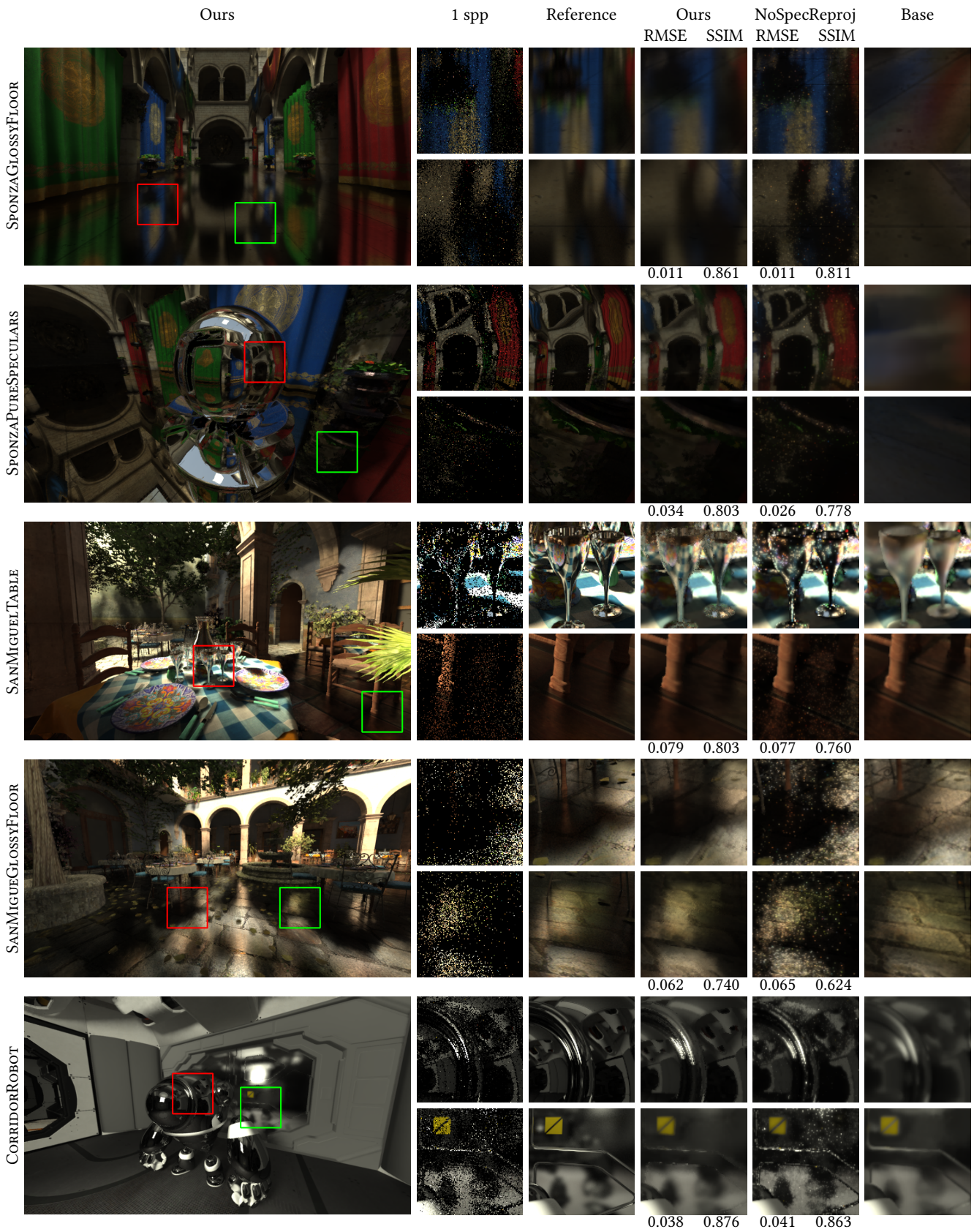


Figure 8: The benchmark frames, taken from animated flythroughs, used to evaluate the quality of filtered reflections. *Base* smears out reflections due to using primary hit motion vectors. *NoSpecReproj* produces temporally unstable reflections that often still contain noise. Our method produces temporally stable reflections without the ghosting artifacts of *Base*. Under each benchmark frame, the RMSE (lower is better) and SSIM (higher is better) of the entire benchmark frame to a 4096 spp render is given.

7.1 Quality and temporal stability of reflections

We evaluate the quality of reflections filtered by our technique using a variety of benchmark frames in various scenes. These benchmark frames are taken from animated flythroughs: this ensure that our results are typical for a frame in an interactive setting. For each frame in the flythrough, one path-traced sample is taken for each pixel. Five benchmark frames are used: SponzaGlossyFloor and SanMiguelGlossyFloor feature glossy specular floors, SponzaPureSpeculars has a floor and an orb that are both purely specular, and SanMiguelTable and CorridorRobot feature both prominent pure and glossy specular reflections. In SanMiguelTable, the cups are purely specular and the brown floor is a glossy specular. CorridorRobot contains a purely specular robot and a door that is a glossy specular.

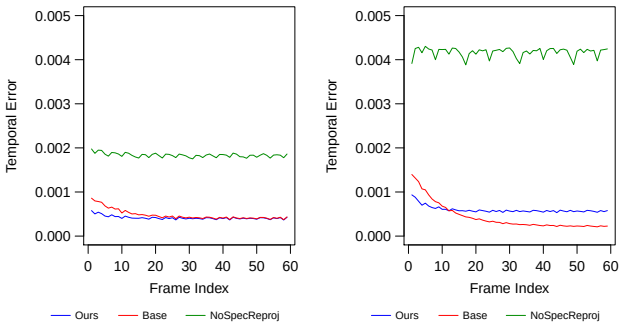
We filter the benchmark frames with our extended SVGF technique (*Ours*) and with our base SVGF implementation (*Base*). Directly comparing the filtered results of *Ours* and *Base* does however not hold much value: *Base* is reprojecting reflected geometry incorrectly, allowing us to arbitrarily increase its reconstruction error by speeding up camera motion. Instead, we compare reconstruction quality of *Ours* to *NoSpecReproj*, which is a version of base SVGF that does not apply reprojection and TAA to specular pixels. The filtered benchmark frames, along with the RMSE, SSIM and several insets comparing the different techniques, are given in Figure 8.

Base is reprojecting reflections incorrectly, leading to smeared out reflections. *NoSpecReproj* and *Ours* do not suffer from this. The reprojection of speculars in our technique reduces the variance of pixel estimates before the filter passes are applied. This leads to less noisy filter output, especially in areas with large amounts of variance, such as the glossy speculars in SponzaGlossyFloor, SanMiguelGlossyFloor and SanMiguelTable. The reprojection makes a substantial difference: *Ours* is able to effectively filter out noise in these areas, whereas *NoSpecReproj* is unable to. This is reflected in the SSIM values of these scenes.

The reduced variance due to reprojection also allows for denoising noisy pure specular reflections, such as the second inset of SponzaPureSpeculars, better than *NoSpecReproj*. In addition, our technique edge stops using feature values of geometry in pure specular reflections and demodulates using albedo of geometry in pure specular reflections before filtering, allowing it to better retain details in pure specular reflections. See for example the first insets of SponzaPureSpeculars and SanMiguelTable. These advantages of *Ours* over *NoSpecReproj*, combined with our modifications for improved robustness, lead to better SSIM values in all scenarios.

Our reprojection is extending our filter into the temporal dimension, leading to a lower variance. However, this additional temporal filtering can also introduce additional bias. *Base* introduces a large amount of bias due to incorrect reprojection. *Ours* introduces much less, but it does still introduce *some* bias. *NoSpecReproj* does not filter temporally, so no temporal bias is introduced. This temporal bias is reflected in the RMSE values of our benchmark frames: *NoSpecReproj* has better, lower RMSE values than *Ours* for two out of five benchmark frames.

Although our technique sometimes produces renders with a higher RMSE than *NoSpecReproj*, it is consistently producing more desirable results. A key property of our approach is that it produces temporally stable results. *NoSpecReproj* does not produce temporally coherent frames due to lack of reprojection: filtered speculars flicker heavily. To measure this difference in temporal stability, we take SponzaGlossyFloor and SponzaPureSpeculars, and measure the average luminance differences of pixels in subsequent frames for a fixed viewpoint. This method is known as measuring the temporal error, and was introduced by Schied et al. [21]. Results are shown in Figure 9.



(a) SponzaGlossyFloor (b) SponzaPureSpeculars

Figure 9: Temporal error in two scenes, measured over 60 frames with a fixed camera and scene. Due to a lack of specular reprojection, *NoSpecReproj* produces much less temporally stable results than our method and base SVGF.

The temporal error for *NoSpecReproj* is on average 4.4 times higher than for our technique in SponzaGlossyFloor, and on average 7 times higher in SponzaPureSpeculars. *Base*, which does apply temporal reprojection for speculars, introducing large amounts of bias in the process, has a comparable temporal error to our technique. In SponzaPureSpeculars, *Base* converges to a lower temporal error than our technique, because it does not perform edge stopping using features of geometry in pure specular reflections, and thus is performing more spatial filtering.

7.2 Improved robustness of the filter

We have made several adjustments to improve SVGF's robustness (see Section 6). We evaluate the image quality with and without our adjustments on two benchmark keyframes from animated flythroughs. Both benchmark keyframes contain areas where paths that return energy occur with a low probability, causing the energy to be focused in a small amount of bright samples. We take frames in which such areas have recently been disoccluded, so that there is little to no history available.

SponzaPillars is taken from an animated sequence in which the camera flies along a series of pillars: each pillar occludes the background for a few frames. We take a frame right after such a disocclusion. The animated sequence of SanMiguelStairs quickly moves the camera past a wall, unveiling a dimly lit area almost exclusively lit by indirect light. The filtered benchmark frames, along with RMSE, SSIM and insets demonstrating the difference our modifications make, are given in Figure 10.

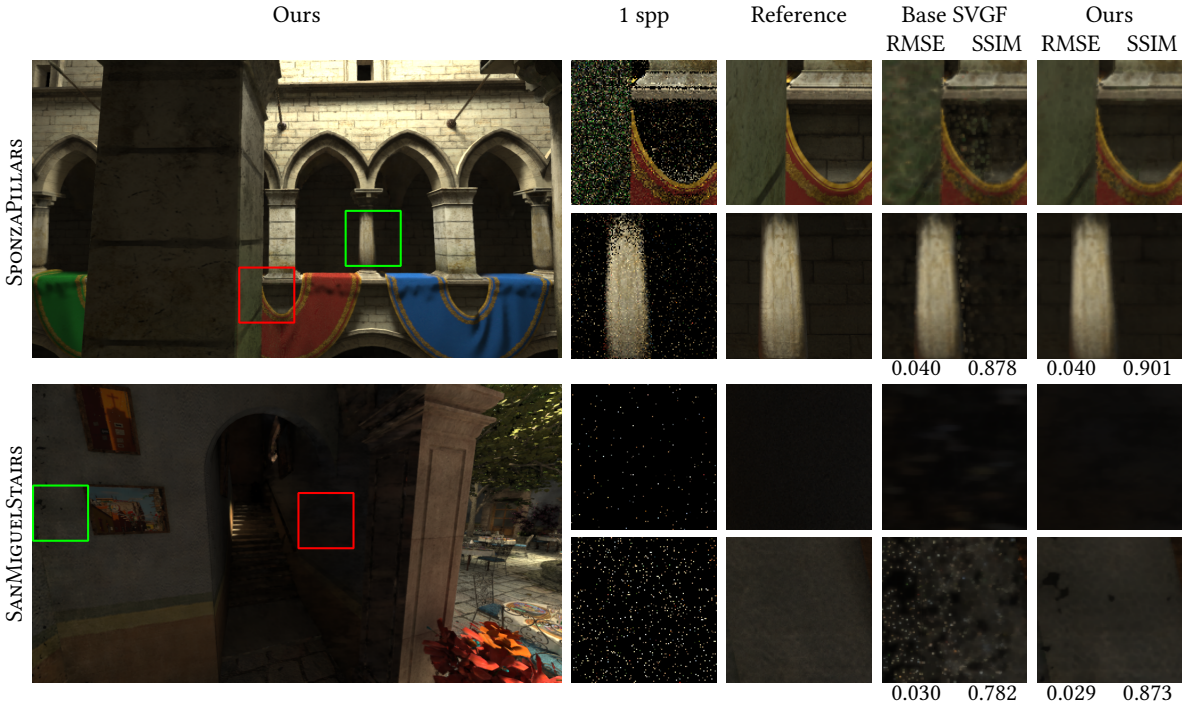


Figure 10: The difficult to filter benchmark frames, taken from animated flythroughs, used to evaluate the impact of our robustness improvements. Base SVGF is unable to effectively denoise parts of these frames when there is no temporal history available (see the insets). With our modifications, artifacts either disappear completely or become much less prevalent. Under each benchmark frame, the RMSE and SSIM of the entire benchmark frame to a 4096 spp render is given. Our method outperforms base SVGF in terms of SSIM, with RMSE staying approximately the same.

The base SVGF method produces noisy areas in both of these frames. SponzaPillars features recently disoccluded areas (above the curtain and next to the circular pillar) where there is no direct illumination and no available history. Variance is underestimated due to the low probability of samples that return energy. Insufficient smoothing occurs, leading to the bright samples still being visibly brighter than their neighborhood in the filtered frame. Similar situations are visible in SanMiguelStairs. The dimly area near the left edge of camera's field of view lacks history: bright samples get smoothed insufficiently and dark areas do not blend with bright areas around them. Bright horizontal strokes are visible on the wall in the center of the frame. These appear because the temporal reprojection has smeared out bright samples that arose from insufficient smoothing.

Our modifications make the filter more robust against these scenarios: artifacts are either completely gone or much less prevalent than before. For our benchmark frames, our modified SVGF method produces higher SSIM values for both filtered frames. The RMSE stays approximately the same.

7.3 Improving results using supersampling

We have introduced supersampling to SVGF by using a modified guide rays approach (see Section 5). We will now evaluate whether taking expensive extra samples with this method is able to improve the quality of filtered results. Our supersampling method should both help to produce sharp anti-aliased results and to decrease variance in the input to the filter, improving reconstruction quality in areas with a low probability of paths that return energy.

We again take benchmark frames from animated flythroughs, which are shown with RMSE, SSIM and various insets in Figure 11. We filter with 1 spp, with and without TAA, and with 4 spp, without TAA. TAA is disabled to test if anti-aliased, non-blurry results can be produced using purely our supersampling. We filter the benchmark frames SanMiguelTable and SanMiguelStairs. SanMiguelTable features purely specular cups and an intricate chair, whose details tend to be overblurred by TAA. SanMiguelStairs features areas with a low probability of paths that return energy, right after a disocclusion. We demonstrated that our robustness modifications help reconstruct these areas, but some artifacts still remain.

At 1 spp, the detailed reflections in the cups and the chair in SanMiguelTable are overblurred by TAA, but without TAA aliasing is prevalent. Our supersampling method with 4 spp produces sharp anti-aliased results. This is reflected in a better SSIM value for the supersampled frame. Supersampling with 4 spp also causes the remaining artifacts in SanMiguelStairs to disappear, again leading to a better SSIM value. RMSE values increase slightly over TAA with 1 sample per pixel: the guide rays approach is setting the filtered color of each non-guide ray sample to the filtered color of its most similar guide ray sample, which can increase bias.

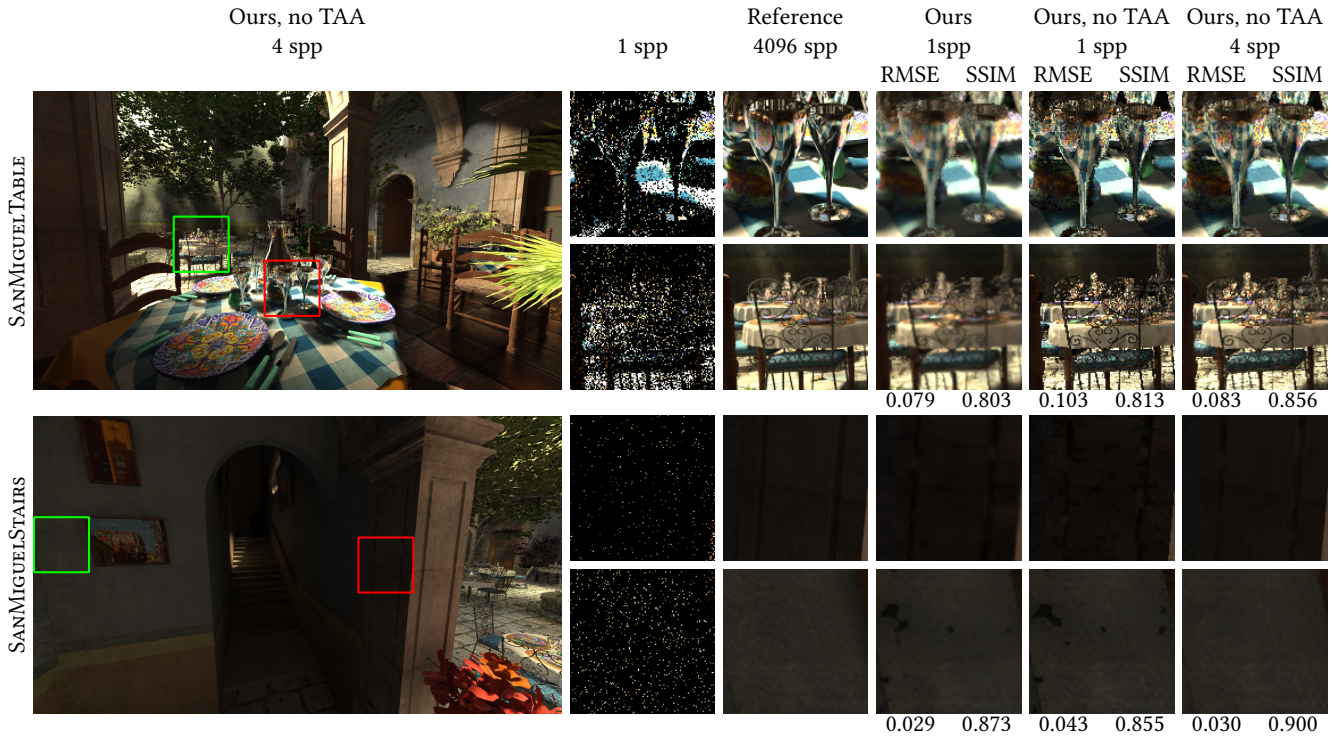


Figure 11: Benchmark frames, taken from animated flythroughs, used to evaluate both advantages of our supersampling method: sharp, effective anti-aliasing (SanMiguelTable) and improved reconstruction quality (SanMiguelStairs). Applying our supersampling technique with 4 spp produces non-blurry anti-aliased results (see SanMiguelTable) and improves reconstruction quality in areas where the probability of sampling a path that returns energy is low (see SanMiguelStairs). Under each benchmark frame, the RMSE and SSIM of the entire benchmark frame to a 4096 spp render is given.

7.4 Motion vector estimation

To evaluate the accuracy of motion vectors for reflected geometry generated using our search algorithm (see Section 4.1), we measure the difference between our specular motion vectors and specular motion vectors generated using a brute-force approach. The brute force approach always finds the best matching pixel q in the previous frame by inspecting a window of 200×200 pixels around the current pixel p (no motion vector in our experiment exceeds 100 pixels). For the brute force approach, we perform the same steps to approximate subpixel accuracy as were described in Section 4.1.

We measure the errors of our motion vectors during flythroughs of two modified version of the Corridor scene: one contains a purely specular orb and the other contains a glossy specular orb. We also evaluate the error of specular motion vectors which would be generated without our LDSP scaling, to determine whether LDSP scaling improves results. No other specular materials are present: otherwise, the brute-force approach might incorrectly find matching geometry in a different reflection. Our approach, which takes search steps of a few pixels, does not suffer from this. Results are given in Figure 12 and Figure 13.

For the purely specular orb the average motion vector error of our approach varies between 0.11 and 1.26 pixels, and for the glossy orb the average error varies between 0.06 and 1.09 pixels. These errors are purely caused by the search algorithm getting stuck in a local minimum before finding the optimal motion vector. Our LDSP scaling makes the algorithm less prone to getting stuck in local minima, consistently reducing the average motion vector error. On average, more than 90% of the found motion vectors in a frame is within one pixel of the optimum, for both the purely specular orb and the glossy specular orb. It is worth noting that, even if our approach finds an incorrect motion vector, our neighborhood clipping step (see Section 4.2.3) will prevent ghosting artifacts by effectively resetting the pixel's history.

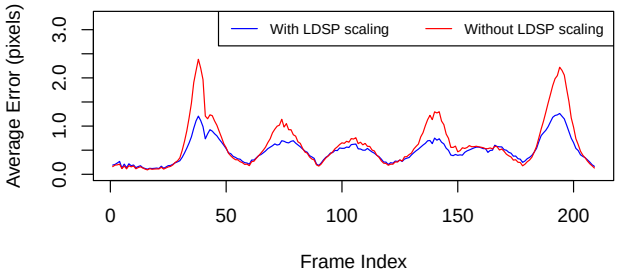


Figure 12: The average error of our motion vectors during a flythrough of Corridor with a purely specular orb. Results are shown with and without our LDSP scaling adjustment.

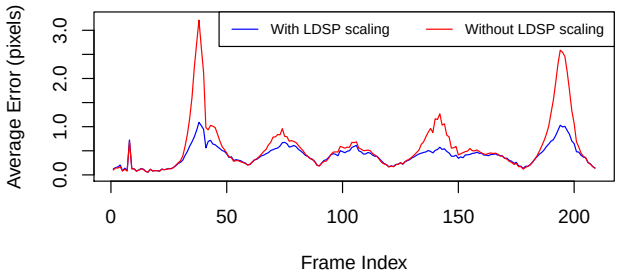


Figure 13: The average error of our motion vectors during a flythrough of Corridor with a glossy specular orb. Results are shown with and without our LDSP scaling adjustment.

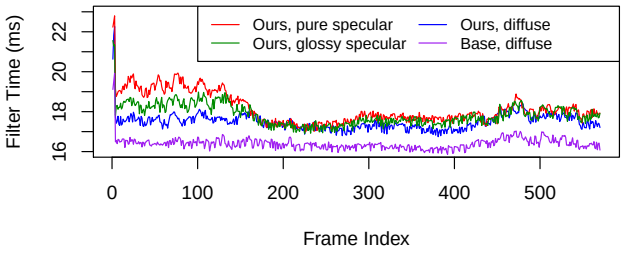


Figure 14: Filter runtime of our extended SVGF implementation and our base SVGF implementation during a flythrough of three variants of Sponza. The first variant contains a diffuse floor, the second a purely specular floor and the third a glossy specular floor. Since the runtime of base SVGF does not depend on the specularity of the filtered materials, its performance is shown only for the diffuse Sponza scene.

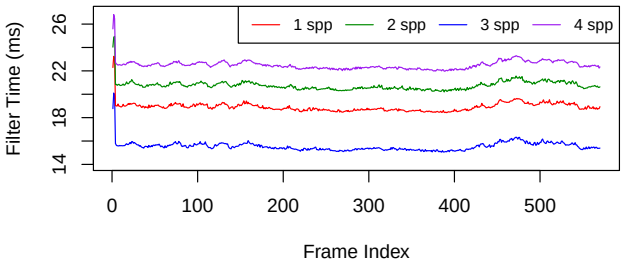


Figure 15: Runtime of our filtering method during a flythrough of Sponza with a diffuse floor for different sample counts. The first additional sample is the most expensive, after which each sample adds a constant amount of runtime. TAA is disabled during these flythroughs, hence the faster runtime at 1 spp compared to Figure 14.

7.5 Impact on performance

The runtime of SVGF varies exclusively due to the spatial variance estimation method, used for pixels with less than 4 successful reprojections, being more expensive than the temporal method. In our modified version of the algorithm, the types of materials hit by primary hits also affect runtime. To measure the performance of our algorithm under different circumstances, we measure the filter runtime during a 570 frame flythrough of three versions of Sponza. We test the regular scene with a diffuse floor, a version with a glossy specular floor (roughness 0.1) and a version with a purely specular floor. Results of the flythrough are shown in Figure 14. We also include the performance of our base SVGF implementation in the scene with the diffuse floor, which excludes any of the modifications and extensions we have proposed. Performance of the base algorithm in the other two scenes is nearly identical and has been omitted.

In the diffuse scene, our modified SVGF approach takes on average 1.1 ms longer to filter than our base SVGF implementation, or about 6.67% of the average runtime of the base SVGF implementation. Pixels with a specular primary hit further increase the runtime. The relationship between the amount of pixels with a specular primary hit and the increase in runtime compared to the diffuse flythrough is roughly linear, with purely specular pixels being more expensive than glossy specular pixels. The cost of a specular pixel is highly dependent on the camera movement and scene, since those influence the amount of steps our diamond search algorithm has to take. In the Sponza flythroughs, the lowest performance, disregarding the start where no history is available, is about 20 ms. Combined with an optimized GPU path tracer that can render 1 spp renders of Sponza in less than 30 ms, a real-time framerate of 20 fps is possible on our hardware.

We also measure the impact of our supersampling method by measuring filtering runtime in a flythrough of Sponza with a diffuse floor with TAA disabled: see Figure 15. Going from 1 to 2 spp increased the filter runtime by 3.38 ms on average, after which every additional sample cost about 1.8 ms. This linear relationship between additional samples and filter runtime is an improvement over the exponential increase in costs caused by additional samples when filtering on the sample level. We argue that, if processing power is available to take additional samples, the extra needed constant filter time per extra sample is likely of little concern.

7.6 Limitations

Our extensions to SVGF enable reprojecting and filtering both glossy and pure specular reflections, producing temporally stable results. However, our technique currently has several important limitations. We will discuss the two limitations which we deem to be the most important.

Usage of only one motion vector per pixel Our technique struggles when a path is able to choose between multiple distinct, dissimilar distributions of outgoing directions, because it is limited to one motion vector per pixel. This means that our technique cannot filter dielectrics effectively: it cannot reproject reflection and transmission components contributing to a pixel color separately.

Overblurring of texture detail in glossy reflections SVGF demodulates the direct and indirect illumination with the surface albedo before filtering. After filtering, the filtered illumination is remodulated. This preserves texture detail. We extended this concept to pure speculars, demodulating using the multiplied surface albedo of both the specular surface and the reflected geometry. However, this approach does not work for glossy specular reflections. A pixel

no longer has one deterministic first diffuse hit, so while we could demodulate our samples, we do not have one texture color with which we could remodulate the pixel after filtering. For this reason, we filter textured illumination components for glossy reflections. This, combined with the fact that we do not use auxiliary features of reflected geometry to edge stop for glossy reflections, can lead to overblur of texture detail in glossy reflections.

8 CONCLUSIONS AND FUTURE WORK

We have extended the SVGF algorithm in several ways. We have presented a technique for reprojection and filtering of both pure and glossy specular reflections. Base SVGF produces temporally stable reflections, but heavily biases the results due to incorrect reprojection of reflected geometry. SVGF can be modified to not reproject specular reflections, which leads to renders without this bias. However, even at modest variance levels, the lack of reprojection has been shown to lead to noisy results, and more importantly, the lack of reprojection leads to temporally unstable, flickering sequences of frames. We have presented a technique that is able to reproject reflected geometry accurately, producing sequences of frames in which reflections are both temporally stable and devoid of ghosting artifacts.

To reproject reflected geometry in a small amount of time we have introduced a motion vector search technique based on diamond search. We have shown the effectiveness of the technique and have demonstrated that our LDSP scaling modification improves accuracy. With several modifications we have made the SVGF algorithm more robust against artifacts in areas where paths that return energy occur with a low probability. Finally, we have introduced supersampling to SVGF, having demonstrated its value for producing sharp anti-aliased images and improving reconstruction quality. Our modifications have made real-time rendering of physically accurate reflections on consumer hardware a reality.

There are several directions for future work. We consider the extension of SVGF to dielectrics the next logical step for the technique: for this, a new reprojection scheme based on multiple motion vectors seems required. A true drawback of our method that we would like to see solved is the large amount of parameters. While our set of parameters was effective on all scenes we have tested, dynamically choosing the best parameters is desirable. Currently, our motion vector estimation approach does not support animated scene content. In the future, we would like to extend our method to support this.

ACKNOWLEDGMENTS

I would like to thank Jacco Bikker for his supervision and guidance during my research, and for authoring the used path tracer. I would also like to thank Frank van der Stappen for fulfilling the role of second examiner. Thanks go out to 3DIMERCE, in particular to Huub van Summeren. The scenes and objects used to evaluate our method were modeled by Guillermo M. Leal Llaguno, Marko Dabrovic, Unity Technologies, Benedikt Bitterli and David Rodriguez. Plots in this paper were created using R [18].

REFERENCES

[1] P. Bauszat, M. Eisemann, S. John, and M. Magnor. 2015. Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field. *Comput. Graph. Forum* 34, 1 (Feb. 2015), 265–276. DOI: http://dx.doi.org/10.1111/cgf.12511

[2] Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Guided Image Filtering for Interactive High-quality Global Illumination. In *Proceedings of the Twenty-second Eurographics Conference on Rendering (EGSR '11)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1361–1368. DOI: http://dx.doi.org/10.1111/j.1467-8659.2011.01996.x

[3] 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. *Comput. Graph. Forum* 35, 4 (July 2016), 107–117. DOI: http://dx.doi.org/10.1111/cgf.12954

[4] 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 (July 2017), 12 pages. DOI: http://dx.doi.org/10.1145/3072959.3073601

[5] Y. Cheng, Xine You, Minlian Xiao, and Minlei Xiao. 2011. A Modified Diamond Search algorithm. In *2011 IEEE International Symposium on IT in Medicine and Education*, Vol. 2. 481–485. DOI: http://dx.doi.org/10.1109/ITiME.2011.6132154

[6] 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, Aire-la-Ville, Switzerland, Switzerland, 67–75. http://dl.acm.org/citation.cfm?id=1921479.1921491

[7] Mauricio Delbracio, Pablo Musé, Antoni Buades, Julien Chauvier, Nicholas Phelps, and Jean-Michel Morel. 2014. Boosting Monte Carlo Rendering by Ray Histogram Fusion. *ACM Trans. Graph.* 33, 1, Article 8 (Feb. 2014), 15 pages. DOI: http://dx.doi.org/10.1145/2532708

[8] Eduardo S. L. Gastal and Manuel M. Oliveira. 2012. Adaptive Manifolds for Real-time High-dimensional Filtering. *ACM Trans. Graph.* 31, 4, Article 33 (July 2012), 13 pages. DOI: http://dx.doi.org/10.1145/2185520.2185529

[9] Homan Igehy. 1999. Tracing Ray Differentials. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 179–186. DOI: http://dx.doi.org/10.1145/311535.311555

[10] James T. Kajiya. 1986. The Rendering Equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. ACM, New York, NY, USA, 143–150. DOI: http://dx.doi.org/10.1145/15922.15902

[11] Brian Karis. 2014. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses* 1 (2014).

[12] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols. 2015. The Path Tracing Revolution in the Movie Industry. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH '15)*. ACM, New York, NY, USA, Article 24, 7 pages. DOI: http://dx.doi.org/10.1145/2776880.2792699

[13] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *ACM Trans. Graph.* 31, 6, Article 194 (Nov. 2012), 9 pages. DOI: http://dx.doi.org/10.1145/2366145.2366213

[14] Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of High Performance Graphics (HPG '17)*. ACM, New York, NY, USA, Article 3, 7 pages. DOI: http://dx.doi.org/10.1145/3105762.3105774

[15] Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *ACM Trans. Graph.* 33, 5, Article 170 (Sept. 2014), 14 pages. DOI: http://dx.doi.org/10.1145/2641762

[16] Bochang Moon, Jose A. Iglesias-Guitian, Sung-Eui Yoon, and Kenny Mitchell. 2015. Adaptive Rendering with Linear Predictions. *ACM Trans. Graph.* 34, 4, Article 121 (July 2015), 11 pages. DOI: http://dx.doi.org/10.1145/2766992

[17] Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *ACM Trans. Graph.* 35, 4, Article 40 (July 2016), 10 pages. DOI: http://dx.doi.org/10.1145/2897824.2925936

[18] R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[19] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive Sampling and Reconstruction Using Greedy Error Minimization. *ACM Trans. Graph.* 30, 6, Article 159 (Dec. 2011), 12 pages. DOI: http://dx.doi.org/10.1145/2070781.2024193

[20] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. 2013. Robust Denoising using Feature and Color Information. *Computer Graphics Forum* 32, 7 (2013), 121–130. DOI: http://dx.doi.org/10.1111/cgf.12219

[21] 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)*. ACM, New York, NY, USA, Article 2, 12 pages. DOI: http://dx.doi.org/10.1145/3105762.3105770

[22] Pradeep Sen and Soheil Darabi. 2012. On Filtering the Noise from the Random Parameters in Monte Carlo Rendering. *ACM Trans. Graph.* 31, 3, Article 18 (May 2012), 15 pages. DOI: http://dx.doi.org/10.1145/2167076.2167083

[23] Peter Shirley, Timo Aila, Jonathan Cohen, Eric Enderton, Samuli Laine, David Luebke, and Morgan McGuire. 2011. A Local Image Reconstruction Algorithm for Stochastic Rendering. In *Symposium on Interactive 3D Graphics and Games (I3D '11)*. ACM, New York, NY, USA, 9–14. DOI: http://dx.doi.org/10.1145/1944745.1944747

[24] C. Tomasi and R. Manduchi. 1998. Bilateral Filtering for Gray and Color Images. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV '98)*. IEEE Computer Society, Washington, DC, USA, 839–. http://dl.acm.org/citation.cfm?id=938978.939190

[25] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.* 13, 4 (April 2004), 600–612. DOI: http://dx.doi.org/10.1109/TIP.2003.819861

[26] Ruifeng Xu and Sumanta N. Pattanaik. 2005. A Novel Monte Carlo Noise Reduction Operator. *IEEE Comput. Graph. Appl.* 25, 2 (March 2005), 31–35. DOI: http://dx.doi.org/10.1109/MCG.2005.31

[27] Shan Zhu and Kai-Kuang Ma. 2000. A New Diamond Search Algorithm for Fast Block-matching Motion Estimation. *Trans. Img. Proc.* 9, 2 (Feb. 2000), 287–290. DOI: http://dx.doi.org/10.1109/83.821744

[28] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Comput. Graph. Forum* 34, 2 (May 2015), 667–681. DOI: http://dx.doi.org/10.1111/cgf.12592

A REFLECTION MOTION VECTOR DIAMOND SEARCH ALGORITHM

ALGORITHM 1: Reflection Motion Vector Diamond Search

Input: Pixel p, world position of pixel p, primary motion vector of pixel p, world position buffer of previous frame.
Output: Estimated motion vector for pixel p.

$stepSize$ = 4;
$bestPos$ = $centerPos$ = *coordinates of pixel* p;
$bestDist$ = $||World_p - PrevWorld_p||$;
pixel m = *pixel at (coordinates of pixel* p *- primary motion vector of pixel* p*)*;
if $||World_p - PrevWorld_m|| < bestDist$ **then**
    $bestDist$ = $||World_p - PrevWorld_m||$;
    $bestPos$ = $centerPos$ = *coordinates of pixel* m;
end
while *true* **do**
    for *each pixel* q *in (LDSP *)* $stepSize$*) centered on* $centerPos$ **do**
        if $||World_p - PrevWorld_q|| < bestDist$ **then**
            $bestDist$ = $||World_p - PrevWorld_q||$;
            $bestPos$ = *coordinates of pixel* q;
        end
    end
    if $bestPos$ == $centerPos$ **then**
        if $stepSize$ == *1* **then**
            break;
        else
            $stepSize$ /= 2 ;
        end
    end
    $centerPos$ = $bestPos$;
end
for *each pixel* q *in SDSP centered on* $centerPos$ **do**
    if $||World_p - PrevWorld_q|| < bestDist$ **then**
        $bestDist$ = $||World_p - PrevWorld_q||$;
        $bestPos$ = coordinates of pixel q;
    end
end
return *coordinates of pixel p -* $bestPos$*;*

B PRELIMINARIES AND LITERATURE STUDY

Utrecht University

Filtering for Path Tracing

Preliminaries and Literature Study

by

Victor Voorhuis

Faculty of Science
Department of Information and Computing Sciences

June 2018

Contents

Chapter 1

Introduction

This document discusses required preliminary knowledge and gives an overview of insights gathered during an extensive literature study. We recommend Chapter 2 to those who are unfamiliar with path tracing and filtering for path tracing. Chapter 3 gives an overview of previous work on filtering path-traced renders, and is recommended to those who desire a more in-depth overview of previous work.

Chapter 2

Preliminaries

This chapter serves as an introduction to path tracing and the filtering of path traced renders. The concept of path tracing will be introduced and contextualized, and several advanced techniques to improve its convergence speed will be discussed in depth. We will conclude with an introduction to the general principles of filtering for path tracing.

2.1 From Rasterization to Physically-based Rendering

Rasterization-based rendering is currently considered to be the algorithm of choice for rendering three-dimensional environments in real-time. The virtual environment, which consists of triangles, is rendered to a two-dimensional image by sequentially processing the triangles in isolation. The vertices of the triangles that lie within the view frustum of the camera are projected to the 2D view of the camera (see Figure 2.1). The fragments covered by the projected triangle are then processed to determine the color they should have in the render: they are *shaded*. To create the illusion of realistic lighting, information about the lights in the scene is often used in combination with normal information to shade the fragments. A z-buffer, which stores depth information for the rendered geometry so far, is used to deal with triangles occluding each other.

Because the triangles in the scene are processed in isolation, dedicated hardware pipelines can quickly draw large amounts of triangles. This makes the approach well-suited for real-time rendering. However, this isolation of the triangles comes at a price. In reality, light does not simply travel from light sources directly to every triangle in the scene. Objects occlude light sources, creating shadows, and reflect light, creating indirect illumination. The processing of triangles in isolation means that such effects do not naturally arise in rasterization.

The desire for more realistic looking graphics has led to the creation of a wide variety of "tricks" to crudely approximate physical phenomena. For example, shadow maps can be used to approximate shadows, cube maps can approximate reflections, and screen-space

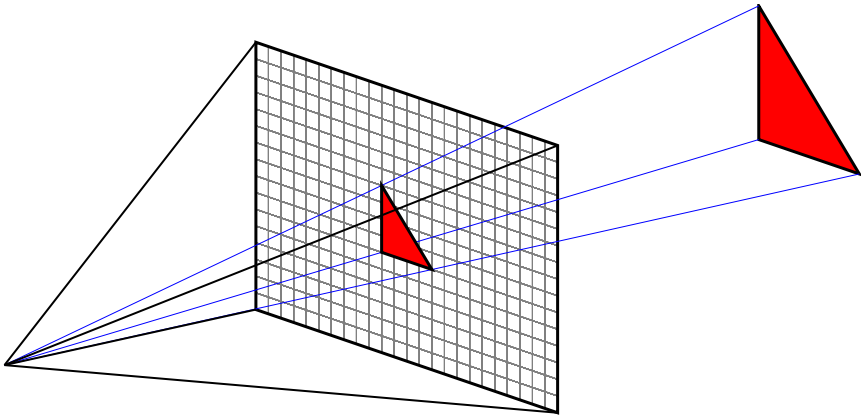


FIGURE 2.1: A demonstration of the working of rasterization. The vertices of the triangle are projected to the screen plane of the camera. Afterwards, fragments falling inside of the triangle are shaded.

ambient occlusion can approximate global illumination. To get results that look physically accurate, a large number of these approximation techniques and tedious tweaking is required. This has made the development of software that renders realistic looking virtual worlds a large undertaking, while the results often leave much to be desired.

Physically-based rendering offers an attractive alternative to the rasterization approach. The general idea is that, by simulating light transport more realistically, using the physical laws that govern light transport in the real world, photorealistic rendering becomes possible without requiring large amounts of tricks and tweaking. Realistic effects, such as soft shadows, indirect illumination, reflections and refractions arise naturally by following rules that are based on those that apply in reality.

2.2 Realistic Light Transport with Path Tracing

The key to physically-based rendering is physically-based light transport. In reality, a surface is not only lit by light traveling directly from light sources to the surface; surfaces themselves also reflect light and receive light that is reflected from other surfaces. Algorithms that simulate this concept are known as *global illumination* algorithms. Crude approximations of global illumination, such as screen space ambient occlusion (SSAO), are used in games of today to achieve more realistic looking graphics in rasterization based engines.

2.2.1 The Rendering Equation

Instead of using crude approximations, we want to simulate physical light transport. Such realistic light transport can be described using the *rendering equation*, first formulated by Kajiya [Kaj86]. It is also often referred to as the *light transport equation*. By

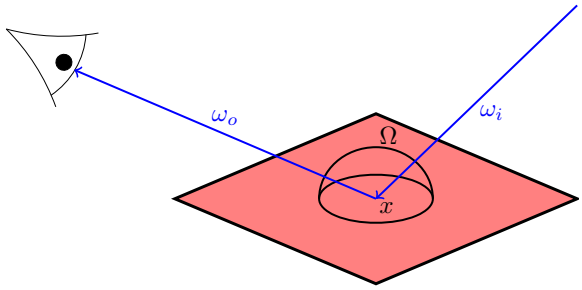


FIGURE 2.2: A visualization of the quantities involved in the rendering equation. Radiance leaves x in direction ω_o. This outgoing radiance consists of the radiance transmitted by x towards ω_o and the reflected radiance. The reflected radiance is determined by integrating over all directions on hemisphere Ω. For each direction, radiance is converted to irradiance and scaled by the BSDF. In this figure, one incoming direction ω_i is shown.

assuming that light travels instantly in straight lines, the equilibrium of light transport in a scene is expressed in an equation. One formulation of this rendering equation is:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_s(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{2.1}$$

The radiance leaving position x in direction ω_o ($L_o(x, \omega_o)$) consists of the radiance that x emits into direction ω_o ($L_e(x, \omega_o)$) and the radiance that is reflected from x towards direction ω_o. The reflected radiance is an integral over the possible incoming directions on the hemisphere Ω. For an incoming direction ω_i, the reflected radiance towards ω_o is the incoming radiance from that direction at x ($L_i(x, \omega_i)$), converted from radiance to irradiance on the surface (by multiplying with $(\omega_i \cdot n)$), and finally multiplied by the bidirectional scattering distribution function (BSDF). The BSDF expresses how the material at x scatters incoming irradiance to outgoing radiance for a pair of directions ($f_s(x, \omega_i, \omega_o)$). See Figure 2.2 for a visualization of these quantities.

The rendering equation expresses the radiance leaving an arbitrary point in an arbitrary direction. This is only one possible formulation, however. For example, the alternative *three-point form* does not integrate over the hemisphere at the intersection, but over all scene surfaces. A visibility term ensures that only light from visible surfaces is integrated. Extensions also exist, such as one that distinguishes between different wavelengths, allowing for spectral rendering.

2.2.2 The Measurement Equation

We are specifically interested in rendering an image, instead of just calculating the amount of radiance exiting certain points in certain directions. This is where the so-called *measurement equation* comes into play:

$$p = \int_{P \times \Omega} W(x, \omega) L_i(x, \omega) dx d\omega \tag{2.2}$$

The measurement equation models a sensor in the virtual scene: it integrates over the surface of the sensor (P) and the possible incoming directions (Ω). $W(x, \omega)$ is the sensitivity of the sensor at x for direction ω. Using this, we can create a virtual camera. For example, a "perfect" pinhole camera would consist of a sensor for each pixel, with each position on the sensor being sensitive to only one direction.

Similar to the rendering equation, different forms of the measurement equation exist. We could simplify it by not integrating over the sensor surface and possible directions, simply assigning each pixel one position and one direction. The lack of an integral over the pixel surface would then result in visible aliasing. But we could also expand it, by for example also integrating over the surface of a lens to achieve depth of field, or by integrating over time to achieve motion blur.

An image with light transport following the rendering equation could now theoretically be rendered by solving the measurement equation for each pixel of our virtual camera. Unfortunately, analytically solving the high-dimensional integral is impossible in every non-trivial scene. Rasterization-based rendering is essentially solving a simplified version of the rendering equation without integrals, in which only light traveling directly from light sources to geometry is considered.

2.2.3 Whitted-style Ray Tracing

Whitted-style ray tracing [Whi79] is a rendering method which does not render scenes by sequentially drawing the different triangles, but by tracing a ray for each pixel. For each pixel, a ray is intersected with the scene geometry to obtain the geometry visible in that pixel. The pixel is then shaded by casting *shadow rays* from the intersection to the different light sources, determining whether light travels from the light source to the geometry being shaded (see Figure 2.3). Fully specular reflections can be rendered by reflecting the ray off the specular surface and tracing the reflected ray. Refraction can also be accurately rendered by transmitting rays into objects.

Shadows, reflections and refractions fit elegantly into Whitted-style ray tracing; the rendering equation is approximated closer than by rasterization-based rendering. However, we are still omitting the integrals in the rendering equation. Distribution effects, such as light arriving from all directions on the hemisphere, area lights, depth of field and motion blur remain unsupported.

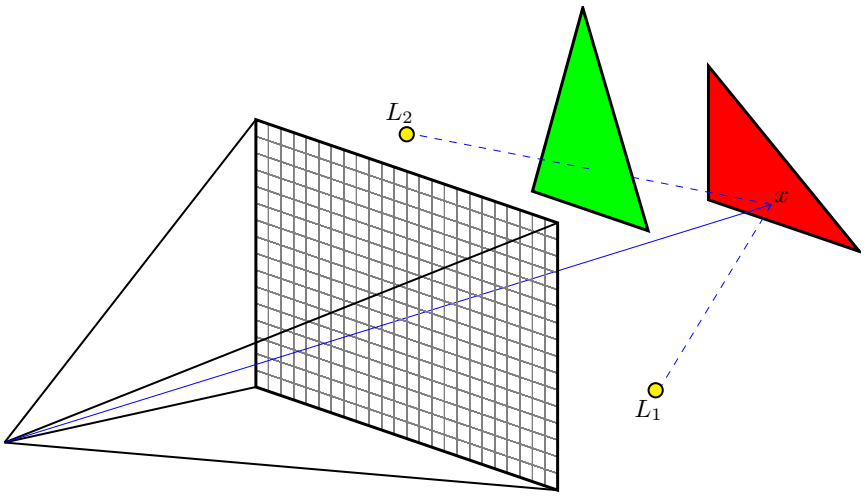


FIGURE 2.3: Whitted-style ray tracing demonstrated. The solid blue ray goes through a pixel and intersects the red triangle at x. Two shadow rays are cast from x: one towards L_1 and the other towards L_2. L_1 is visible, but the shadow ray to L_2 intersects the green triangle, indicating that L_2 is not visible from x.

2.2.4 Rendering with Monte Carlo Integration

Path tracing is a rendering method introduced by Kajiya [Kaj86]. Unlike rasterization and Whitted-style ray tracing, the path tracing technique does not simplify the rendering equation to make rendering the scene possible. Instead, it employs Monte Carlo integration to approximate the high dimensional integral. The idea of Monte Carlo integration is to approximate the value of the integral by taking random samples from the integral:

$$\int_A^B f(x)dx \approx \frac{B-A}{N}\sum_{i=1}^{N} f(X_i), \text{where } X_1, ..., X_N \in [A, B]. \tag{2.3}$$

The key here is that we approximate the integral by performing a stochastic experiment. N random samples of the function to be integrated are taken, which are then combined with the formula to obtain an estimate of the integral. The expected value of this experiment is the actual value of the integral. The variance of the outcome of the experiment decreases linearly with the number of samples N, and thus, the standard deviation of the outcome decreases with a rate of $\frac{1}{\sqrt{N}}$. As the amount of samples approaches infinity, the standard deviation of the estimator approaches 0 and thus, the outcome of the experiment approaches the true value of the integral.

Path tracing applies Monte Carlo integration to approximate the incoming light according to the high dimensional integral (formed by the measurement equation and rendering equation) for each pixel of the virtual camera. For each pixel, a ray is cast, similar to

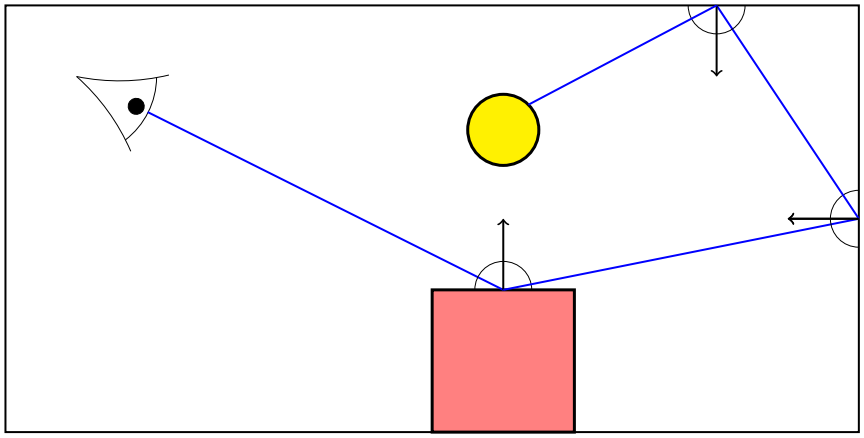


FIGURE 2.4: A path is recursively traced with a simple path tracing algorithm. First, a ray is shot from the camera which intersects the red box. Then, a random direction on the hemisphere is chosen and a new ray is shot. This continues until a light source is hit or a ray leaves the scene. At each intersection, the potential throughput is scaled with the BSDF $f_s(x, \omega_i, \omega_o)$ and a factor $(\omega_i \cdot n)$.

Whitted-style ray tracing. The ray is intersected with the scene, and the rendering equation is evaluated for the intersection x. This is where the integral in the rendering equation appears: we cannot evaluate the integral over the hemisphere at x directly. So instead, we simply choose a random direction and cast a new ray, evaluating the light that travels along that ray to x. The radiance found from this recursive call is scaled by the BSDF $f_s(x, \omega_i, \omega_o)$ and a factor $(\omega_i \cdot n)$. A light path terminates when a light source is hit: at that point, light transport from that light source to the eye is found. This sample is used to update the current estimate for this pixel. This scheme is repeated, sending more and more rays through the virtual pixels, establishing paths to light sources, so that each pixel color estimate is based on more and more samples. See Figure 2.4 for a visualization of a path being traced.

As the number of rays through a virtual pixel approaches infinity, each pixel's color estimate eventually converges to the true value. An estimator is *unbiased* if the expected error of the estimator equals zero, which is the case for path tracing. The error in an unbiased estimator completely consist of variance. If the expected error approaches zero as the amount of samples increases, the estimator is *consistent*. Note that it is possible for an estimator to be biased, yet consistent: then, the bias vanishes as the amount of samples approaches infinity.

Path tracing can be used to approximate a variety of difficult effects. For example, refractive objects can be accurately simulated by "rolling dice" to determine whether a refracted or reflected ray should be chosen. We can get smooth, anti-aliased results by integrating over the surface of the virtual pixel in the measurement equation: we then choose a random position on the pixel for each sample. Depth of field can be accurately

simulated by integrating over the surface of a virtual lens in the measurement equation: we choose a random position on the lens for each sample.

Unfortunately, for the variance of our pixel estimates to become reasonably small, a large amount of samples can be required. The variance in the pixel estimates, caused by under-sampling of the integrals, shows up in the rendered images as noise. The introduction of additional dimensions to the integral, such as for anti-aliasing, depth of field or motion blur, makes the problem worse: more samples will be required to reach convergence. Fortunately, techniques exist that lead to faster convergence. Also, it is possible to use filters to try to filter this noise out of the images.

2.3 Variance Reduction

An important observation to make is that in path tracing, some samples will return significantly more energy than others. The more bounces before a light source is hit, the smaller the light transport along the path will generally be. After all, non-fully specular surfaces scatter light as they reflect it. And there are many other factors: some lights transmit more energy than others, certain materials reflect more light along certain directions, and the list goes on.

2.3.1 Importance Sampling

Fortunately, we don't have to select our samples with uniform probability. We can select our samples using a probability density function (pdf) $p(x)$ of our choosing, as long as the pdf is positive for any sample with a positive value (=positive light transport) and integrates to one. The Monte Carlo integration then works as follows:

$$\int_{\Omega} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}. \tag{2.4}$$

Samples with a higher pdf value may now be chosen more often, but they will not contribute disproportionately to the final estimate, because they are also divided by a higher pdf value. Now we can distribute the samples almost completely freely. It is a good idea to sample the paths with a higher contribution with a higher probability, because those paths have the most influence on the final color estimate. Doing this will decrease the variance of our estimates: having less accurate estimates of the light transport through less influential paths does not decrease quality of our estimates significantly, while having more accurate estimates of the light transport through more influential paths leads to better estimates. For example, it is a good idea to sample paths according to their value for the $(\omega_i \cdot n)$ term, or according to the value of the BSDF function (see Figure 2.5).

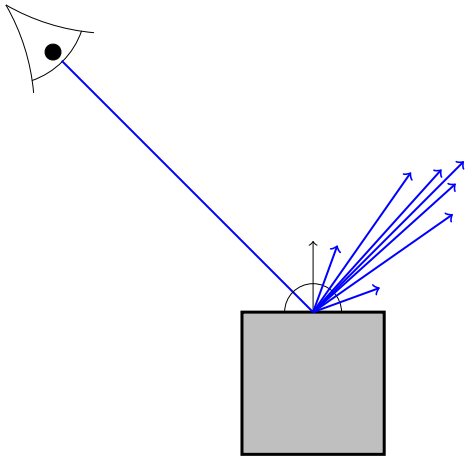


FIGURE 2.5: An example of importance sampling: more samples are sent to directions on the hemisphere which are likely to have a higher contribution.

2.3.2 Russian Roulette

Like stated before, the more bounces, the less throughput a path will generally have. For unbiased estimates, we would technically have to keep bouncing until a light source is hit or a ray leaves the scene. This means that we could be tracing long paths with little light transport.

Fortunately, we can again apply importance sampling. We want to consider shorter paths, which generally have a higher throughput, as more important. A simple technique to realize this is to have a termination probability at each bounce. If the path survives, the contribution of the path is scaled by dividing by this termination probability. Because of this, paths that are unlikely to be sampled will be scaled up to compensate for that fact. Each path still has a non-zero probability, which means that the result remains unbiased.

2.3.3 Next Event Estimation

Consider an intersection x with a surface along one of the paths that is being traced. If the surface is diffuse, it is easy to see that direct light, light that travels directly from a light source to x, will likely contribute more light transport than light that travels to x via one or more bounces. Thus, the rays from x towards light sources are likely important.

We can exploit this fact by separating the sampling of direct and indirect light traveling to x. Instead of just sending a ray towards somewhere on the hemisphere, we now also send a ray directly towards a random point on a light source. The contributions of these two rays are simply summed together. If a random ray sampled with the hemisphere hits a light, we ignore it to make sure that the contributions from the light are not sampled

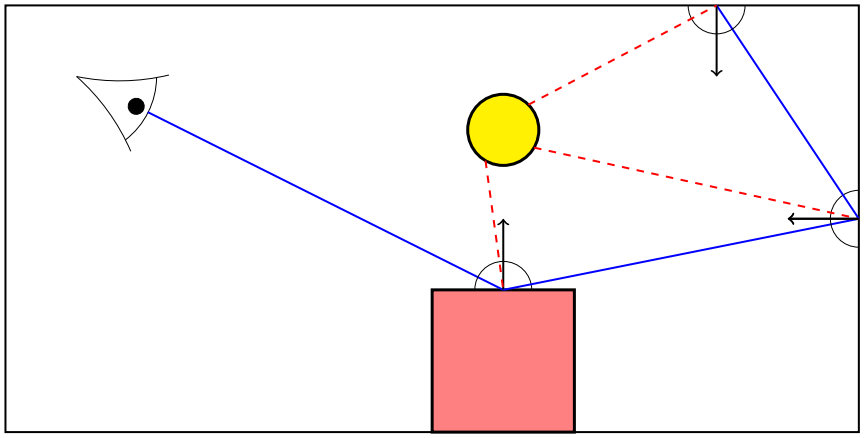


FIGURE 2.6: Next event estimation in action: direct and indirect lighting are now estimated separately. From each vertex, a random ray on the hemisphere is cast (blue) and a ray towards a random point on a light source is cast (dashed red).

more than once. Similarly, rays send directly towards a light source that do not hit the light source are ignored. For an illustration, see Figure 2.6.

For the direct light sampling we can now use a different pdf than for the indirect light sampling. A good value is the solid angle of the light on the hemisphere of x multiplied by the luminance of the light source (the bigger this value, the more light transport there could be).

2.4 Filtering for Path Tracing

Despite the existence of several effective variance reduction techniques, it can still take a prohibitively large amount of samples to generate noise-free images with a path tracer. This makes path tracing unsuitable for real-time applications, and even for offline rendering, convergence can take prohibitive amounts of time. With more complex scenes and more complex effects, this problem only becomes larger.

The idea of filtering for path tracing is to combine the estimates of multiple Monte Carlo estimators, assuming that they converge to similar values, to reduce their variance. This can help get rid of the visually disturbing noise, so that noise-free images resembling the converged result can be obtained quickly. A popular way of doing this is to work in image space and blend nearby pixels together, exploiting the spatial coherence in a render:

$$\hat{\boldsymbol{c}}_i = \frac{\sum_{j \in \mathcal{N}_i} \boldsymbol{c}_j w(i,j)}{\sum_{j \in \mathcal{N}_i} w(i,j)}. \tag{2.5}$$

In this formula, $\hat{\boldsymbol{c}}_i$ is the filtered estimate of the color of pixel i, replacing the unfiltered estimate $\boldsymbol{c}_i$. The estimate is a weighted average of the colors of the pixels in a window $\mathcal{N}_i$ which is centered on pixel i. $w(i,j)$ denotes the weight between pixels i and j.

However, one has to be careful. The assumption that neighboring pixels converge to the same value is usually not completely true, which means that we introduce bias by combining neighboring pixels. When the assumption is completely wrong, for example if we were to combine samples of pixels on opposite sides of an edge, we introduce a large amount of bias. The goal of these filtering algorithms is to reduce variance while keeping the introduced bias small. Visible noise has to be smoothed away, while sharp features that would be present in the converged render are retained. This is a significant challenge, since we do not know what the converged render will look like.

It is worth noting that, while filtering on the pixel level is a popular method, alternatives exist. When taking multiple samples per pixel, it is for example possible to filter on the sample level instead. It is also possible to perform filtering in world space or in path space instead of in image space. Filtering on the pixel level is however attractive, because of low computational cost and easy integration into existing renderers.

2.4.1 Only Combining Similar Pixels

We have seen that to filter successfully, we *only* want to combine pixels that will converge to similar values. This is where the weight term $w(i,j)$ in Equation 2.5 comes in: we can use it to weigh similar pairs of pixels highly and diminish the weight of dissimilar pairs.

A simple choice would be to use a Gaussian kernel:

$$w(i,j) = \exp\left(\frac{-||\boldsymbol{p}_i - \boldsymbol{p}_j||^2}{2\sigma_d^2}\right), \tag{2.6}$$

where $\boldsymbol{p}_i$ is the screen-space position of pixel i and σ_d is the spatial standard deviation of the Gaussian kernel. This weighting function will weigh pixels that are close to the current pixel more severely than pixels that are further away. This is a good idea, since pixels that are close together are more likely to be similar than pixels that are far apart. σ_d is the "bandwidth" parameter that determines the size of the Gaussian kernel. Filtering with a single Gaussian kernel is generally insufficient: a large kernel is required for sufficient smoothing in noisy areas, which will blur out high-frequency details.

A popular image denoising algorithm is the non-linear bilateral filter [TM98]. Traditional low pass filters for images, such as the Gaussian filter, blend pixels together using domain kernels: pixels are blended together with their neighbors, with closer neighbors being weighed more severely. The bilateral filter extends this by also letting the weights depend on differences in range: the colors of the pixels.

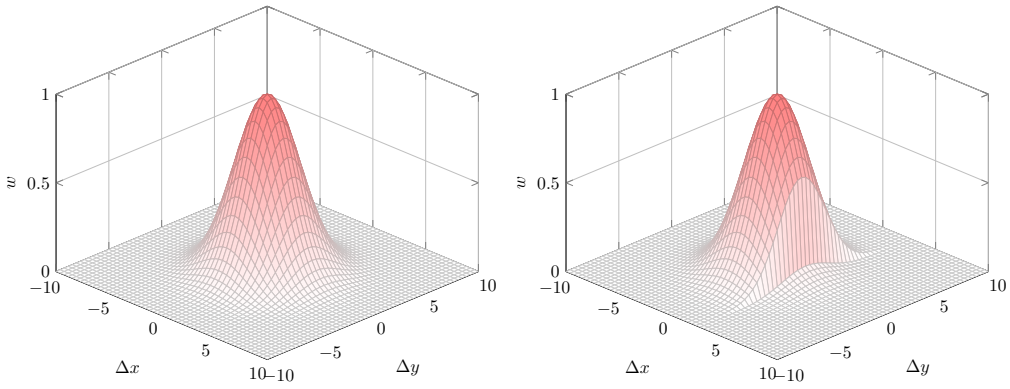


FIGURE 2.7: Two different filtering kernels visualized. The current pixel is located at the center, and $\sigma_d = 5$. (a) is a Gaussian kernel: the weights for pixels around the current pixel only depend on the difference in spatial coordinates. (b) is a bilateral kernel. A hard edge is present at $\Delta x = 2.5$: weights for pixels on the other side of the edge are diminished.

$$w(i,j) = \exp\left(\frac{-||\boldsymbol{p}_i - \boldsymbol{p}_j||^2}{2\sigma_d^2}\right) \times \exp\left(\frac{-||\boldsymbol{c}_i - \boldsymbol{c}_j||^2}{2\sigma_r^2}\right). \tag{2.7}$$

Now, the weight between pixels i and j also depends on their difference in color $||\boldsymbol{c}_i - \boldsymbol{c}_j||$. A corresponding bandwidth parameter σ_r has been introduced, which determines how severely differences in pixel color influence the weight. Edges are now preserved, because the pixels of two distinct regions, with different color values, will not blend together. For a visualized Gaussian and bilateral kernel, see Figure 2.7. Unfortunately, the bilateral filter is not robust against severe noise that might be present in our renders. A difference in color could not only have been caused by the pixels actually converging to different colors, but also by undersampled Monte Carlo integration. Outlier pixels in a noisy input do not blend with their neighbors, even if they would converge to similar values.

We can also use more information than just the color to "steer" our filter. The joint/cross bilateral filter [ED04, PSA*04] derives its range distance from one or more separate guide images. This means that we could use less noisy inputs, such as the normals or depths at the first hits, to "steer" our filter. Such extra geometric data is often referred to as auxiliary features. We can formulate a new weight function that uses these auxiliary features in combination with spatial and color distances:

$$\begin{aligned} w(i,j) = \exp\left(\frac{-||\boldsymbol{p}_i - \boldsymbol{p}_j||^2}{2\sigma_d^2}\right) \times \exp\left(\frac{-||\boldsymbol{c}_i - \boldsymbol{c}_j||^2}{2\sigma_r^2}\right) \\ \times \prod_{k=1}^{K} \exp\left(\frac{-||\boldsymbol{f}_{k,i} - \boldsymbol{f}_{k,j}||^2}{2\sigma_k^2}\right), \end{aligned} \tag{2.8}$$

where $\boldsymbol{f}_{k,i}$ is the feature vector of the k'th feature at pixel i (e.g. the normal at pixel i) and σ_k is the bandwidth parameter for feature k. The usage of auxiliary features

can reduce introduced bias. If we, for example, introduce the depth of the first hit as a feature, pixels with dissimilar depths would no longer be incorrectly blended together, even if they have a similar color.

Unfortunately, auxiliary features cannot always be applied effectively. If we make use of stochastic primary ray effects, such as depth of field or motion blur, or if we spread our samples over a pixel's surface (performing anti-aliasing by supersampling), one pixel no longer has one clear value for each feature. The different samples of the pixel might, after all, have different feature values. A popular way to handle this is to simply average the feature values together for each pixel, but this means that our feature values are no longer noise-free. This noise might in turn introduce noise in the filter output. Some approaches which use auxiliary features do not support stochastic primary ray effects for this reason, while others provide methods to filter effectively even with noise in the features.

2.4.2 Real-time Performance using Filtering

Traditionally, path tracing was exclusively used in offline scenarios, simply because real-time path tracing was not feasible. Thus, most earlier Monte Carlo denoising methods have focused on this scenario as well. Only more recently has more research focused on filtering in a real-time setting. With faster and faster graphics hardware being available and GPU path tracers becoming the norm, path tracing can now be done in real-time, albeit with low sample counts. This has sparked an interest in filtering algorithms that run fast and can handle high variance input.

2.4.3 Wavelets and the Discrete Wavelet Transform

The general filtering techniques discussed so far filter out noise by blending pixel colors with those of their neighbors. However, that is not the only effective strategy. The discrete wavelet transform (DWT) is a powerful tool for denoising signals, including images. An image can be denoised by first transforming it into a series of wavelet coefficients, then modifying the wavelet coefficients and finally transforming the wavelet coefficients back into an image. For the reader unfamiliar with the discrete wavelet transform, we will give a brief, practical introduction here. We will leave out some details such as scaling functions. For those who desire a more complete theoretical understanding of wavelets, we recommend the guide by Valens [Val99].

Fourier expansion is a popular technique which allows us to represent a signal as a sum of sines and cosines with different frequencies. The wavelet transform is similar, but does not represent the signal using a series of unlimited sines and cosines, but using a series of wavelets. Wavelets are waves limited in the space/time dimension. By taking one *mother wavelet*, scaling and translating it in every possible way, and then convoluting

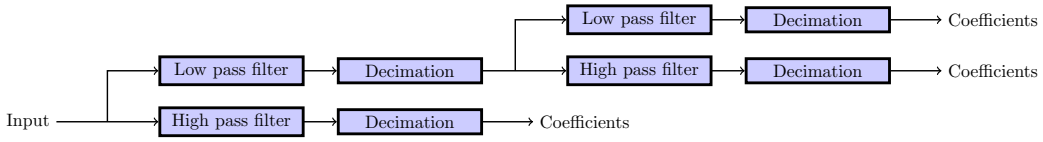


FIGURE 2.8: Calculating the 2-level discrete wavelet transform of a 1D signal with Mallat's algorithm. The output of the high pass filter forms the coefficients, while the output of the low pass filter is passed on to the next level, if there is one.

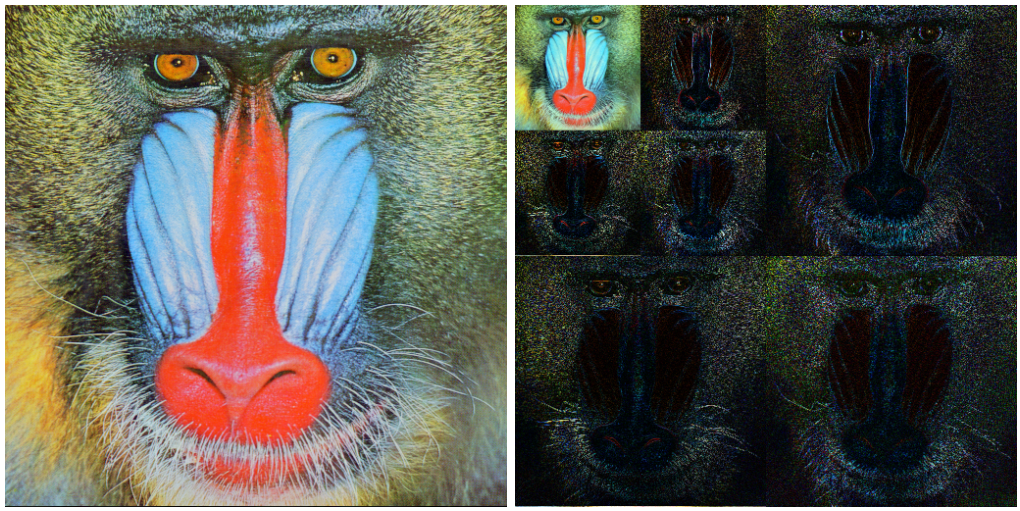
these translated/scaled wavelets with the signal, we obtain a representation of the signal that is highly redundant. This is the *continuous wavelet transform* (CWT).

The *discrete wavelet transform* (DWT) is more practical than the CWT: we obtain a less redundant representation by not convoluting with all possible wavelets, but by a limited amount. The scales of the used wavelets are limited to the powers of two. One can use a wavelet of each scale for all possible integer translations (the *undecimated* DWT), or reduce redundant overlap between wavelets by spacing them according to their scale (the *decimated* DWT). The amount of produced coefficients by the undecimated DWT is the same as the amount of input coefficients.

The representation of the signal obtained by the undecimated DWT is *sparse*: many of the coefficients will be small, while most information is captured in a small amount of coefficients. For images, the small coefficients often capture noise, while large coefficients capture the actual structure of the image. This allows us to perform smoothing. By setting the small coefficients to zero, we get rid of the small noisy features and retain the structure of the image. Hard-thresholding sets all coefficients under a threshold to 0, while soft-thresholding subtracts the threshold from all coefficients, also shrinking coefficients above the threshold. This denoising method is known as *wavelet shrinkage*.

The different wavelet scales used in the DWT capture different bands in the frequency spectrum. It turns out that because of this, the DWT can be efficiently calculated by repeatedly convoluting the image with a low pass and a high pass filter. This is known as Mallat's Algorithm [Mal89]. We begin by applying the low pass and high pass filter, storing the result of the high pass filter as wavelet coefficients. The result of the low pass filter is again filtered using the low pass and the high pass filter. After the filter applications, the results are downsampled by a factor two (this is decimation). The process is repeated for a certain amount of levels (see Figure 2.8). The DWT can be implemented efficiently using this iterated filtering, while the wavelets do not even have to be explicitly specified anymore.

For 2D images, the process is similar, but in each iteration, four decimated images are obtained instead of two. The filters are combinations of vertical and horizontal high pass and low pass filters. We obtain high pass vertical-high pass horizontal coefficients, high pass vertical-low pass horizontal coefficients, low pass vertical-high pass horizontal coefficients, and low pass vertical-low pass horizontal coefficients. The decimated low



(a) (b)

FIGURE 2.9: The discrete wavelet transform applied to the baboon image. The input (a) is transformed into wavelet coefficients visible in (b). The wavelet coefficients can be transformed back into (a) without any loss of information.

pass-low pass results are passed on to the next iteration. The result of the DWT for an image is shown in Figure 2.9.

Chapter 3

Literature study

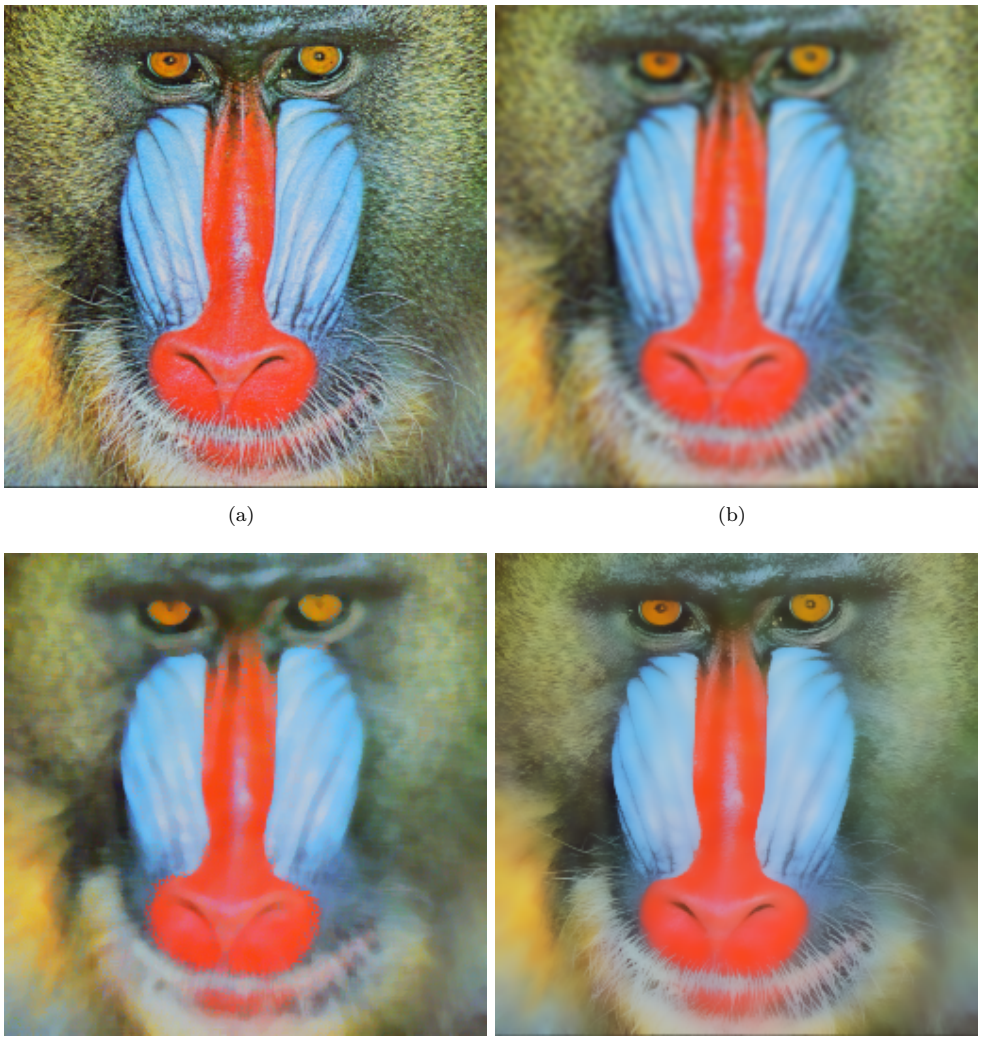
The following extensive literature study will give a detailed overview of previous work focusing on denoising path-traced renders. The discussion is split into several main categories, but it is worth noting that many of these techniques could easily fit under multiple of the subsections. Influential older papers, several main categories of approaches and several state-of-the-art methods will be discussed. Finally, we conclude with an assessment and summary of the discussed work.

3.1 Early Influential Work

Lee and Redner [LR90] were some of the first to successfully apply filtering to images rendered using stochastic sampling. They observe that linear filters have several important drawbacks: when the filter kernels are too narrow, not all noise is filtered out, but when the kernels become bigger, edges are smeared out (see Figure 3.1 b). Besides this, "pops" (pixels that differ significantly from their neighborhood, which we call outliers) are not smoothed sufficiently, leaving clearly too bright or too dark areas.

To deal with these issues, they discuss two approaches. A median filter (see Figure 3.1 c) simply takes the median of the window around a pixel, and an alpha-trimmed mean filter takes the average of the values in the window, while disregarding a fraction of the most outlying values. Both these approaches effectively smooth the image, removing visible outliers by ignoring them. In a wide range of signal processing contexts, these outliers can reasonably be considered to be measurement errors, but in path tracing, all samples, even outliers, are valid. Removing samples causes energy to be lost, biasing the result.

Rushmeier and Ward [RW94] suggested an alternative filter that *does* preserve energy. They still use a nonlinear filter, but instead of discarding outliers, they identify outliers, use the average of the neighbors to guess what fraction of the energy is "excessive", and then spread this excessive energy over an area around the pixel. To avoid obviously



(a) (b)

(c) (d)

FIGURE 3.1: Different filters applied to the baboon image (a). (b): A Gaussian filter with a 7×7 kernel. Note that edges are blurred. (c): A median filter with a 7×7 window. The median filter is effective at removing outliers, but introduces noisy results in noisy areas. (d): A bilateral filter with a 7×7 spatial window and a bandwidth of 0.3 . The bilateral filter smooths surfaces with similar colors, and thus, outliers are preserved.

brighter patches around outliers, they adjust the size of the window accordingly. The energy can still spread over edges, which means that when outliers occur near edges, results are heavily biased.

A later approach that is energy-preserving and respects edges was introduced by McCool [McC99]. McCool applies anisotropic diffusion to the noisy render: the energy of the pixels essentially diffuses, smoothing the image in the process. To maintain edges, McCool applies a color coherence metric and a coherence map. If two pixels have dissimilar colors and low estimated variances, the color coherence metric, and consequently the diffusion between the pixels, will be low. The coherence map uses geometric information (including normals and depths) to lower diffusion at edges. McCool was one of the first to use (less noisy) geometric features to "steer" his filter, a concept that would become prevalent in future work. The technique works well and preserves edges, but works using iterations that smooth the image more and more. It is not trivial to know when to stop iterating.

3.2 Bilateral Filtering for Monte Carlo Denoising

Xu and Pattanaik [XP05] stated that Monte Carlo noise appears both as inter-pixel incoherence and outliers, while noting that Monte Carlo denoising techniques at the time (including [RW94] and [McC99]) could not deal with both types in a unified way. They extended the bilateral filter to not only deal with inter-pixel coherence, but also with outliers, which the bilateral filter does not suppress directly.

The bilateral filter does not smooth outliers sufficiently because the neighbors do not have similar color estimates. They resolve this by first applying a Gaussian filter to the image, which is used as input for the illumination weight calculation. Unfortunately, the low-pass Gaussian filter smooths out sharp features, which in turn smooths out these features in the filter output. Sharp features can thus not be retained satisfyingly.

3.3 Wavelet Shrinkage for Monte Carlo Denoising

A technique that employs wavelet shrinkage for Monte Carlo denoising is Adaptive Wavelet Rendering (AWR), introduced by Overbeck et al. [ODR09]. For a discussion on wavelets and wavelet shrinkage, see Section 2.4.3. Overbeck et al. perform a discrete wavelet transform with Daubechies wavelets and perform soft thresholding to smooth the produced images, but make an important modification. In path tracing renders, the noise is generally not uniform over the image. To handle this, Overbeck et al. use a threshold that varies for different positions in the image. They conservatively estimate the variances of the different wavelet coefficients and use the standard deviations as the thresholds. This means that the wavelet coefficients, which encode high frequency details, are set to the smallest value within a standard deviation of the current noisy

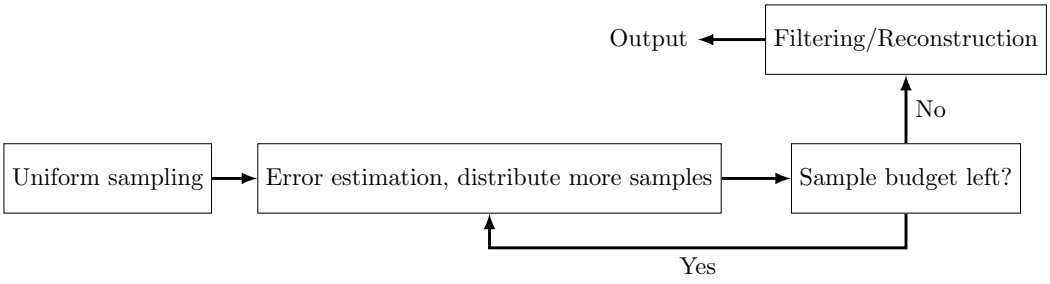


FIGURE 3.2: The general idea of the adaptive sampling approach by Overbeck et al. [ODR09]. Many later techniques employ a similar scheme.

estimate: the constructed image is effectively the smoothest image that is consistent with the samples.

3.4 An Iterative Adaptive Sampling Strategy

Besides the described denoising approach, Overbeck et al. [ODR09] also use the wavelet representation to *adaptively distribute samples*. The idea of adaptive sampling is to adjust the sampling density dynamically, allocating more samples to "difficult pixels" to speed up convergence. Overbeck et al. perform adaptive sampling iteratively by setting a *sample budget* for each rendered frame. First, each pixel is sampled with 4 samples. The output, represented in the wavelet basis, is then used to calculate priority values for all the wavelets, which are based on the variances of the wavelet coefficients.

In each iteration, we pick the wavelet with the highest priority and send samples to the pixels corresponding to the wavelet. This could for example be a large wavelet in a continuous but noisy area, or a small wavelet at a noisy edge. The wavelet coefficients are then updated, resulting in new priorities. More samples are distributed to the highest priority wavelet. This continues until the sample budget runs out.

The general idea of Overbeck et al.'s adaptive sampling (see Figure 3.2: start with uniform sampling, then iteratively allocate more samples until the sampling budget runs out) has been adopted by several later approaches. We will discuss several of these in the next subsection.

3.5 Filter Selection and Adaptive Sampling using Error Estimation

We will now investigate a collection of denoising approaches that all follow the same blueprint. Renders are filtered using a screen-space filtering, roughly following Equation 2.6, 2.7 or 2.8. These filters have different bandwidth parameters which influence the filtering process. The following approaches do not choose one set of parameter values,

but create a filterbank, which contains multiple filters with each different parameter values. Using an error estimate, the filter in the filterbank which will likely produce the result with the lowest error is selected on a per-pixel basis. This error estimate can then also be used to perform adaptive sampling in a similar way as was done by Overbeck et al. [ODR09].

3.5.1 Greedy Error Minimization (GEM)

Rousselle et al. [RKZ11] follow this blueprint with their Greedy Error Minimization (GEM) approach. The technique makes use of a bank of Gaussian filters, with each a different kernel size. They first filter each pixel with the finest filter, and then estimate per-pixel whether using the next filter in the filterbank, with a larger kernel, would reduce the error. Each pixel greedily chooses larger and larger kernels, until the error estimate indicates an increase of error. Now, pixels in areas with high-frequency detail will use small kernels, while pixels in noisy areas which lack high-frequency detail will use large kernels.

Rousselle et al. filter the filter selection maps with a Gaussian filter to prevent noise appearing due to a noisy filter selection. Rousselle et al. also use their error estimation technique to apply adaptive sampling in the same way as Overbeck et al. [ODR09], allocating samples to the pixels which are estimated to have the highest potential improvement. Rousselle et al. show that their approach produces consistently better results than Adaptive Wavelet Rendering [ODR09].

3.5.2 SURE-Based Filtering

The reason that GEM [RKZ11] uses Gaussian filters, is that the used error estimation approach only works for symmetric filters. Li et al. [LWC12] replaced this error estimate with Stein's Unbiased Risk Estimator (SURE) [Ste81], a technique from statistics that is able to estimate the MSE (mean-squared error) of a wide variety of estimators in an unbiased manner. This allows them to instead use a filterbank of cross-bilateral filters, which also factor the similarity of illumination and auxiliary features into the weight calculation. To prevent a noisy filter selection, the MSE estimates are filtered with a cross-bilateral filter. Apart from these differences, the approach is very similar to GEM: filters in the filterbank vary purely in spatial support and iterative adaptive sampling is done using the error estimate. The usage of cross-bilateral filters leads to the SURE-based filter producing better results than GEM.

As discussed in Section 2.4.1, geometric features can become noisy when stochastic primary ray effects such as depth of field and motion blur are present. To deal with the noise in these features, Li et al. use the local sample variances of the features to diminish their weight. For example, when strong depth of field is present, the depth feature might normally stop adjacent pixels from blending together while they safely

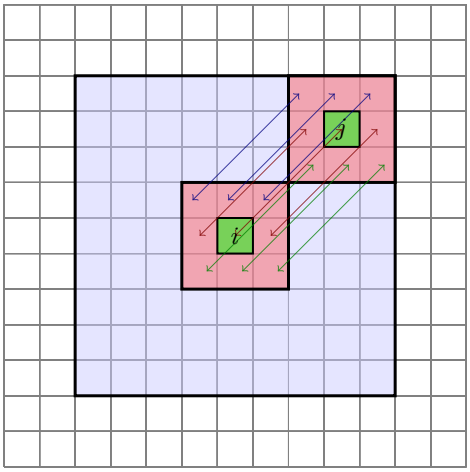


FIGURE 3.3: When using the NL-means filter, a pixel i is blended with the pixels in the filtering window (the blue square). The color distance of a pixel i and a nearby pixel j depends on how similar in color the patches around the pixels (the red squares) are.

could be. Because of its high local variance, Li et al. diminish the weight of the depth feature, allowing for the pixels to be blended.

3.5.3 NL-Means Filtering

An effective technique that does not use geometric information was proposed by Rousselle et al. [RKZ12]. They employ the non-local means (NL-means) filter, which is a generalized bilateral filter. The bilateral filter determines the color weight of two pixels using the colors of the pixels themselves. The NL-means filter instead determines color weights using neighborhoods, or, patches, centered on the two pixels (see Figure 3.3). Because of this, the filter is more robust against noise, but also more expensive.

Rousselle et al. apply the NL-means filter to denoise the Monte Carlo renders. The NL-means filter uses sample variance estimates to account for the overestimation of color distances when the colors are noisy. The original NL-means filter [BCM05] assumes uniform noise levels, but that is not the case here. So instead, Rousselle et al. estimate variance per pixel by splitting the samples into two buffers and analyzing the differences between the buffers. The two separately filtered buffers are also used for an error estimate which drives iterative adaptive sampling.

3.5.4 Robust Denoising using Features and Color (RDFC)

Rousselle et al. [RMZ13] later built upon the work by Li et al. [LWC12]. They also use SURE to locally select the best filter, but instead of using filters with varying spatial support, they use three carefully designed filters with varying sensitivity to color differences. The filter weights used are the minimum of a feature weight and a color

weight. One filter exclusively uses a feature weight and is thus very robust against noise in the colors, while another filter uses a feature weight and a color weight, being very sensitive to noise in the colors. The third filter lies somewhere in between; the used color weight is determined using NL-means weighting, which is more robust against noise than using the colors of two pixels.

Rousselle et al. handle noisy features by applying NL-means filters to the feature buffers. Similar to Li et al. [LWC12], the filter selection maps are filtered to prevent noisy filter selection and influence of noisy features is diminished. The usage of iterative adaptive sampling using the error estimate is again supported. RDFC outperforms both GEM and SURE-based filtering, but because it uses computationally expensive NL-means filters, its computational costs are higher.

3.5.5 General and Robust Error Estimation and Reconstruction

Bauszat et al. [BEEM15] proposed a technique that is able to estimate which filter of a selection of arbitrary filters performs the best. This means that a wide variety of different filter types can be combined. The technique works by assigning several pixels as *filter caches*. For these pixels, a larger amount of samples is taken than for the other pixels. These extra samples are used as an approximation of the ground truth of a filter cache. By applying the different filters on the filter caches and comparing the results to the approximation of the ground truth, error estimates for the different filters at these filter caches are obtained.

To select the best filter for each pixel, the error estimates are extended to all other pixels using an inpainting method based on Delaunay triangulation. However, applying the per-pixel best filter would lead to visible seams in the image. Instead, a multi-labeling optimization problem is solved using graph-cut optimization, giving smoother results. Bauszat et al. show that their technique outperforms the SURE-based filter. RDFC is outperformed for some renders, while for others RDFC performs better.

3.6 Random Parameter Filtering

Consider once again the usage of auxiliary features to steer a filter. When stochastic primary ray effects such as depth of field or motion blur are present, these auxiliary features can contain noise. We have already seen some possible solutions for this: weights of noisy features can be diminished and the feature values can be filtered.

Sen and Darabi [SD12] introduced an alternative technique to deal with the noise in these features, known as Random Parameter Filtering (RPF). The general idea of RPF is to measure how reliant the features (such as world position, depth etc.) are on the random parameters. They do this by calculating the mutual information of the values. When features have a high mutual information with the random parameters, this indicates

that they "got corrupted" by the random parameters and are unreliable for steering the filter. RPF diminishes the influence of such features. Filtering is done using multiple cross-bilateral filter iterations with a decreasing window size. The used bandwidths are based on user-set constants and the calculated mutual information of the features.

RPF is able to denoise effectively, even with low sample counts, but because of its multiple filter iterations the results tend to get overblurred. There is no clear error estimation to drive adaptive sampling. Mutual information is computed at the *sample level*, which costs large amounts of memory and is computationally expensive. Park et al. [PMKY13] later modified the approach to have a performance independent of the sample count. At low sample counts, RPF may outperform approaches such as the SURE-based filter, since it is quite robust against feature noise which is prevalent at low sample counts. At higher sample counts the results are generally worse due to its lack of an error estimate.

3.7 Filtering without Geometric Information

Instead of attempting to deal with noise in the auxiliary features, some approaches use alternate methods to prevent bias being introduced from dissimilar pixels blending together. These approaches are generally consider to be less effective than the ones that use features, especially at low sample counts, but are interesting nevertheless.

3.7.1 Ray Histogram Fusion

Delbracio et al. [DMB*14] observe that to identify which pixels can be combined without introducing large amounts of bias, a good approach is to look at the distribution of their samples. To prevent having to store all the gathered samples, their approach, known as Ray Histogram Fusion (RHF), stores a histogram for every pixel. They then apply the NL-means filter, similar to the approach by Rousselle et al. [RKZ12]. However, they do not use it with the color distances of the pixels in the patches, but with the χ^2 distances of the histograms of the pixels in the patches. They apply the filter on multiple scales to deal with low frequency noise.

Because no geometric information is used, even scenes with multiple simultaneous effects (e.g. global illumination and depth of field) can be denoised. However, because the decision of sharing samples between two pixels is purely based on the color histogram, quite some samples are required for the histograms to become reliable. This makes the technique less suitable for low sample counts than for example Random Parameter Filtering [SD12].

3.7.2 General Image Denoising (GID)

Kalantari and Sen [KS13] observed that most path tracing denoising techniques at the time used relatively simple filters, while in the image denoising field many newer, more

powerful filters exist. They offer an explanation for this: many of these newer techniques assume spatially-invariant noise levels, which is not the case for Monte Carlo renders. They introduce General Image Denoising (GID), which enables the usage of denoising techniques that assume spatially-invariant noise for path tracing denoising.

GID employs the *median absolute deviation* (MAD) [DJ94] to estimate the variance for every pixel. MAD uses the coefficients of the finest level of a wavelet transform to estimate the variance, since these coefficients typically only represent noise. Once the noise levels in the image have been estimated, a cumulative distribution function is used to generate a set of representative noise levels. The image is then denoised multiple times using the image denoising method, once for every noise level. Each pixel in the final image is then constructed by alpha-blending the denoised values of the two denoised images with closest noise levels.

Kalantari and Sen also use MAD, combined with the contrast metric by Hachisuka et al. [HJW*08] to perform adaptive sampling. By employing the BM3D denoising method [DFKE06], they demonstrate improvements over Random Parameter Filtering [SD12], Greedy Error Minimization [RKZ11] and Adaptive Wavelet Rendering [ODR09]. Unfortunately, because the noise estimate is purely based on color, noisy textures tend to be blurred by the algorithm.

3.8 Real-time Denoising with the Edge-Avoiding À-Trous Wavelet Transform

All of the approaches that we have discussed so far are designed to be used in an offline rendering context, running in the order of seconds or minutes. One of the first approaches targeted for real-time performance is the method by Dammertz et al. [DSHL10], who employ the fast undecimated À-Trous wavelet transform to denoise Monte Carlo renders effectively in a short amount of time.

Instead of calculating the decimated wavelet transform with Mallat's algorithm (see Section 2.4.3), the *undecimated* wavelet transform is calculated using the *algorithme à-trous*. This algorithm produces wavelet coefficients by performing multiple filtering iterations. In each iteration, one level of the wavelet coefficients is calculated by convoluting the input with a kernel, smoothing the image. The difference between the image before and after the filtering form the coefficients of the current level. After each iteration, the kernel size is doubled. The key is that, instead of eventually having expensive kernels covering lots of pixels, the amount of non-zero entries in the kernel is the same in every iteration. The kernels are increased in size by adding more zeroes in between the non-zero entries. In iteration i, the distance between the non-zero entries is 2^i (see Figure 3.4).

Dammertz et al. introduce some important changes and extensions to apply the transform for Monte Carlo denoising. First of all, they let the filtering weights not only

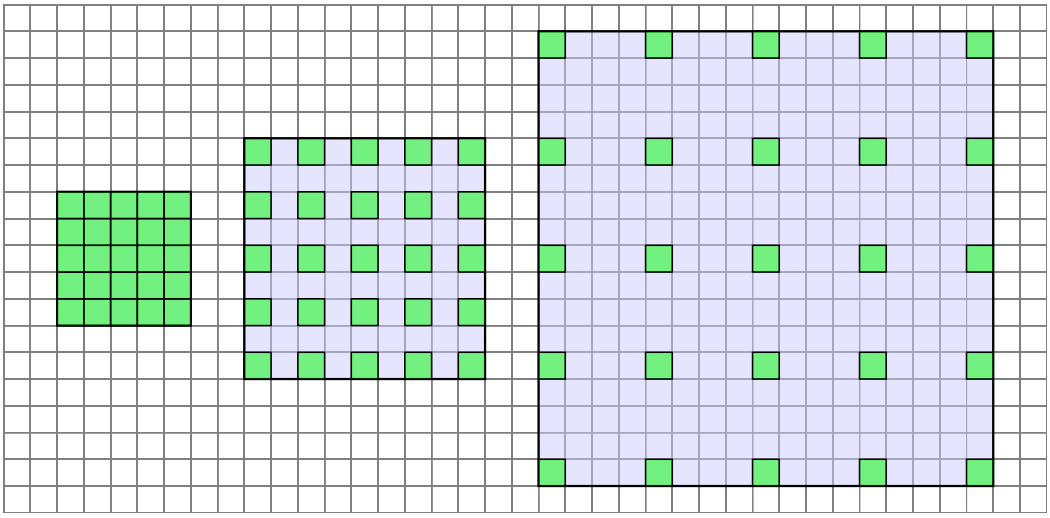


FIGURE 3.4: Three kernel levels used in the À-Trous wavelet transform. In each iteration, a larger kernel is used, but the amount of non-zero entries (shown in green) remains the same. The distance between the non-zero entries is 2^i. Here, we show the kernels at $i = 0$, $i = 1$ and $i = 2$.

depend on the spatial position in the kernel, but also on auxiliary features distances. These weights are known as edge-stopping functions. Note that this is just like what happens in the joint-bilateral filter. Three edge-stopping functions are used: one for the normal, one for the world position and one for the color.

But one crucial step remains: the wavelet coefficients produced by the transform have to be turned into a smoothed image. Instead of shrinking wavelet coefficients and transforming the representation back to an image, the filtered image at the last level of the transform is simply used as the output. This means that in practice, the approach comes down to applying a joint-bilateral filter repeatedly, with more and more zeroes between the coefficients of the kernel. The approach approximates a bilateral filter with a large kernel, which would be expensive to compute.

Unlike some of the more complex techniques discussed previously, the same filter is applied at every pixel. The bandwidth parameters for the normal and world position are fixed and user-set, and the color bandwidth is divided by two in every iteration to allow for smoothing smaller variations in illumination. To prevent textures from getting blurred out, the filter can be applied to the incident illumination at each pixel, instead of to the final shaded image. After the filtering, the material evaluation is performed for every pixel, yielding the final image. This can however only be done for diffuse surfaces.

Dammertz et al. are able to produce filtered images quickly. However, their algorithm assumes that the feature buffers are noise-free, making it incompatible with stochastic primary ray effects such as motion blur and depth of field. The approach is significantly faster than many of the other techniques, making it the first approach that is actually

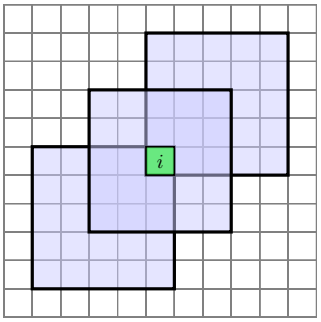


FIGURE 3.5: When applying the guided image filter, pixel i is a part of many different windows. For each window a linear regression model is fitted, approximating the pixels inside of it. The filtered value of pixel i is the average of its approximations in all windows it is a part of. Three such windows are drawn in this figure.

suitable for denoising Monte Carlo renders in real-time. The approach makes no attempt to make subsequent frames coherent with one another, which means that flickering can occur. This was addressed later in the work by Schied et al. [SKW*17], who extended the approach.

3.9 Real-time Denoising with the Guided Image Filter

An alternative denoising approach that targets real-time framerates was introduced by Bauszat et al. [BEM11]. This approach does not use a bilateral filter or an approximation of it, but instead applies the guided image filter [HST10]. The guided image filter assumes that the filtered image is a local linear transformation of the geometric information:

$$q_i = a_k I_i + b_k, \forall i \in \omega_k. \tag{3.1}$$

Here, for a pixel i in a local window ω_k, the filtered pixel q_i is a linear transformation of the corresponding geometric information I_i. For the window ω_k, a_k and b_k are calculated using linear regression, so that the linear transformation approximates the noisy image in the window ω_k. A window is placed around every pixel in the image and a linear regression model is fitted for each window. Each pixel now has an approximation q_i in multiple windows (all the windows the pixel is a part of, see Figure 3.5). The final filtered image is created by averaging all approximations for each pixel.

The guided image filter smooths the input, because the linear regression models are linear transformations of the input (the geometric features). Thus, noise in the render, which is not present in the geometric features, disappears in the fitted linear regression models. The guided filter has some important advantages over the bilateral filter. Its runtime is independent of kernel size and intensity range, and it does not exhibit the

ringing and gradient reversal artifacts of the bilateral filter. It is also faster than high-dimensional bilateral filters (although high-dimensional bilateral filters can be sped up significantly, for example with adaptive manifolds [GO12]). Similar to the approach by Dammertz et al. [DSHL10], the feature buffers are assumed to be noise-free, making the approach incompatible with stochastic primary ray effects.

Bauszat et al. [BEM11] split indirect illumination and direct illumination, and then use the guided image filter to filter the noisy indirect illumination. They note that supersampling (using multiple samples per pixel, spreading them over the pixel surface to obtain anti-aliased images) is not trivial to combine with the filter. This is because the different samples might have significantly different hit points for a single pixel, and thus different geometric information. Supersampling introduces noise in the geometric features, which the filter does not support. To support supersampling they send one *guide ray* for each pixel, which goes through the center of the pixel. The results of these guide rays, along with their geometric information, are filtered with the guided filter. The filtered values of the other samples are determined by selecting the filtered value of the adjacent guide ray with the most similar geometric features.

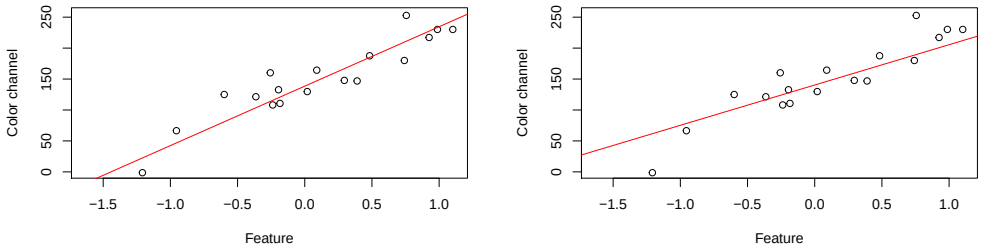
3.10 Local Regression-Based Filtering

The discussed guided image filter is smoothing renders by locally fitting linear regression models that map geometric feature space to a color channel. The guided image filter uses *first-order regression models*. The priorly discussed bilateral filter and NL-means filter can be seen as *zero-order regression models*: they do not approximate spatial windows with a first-order function of the input, but with a constant. We will now discuss several approaches that employ first-order and higher-order regression models.

3.10.1 Employing Weighted Local Regression

Moon et al. [MCY14] proposed a more complex regression-based filtering technique than the guided image filter. Instead of applying unweighted regression models, they apply *weighted* regression models, which prioritize fitting to pixels which have feature values similar to the center pixel (see Figure 3.6 for a 1D example). This leads to better results. A pixel is no longer based on its approximation in all windows containing the pixel, but exclusively on the regression model of the window centered on it.

The filter is able to preserve texture detail by including the texture value as one of the input features. To deal with noisy features, Moon et al. perform a Truncated Singular Value Decomposition (TSVD). By basing the singular value threshold (which determines which singular values are set to zero) on perturbation theory, they remove dimensions that are likely a result of noise. Now, the set of used features is less noisy, making the approach compatible with stochastic primary ray effects. In addition to this, the



(a) (b)

FIGURE 3.6: Two linear regression models. A linear function of the feature value is created (shown in red), which is fit as closely as possible to the noisy color channel. In these plotted models, only one feature was used, but in the filtering approaches, multiple features are used. (a) is an unweighted linear regression (used in the guided image filter) and (b) is a weighted linear regression. The weighted linear regression model prioritizes fitting to pixels with more similar feature values to the center pixel (which has feature value 0 here).

importance of different features in the fitting of the weighted regression model is scaled using an error estimate.

Moon et al. also performed an analysis to predict how the relative MSE (rMSE) [RKZ11] of a pixel would decrease by giving it one more sample, which drives iterative adaptive sampling. Moon et al. demonstrate that their approach produces better results than Random Parameter Filtering [SD12], the SURE-based filter [LWC12] and the NL-means filter [RKZ12]. However, unlike the guided image filter approach by Bauszat et al. [BEM11], the approach is unsuitable for real-time framerates.

3.10.2 Adaptive Rendering with Linear Predictions

A similar but faster method was later introduced by Moon et al. [MIGYM15]. This alternative technique does not fit a linear regression model at every pixel, but instead fits a smaller amount of models, estimating multiple pixel values with each model. The models are iteratively placed over the image until every pixel is predicted by at least one model. Each model has an adaptive window size, which is determined using a novel estimate for the prediction error that splits the pixels in a training and test set. Again, adaptive sampling is performed based on the estimated error, and the TSVD introduced by Moon et al. [MCY14] is used to handle noisy features.

The approach is able to produce high quality results in significantly less time than SURE [LWC12], NLM [RKZ12] and LWR [MCY14]. Because the filter takes less time, it is able to take more samples in an equal-time comparison. It also produces better results than real-time filtering methods by Dammertz et al. [DSHL10] and Bauszat et al. [BEM11], but is not suitable for interactive framerates.

3.10.3 Fitting Polynomials

Later, Moon et al. [MMMG16] proposed another regression-based technique. The approach uses higher-order regression models: polynomial functions are fitted instead of linear functions. Polynomial functions can adapt more effectively to the data: if we increase the order of the polynomial enough, the polynomial will fit exactly to our data and not perform any smoothing. Moon et al. propose a method to locally estimate the optimal order of the polynomial, which reduces variance, without introducing large amounts of bias. Similar to Moon et al. [MIGYM15], they let one model estimate multiple pixels in the window, but they do not dynamically adjust the window sizes. To deal with noise in the features, features are pre-filtered using a similar technique that uses polynomial functions. Adaptive sampling is applied in the same iterative manner as other discussed techniques.

Moon et al. achieve consistently better results than several earlier approaches ([MCY14], [MIGYM15], [KBS15]) with the same amount of input samples. What is remarkable is that no dynamic bandwidth selection is performed at all: only the order of the polynomials is dynamically selected. Unfortunately, the approach is still slow and not suitable for interactive framerates.

3.10.4 Nonlinearly Weighted First-Order Regression

Bitterli et al. [BRM*16] find that in practice, zero-order models, such as the RDFC technique by Rousselle et al. [RMZ13], often outperform first-order regression approaches such as locally weighted regression (LWR) [MCY14]. They give several explanations. The TSVD is effectively discarding noisy features, while RDFC denoises them and still obtains useful information from them. RDFC uses the noisy color estimates to steer filtering, which are ignored by LWR. Finally, the selection from three carefully chosen filters turns out to be more stable in practice than intricate spatial per-pixel estimation.

Bitterli et al. use these insights to design a hybrid approach. A first-order regression model is employed, but instead of using a TSVD, features are pre-filtered using an NL-means filter. To prevent residual noise artifacts in the feature buffers, samples are split into two feature buffers and filtered using weights derived from the other buffer.

For each of the two color buffers, a first order regression models is fitted. Again, to reduce artifacts, features from one buffer are used to fit the regression models for the other buffer, decorrelating feature noise from color noise. The regressions are weighted, with only the pixel color playing a role in this weight. NL-means weighting is used for the color, making the approach more robust against noise in the color data. Each fitted model (one model for each pixel, with a window around it) is used to predict color values for all pixels in the window. Unlike the approach by Bauszat et al. [BEM11], the estimates for a pixel are combined using a *weighted* average, reusing the regression

weights. After filtering the two buffers, they are combined and filtered one last time using a first-order NL-means filter. No adaptive sampling is performed.

The proposed approach improves consistently upon both existing first-order and zero-order approaches when provided with the same amount of samples. However, the approach takes significantly longer to run (in the order of minutes) and requires more memory than previous approaches.

3.11 The Learning-Based Filter

Kalantari et al. [KBS15] state that the most successful filtering techniques for path tracing use feature-based filters. These use a spatial bandwidth, a range bandwidth and feature bandwidths. In Equation 2.6, 2.7 and 2.8 these are σ_d, σ_r and σ_k, respectively. Kalantari et al. [KBS15] introduce the learning-based filter (LBF), which attempts to choose effective values for these parameters automatically. They select parameters for a large cross-bilateral filter, but show that the approach also works with other filters, such as the NL-means filter.

Kalantari et al. train a multilayer perceptron neural network on a wide variety of scenes (aiming to minimize the relative mean squared error [RMZ13]). The neural network learns to predict effective feature bandwidths using a set of secondary feature values for the pixel. The secondary feature values are derived from primary feature values, which are the geometric features used to steer the filter. Examples of secondary feature values are the mean, standard deviation and median absolute deviation (MAD) of the features. The training happens offline, while the trained neural network is then used online to determine the parameters to use while filtering.

The range bandwidth, or color bandwidth, is not predicted but fixed to prevent overfitting. The primary features are filtered using an NL-means filter before the secondary features are derived from them. Now, effective bandwidth parameter values can be predicted automatically. The approach is effective, outperforming NLM [RKZ12], SURE-based filtering [LWC12], LWR [MCY14] and RDFC [RMZ13]. Unfortunately, it is too slow for interactive framerates.

3.12 Filtering Depth of Field and Hemisphere Noise Separately

Bauszat et al. [BEJM15] observe that several techniques exist that can produce good results, even when multiple effects are present (e.g. global illumination and depth of field). However, none of these techniques is suitable for interactive settings. Bauszat et al. propose a new technique, specifically designed for denoising Monte Carlo renders in presence of both global illumination and depth of field in real-time.

Bauszat et al. state that two types of noise are present in these renders: noise created by undersampling the hemispheres of the sample points and noise created by undersampling of the virtual lens. They filter both these types of noise out separately. First, they filter the *samples* to recover the global illumination for each sample. Samples are mixed with adjacent samples with similar color and geometric features. Employing a cross-bilateral filter to filter on the sample-level would be rather expensive. Instead, a modified version of the adaptive manifolds approach introduced by Gastal and Oliveira [GO12] is employed, which approximates the cross-bilateral filter but is faster.

The adaptive manifolds approach is efficient, because it avoids filtering in higher dimensional space. The approach creates a collection of manifolds in this higher dimensional space, which adapt to the signal. The manifolds are still discretized using the resolution of the image. The colors of the pixels are *splatted* onto these manifolds using a Gaussian falloff in the higher dimensional space. The manifolds are then filtered with a filter that takes the curvature of the manifold into account. The filtered color for each sample is then obtained using a *slicing* step, which blends the colors on the filtered manifolds using the same weights as used in the splatting step.

The adaptive manifolds approach allows for efficient approximation of high dimensional filters, because its total cost is both linear in the amount of pixels and the amount of dimensions. The second step Bauszat et al. perform is a modified version of the sweep-blur algorithm to filter out the noise from undersampling of the virtual lens. The sweep-blur algorithm was introduced by Shirley et al. [SAC*11] and is able to reconstruct stochastic motion blur and depth of field effects from small amounts of samples. Bauszat et al. speed up the algorithm by employing linear manifolds.

By combining adaptive manifolds and sweep-blur, Bauszat et al. are able to denoise Monte Carlo renders with global illumination and depth of field in real-time. The approach essentially blurs the out-of-focus samples twice, which leads to slight overblurring. This means that, at 4 samples per pixel, the approach leads to slightly higher MSE values than RDFC [RMZ13] and SURE-based filtering [LWC12] (see Section 3.5). However, the results from these two offline denoising methods contain visually disturbing outliers that are not present in the results by Bauszat et al. No effort is done to ensure temporal coherence between consecutive frames, which means that flickering between consecutive frames can occur.

3.13 The Virtual Flash Image

The usage of geometric information seems to have pros and cons: they can drastically improve results when the amount of samples is small, but noise in these features can be introduced by stochastic primary ray effects such as depth of field, and not all effects (such as caustics) can be represented by the geometric information. Moon et al.

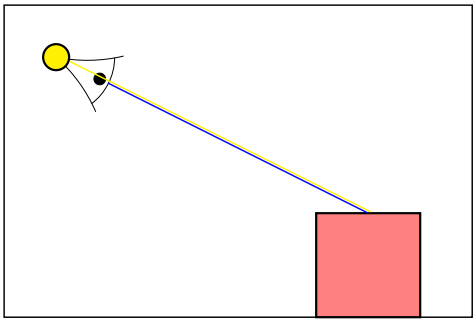


FIGURE 3.7: The virtual flash image is rendered by reusing the direct illumination paths used for the render and adding an additional "virtual flash light source" at the camera position. No rays are traced for this extra light to avoid expensive raycasts and the introduction of shadows not present in the render.

[MJL*13] propose an interesting middle ground. They do not use geometric features directly to steer their filtering, but render a *virtual flash image*.

The virtual flash image is rendered by taking the direct illumination and adding a "flash" point light at the camera to light the areas that are not lit by direct illumination (see Figure 3.7). This absence of indirect illumination makes the virtual flash image less noisy than the actual image, which makes it suitable for steering an NL-means filter. The advantage of the virtual flash image is that it contains features not represented in the geometric features, such as refractions. The image can be rendered relatively cheaply: no visibility tests are performed for the virtual flash light to avoid introducing shadows that do not exist in the render, so no additional ray samples are needed.

Moon et al. note that most existing methods require manual tweaking to find a window size that denoises effectively but does not blur excessively. In their work, they do not adjust the window size to prevent overblurring, but only blend with the *homogeneous pixels* in a large window. The homogeneous pixels of a pixel p are the pixels that are statistically equivalent to p. They identify these by calculating the confidence interval of p and comparing the sample means of other pixels to this interval, essentially performing a t-test. Moon et al. demonstrate improvements over the work by McCool [McC99], Xu and Pattanaik [XP05], Overbeck et al. [ODR09] and Dammertz et al. [DSHL10]. Unfortunately, the approach does not work well with low sample counts, making it unsuitable for real-time filtering.

3.14 Alternatives to Filtering Pixel Colors

Most approaches discussed so far apply filters directly to the renders in image space, filtering the current pixel color estimates. This is an efficient and straightforward way of filtering. However, approaches that take a different approach can have large advantages. An example is the in Section 3.12 discussed method by Bauszat et al. [BEJM15], which

employs adaptive manifolds to filter on the sample level instead. More approaches exist that do not filter pixel colors directly. Often, these approaches are too complex and thus too slow for real-time usage. We will briefly discuss some of them.

3.14.1 Radiance Filtering

In Schwenk's thesis [Sch13], the *radiance filtering* approach is discussed. This technique does not filter the pixel colors or irradiance values, but instead filters the incident radiance of the first non-specular hit. The filter works as follows. First, a path tracing pass is performed. For each pixel, the incident radiance of the first non-specular hit is stored. This stored information is known as a radiance sample. Then, to filter a pixel, stored radiance samples of neighboring pixels are retrieved. The radiance sample of the current pixel is filtered with the neighboring radiance samples *in world-space*, using a projection of the kernel in world space. The radiance samples together lead to a filtered estimate of the exitant radiance for our current radiance sample.

3.14.2 Path Space Filtering

It is also possible to go even further and filter in path space instead of in image space. Keller et al. [KDB16] introduce such a method. A path is filtered by taking a 3D sphere around the first sufficiently diffuse hit and filtering with other path vertices in the sphere. These other path vertices are found by performing a search in world space (using a hash grid, bvh, kd-tree or divide-and-conquer method).

To prevent blurring across geometry, the contribution of found paths is weighted using similarity of normals. Contributions are only included when visibility of point light sources is the same, in order to prevent blurred hard shadow boundaries. To reduce texture blur, the contribution of a neighboring sample is evaluated with the current vertex's surface properties. In cases where that is considered to be too costly, one can also only consider samples with similar contributions and, if the surface is specular, similar observation directions.

Unfortunately, path space filtering has to perform expensive searches in world space and requires deep integration with the rendering process itself. Often, a standalone, decoupled filter is desirable.

3.14.3 Multidimensional Adaptive Sampling and Reconstruction

The Multidimensional Adaptive Sampling and Reconstruction approach by Hachisuka et al. [HJW*08] operates in the high-dimensional sample space instead of in image space. For example, in a scene with depth of field and motion blur this is a 5D space. The algorithm starts with a coarse sampling of this space, storing samples in a kd-tree data structure. Adaptive sampling is then performed by iteratively sending more

samples to the kd-tree leaf node with the highest sample variance, subdividing the leaf node if necessary. A pixel color can be estimated using an anisotropic nearest neighbor reconstruction technique.

Unfortunately, the usage of kd-trees makes the approach scale poorly to higher dimensions. The approach seems similar to many a priori methods (see Section 3.15), which mostly also operate in higher dimensional space. However, it is more general, making it applicable to arbitrary combinations of effects. This generality comes at a price: the a priori methods that focus on specific effects tend to produce better results.

3.15 A Priori Methods

In the survey on adaptive sampling and reconstruction for Monte Carlo rendering by Zwicker et al. [ZJL*15], a distinction is made between *a priori* and *a posteriori* methods. A priori methods perform an analysis of the light transport equations and use this to perform adaptive sampling and reconstruction, while a posteriori methods instead analyze samples generated by the renderer to do this. A priori methods tend to require deep integration with the renderer, while a posteriori methods can easily be added to existing renderers. Furthermore, a priori methods are often restricted to a small selection of effects to keep the analysis tractable, while a posteriori methods can often be applied to arbitrary combinations of effects. For these reasons, we focus on a posteriori methods in this work. The discussed approaches so far have all been a posteriori. For completeness, we will now briefly discuss some a priori methods. For a more complete discussion, we refer to the survey by Zwicker et al. [ZJL*15].

An interesting line of work was started by Durand et al. [DHS*05]. In their work, they perform a frequency analysis of light transport. They look at the spectra of radiance present in the scene, and at how these change by phenomena such as light transport, occlusion and reflection. Durand et al. show how such an analysis can be useful for filtering and adaptive sampling. By analyzing the light transport in a simple scene, they estimate local bandwidths of the image signal. They then use these estimates to scale the spatial bandwidth of a bilateral filter and distribute samples effectively.

The approach was later extended to a variety of effects. Egan et al. [ETH*09] employed spatio-temporal sheared filters to filter motion blur. Later work [EHDR11, EDR11] extended the approach to more effects, such as soft shadows. Unfortunately, the sheared filters employed in these works tend to be expensive. As an alternative, the usage of a cheaper axis-aligned filter was proposed by Mehta et al. [MWR12], who focused on the rendering of soft shadows. This approach has been extended with more effects, with a later iteration [MYRD14] supporting depth of field, soft shadows and indirect illumination. While axis-aligned filtering is faster than sheared filtering, it does require more samples. Yan et al. [YMRD15] later proposed a faster sheared filter, allowing

them to achieve impressive results with smaller amounts of samples than required for axis-aligned filters in a comparable amount of time.

Other a priori methods exist which do not perform a frequency analysis of light transport. Lehtinen et al. [LAC*11] efficiently render images with motion blur, depth of field and soft shadows using low sample counts by reconstructing the temporal light field from a sparse set of samples. The approach has later been extended to support diffuse indirect illumination [LALD12].

3.16 Recent Work

In the last few years, the rise of GPU-based path tracing has made real-time path tracing a reality. However, at interactive frame rates, the amount of samples for each pixel per frame is still small. Currently, taking the large amount of samples required to generate converged images in real-time seems unrealistic. Effective, fast filtering of renders with small amounts of samples is required to render noise-free images with path tracing in real-time.

So far, we have only discussed three approaches that target these low sample counts and interactive framerates: the À-Trous wavelet filter by Dammertz et al. [DSHL10], the Guided Image Filter approach by Bauszat et al. [BEM11] and the approach by Bauszat et al. [BEJM15] that filters depth of field and hemisphere noise separately. We will now take a look at some more recent work that has focused on this area.

3.16.1 Spatiotemporal Variance-Guided Filtering (SVGF)

The recent work by Schied et al. [SKW*17] introduces Spatiotemporal Variance-Guided Filtering. This work is closely related to our own work: we implement SVGF and introduce a number of extensions and modifications.

Schied et al. build upon the the fast edge-avoiding À-Trous wavelet filter by Dammertz et al. [DSHL10]. Just like Dammertz et al., they steer their filter using geometric features which they assume to be noise-free. This means that stochastic primary ray effects such as depth of field are unsupported. Schied et al. assume that in the near future, only 1 sample per pixel (1 spp) will be practical at desired real-time framerates. Thus, they design their filter with this constraint in mind. See Figure 3.8 for an overview of the working of the algorithm.

The filter is applied to the direct and indirect illumination separately. Instead of filtering the pixel colors directly, Schied et al. first demodulate the surface albedo of directly visible surfaces. This means that they will filter the untextured illumination components. Now, the filter will not blur texture detail and there is more possible spatial reuse for neighboring samples. After the filtering step, indirect and direct illumination are combined and texturing is reapplied to create the final image.

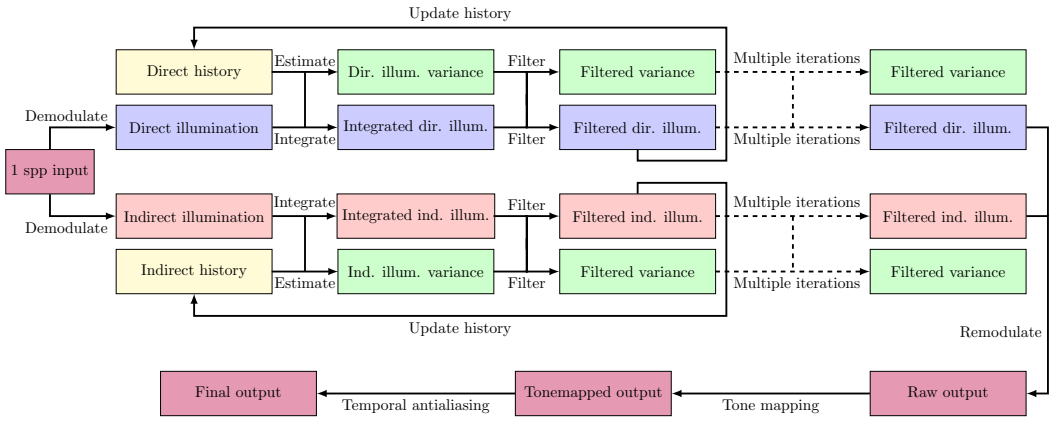


FIGURE 3.8: An overview of the SVGF approach [SKW*17]. The 1 spp input is demodulated, with direct and indirect illumination being filtered separately. Variance is estimated using history when history is available (otherwise a spatial estimate is used), which controls the illumination edge stopping. After several filtering iterations, illumination is combined, remodulated and tone mapped. Finally, a temporal antialiasing step is performed.

The filter first applies a *temporal filtering* step. This step is inspired by the popular temporal antialiasing technique (TAA) [Kar14]. Schied's filter stores a history buffer, which contains a value for each pixel. By performing reprojection (based on motion vectors), the history buffer is examined to obtain the corresponding color for the current pixel in the previous frame. This effectively increases the amount of samples used. By updating the history buffer in each frame, blending in our current samples using an exponential moving average, samples are accumulated over time, even when the viewpoint changes or objects in the scene move. It also drastically increases the temporal coherence between frames, reducing flickering. TAA uses color information to determine if reprojection was successful. This is not suitable for our noisy renders with 1 sample per pixel, so instead, Schied et al. use a set of consistency tests based on the geometric features. To improve reprojection performance, a bilinear filter is used to sample the history buffer.

Schied et al. use different edge stopping functions (which serve to steer the filter using geometric features) than Dammertz et al. The first one uses the depth difference, scaled using the screen-space clip-space depth derivative to account for different local depth models. The second edge-stopping function is a simple dot-product of normals. The final one is based on the difference in luminance (and is thus similar to the range distance in a bilateral filter). The luminance difference is scaled using a local estimate of the standard deviation of the pixel's luminance. The bandwidth parameters are set by the user: Schied et al. give bandwidth parameter values which they found worked well on all tested scenes.

The local pixel variance is estimated by accumulating the first and second raw moments of color luminance. However, when disocclusions occur, reprojection fails. The variance

cannot be estimated with only one sample. So, when there is only a small amount of history, variance is instead estimated using a 7×7 bilateral filter steered by depths and normals. The variance estimate can be rather noisy, so it is smoothed with a 3×3 Gaussian filter.

To produce the final render, filtered indirect and direct illumination are combined and remodulated with surface albedo. Tone mapping is performed and finally, temporal antialiasing is applied to increase temporal stability and decrease aliasing. The filter is fast, making filtering at interactive framerates possible.

The temporal accumulation assumes that a position in world-space will have similar illumination in the previous frames as in the current frame. When this is not the case, the temporal accumulation introduces bias, resulting in ghosting artifacts. The motion vectors and geometric features are generated with a rasterization pass, which means that the motion vectors and geometric features describe the motion and properties of a path's primary hit. This leads to ghosting artifacts for geometry visible in specular reflections, something which we address in our work.

3.16.2 Separate Filtering Chains for Matte and Glossy Rays

The approach by Mara et al. [MMBJ17] seems similar to the approach by Schied et al. at first glance. It also targets real-time performance, works with only one sample per pixel and performs temporal filtering. Mara et al. separate illumination into direct and indirect illumination and only filter the indirect illumination. The indirect illumination is further separated: by assuming that the BRDF is divided into matte and glossy terms, the estimator of the pixel color is divided into a matte and a glossy part. An approximation of the two estimators is used, factoring out several material-dependent factors. This is similar to what Schied et al. do: untextured illumination components are filtered.

Mara et al. have now separated the estimator into two: a matte estimator and a glossy estimator. These estimators are filtered using separate filtering chains. The matte estimator is first temporally filtered, blending in history dependent on a confidence value. The result is filtered using a sparse cross-bilateral filter for efficiency reasons and blended back into the stored historical buffer. However, that is not the image shown to the user. The presented image is first filtered with a median filter to suppress fireflies and then filtered with a sparse bilateral filter, which gets an expanded radius when there was a low confidence for the reprojection. It is worth noting that normal and depth feature values are also used to steer all bilateral filters.

The glossy estimator is filtered differently. The temporal filter is now applied to the *virtual positions* of the reflected objects: if the virtual position of a considered pixel in a previous frame is similar to the current pixel being filtered, it is blended in more significantly. No median filter is applied: the authors state that fireflies generally occur

because of caustics, which appear on matte surfaces, while bright reflected spots on glossy surfaces are mostly valid. No sparse bilateral filter is applied either: because of less effective reprojection for glossy surfaces, artifacts become visible when using this filter. Instead, a bilateral filter that is separated in two 1D passes is applied.

Mara et al. achieve impressive results in small amounts of time. Their method runs in a similar amount of time as the approach by Schied et al., but is slightly slower. Unfortunately, both the approach by Schied et al. and the approach by Mara et al. are unable to accurately generate motion vectors for reflected objects. Both approaches only have one motion vector per pixel, while there could be multiple relevant ones (e.g. for the primary hit and for a reflected object). Zimmer et al. [ZRJ*15] have proposed a solution for this problem where the pixel colors are separated into multiple components (using a classification of paths), with each component receiving separate motion vectors. Specular motion vectors are not trivial to derive; they employ a version of manifold exploration [JM12] to find these. Unfortunately, Schied et al. state that the method requires precomputed frames, and is thus not effective for real-time usage, where only previous frames are known.

3.16.3 Reconstruction using a Recurrent Denoising Autoencoder

Another recent approach that targets interactive framerates and low sample counts is the approach by Chaitanya et al. [CKS*17]. The approach makes use of machine learning, using artificial neural networks, but differs significantly from the earlier discussed Learning-Based Filter by Kalantari et al. [KBS15]. The whole denoising process is performed by a convolutional neural network, while Kalantari et al. only use an artificial neural network to determine certain filter parameters. We will forego an in-depth discussion of convolutional neural networks here: see [CKS*17] and its references for a deeper explanation.

Although the approach by Chaitanya et al. is different from the approaches by Schied et al. [SKW*17] and Mara et al. [MMBJ17], upon closer inspection many similarities become evident. Chaitanya et al. use a recurrent denoising autoencoder. This architecture, which is a convolutional neural network, takes the untextured illumination as its input. Just like in SVGF, the render is demodulated by the albedo of the first hit surface. Besides this image, it also takes a variety of auxiliary inputs: normal, depth and roughness. These are assumed to be noise-free, making stochastic primary ray effects such as depth of field unsupported. Contrary to other approaches which use these features explicitly, Chaitanya et al. simply give these to the convolutional neural network (each pixel gets 7 scalar input values) and trust the convolutional neural network to learn how to use these effectively.

The recurrent denoising autoencoder is a convolutional neural network (CNN) with many different layers. The input is transformed into an internal representation ("encoded")

in the first few layers, which each transform to a lower and lower spatial resolution. That internal representation is then transformed into the denoised result ("decoded") by multiple layers, which transform into a higher and higher spatial resolution. This approach has many layers which operate on lower resolutions, making the network faster than the CNN image restoration by Mao et al. [MSY16]. Skip connections are used between the layers to make training easier. The used CNN contains recurrent connections which connect to the state of the architecture in the previous frame. These inputs allow the CNN to use this information to create temporally stable frames. Again, note the similarities to the approaches by Schied et al. [SKW*17] and Mara et al. [MMBJ17], who use a history buffer to achieve temporal stability.

The CNN is trained using a large amount of frames generated in flythroughs in several scenes. Reference images are generated and for each of them ten different noisy images are generated, which the CNN should all reconstruct as close to the reference image. Different training sequences are created, which consist of an area of 128×128 pixels in 7 consecutive frames. A wide variety of training sequences is used, with different scenes, different parts of the flythroughs, different camera orientations etc. The denoised images during training are evaluated using a combination of loss terms: one is based on the L_1 distances per pixel, another is based on the differences in gradients and the final one is based on the differences in temporal derivatives.

It takes a long time to train the CNN (it took the authors approximately 16 hours on a GPU), but once trained, it runs in significantly less time than the Learning-Based Filter by Kalantari et al. [KBS15]. It is almost 30 times faster in the author's experiments, while producing better results. Interactive framerates become possible, although the performance numbers given seem slower than the approaches by Schied et al. [SKW*17] and Mara et al. [MMBJ17].

3.17 Assessment and Summary

Having discussed a wide variety of related work, we will now provide an assessment of the discussed work. Note that this assessment is not based on any empirical experiments, but purely on the literature itself. See Table 3.1 for a list of the discussed a posteriori methods.

Many effective path tracing denoising approaches exist for a wide variety of use cases. Often, adaptive sampling is performed in combination with such a denoising approach. The adaptive sampling can improve results significantly, but remains only useful in offline rendering settings. The main reasons for this seem to be a lack of samples to adaptively distribute in each frame and the requirement of an accurate error estimation, which is not easy to derive at low sample counts.

The usage of geometric features can make our filter weights more stable against noise, even when noise is present in these features. The Robust Denoising using Feature and

Color Information (RDFC) approach by Rousselle et al. [RMZ13] is one of the most effective methods for offline contexts, combining the NL-means filter with a robust method of varying filter bandwidths dynamically. More recently, methods that employ local first-order regression models have further increased filtering quality. The most effective of these methods seems to be the Nonlinearly Weighted First-Order Regression method by Bitterli et al. [BRM*16], which combines the best aspects of the local regression-based filters and RDFC.

For real-time filtering, the À-Trous wavelet filter approach by Dammertz et al. [DSHL10] and the Guided Image Filter approach by Bauszat et al. [BEM11] were long the only viable approaches. More recently, Schied et al. [SKW*17] have proposed an updated version of the À-Trous wavelet filter with their Spatiotemporal Variance-Guided Filtering approach. The approach by Mara et al. [MMBJ17] operates differently, but employs many of the same concepts as Schied et al. Chaitanya et al. [CKS*17] employed convolutional neural networks successfully for denoising Monte Carlo renders, but the approach requires long training times and is slower than the other real-time approaches. The approach by Bauszat et al. [BEJM15] remains the only approach that can handle both noise from undersampling hemispheres and undersampling the lens domain.

A number of interesting alternative, "less mainstream" approaches have been proposed over the years. The virtual flash image [MJL*13, Sch13] seems to offer benefits over geometric feature buffers. Kalantari et al. [KBS15] showed that using machine learning to predict bandwidth parameter values can be effective. A wide variety of a priori filtering methods can produce high quality results, but unfortunately these require deep integration with the renderer and are restricted to certain effects.

For our research, we built upon the Spatiotemporal Variance-Guided Filtering approach by Schied et al. [SKW*17]. SVGF is both effective and can easily be integrated with existing renderers. The approach by Mara et al. [MMBJ17] seems more restrictive, since it assumes that the BRDF can be divided into matte and glossy terms. The convolutional neural network by Chaitanya et al. [CKS*17] is effective, but seems to require more time than SVGF.

TABLE 3.1: A chronological overview of the discussed a posteriori methods. For each method, we list the used smoothing method and whether the technique was designed with a real-time performance target. We do not list reported runtimes: the increasing capabilities of hardware mean that reported runtimes cannot be compared in a fair manner.

Work	Known as	Smoothing method	Real-time target
[LR90]	Nonlinear Filtering in Computer Graphics	Median, alpha-trimmed mean filter	No
[RW94]	Energy Preserving Non-Linear Filters	Energy-preserving non-linear filter	No
[McC99]	Anisotropic Diffusion for Monte Carlo Noise Reduction	Anisotropic diffusion	No
[XP05]	Novel Monte-Carlo Noise Reduction Operator	Bilateral filter	No
[HJW*08]	Multidimensional Adaptive Sampling and Reconstruction	Anisotropic reconstruction	No
[ODR09]	Adaptive Wavelet Rendering (AWR)	DWT, soft thresholding	No
[DSHL10]	Edge-Avoiding À-Trous Wavelet Filter (EAW)	À-Trous wavelet filter	Yes
[BEM11]	Guided Image Filtering for Global Illumination	Guided image filter	Yes
[RKZ11]	Greedy Error Minimization (GEM)	Gaussian filterbank	No
[SD12]	Random Parameter Filtering (RPF)	Cross-bilateral filter iterations	No
[LWC12]	SURE-Based Filter	Cross-bilateral filterbank	No
[RKZ12]	Adaptive Rendering with NL-Means Filtering (NLM)	NL-means filter	No
[MJL*13]	Denoising using a Virtual Flash Image	NL-means, homogeneous pixels	No
[KS13]	General Image Denoising (GID)	E.g. BM3D	No
[Sch13]	Radiance Filtering	Filter with radiance samples neighbors	No
[RMZ13]	Robust Denoising (RD/RDFC)	Filterbank, varying color sensitivity	No
[DMB*14]	Ray Histogram Fusion (RHF)	NL-means filter	No
[MCY14]	Local Weighted Regression (LWR)	First-order regression	No
[BEJM15]	Sample-Based Manifold Filtering	Adaptive manifolds, sweep-blur	Yes
[BEEM15]	General and Robust Error Estimation and Reconstruction	Filter caches, filterbank	No
[KBS15]	Learning-Based Filter (LBF)	Cross-bilateral filter	No
[MIGYM15]	Adaptive Rendering with Linear Predictions	First-order regression	No
[KDB16]	Path Space Filtering	3D world-space kernel	No
[MMMG16]	Adaptive Polynomial Rendering	Higher-order regression	No
[BRM*16]	Nonlinearly Weighted First-Order Regression (NFOR)	First-order regression	No
[CKS*17]	Reconstruction using a Recurrent Denoising Autoencoder	Recurrent denoising autoencoder	Yes
[SKW*17]	Spatiotemporal Variance-Guided Filtering (SVGF)	À-trous wavelet filter	Yes
[MMBJ17]	Efficient Denoising for Global Illumination	Cross-bilateral filter	Yes

Bibliography

[BCM05] Buades A., Coll B., Morel J. M.: A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation 4*, 2 (2005), 490–530.

[BEEM15] Bauszat P., Eisemann M., Eisemann E., Magnor M.: General and robust error estimation and reconstruction for monte carlo rendering. *Comput. Graph. Forum 34*, 2 (May 2015), 597–608.

[BEJM15] Bauszat P., Eisemann M., John S., Magnor M.: Sample-based manifold filtering for interactive global illumination and depth of field. *Comput. Graph. Forum 34*, 1 (Feb. 2015), 265–276.

[BEM11] Bauszat P., Eisemann M., Magnor M.: Guided image filtering for interactive high-quality global illumination. In *Proceedings of the Twenty-second Eurographics Conference on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2011), EGSR '11, Eurographics Association, pp. 1361–1368.

[BRM*16] Bitterli B., Rousselle F., Moon B., Iglesias-Guitián J. A., Adler D., Mitchell K., Jarosz W., Novák J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. *Comput. Graph. Forum 35*, 4 (July 2016), 107–117.

[CKS*17] Chaitanya C. R. A., Kaplanyan A. S., Schied C., Salvi M., Lefohn A., Nowrouzezahrai D., Aila T.: Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph. 36*, 4 (July 2017), 98:1–98:12.

[DFKE06] Dabov K., Foi A., Katkovnik V., Egiazarian K.: Image denoising with block-matching and 3D filtering. In *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning* (Feb. 2006), vol. 6064 of *Proceedings of the SPIE*, pp. 354–365.

[DHS*05] Durand F., Holzschuch N., Soler C., Chan E., Sillion F. X.: A frequency analysis of light transport. *ACM Trans. Graph. 24*, 3 (July 2005), 1115–1126.

[DJ94] Donoho D. L., Johnstone J. M.: Ideal spatial adaptation by wavelet shrinkage. *Biometrika 81*, 3 (1994), 425–455.

[DMB*14] Delbracio M., Musé P., Buades A., Chauvier J., Phelps N., Morel J.-M.: Boosting monte carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 8:1–8:15.

[DSHL10] Dammertz H., Sewtz D., Hanika J., Lensch H. P. A.: Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2010), HPG '10, Eurographics Association, pp. 67–75.

[ED04] Eisemann E., Durand F.: Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 673–678.

[EDR11] Egan K., Durand F., Ramamoorthi R.: Practical filtering for efficient ray-traced directional occlusion. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 180:1–180:10.

[EHDR11] Egan K., Hecht F., Durand F., Ramamoorthi R.: Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2 (Apr. 2011), 9:1–9:13.

[ETH*09] Egan K., Tseng Y.-T., Holzschuch N., Durand F., Ramamoorthi R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28*, 3 (July 2009), 93:1–93:13.

[GO12] Gastal E. S. L., Oliveira M. M.: Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph. 31*, 4 (July 2012), 33:1–33:13.

[HJW*08] Hachisuka T., Jarosz W., Weistroffer R. P., Dale K., Humphreys G., Zwicker M., Jensen H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 33:1–33:10.

[HST10] He K., Sun J., Tang X.: Guided image filtering. In *Proceedings of the 11th European Conference on Computer Vision: Part I* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 1–14.

[JM12] Jakob W., Marschner S.: Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Trans. Graph. 31*, 4 (July 2012), 58:1–58:13.

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 143–150.

[Kar14] Karis B.: High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses 1* (2014).

[KBS15] Kalantari N. K., Bako S., Sen P.: A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph. 34*, 4 (July 2015), 122:1–122:12.

[KDB16] Keller A., Dahm K., Binder N.: Path space filtering. In *Monte Carlo and Quasi-Monte Carlo Methods* (Cham, 2016), Springer International Publishing, pp. 423–436.

[KS13] Kalantari N. K., Sen P.: Removing the noise in monte carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32*, 2pt1 (2013), 93–102.

[LAC*11] Lehtinen J., Aila T., Chen J., Laine S., Durand F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (July 2011), 55:1–55:12.

[LALD12] Lehtinen J., Aila T., Laine S., Durand F.: Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31*, 4 (July 2012), 51:1–51:10.

[LR90] Lee M. E., Redner R. A.: Filtering: A note on the use of nonlinear filtering in computer graphics. *IEEE Comput. Graph. Appl. 10*, 3 (May 1990), 23–29.

[LWC12] Li T.-M., Wu Y.-T., Chuang Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 194:1–194:9.

[Mal89] Mallat S. G.: A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell. 11*, 7 (July 1989), 674–693.

[McC99] McCool M. D.: Anisotropic diffusion for monte carlo noise reduction. *ACM Trans. Graph. 18*, 2 (Apr. 1999), 171–194.

[MCY14] Moon B., Carr N., Yoon S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (Sept. 2014), 170:1–170:14.

[MIGYM15] Moon B., Iglesias-Guitian J. A., Yoon S.-E., Mitchell K.: Adaptive rendering with linear predictions. *ACM Trans. Graph. 34*, 4 (July 2015), 121:1–121:11.

[MJL*13] Moon B., Jun J. Y., Lee J., Kim K., Hachisuka T., Yoon S.-E.: Robust image denoising using a virtual flash image for monte carlo ray tracing. *Computer Graphics Forum 32*, 1 (2013), 139–151.

[MMBJ17] Mara M., McGuire M., Bitterli B., Jarosz W.: An efficient denoising algorithm for global illumination. In *Proceedings of High Performance Graphics* (New York, NY, USA, 2017), HPG '17, ACM, pp. 3:1–3:7.

[MMMG16] Moon B., McDonagh S., Mitchell K., Gross M.: Adaptive polynomial rendering. *ACM Trans. Graph. 35*, 4 (July 2016), 40:1–40:10.

[MSY16] Mao X., Shen C., Yang Y.: Image restoration using convolutional auto-encoders with symmetric skip connections. *CoRR abs/1606.08921* (2016).

[MWR12] Mehta S. U., Wang B., Ramamoorthi R.: Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 163:1–163:10.

[MYRD14] Mehta S. U., Yao J., Ramamoorthi R., Durand F.: Factored axis-aligned filtering for rendering multiple distribution effects. *ACM Trans. Graph. 33*, 4 (July 2014), 57:1–57:12.

[ODR09] Overbeck R. S., Donner C., Ramamoorthi R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 140:1–140:12.

[PMKY13] Park H., Moon B., Kim S., Yoon S.-E.: P-rpf: Pixel-based random parameter filtering for monte carlo rendering. In *Proceedings of the 2013 International Conference on Computer-Aided Design and Computer Graphics* (Washington, DC, USA, 2013), CADGRAPHICS '13, IEEE Computer Society, pp. 123–130.

[PSA*04] Petschnigg G., Szeliski R., Agrawala M., Cohen M., Hoppe H., Toyama K.: Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 664–672.

[RKZ11] Rousselle F., Knaus C., Zwicker M.: Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 159:1–159:12.

[RKZ12] Rousselle F., Knaus C., Zwicker M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 195:1–195:11.

[RMZ13] Rousselle F., Manzi M., Zwicker M.: Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7 (2013), 121–130.

[RW94] Rushmeier H. E., Ward G. J.: Energy preserving non-linear filters. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 131–138.

[SAC*11] Shirley P., Aila T., Cohen J., Enderton E., Laine S., Luebke D., McGuire M.: A local image reconstruction algorithm for stochastic rendering. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 9–14.

[Sch13] Schwenk K.: *Filtering Techniques for Low-Noise Previews of Interactive Stochastic Ray Tracing*. PhD thesis, Technische Universität, Darmstadt, August 2013.

[SD12] Sen P., Darabi S.: On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31*, 3 (May 2012), 18:1–18:15.

[SKW*17] Schied C., Kaplanyan A., Wyman C., Patney A., Chaitanya C. R. A., Burgess J., Liu S., Dachsbacher C., Lefohn A., Salvi M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics* (New York, NY, USA, 2017), HPG '17, ACM, pp. 2:1–2:12.

[Ste81] Stein C. M.: Estimation of the mean of a multivariate normal distribution. *Ann. Statist. 9*, 6 (11 1981), 1135–1151.

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), ICCV '98, IEEE Computer Society, pp. 839–.

[Val99] Valens C.: A really friendly guide to wavelets. *ed. Clemens Valens* (1999).

[Whi79] Whitted T.: An improved illumination model for shaded display. *SIGGRAPH Comput. Graph. 13*, 2 (Aug. 1979), 14–.

[XP05] Xu R., Pattanaik S. N.: A novel monte carlo noise reduction operator. *IEEE Comput. Graph. Appl. 25*, 2 (Mar. 2005), 31–35.

[YMRD15] Yan L.-Q., Mehta S. U., Ramamoorthi R., Durand F.: Fast 4d sheared filtering for interactive rendering of distribution effects. *ACM Trans. Graph. 35*, 1 (Dec. 2015), 7:1–7:13.

[ZJL*15] Zwicker M., Jarosz W., Lehtinen J., Moon B., Ramamoorthi R., Rousselle F., Sen P., Soler C., Yoon S.-E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Comput. Graph. Forum 34*, 2 (May 2015), 667–681.

[ZRJ*15] Zimmer H., Rousselle F., Jakob W., Wang O., Adler D., Jarosz W., Sorkine-Hornung O., Sorkine-Hornung A.: Path-space motion estimation and decomposition for robust animation filtering. In *Proceedings*

of the 26th Eurographics Symposium on Rendering (Aire-la-Ville, Switzerland, Switzerland, 2015), EGSR '15, Eurographics Association, pp. 131–142.