Denoising

Monte Carlo rendering often leads a lot of noise when we don't have many samples per pixel.

Image-Based Denoising

In the image-based denoising, we don't touch the rendering pipeline, but manipulate only on the rendered image.
The main idea is to take advantage of the neighboring pixels in the meantime preserve the structure(not blurring).

Bilateral Filter

Linear Filtering with Gaussian Blur(GB)

Convolution by a positive kernel is the basic operation in linear image filtering. It amounts to estimate at each position a local average of intensities and corresponds to low-pass filtering. The Gaussian blur(GB) operator is defined by:

$$\mathrm{GB}[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma}(\|\mathbf{p} - \mathbf{q}\|) I_{\mathbf{q}}$$

where $G_{\sigma}(x)$ denotes the 2-dimensioanl Gaussian kernel:

$$G_{\sigma}(x) = \frac{1}{2\pi\sigma^2} e^{-x^2/2\sigma^2}$$

Gaussian filtering is a weighted average of the intensity of the adjacent positions with a weight decreasing with the spatial distance to the center position $\mathbf{p}$.

Nonlinear Filtering with Bilateral Filter(BF)

Similarly to the Gaussian convolution, the bilateral filter is also defined as a weighted average of pixels. The difference is that the bilateral filter takes into account the variation of intensities to preserve edges.

The rationale of bilateral filtering is that two pixels are close to each other not only if they occupy nearby spatial locations but also if they have some similarity in the photometric range.

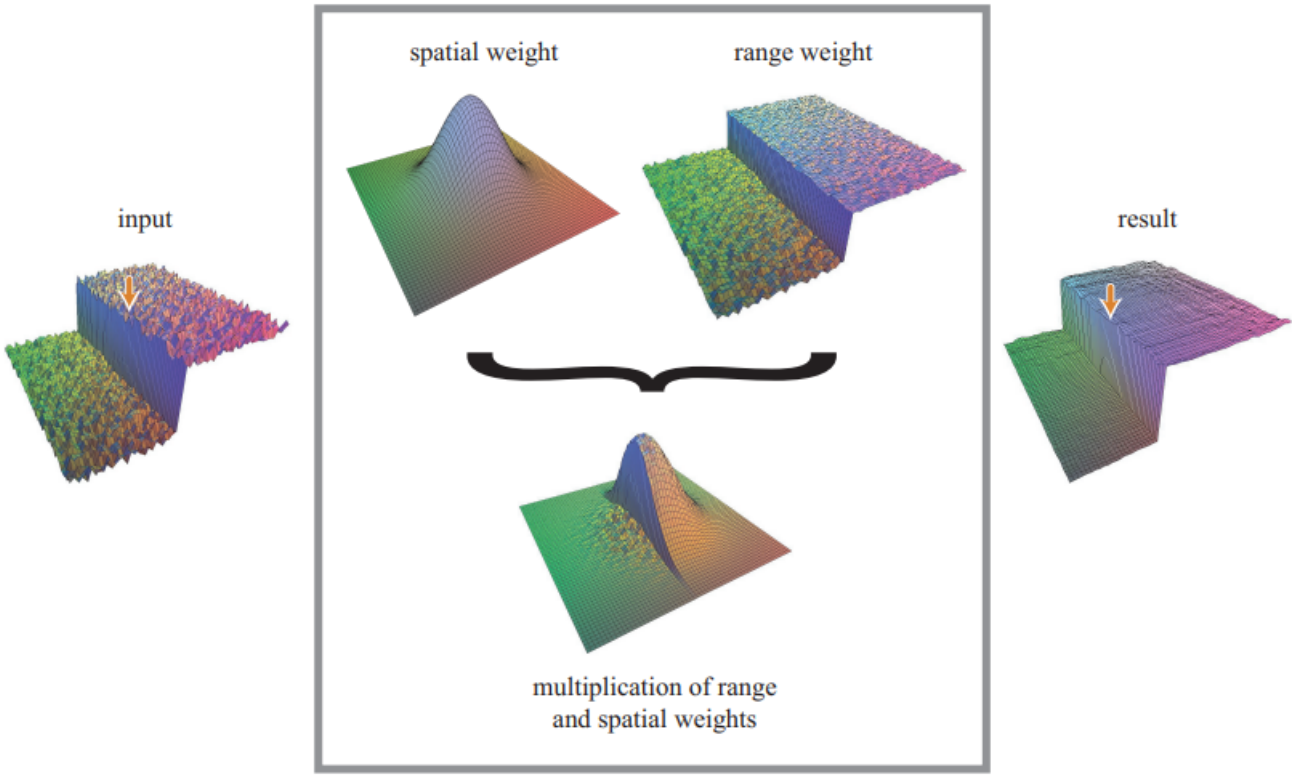
The bilateral filter, denoted by $\mathrm{BF}[\cdot]$ is defined by:

$$\mathrm{BF}[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(I_{\mathbf{p}} - I_{\mathbf{q}}) I_{\mathbf{q}}$$

where $W_{\mathbf{p}}$ is a normalization factor:

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(I_{\mathbf{p}} - I_{\mathbf{q}})$$

Parameters σ_s and σ_r will measure the amount of filtering for the image I. G_{σ_s} is a spatial Gaussian that decreases the influence of distant pixels, G_{σ_r} a range Gaussian that decreases the influence of pixels $\mathbf{q}$ with an intensity value different from $I_{\mathbf{p}}$. Note that the term *range* qualifies quantities related to pixel values, by opposition to *space* which refers to pixel location.

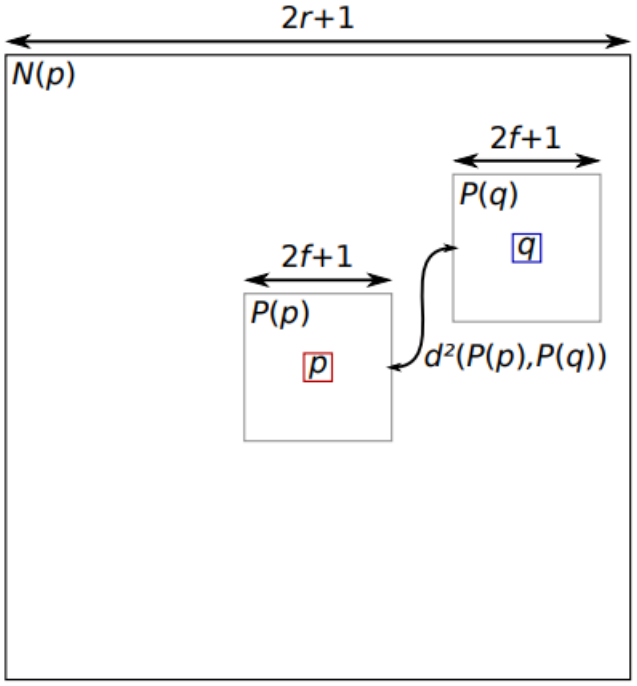


bilateral filter weights of the central pixel

- As the range parameter σ_r increases, the bilateral filter becomes closer to Gaussian blur because the range Gaussian is flatter i.e., almost a constant over the intensity interval covered by the image.
- Increasing the spatial parameter σ_s smooths larger features.

An important characteristic of bilateral filtering is that the weights are multiplied, which implies that as soon as one of the weight is close to 0, no smoothing occurs.

NL-Means



Distances are computed by image patches

The NL-Means filter computes the filtered value $\hat{u}(p)$ of a pixel p in a color image $u = (u_1, u_2, u_3)$ as a weighted average of pixels in the square neighborhood of size $(2r + 1) \times (2r + 1)$ centered at p

$$\hat{u}_i(p) = \frac{1}{C(p)} \sum_{q \in N(p)} u_i(q) w(p,q)$$

where

- $N(p)$ is the square neighborhood centered on p
- $w(p,q)$ is the weight of contribution of q to p
- i is the index of color channel
- $C(p)$ is the normalization factor: $C(p) = \sum_{q \in N(p)} w(p,q)$

The weight $w(p,q)$ of a neighbor pixel q is determined by the distance between a pair of small patches of size $(2f+1) \times (2f+1)$ centered at p and q. Note that the distance here is a range distance, meaning we only care about pixel similarity.

The patch distance $d^2(P(p), P(q))$ is the average of the per-pixel and per color channel squared distances $d_i^2(p,q)$ over patches

$$d_i^2(p,q) = (u_i(p) - u_i(q))^2$$
$$d^2(P(p), P(q)) = \frac{1}{3(2f+1)^2} \sum_{i=1}^{3} \sum_{n \in P(0)} d_i^2(p+n, q+n)$$

where $P(p)$ is the patch centered on p, $P(0)$ represents the offsets to each pixel within a patch. When $f = 1$, NL-Means reduces to bilateral filter.

An important observation is that, because the input signal is noisy, the measured squared distances are biased. Therefore, the original NL-Means filter substracts the variance of the measured squared distances to cancel out the noise contribution from the patch distance. Assuming uniform pixel noise(every pixel value has the same variance) with variance σ^2 and uncorrelated pixels p and q, the modified patch distance is

$$\max(0, d^2(P(p), P(q)) - 2\sigma^2)$$

The weight $w(p,q)$ of the contribution of pixel p to q is then obtained using an exponential kernel

$$w(p,q) = e^{-\frac{\max(0, d^2(P(p),P(q)) - 2\sigma^2)}{k^2 2\sigma^2}}$$

where k is a user specified damping factor that controls the strength of the filter. A lower k value yields a more conservative filter.

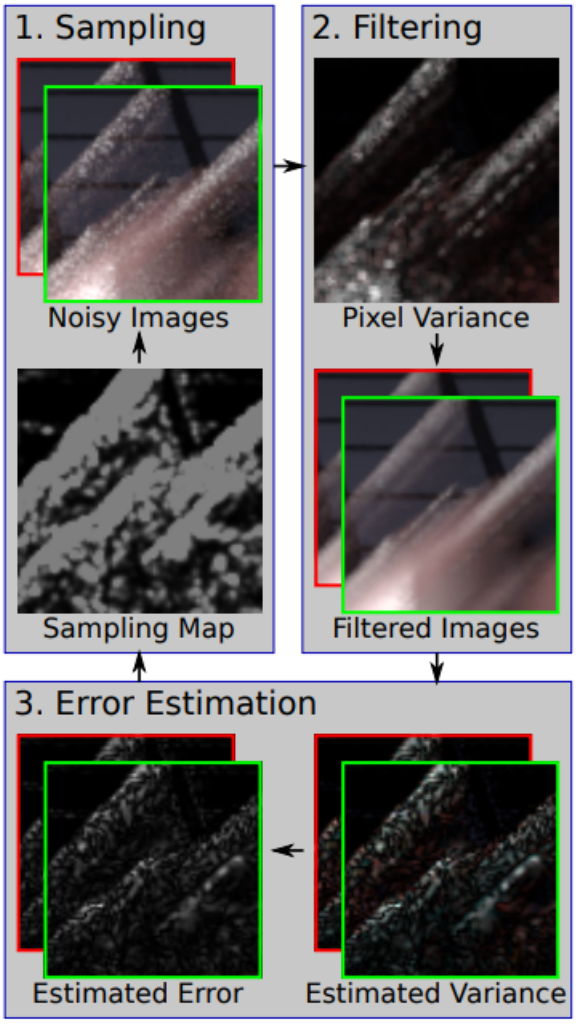
The patchwise extension produces slightly smoother outputs. In the patchwise implementation the final weight $W(p,q)$ for a pair of pixels is simply the average over all weights that involve these two pixels

$$W(p,q) = \frac{1}{(2f+1)^2} \sum_{n \in P(0)} w(p+n, q+n)$$

Adaptive Rendering with Non-Local Means Filtering

The main idea of this paper is to adaptively alter the NL-Means filter and sampling during rendering. They use a variant of the NL-Means filter and use a dual buffer to estimate the variance of each pixel. They then estimate error in each iteration and give more samples to pixel with high error in the next iteration.

Non-uniform Variance NL-Means



Overview of the algorithm

While the original NL-Means formulation assumes uniform noise over input images, Monte Carlo rendering generally leads to highly non-uniform noise patterns. The type of noise and its magnitude depend on the scene geometry, object materials, and light transport and lens effects.

Denote per pixel variance of color channel i at pixel p by $\mathrm{Var}_i[p]$, replacing the uniform variance σ

$$d_i^2(p,q) = \frac{(u_i(p) - u_i(q))^2 - \alpha(\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])}{\epsilon + k^2(\mathrm{Var}_i[p] + \mathrm{Var}_i[q])} \qquad (*)$$

where α controls the strength of variance cancellation. In addition, define $\mathrm{Var}_i[p,q] = \min(\mathrm{Var}_i[p], \mathrm{Var}_i[q])$ to clamp the variance at position q to the variance at p. This ensures that the potentially large variance of brighter neighbors does not cancel out the measured squared difference, and it prevents bright regions from blurring into dark ones.

Variance Estimation

Estimating the pixel variance $\mathrm{Var}[p]$ is a key component of the algorithm (omit the index i of the color channel for better readability from now on). Assuming that samples are drawn from a random distribution, we could estimate the pixel variance using the *empirical variance* of the samples contributing to the pixel, which we denote by $\Sigma[p]$. Random sampling, however, may lead to significantly higher variance than more sophisticated sampling strategies such as low-discrepancy sequences.

Dual-buffer supports low-discrepancy sampling. We obtain an initial estimate $\Delta[p]$ of the pixel variance $\mathrm{Var}[p]$ using the squared difference between the two buffers, $\Delta[p] = (A(p) - B(p))^2/2$. This is an unbiased estimate, but it is rather noisy because it is based on just two samples, that is, the two

buffers. We found that we can improve our results by applying an additional smoothing step to the initial variance estimates $\Delta[p]$.

Dual Buffer

In each iteration we sample N samples in total and separate them evenly into two images, i.e. each image has $\frac{N}{2}$ samples. In the end we average the two images to get the final image.

Robust Denoising using Feature and Color Information

Filtering based on feature buffers, such as per pixel normal, texture, or depth, has proven extremely effective, in particular for images rendered with very few samples per pixel and high noise levels in the Monte Carlo output.

Denote feature buffers such as normals, textures, or depth, denoised and normalized to unit range, as f_j. The feature distance $\Phi_j^2(p,q)$ for feature j between pixels p and q is based on the squared feature difference including variance cancellation similar to equation (*)

$$\Phi_j^2(p,q) = \frac{(f_j(p), f_j(q))^2 - (\mathrm{Var}_j[p] + \mathrm{Var}_j[p,q])}{k_f^2 \max(\tau, \mathrm{Var}_j[p], \|\mathrm{Grad}_j[p]\|^2)}$$

The distance is normalized by two factors:

- User parameter k_f controls the sensitivity of the filter to feature differences
- $\max(\tau, \mathrm{Var}_j[p], \|\mathrm{Grad}_j[p]\|^2)$ depends on the residual variance of the prefiltered feature denoted by $\mathrm{Var}_j[p]$ and the squared gradient magnitude $\|\mathrm{Grad}_j[p]\|$, thresholded to a minimum value τ. This factor normalizes the feature distance relative to residual noise left in the filtered feature and the local feature contrast. measured by its gradient magnitude.

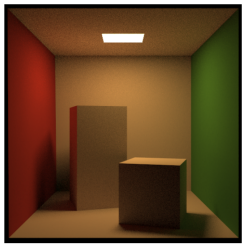
The overall distance $d_f(p,q)$ is obtained by taking the maximum distance over all M features

$$d_f^2(p,q) = \underset{j \in [1,\ldots,M]}{\arg\max}\ \Phi_j^2(p,q)$$

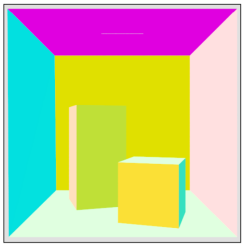
and the final feature weight $w_{f(p,q)}$ is obtained using an exponential kernel.

We determine the final filter weights by taking the minimum of the color and feature weights, that is

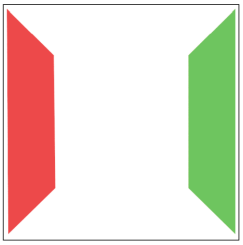
$$w(p,q) = \min(w_c(p,q), w_f(p,q))$$



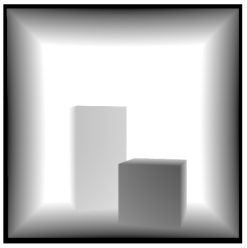
100 samples per pixel, 26s



Normal, remapped to [0,1]



Albedo



Depth, remapped to [0,1]

Data Driven Denoising

Please refer to
Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings-Paper22.pdf (disneyresearch.com)
Denoising with Kernel Prediction and Asymmetric Loss Functions (pixar.com)

References

A Gentle Introduction to Bilateral Filtering and its Applications
Robust Denoising using Feature and Color Information
Adaptive Rendering with Non-Local Means Filtering (umd.edu)
IPOL Journal · Non-Local Means Denoising