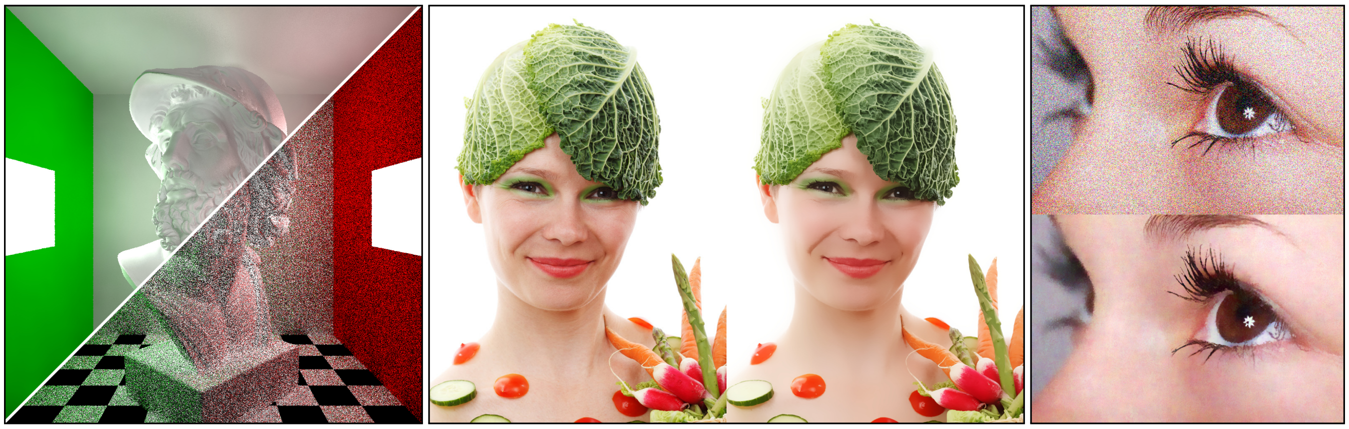
Adaptive Manifolds for Real-Time High-Dimensional Filtering

Eduardo S. L. Gastal* Manuel M. Oliveira†
Instituto de Informática – UFRGS



(a) Filtering using geometric information **(b)** Edge-aware smoothing **(c)** Non-local means denoising

Figure 1: *Examples of filtering results produced with our adaptive-manifold filter.* (a) *Filtering (performed in 8-D) of a noisy undersampled image generated using path tracing. The split rendition compares the input (bottom right) and the filtered result (top left). Note the smoothness of the shading.* (b) *Edge-aware filtering of color images (performed in 5-D) showing large regions smoothed out, and sharp edges preserved.* (c) *Denoising of natural images using non-local-means (performed in 27-D). Notice the noise reduction while retaining fine details.*

Abstract

We present a technique for performing high-dimensional filtering of images and videos in real time. Our approach produces high-quality results and accelerates filtering by computing the filter's response at a reduced set of sampling points, and using these for interpolation at all N input pixels. We show that for a proper choice of these sampling points, the total cost of the filtering operation is linear both in N and in the dimension d of the space in which the filter operates. As such, ours is the first high-dimensional filter with such a complexity. We present formal derivations for the equations that define our filter, as well as for an algorithm to compute the sampling points. This provides a sound theoretical justification for our method and for its properties. The resulting filter is quite flexible, being capable of producing responses that approximate either standard Gaussian, bilateral, or non-local-means filters. Such flexibility also allows us to demonstrate the first hybrid Euclidean-geodesic filter that runs in a single pass. Our filter is faster and requires less memory than previous approaches, being able to process a 10-Megapixel full-color image at 50 fps on modern GPUs. We illustrate the effectiveness of our approach by performing a variety of tasks ranging from edge-aware color filtering in 5-D, noise reduction (using up to 147 dimensions), single-pass hybrid Euclidean-geodesic filtering, and detail enhancement, among others.

CR Categories: I.4.3 [Image Processing and Computer Vision]: Enhancement—Filtering

Keywords: high-dimensional filters, Euclidean filters, bilateral filters, non-local-means filters, hybrid Euclidean-geodesic filters.

*eslgastal@inf.ufrgs.br
†oliveira@inf.ufrgs.br

Links: DL PDF WEB CODE

1 Introduction

High-dimensional filtering has recently received significant attention in the image processing, computer vision, and computational photography communities. Such filters are fundamental building blocks for several applications, including *tone mapping* [Durand and Dorsey 2002], *denoising* [Buades et al. 2005], *detail manipulation* [Bae et al. 2006; Fattal et al. 2007], *upsampling* [Kopf et al. 2007], *spatio-temporal filtering* [Bennett and McMillan 2005; Richardt et al. 2010], *photon-map filtering* [Weber et al. 2004; Bauszat et al. 2011], *alpha matting* [Gastal and Oliveira 2010; He et al. 2011], *recoloring* [Chen et al. 2007], and *stylization* [Winnemöller et al. 2006]. Due to their wide applicability, several high-dimensional filters have been proposed. The most popular one is the bilateral filter [Aurich and Weule 1995; Smith and Brady 1997; Tomasi and Manduchi 1998], which works by weight averaging the colors of neighbor pixels based on their distances in image and color space. For 2-D RGB images, it operates in a 5-D space [Barash 2002], and a naïve implementation is too slow for many practical uses. As a result, several acceleration techniques have been suggested. While they clearly improve performance,

ACM Reference Format
Gastal, E., Oliveira, M. 2012. Adaptive Manifolds for Real-Time High-Dimensional Filtering. *ACM Trans. Graph. 31* 4, Article 33 (July 2012), 13 pages. DOI = 10.1145/2185520.2185529 http://doi.acm.org/10.1145/2185520.2185529.

these solutions natively only handle grayscale images [Durand and Dorsey 2002; Chen et al. 2007; Yang et al. 2009], are still not sufficiently fast for real-time applications [Paris and Durand 2009; Adams et al. 2009; Adams et al. 2010], or may introduce artifacts by not using true Euclidean distances [He et al. 2010].

We present a new approach for efficiently performing *high-quality high-dimensional filtering* that avoids the shortcomings found in previous techniques. Our solution accelerates filtering by evaluating the filter's response on a reduced set of sampling points and using these values to interpolate the filter's response at all N input pixels. We show that, given an appropriate choice of sampling points, the image can be filtered in $O(dNK)$ time, where d is the dimension of the space in which the filter operates, and K is a value independent of N and d. For color images, K typically varies from 3 to 15. Thus, the resulting filters are the *first high-dimensional filters with linear cost both in N and in d*. We present a derivation for the equations that define our method, providing a solid theoretical justification for the technique and for its properties. We also show that the response of our filter can easily approximate either a standard Gaussian, a bilateral, or a non-local-means filter [Buades et al. 2005]. This kind of versatility has also been described by Adams et al. [2009; 2010]. However, our filters are faster and require less memory than previous approaches. For instance, we can "bilateral-like" filter a 10-Megapixel full-color image in real time (50 fps) on modern GPUs.

Figure 1 shows some filtering examples obtained using our technique. Figure 1(a) depicts the result of filtering indirect illumination from an undersampled scene, rendered with path tracing. The filter works in an 8-D space, composed of two spatial dimensions and six range dimensions (3-D scene position and normal vector). Note the quality of the filtered image, despite the highly-noisy input. Figure 1(b) shows an example of edge-aware smoothing of an RGB color image (5-D space). Note how the skin of the model has been smoothed out, while important high-frequency features have been preserved (*e.g.*, the rugged leaves on the model's head). The filtered image maintains the artistic integrity of the photograph. Figure 1(c) shows the result of applying a non-local-means filter to obtain a denoised version (bottom) of a corrupted image (top). For this example, the affinity among pixels was computed using a 7×7-pixel neighborhood, reduced through PCA to 25-D, plus two spatial dimensions. Note how delicate features were correctly preserved.

The **contributions** of our work include:

- An efficient approach for performing high-dimensional filtering. Our solution produces high-quality results, and is faster and requires less memory than previous approaches (Section 4). It is the first filter with linear complexity in both the number of pixels N, and in the dimension d of the space in which the filter operates (Section 6.1);
- A theoretical justification for the method and for its properties, by means of a formal derivation of the equations that define it (Appendix A);
- An algorithm for hierarchically computing nonlinear manifolds adapted to an input signal (Section 4);
- A mechanism for trading off accuracy and speed, which also lends to a filter with outlier-suppression properties (Section 5);
- The first demonstration of a hybrid Euclidean-geodesic filter that runs in a single pass (Section 8).

2 High-Dimensional Filtering

High-dimensional filters can be classified as *Euclidean* or *geodesic*, according to how they compute distances between samples. The main difference between the two groups is the filter behavior near strong discontinuities (edges) in the signal. In general, Euclidean filters allow for samples belonging to different sides of a discontinuity to be combined, while geodesic filters do not. Thus, each filter type provides optimal results for different applications. For instance, Euclidean response is better suited for recoloring disjoint elements in an image [Chen et al. 2007], while geodesic response is optimal for adding colors to grayscale images [Levin et al. 2004].

Many geodesic filters were recently proposed [Farbman et al. 2008; Fattal 2009; Gastal and Oliveira 2011]. In this paper, *we focus on efficiently performing filtering operations with Euclidean-like response*. However, given the flexibility of our approach, we also demonstrate the first single-pass hybrid Euclidean-geodesic filter.

Notation Used in the Paper Let $f : \mathcal{S} \subset \mathbb{R}^{d_\mathcal{S}} \to \mathcal{R} \subset \mathbb{R}^{d_\mathcal{R}}$ be a signal associating each point from its $d_\mathcal{S}$-dimensional *spatial* domain $\mathcal{S}$ to a value in its $d_\mathcal{R}$-dimensional *range* $\mathcal{R}$. Examples of such a signal include grayscale images ($d_\mathcal{S} = 2, d_\mathcal{R} = 1$), RGB color images ($d_\mathcal{S} = 2, d_\mathcal{R} = 3$), RGB color videos ($d_\mathcal{S} = 3, d_\mathcal{R} = 3$), and 3D tomographic images ($d_\mathcal{S} = 3, d_\mathcal{R} = 1$). For digital manipulation, the domain $\mathcal{S}$ must be discretized. Thus, let $\{p_1, \ldots, p_N\}$ be the set of N samples obtained by sampling $\mathcal{S}$ using a regular grid. We refer to each p_i as a *pixel*, even for signals with $d_\mathcal{S} \neq 2$. We also adopt the abbreviated notation $f_i = f(p_i)$, and generically refer to f_i as the *color* of pixel p_i. Furthermore, let $\hat{p}_i \in \mathcal{S} \times \mathcal{R}$ be the point in a d-dimensional space ($d = d_\mathcal{S} + d_\mathcal{R}$) with coordinates given by the concatenation of the spatial coordinates $p_i \in \mathcal{S}$ and the range coordinates $f_i \in \mathcal{R}$. For example, in an RGB image, a pixel $p_i = (x_i, y_i)^T$ with color value $f_i = (r_i, g_i, b_i)^T$ has $\hat{p}_i = (x_i, y_i, r_i, g_i, b_i)^T$. This concatenation of coordinates will also be denoted as $\hat{p}_i = (p_i, f_i)$.

2.1 Euclidean Linear Filters

Linear filtering produces a new set of pixel colors as weighted averages of the colors of the input pixels. The weights of this linear combination are given by a function ϕ called the *filter kernel*. For Euclidean filters, the weights decrease with the Euclidean distance. Filtering a signal f with ϕ gives a new signal g:

$$g_i = \sum_{p_j \in \mathcal{S}} \phi(\hat{p}_i - \hat{p}_j)\, f_j \Bigg/ \sum_{p_j \in \mathcal{S}} \phi(\hat{p}_i - \hat{p}_j). \tag{1}$$

Non-negative weights ensure that the linear combination is convex. A common choice for ϕ is an axis-aligned Gaussian function:

$$\phi_\Sigma(\hat{p}_i - \hat{p}_j) = \exp\left(-\frac{1}{2}(\hat{p}_i - \hat{p}_j)^T \Sigma^{-1} (\hat{p}_i - \hat{p}_j)\right), \tag{2}$$

where Σ is a $d \times d$ diagonal covariance matrix that controls, for each dimension, how fast the weights decrease with distance.

If the signal f is an image, Eqs. (1-2) describe the standard *bilateral filter* [Aurich and Weule 1995; Smith and Brady 1997; Tomasi and Manduchi 1998]. A *joint bilateral filter* [Eisemann and Durand 2004; Petschnigg et al. 2004] can be obtained by taking the vectors $\hat{p}_i$ and $\hat{p}_j$ from some image other than f (which may include depth and normal images [Weber et al. 2004]). A *non-local-means filter* [Buades et al. 2005] can be obtained by replacing $\hat{p}_i$ and $\hat{p}_j$ with neighborhoods around the corresponding pixels. Other spatial dimensions can also be taken into account, such as time in a video-sequence [Bennett and McMillan 2005].

Naïvely evaluating Eq. (1) for N pixels requires $O(dN^2)$ operations, which is impractical for most applications. For this reason, many acceleration techniques have been proposed in recent years.

For grayscale image filtering Durand and Dorsey [2002] linearly interpolate several discretized range values, each filtered with a

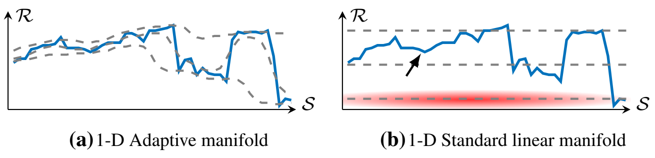


Figure 2: *Our adaptive sampling versus standard linear sampling, applied to a 1-D signal shown in blue.* (a) *Our adaptive manifolds adjust themselves to the signal in high-dimensional space.*

Gaussian kernel in the frequency domain. Porikli [2008] uses summed-area tables to filter each intensity level. Yang et al. [2009] push the idea further, and avoid representing the entire filtering space. Since the time complexity of these methods grows exponentially with $d_{\mathcal{R}}$, they can only be applied to grayscale images.

For color image filtering Paris and Durand [2009] use a 5-D *bilateral grid* and perform filtering by downsampling, which makes Eq. (1) tractable for kernels ϕ_Σ with large support. For 3-D (grayscale) bilateral filtering, this technique has been shown to achieve real-time performance on GPUs [Chen et al. 2007]. Pham and Vliet [2005] approximate Eq. (1) for small kernels as a separable operation, and apply it to video preprocessing.

For higher-dimensional filtering Adams et al. [2009] use a *kd-tree* to sparsely gather the filter response. Adams et al. [2010] use a *permutohedral lattice* to tessellate the high-dimensional space using uniform simplicies, and a hash table to store pixel colors. Other approaches, such as the *guided filter* [He et al. 2010], reduce the dimensionality of the problem by comparing pixels indirectly by their relation to a third point. Although its response is not exactly Euclidean, it has successfully been used in applications such as global-illumination filtering [Bauszat et al. 2011]. The *improved fast Gauss transform* [Yang et al. 2003] computes the filter response as a series expansion around a few clusters. Its high accuracy is obtained at the expense of long computational times. With the exception of the *guided filter*, all these techniques achieve interactive rates, but are still not fast enough for real-time applications.

3 Adaptive Manifolds

The filtering operation described by Eq. (1) can be greatly accelerated by computing the filter's response at a set of M sampling points $\hat{\eta} \in \mathcal{S} \times \mathcal{R}$ and using them to interpolate the filter's response at all N pixels $\hat{p}_i$. This approach shows performance gains when $M \ll N$ [Paris and Durand 2009; Adams et al. 2009; Adams et al. 2010], and/or the M points $\hat{\eta}$ are distributed in a structured way on some flats (generalized planes) embedded in $\mathcal{S} \times \mathcal{R}$ [Paris and Durand 2009; Yang et al. 2009; Adams et al. 2010]. We call such points a *structured set of points*. In this case, using linear-time filtering approaches [Deriche 1993; Heckbert 1986; Yang et al. 2009] and the separability of the Gaussian function, the filter's response for the M sampling points can be computed in $O(dN + dM)$ time instead of $O(dNM)$.

Previous approaches build a structured set of points by laying them onto oriented $d_{\mathcal{S}}$-dimensional flats in $\mathcal{S} \times \mathcal{R}$, either axis-aligned [Paris and Durand 2009; Yang et al. 2009] or with a few discrete orientations [Adams et al. 2010]. These flats define a tessellation of $\mathcal{S} \times \mathcal{R}$ into cells given by hyperrectangles or simplices, respectively. To compute the filter's response for the original pixels, one then performs multi-linear or barycentric interpolation. Since the signal being filtered is rarely a linear manifold, as $d_{\mathcal{R}}$ increases, one needs more flats to enclose the original pixels $\hat{p}_i$ into cells. Furthermore, many of the defined cells will not contain any pixels, and thus any work performed on them is wasted. While this can be avoided using representations such as hash tables [Adams et al. 2010], this makes the algorithm's cost quadratic in the dimensionality d, and its implementation less parallelizable, specially on GPUs. *Kd-trees* [Adams et al. 2009] can also be used to reduce the amount of wasted work. But in this case, Eq. (1) has to be evaluated using nearest-neighbor queries, which generates significant overhead to the algorithm, making its cost superlinear in the number of pixels.

Our approach computes a structured set of sampling points adapted to the signal. This is done by *laying the sampling points on nonlinear $d_{\mathcal{S}}$-dimensional manifolds adapted specifically to each signal and constructed considering the spatial standard deviations of the high-dimensional filter*. To obtain a good approximation for Eq. (1), we show (Appendix A) that it is sufficient for such manifolds to be only approximately linear in all local neighborhoods. The filter's response is computed using a *normalized convolution* interpolator [Knutsson and Westin 1993]. The **benefits of our approach** are:

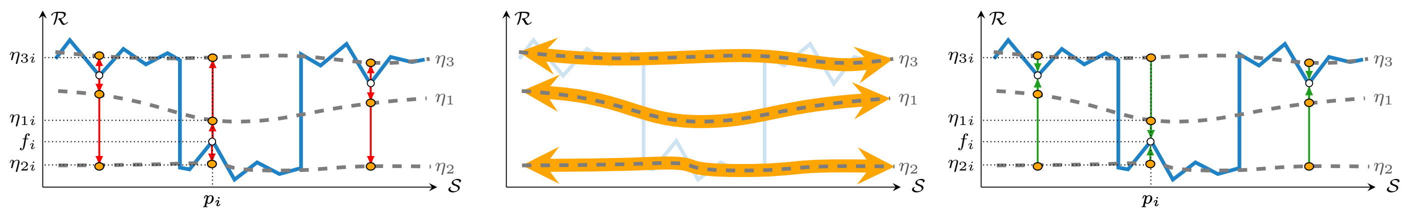
1. *One can sample the high-dimensional space at scattered locations* (on the manifolds), without having to worry about enclosing pixels into cells to perform multi-linear or barycentric interpolation. This is a **key factor for the performance of our method** and results from the use of normalized convolution;
2. *Computation is performed only where it is needed.* We obtain this by adapting the manifolds (and, in turn, the sampling points) to each specific signal;
3. *The filter response can be computed in linear time.* This is possible since the sampling points are structured on $d_{\mathcal{S}}$-dimensional manifolds;
4. *The number of manifolds required to compute the filter's response is independent of the dimension, d, of the space in which the filter operates.* As the manifolds are $d_{\mathcal{S}}$-dimensional, they adapt to the signal equally well, regardless of $d_{\mathcal{R}}$.

Thus, *our filter is the first high-dimensional filter with linear cost both in the number of pixels N and in dimensionality d* (Section 6.1). Furthermore, its implementation is faster and requires less memory than previous approaches. Figure 2(a) illustrates our approach for a 1-D signal ($d_{\mathcal{S}} = d_{\mathcal{R}} = 1$). Our adaptive manifolds (dashed lines) adjust themselves to the underlying signal (in blue) in high-dimensional space (in Figure 2(a), $d = 2$). This defines pathways to exchange information among pixels with similar range values. In contrast, the linear manifolds shown in (b) do not represent well certain regions of the signal (black arrow), while performing unnecessary work in other regions (in red). The occurrence of these issues tend to increase with the dimensionality of $\mathcal{R}$.

3.1 Euclidean Filtering Using Adaptive Manifolds

The steps for computing Eq. (1) using our nonlinear adaptive manifolds are illustrated in Figure 3 for a 1-D signal. To help the readers establish an analogy and compare it with other approaches (*e.g.*, the Gaussian *kd-trees*, the *permutohedral lattice*, and the *5-D bilateral grid*), we use the terminology adopted by Adams et al. [2009; 2010]. Furthermore, to simplify the exposition, this section presents the final form of our equations. Please refer to Appendix A for their derivations.

Let K be the total number of adaptive manifolds that will be used to filter a signal f (Section 5.1 shows how to compute this number). Each pixel $p_i \in \mathcal{S}$ has an associated sampling point $\hat{\eta}_{ki} \in \mathcal{S} \times \mathcal{R}$ which lies on the k-th adaptive manifold. The point $\hat{\eta}_{ki}$ has the same spatial coordinates of p_i, but its range coordinates are given by $\eta_{ki} \in \mathcal{R}$ (note the absence of the hat '^'). Thus, $\hat{\eta}_{ki} = (p_i, \eta_{ki})$. Section 4 discusses how the range coordinates η_{ki} are computed.



(a) Splatting **(b)** Blurring **(c)** Slicing

Figure 3: *Steps of our adaptive-manifold filter (in 1-D), using the terminology from Adams et al. [2009; 2010].* (a) ***Splatting*** *performs a distance-weighted projection of the colors f_i onto each adaptive manifold.* (b) ***Blurring*** *performs Gaussian filtering over each manifold, mixing the distance-weighted projections from all pixels.* (c) ***Slicing*** *computes the filter response for each pixel using normalized convolution.*

The three steps of our filter are: *splatting*, *blurring*, and *slicing*. **Splatting** performs a Gaussian distance-weighted projection of the colors f_i of all pixels p_i onto *each* adaptive manifold (Figure 3(a)). The projected values are stored at the sampling points $\hat{\eta}_{ki}$ associated with p_i:

$$\Psi_{splat}(\hat{\eta}_{ki}) = \phi_{\frac{\Sigma_{\mathcal{R}}}{2}}(\eta_{ki} - f_i)\, f_i. \tag{3}$$

$\Sigma_{\mathcal{R}}$ is the $d_{\mathcal{R}} \times d_{\mathcal{R}}$ diagonal covariance matrix which controls the decay of the Gaussian kernel ϕ in the dimensions of $\mathcal{R}$. The need for scaling $\Sigma_{\mathcal{R}}$ by $1/2$ is explained in Appendix A.

Blurring performs Gaussian filtering over each adaptive manifold, mixing the splatted values Ψ_{splat} from all sampling points $\hat{\eta}_{ki}$ (Figure 3(b)). This results in a new value $\Psi_{blur}(\hat{\eta}_{ki})$ stored at each $\hat{\eta}_{ki}$. Distances for this filtering process on the manifolds are computed taking into account the manifold's curvature in a *scaled* version of the space $\mathcal{S}\times\mathcal{R}$. This scaling maps the (possibly) anisotropic, axis-aligned Gaussian onto an isotropic Gaussian with identity covariance matrix. This is a common operation when implementing high-dimensional filters [Chen et al. 2007; Adams et al. 2010; Gastal and Oliveira 2011].

Slicing computes the final filter response g_i for each pixel p_i by interpolating blurred values Ψ_{blur} gathered from *all* adaptive manifolds (Figure 3(c)). We gather from the same sampling points $\hat{\eta}_{ki}$ used for splatting p_i (Figure 3(a)). Interpolation is done with normalized convolution, described by Knutsson and Westin [1993] in the context of missing data. *g_i is then computed using nearby values known at positions $\hat{\eta}_{ki}$*:

$$g_i = \frac{\sum_{k=1}^{K} w_{ki}\, \Psi_{blur}(\hat{\eta}_{ki})}{\sum_{k=1}^{K} w_{ki}\, \Psi^0_{blur}(\hat{\eta}_{ki})}, \qquad w_{ki} = \phi_{\frac{\Sigma_{\mathcal{R}}}{2}}(\eta_{ki} - f_i). \tag{4}$$

The value Ψ^0_{blur} is the blurred version of Ψ^0_{splat}:

$$\Psi^0_{splat}(\hat{\eta}_{ki}) = \phi_{\frac{\Sigma_{\mathcal{R}}}{2}}(\eta_{ki} - f_i). \tag{5}$$

Contrast Ψ^0_{splat} from Eq. (5) with Ψ_{splat} from Eq. (3), and note the absence of the color f_i on the far right of Ψ^0_{splat}. $\square$

Appendix A provides a detailed derivation of these equations and shows that this process indeed approximates the brute-force evaluation of Eq. (1). Next, we discuss how to compute the sampling points $\hat{\eta}_{ki} = (p_i, \eta_{ki})$ that define the adaptive manifolds.

4 Computing Adaptive Manifolds

The k-th $d_{\mathcal{S}}$-dimensional adaptive manifold (embedded in $\mathcal{S}\times\mathcal{R}$) is represented by a graph (p_i, η_{ki}). The manifold value $\eta_{ki} \in \mathcal{R}$ associated with pixel $p_i \in \mathcal{S}$ is defined by the evaluation of a function $\eta_k : \mathcal{S} \to \mathcal{R}$ at p_i: $\eta_{ki} = \eta_k(p_i)$. Algorithm 1 generates manifolds (*i.e.*, functions η_k) with the following **properties**:

- *They are approximately linear in all local neighborhoods.* As we show in Appendix A, this is required to obtain a good approximation for Eq. (1) for all pixels in $O(dNK)$ time;
- *They approximate the input signal in the high-dimensional space.* This maximizes the number of sampling points with significant interpolation weights w_{ki} in Eq. (4), producing good estimates for Eq. (1). It also reduces bias, as most pixels are well represented by the manifolds (see Section 5 for a discussion on outliers).

The idea behind obtaining manifolds with these properties is to *locally average pixel values from the input signal*. Since the sample mean is a good estimator for the population mean, these averages are good representatives of their corresponding neighborhoods. Furthermore, since local averages define a low-pass filter, the resulting manifolds are guaranteed to be approximately linear in all local neighborhoods (see the discussion at the end of this section). However, close to an edge, the local average will not be a good neighborhood representative. At these locations, one needs *more than one mean estimate* to represent the local mixture of two (or more) populations. For this reason, we use a *hierarchical* segmentation approach to *iteratively separate pixels from different populations into different clusters*. Averaging values only from pixels belonging to the same cluster generates better estimates for the local population means.

An **algorithm** for creating adaptive manifolds for an input signal f can be summarized as follows:

Step 1: Generate the first manifold, η_1, by low-pass filtering the input signal: $\eta_1(p_i) = (h_{\Sigma_{\mathcal{S}}} * f)(p_i)$, where $*$ is convolution, and $h_{\Sigma_{\mathcal{S}}}$ is a low-pass filter in $\mathcal{S}$ with covariance matrix $\Sigma_{\mathcal{S}}$. Recall that $\Sigma_{\mathcal{S}}$ is a $d_{\mathcal{S}} \times d_{\mathcal{S}}$ diagonal matrix that controls the decay of the Gaussian kernel ϕ in the $\mathcal{S}$ dimensions.

Step 2: Compute the direction $v_1 \in \mathbb{R}^{d_{\mathcal{R}}}$ that summarizes the variations of pixel colors f_i about the manifold η_1. Direction v_1 corresponds to the eigenvector associated with the largest eigenvalue of the covariance matrix $(\mathbf{f}_1 - \boldsymbol{\eta}_1)(\mathbf{f}_1 - \boldsymbol{\eta}_1)^T$, where $\mathbf{f}_1 = (f_1 \ldots f_N)$ is a $d_{\mathcal{R}} \times N$ matrix containing all pixel colors f_i, and $\boldsymbol{\eta}_1 = (\eta_{11} \ldots \eta_{1N})$ is a $d_{\mathcal{R}} \times N$ matrix containing all manifold values η_{1i} associated with each pixel p_i. Appendix B shows how to approximate v_1 efficiently in $O(d_{\mathcal{R}} N)$ time.

Step 3: Segment the pixels into two clusters $\mathcal{C}_-$ and $\mathcal{C}_+$ using the sign of the dot product $dot = v_1^T(f_i - \eta_{1i})$:

$$\begin{cases} p_i \in \mathcal{C}_- & \text{if } dot < 0, \\ p_i \in \mathcal{C}_+ & \text{otherwise.} \end{cases} \tag{6}$$

This can be intuitively understood as segmenting the pixels of the input signal into two subsets: one that is locally above the manifold and another that is locally below the manifold. This defines two

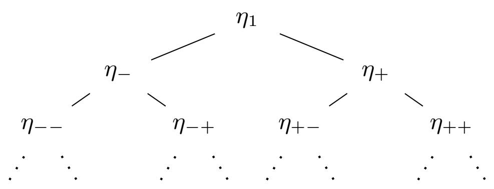


Figure 4: *Manifold tree constructed using Algorithm 1.*

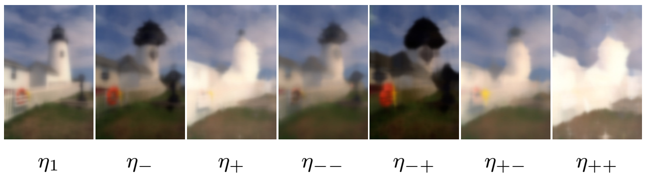


Figure 5: *The manifolds used by our filter to compute Figure 8(b).*

distinct populations. Note that this is just an intuition, since *above* and *below* are only defined for flats with dimension $(d-1)$ in $\mathbb{R}^d$.

Step 4: Compute a new manifold η_- by (weighted) low-pass filtering the input signal, giving weight zero to pixels *not* in $\mathcal{C}_-$:

$$\eta_-(p_i) = \sum_{p_j \in \mathcal{C}_-}^{N} W_-(p_j)\, f_j \Bigg/ \sum_{p_j \in \mathcal{C}_-}^{N} W_-(p_j), \tag{7}$$
$$W_-(p_j) = \theta(\eta_{1j} - f_j)\, h_{\Sigma_\mathcal{S}}(p_i - p_j).$$

$h_{\Sigma_\mathcal{S}}$ is the low-pass filter used to generate η_1, and θ is a function that gives more weight to pixels p_j not well represented by the manifold η_1:

$$\theta(\eta_{1j} - f_j) = 1 - w_{1j}. \tag{8}$$

The values w_{1j} are the interpolation weights from Eq. (4). The construction of the manifold η_+ associated with $\mathcal{C}_+$ is done similarly.

Step 5: Based on the stopping criterion discussed in Section 5.1, decide whether more manifolds are needed. If yes, recursively repeat Step 2, replacing every occurrence of η_1 with η_-, and only using pixels $p_i \in \mathcal{C}_-$ to build the matrices $\mathbf{f}_-$ and $\boldsymbol{\eta}_-$, and clusters $\mathcal{C}_{--}$ and $\mathcal{C}_{-+}$. Do the same for η_+ using pixels $p_i \in \mathcal{C}_+$. □

These steps are shown in Algorithm 1, and define manifolds in a hierarchical tree (Figure 4). Going further down this tree yields manifolds better adapted to local populations: each cluster will contain fewer and more correlated pixels, and Eq. (7) will perform more robust local mean estimates. Figure 5 shows the first three levels of the *manifold tree* computed to filter the image in Figure 8(a).

Low-Pass Filtering Appendix A shows that a good approximation for Eq. (1) can be obtained using nonlinear manifolds if they are approximately linear in all local neighborhoods. Although not true in general, for natural images such approximately-linear manifolds can always be obtained by applying a low-pass filter h to the original pixels, and by making the standard deviation of this filter adequately large (see supplementary materials). According to our experience, using (in Steps 1 and 4 of our algorithm) a low-pass filter $h_{\Sigma_\mathcal{S}}$ with covariance matrix given by $\Sigma_\mathcal{S}$ (*i.e.*, the spatial variance of ϕ_Σ) produces adaptive manifolds which are sufficiently linear (according to *Proposition* A.2 in Appendix A) and generate good filtering results. Section 6 discusses implementation details for this low-pass filter.

Algorithm 1 Computing the Adaptive Manifolds

```
$\eta_1 \leftarrow h_{\Sigma_\mathcal{S}} * f$                    (∗ First manifold ∗)
$\mathcal{C}_1 \leftarrow \{p_1, \ldots, p_N\}$  (∗ First pixel cluster contains all pixels ∗)
return $\{\eta_1\} \cup$ build_manifolds($\eta_1$, $\mathcal{C}_1$)

function build_manifolds($\eta_k$, $\mathcal{C}_k$)
    (∗ Build two pixel clusters by a segmentation of $\mathcal{R}$ ∗)
    Build matrices $\mathbf{f}_k$, $\boldsymbol{\eta}_k$ only using pixels in $\mathcal{C}_k$
    Find largest eigenvector $v_1$ of matrix $(\mathbf{f}_k - \boldsymbol{\eta}_k)(\mathbf{f}_k - \boldsymbol{\eta}_k)^T$
    $\mathcal{C}_- \leftarrow \emptyset$; $\mathcal{C}_+ \leftarrow \emptyset$;
    for all pixels $p_i \in \mathcal{C}_k$ do
        (∗ Use a dot product for segmentation ∗)
        if $v_1^T\,(f_i - \eta_{ki}) < 0$ then
            $\mathcal{C}_- \leftarrow \mathcal{C}_- \cup \{p_i\}$
        else
            $\mathcal{C}_+ \leftarrow \mathcal{C}_+ \cup \{p_i\}$
        end if
    end for
    (∗ Build new manifolds ∗)
    Compute $\eta_-$ and $\eta_+$ using Eq. (7)
    (∗ Recurse if more manifolds are needed ∗)
    $\mathcal{M}_- \leftarrow \{\eta_-\}$; $\mathcal{M}_+ \leftarrow \{\eta_+\}$;
    if stopping criterion not reached (Section 5.1) then
        $\mathcal{M}_- \leftarrow \mathcal{M}_- \cup$ build_manifolds($\eta_-$, $\mathcal{C}_-$)
        $\mathcal{M}_+ \leftarrow \mathcal{M}_+ \cup$ build_manifolds($\eta_+$, $\mathcal{C}_+$)
    end if
    (∗ Return the set of all constructed manifolds ∗)
    return $\mathcal{M}_- \cup \mathcal{M}_+$
end function
```

5 Filter Behavior Analysis

It is instructive to analyze the behavior of our filter by contrasting its response to the response of the brute-force filter defined by Eq. (1), in the presence of outliers. Outlier pixels have range $\mathcal{R}$ coordinates very different from the ones of their neighbors in the spatial domain $\mathcal{S}$. Given an outlier pixel p_i, the filter defined by Eq. (1) *preserves its colors* f_i (*i.e.*, $g_i = f_i$), since p_i has no neighbors in the high-dimensional space $\mathcal{S} \times \mathcal{R}$. This is different from the result obtained with Eq. (4), which *suppresses contributions from outliers*. To understand the source of this *suppressive property*, recall that outliers do not fit the local population and thus, the adaptive manifolds generated by Algorithm 1 do not represent them well. As a result, the values of their colors f_i are highly attenuated during their projections onto the manifolds (Eq. (3), illustrated in Figure 3(a)).

One can handle outlier pixels in **two ways**: *accept* the values computed by Eq. (4), or *change* the filtered values to better approximate the response of Eq. (1). The **first alternative** forces outliers to *behave like their neighborhoods*, thus, as discussed above, suppressing their influence in the filtered signal. This is desirable for some applications, such as denoising. Figure 12 shows an example where our approach was used to implement a *non-local-means* filter [Buades et al. 2005]. Note how the resulting filter (c) is slightly more aggressive in reducing noise than a conventional *non-local-means* filter (d). While algorithms designed specifically for denoising [Dabov et al. 2007] are expected to produce better results than *non-local means*, they tend to be slow. Thus, performing *non-local-means* denoising with our approach offers a good alternative for interactive applications.

The **second alternative** adjusts the response of Eq. (4) to better approximate the response of Eq. (1), which preserves the colors f_i from outliers. (*i.e.*, $g_i = f_i$). Since outlier pixels are far away from all manifolds, this adjustment is performed as

$$g_i = \alpha_i\,\tilde{g}_i + (1-\alpha_i)\,f_i, \quad \alpha_i = \max_k\big(\phi_\Sigma(\hat{p}_i - \hat{\eta}_{ki})\big); \quad (9)$$

where $\tilde{g}_i$ is the result of Eq. (4) for pixel p_i, and α_i is the maximum kernel response for the distance of pixel p_i to its associated sampling point on all manifolds. Since $\alpha_i \in [0,1]$ (recall that the peak of the Gaussian ϕ_Σ in Eq. (2) is 1), g_i becomes a linear interpolation of $\tilde{g}_i$ and f_i. For an outlier pixel, $\alpha_i \approx 0$ and $g_i \approx f_i$, while for a non-outlier pixel, $\alpha_i \approx 1$ and $g_i \approx \tilde{g}_i$.

5.1 Stopping Criteria

According to *Proposition* A.2 (Appendix A), the size of the neighborhood where the manifolds should be approximately linear is proportional to the standard deviations of the filter in $\mathcal{S}$ (*i.e.*, the square-root of the values in $\Sigma_\mathcal{S}$), which are parameters of the filter. As these values increase, the manifolds slowly become flats in $\mathcal{S} \times \mathcal{R}$, and, in turn, become less adapted to the signal. This means that *more manifolds will be needed to accurately represent the pixel population when the spatial standard deviation of the filter is large*. Furthermore, when the values in $\Sigma_\mathcal{R}$ increase, the interpolation weight w_{ki} given by each pixel p_i to its associated sampling point $\hat{\eta}_{ki}$ on the k-th manifold (Eq. (4)) also increases. This means that *fewer manifolds will be needed to estimate Eq. (1) with good accuracy when the range standard deviation of the filter is large*.

With these *guidelines* in mind, the next paragraphs discuss how to select the number of manifolds (K) for different applications. This custom selection provides optimal control over the filter based on properties such as expected outlier behavior and performance goals. We discuss how to compute the height H of the manifold tree (Figure 4), from which K can be obtained as $K = 2^H - 1$.

For **RGB color image filtering**, the tree height is computed as

$$H = \max\big(2, \lceil H_\mathcal{S}\, L_\mathcal{R} \rceil\big), \quad (10)$$

where $H_\mathcal{S}$ is a height computed from the spatial standard deviation of the filter, and $L_\mathcal{R}$ is a linear correction computed from the range standard deviation: $H_\mathcal{S} = \lfloor \log_2(\sigma_{\mathcal{S}_{max}}) \rfloor - 1$, $L_\mathcal{R} = 1 - \sigma_{\mathcal{R}_{min}}$. The standard deviations $\sigma_{\mathcal{S}_{max}}$ and $\sigma_{\mathcal{R}_{min}}$ are given by

$$\sigma_{\mathcal{S}_{max}} = \sqrt{\max(\Sigma_\mathcal{S})}, \quad \text{and} \quad \sigma_{\mathcal{R}_{min}} = \sqrt{\min(\Sigma_\mathcal{R})}. \quad (11)$$

One can verify that these equations follow the guidelines from the beginning of this section. The supplementary materials present a table computed for various combinations of values for $\sigma_{\mathcal{S}_{max}}$ and $\sigma_{\mathcal{R}_{min}}$. Using the tree height given by Eq. (10) and treating outliers according to Eq. (9), one achieves good accuracy against brute-force bilateral filtering: average PSNR above 40 dB for most combinations of σ_s and σ_r parameters (Table 1). Towards the bottom-right of Table 1, the adaptive-manifold filter's results diverge from the bilateral filter's results due to our choice of low-pass filter over the manifolds (RF filter of Gastal and Oliveira [2010], discussed in Section 6), which is not Gaussian, but approximates it. For combinations of large values for both σ_s and σ_r (bottom-right portion of Table 1), the adaptive-manifold filter behaves more like a low-pass filter than as an edge-preserving one. A similar behavior is also observed for the bilateral filter (Figure 6). The supplementary materials provide extensive comparisons of the impact, on the filtered images, of varying the values of the parameters σ_s and σ_r.

For **natural image denoising** using the non-local-means algorithm [Buades et al. 2005], more manifolds are required to deal with the noisy data. Good results are achieved with a small increment to the tree height used for color bilateral filtering:

	σ_s						
σ_r	1	4	8	16	32	64	128
0.01	57.8	52.7	51.2	50.9	50.6	50.4	50.3
0.05	50.7	44.8	42.5	43.3	43.6	43.5	43.3
0.10	47.9	43.7	41.1	41.7	42.1	41.6	40.6
0.15	46.3	43.8	41.2	40.8	41.4	40.4	39.2
0.20	45.3	44.1	41.6	40.2	41.0	39.4	38.0
0.40	43.0	44.5	42.1	40.1	38.6	37.3	36.5
0.60	41.8	43.7	41.7	40.1	39.0	37.4	35.5
1.00	40.8	42.4	41.1	39.7	38.8	37.2	35.1

Table 1: *Mean PSNR of the adaptive-manifold filter (computed with respect to brute-force RGB color bilateral filtering) for various combinations of σ_s and σ_r values. The mean values were obtained from a set of 24 images (included in the supplementary materials). The number of manifolds, K, was computed using the tree height given by Eq. (10). Section 7 defines the parameters σ_s and σ_r.*

$$H = 2 + \max\big(2, \lceil H_\mathcal{S}\, L_\mathcal{R} \rceil\big). \quad (12)$$

Note that *the number of manifolds is independent of the filter dimensionality*. In our tests, we achieve consistent results by using the same tree height (Eq. (12)) for denoising in dimensions from 6 to 147 (Figure 12). Finally, since outliers will be mostly noise, they should be suppressed (*i.e.*, do not use Eq. (9)).

For **global illumination filtering** one can play with the number of manifolds to trade off *performance* and *quality*. This is specially useful in games and other real-time applications. In this case, outliers should also be suppressed. Figure 1 and Figure 14 show an example of noise reduction applied to an image rendered using path tracing, which will be discussed in Section 8.

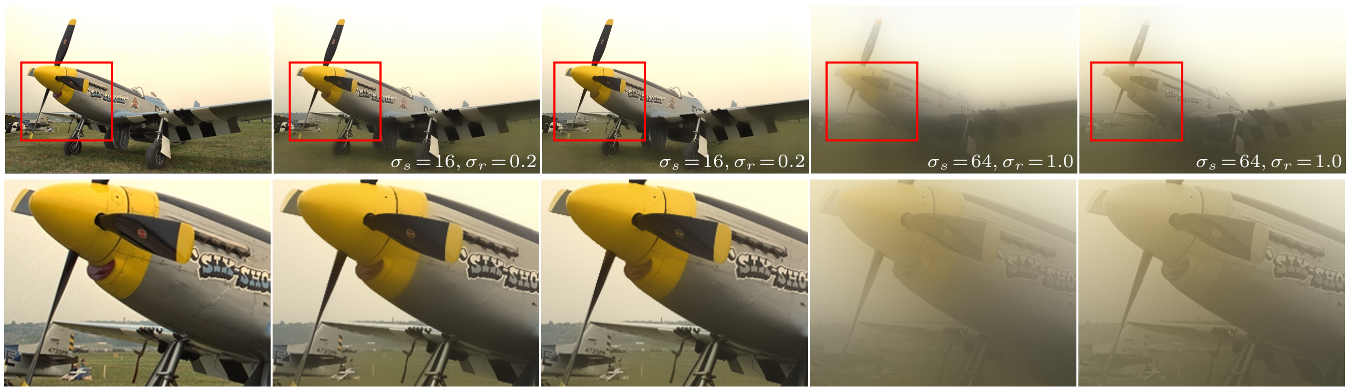
For **other applications**, the number of manifolds can be dynamically computed. A simple and effective approach would be to traverse the manifold tree (Figure 4) breadth-first and stop generating manifolds when a high-percentage of pixels are within close range to at least one manifold. We define close as having a Mahalanobis distance $\|\hat{p}_i - \hat{\eta}_{ki}\|_\Sigma$ less than 1. This is equivalent to being within the range of 1-sigma in an isotropic Gaussian kernel. The pixels outside this range are considered outliers in the pixel color distribution, and should be treated as discussed in Section 5 in accordance with the desired filter response.

6 Implementation Details

For the **low-pass filter** $h_{\Sigma_\mathcal{S}}$ used to compute the adaptive manifolds η_k, we use a linear-time recursive filter with exponential decay:

$$out[i] = in[i] + \exp\left(-\sqrt{2}/\sigma_l\right)(in[i-1] - in[i]). \quad (13)$$

This 1-D filter is applied along each S dimension of the signal being filtered. Since its impulse response is not symmetric, it must be applied twice (once in each direction). For example, in a 2-D image, Eq. (13) is performed left-to-right and then right-to-left for the horizontal dimension. The value σ_l is the square root of the variance at the diagonal position (l,l) of matrix $\Sigma_\mathcal{S}$, where $l = 1, \ldots, d_\mathcal{S}$ is the current $\mathcal{S}$ dimension being filtered. Furthermore, since the samples η_{ki} of the k-th adaptive manifold are computed applying the low-pass filter $h_{\Sigma_\mathcal{S}}$ to an already band-limited signal (the discrete input f in Eq. (7)), they define an "even more" band-limited signal. Thus, one can correctly represent the set $\{\eta_{ki}\}$ using fewer than N samples. For the filter in Eq. (13), one can use $min(N, 4\,N/\sigma_l)$ samples (see the supplementary materials for the derivation of this bound).


$\sigma_s = 16, \sigma_r = 0.2$
$\sigma_s = 16, \sigma_r = 0.2$
$\sigma_s = 64, \sigma_r = 1.0$
$\sigma_s = 64, \sigma_r = 1.0$

(a) Input **(b)** Ours, PSNR 41 dB **(c)** Bilateral filter **(d)** Ours, PSNR 37 dB **(e)** Bilateral filter

Figure 6: *For values of the parameters σ_s and σ_r preserving image edges (b and c), the adaptive-manifold (AM) filter obtains results visually very similar to the bilateral filter (BF). For large values of both σ_s and σ_r (d and e — bottom-right portion of Table 1), the AM filter results diverge from the BF results (lower PSNR) due to our choice of low-pass filter used over the manifolds (see Section 6).*

For **blurring** over each manifold η_k (Figure 3(b)), we use the recursive filter (RF) from Gastal and Oliveira [2011]. This filter uses a domain transform to isometrically map geodesic curves from the high-dimensional manifold onto the real line, filtering in linear time and achieving real-time performance. Although the response of such a filter for computing Ψ_{blur} is exponential instead of Gaussian, we found that results are visually similar and of equal quality. In fact, such a solution achieves average PSNR above 40 dB against the brute-force Gaussian bilateral filter for RGB images (Table 1). If a more Gaussian-like response is needed, one can use iterations of recursive or box filters [Heckbert 1986; Gastal and Oliveira 2011].

In general, the output of the RF filter is not band-limited. However, since the filtering process takes place on *band-limited manifolds* (*i.e.*, smooth manifolds), the output of RF will also be band-limited. Thus, one can safely downsample the signal defined by Ψ_{splat} before using RF to compute Ψ_{blur}, which is then upsampled for slicing. Due to the nonlinear nature of RF, one cannot find a closed-form expression for the Nyquist sampling rate. Using $min(N, N/r)$ samples generates good results for color-image filtering, where $r = min\,(\sigma_{\mathcal{S}_{min}}/4,\ 256\ \sigma_{\mathcal{R}_{min}})$, and $\sigma_{\mathcal{S}_{min}}$ and $\sigma_{\mathcal{R}_{min}}$ are the minimum spatial and range standard deviations.

6.1 Complexity Analysis

Since each pixel appears in a single cluster at each level of the binary tree in Figure 4, clustering takes $O(d\,N\,log\,K)$ time. Computing the manifolds using the filter from Eq. (13) takes $O(d\,N\,K/\sigma_{\mathcal{S}_{min}})$ time , since they are band-limited. Performing the filtering *over* all manifolds has a cost of $O(dNK + dNK/\sigma_{\mathcal{S}_{min}})$ since splatting (Eq. (3)) is evaluated for all pixels, and the RF filter is linear-time. Finally, evaluating the summations in Eq. (4) for all pixels has cost $O(dNK)$. Thus, the total cost of our filter is $O(dNK)$.

Only one manifold has to be kept in memory at each moment, since manifolds can be discarded after being used for blurring (Section 3.1) and clustering (Section 4). To evaluate the filter, one needs the following amounts of memory: (i) $d_{\mathcal{R}}N/\sigma_{\mathcal{S}_{min}}$, to store the current manifold; (ii) $(d_{\mathcal{R}}+1)N/\sigma_{\mathcal{S}_{min}}$, to store the Ψ_{blur} and Ψ^0_{blur} values for the current manifold; (iii) $(d_{\mathcal{R}}+1)N$, to sequentially accumulate Ψ_{blur} and Ψ^0_{blur} from all manifolds (scaled by w_{ki}), and perform the final division (Eq. (4)); and (iv) N, to store the clusters belonging to the current manifold-tree branch.

7 Performance Evaluation

We have implemented three versions of the high-dimensional filter described in the paper, and tested them on a large number of images and videos. These implementations include two CPU versions, one written in C++ and one in MATLAB, and a GPU version written in CUDA. The performance numbers reported in the paper were measured on a 3.3 GHz Intel Core i5 2500K processor with 16 GB of memory, and on two GPUs: a GeForce GTX 280 with 1 GB of memory, and a GeForce GTX 580 with 1.5 GB of memory. All comparisons with other techniques were done on the same machine, using a single CPU core, and using code provided by the authors. Furthermore, we adopt the conventional isotropic kernels in $\mathcal{S}$ and $\mathcal{R}$, with standard deviations σ_s and σ_r, respectively. In this case, covariance matrices are obtained as $\Sigma_{\mathcal{S}} = \sigma_s^2\,I_{d_{\mathcal{S}}\times d_{\mathcal{S}}}$ and $\Sigma_{\mathcal{R}} = \sigma_r^2\,I_{d_{\mathcal{R}}\times d_{\mathcal{R}}}$, where $I_{m\times m}$ is the $m \times m$ identity matrix. σ_s is measured in pixels, and σ_r is measured in normalized units (*e.g.*, for RGB color filtering, the RGB cube is given by $[0, 1]^3$).

RGB color image filtering on the CPU Computing the number of manifolds K using Eq. (10), our CPU implementation of the *adaptive-manifold* (AM) filter processes a 1-megapixel color image in about 0.2 seconds for a wide range of filtering parameters. The fastest color bilateral filter currently available is the *permutohedral lattice* (PL) of Adams et al. [2010]. Figure 7 shows that our AM filter is 2 to 5× faster than PL for a wide range of filtering parameters. Furthermore, the AM filter is slightly faster than the *guided filter* (GF) of He et al. [2010], while generating results indistinguishable from brute-force bilateral filtering (Figure 8). The *guided filter* does not compute true 5-D Euclidean distances between pixels. Instead, it computes their similarity with respect to a third point (a local average). As a result, GF may introduce artifacts in the filtered images (Figure 8(d)). In fact, in some cases, it completely ignores color differences between pixels (Figure 9).

For a combination of large spatial standard deviations *and* small range standard deviations, our filter is slower than PL (Figure 7). This happens because, in such situations, the manifolds become linear almost everywhere, loosing their ability to adapt to the signal. To compensate for this, more manifolds are needed to guarantee a sufficient number of sampling points with significant integration weights (see Appendix A), which affects performance. However, as noted by other researchers [Adams et al. 2009; Farbman et al. 2008], to achieve best results, applications that use Euclidean filters usually require spatial standard deviations of small to medium sizes.

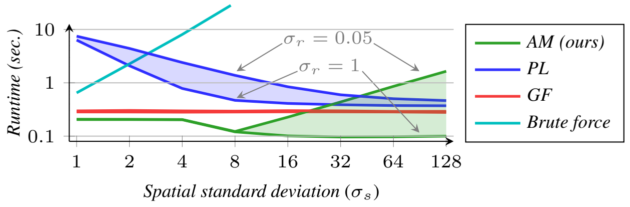


Figure 7: CPU performance *of our adaptive-manifold (AM) filter versus the permutohedral lattice (PL) and the guided filter (GF). The vertical axis shows time, in seconds, to filter a 1-megapixel color image. The shaded areas represent performance changes when σ_r varies from 1 (bottom curve) to 0.05 (top curve).*

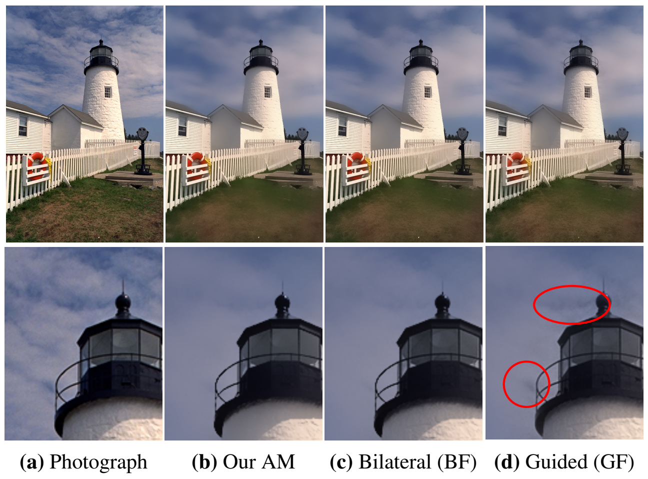


Figure 8: *Comparison of smoothing quality (best viewed in the electronic version). Our filter generates results virtually indistinguishable from the brute-force bilateral filter (BF). The guided filter (GF) failed to smooth certain regions of the sky. Results obtained with the permutohedral lattice and Gaussian kd-trees (not shown) are very similar to ours. Filtering parameters: $\sigma_s = 16$ and $\sigma_r = 0.2$ for our AM and for BF; $r = 16$ and $\epsilon = 0.18^2$ for GF (chosen to maximize overall visual similarity of the results).*

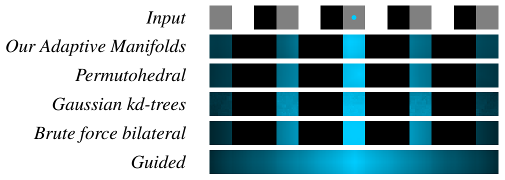


Figure 9: *Response of filtering the blue impulse in the top image using small $\Sigma_{\mathcal{R}}$ values and the edges from the grayscale image. Our filter and bilateral filters preserve discontinuities from the input signal. The guided filter does not manage to do the same, because the gray shade is halfway between white and black. Black regions in the filtered signal indicate $g_i = 0$.*

Thus, *our filter provides the fastest alternative for the most common situations*.

RGB color image filtering on the GPU Due to the simple and parallel operations used by our approach, our filter achieves significant performance gains on GPUs. We implemented our AM filter using CUDA. On a GTX 280 GPU, the total time required for filtering a 1-megapixel color image ranges from 0.001 to 0.036 seconds (considering $\sigma_s \in [1, 128]$, and $\sigma_r \in [0.05, 1.0]$). This represents a speedup from 15 to 50× compared to our one-core CPU implementation. Since a 1-megapixel image is relatively small for a modern GPU, the performance of our filter scales sub-linearly with image size. Our approach can filter a 10-megapixel image under 0.04 seconds. Finally, the bottleneck of our GPU implementation are the recursive filters from Section 6 (*i.e.*, Eq. (13) and the RF filter). Its performance can be further improved using recent approaches for GPU recursive filtering [Nehab et al. 2011].

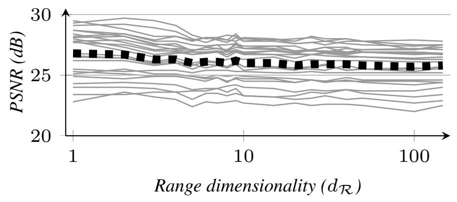


Figure 10: *PSNR (with respect to ground-truth) for non-local means denoising using the adaptive-manifold filter with different range dimensions, using a fixed number of manifolds ($K = 15$, $\sigma_s = 8, \sigma_r = 0.35$). Each gray line represents one of the 24 test images. The black dashed line shows the mean PSNR of the denoised images (~26 dB). The mean PSNR for the noisy input images is ~14 dB. As noted by Tasdizen [2008], lower dimensions provide best results for non-local means denoising, which, as can be seen on the graph, also holds true for the adaptive-manifold filter. Furthermore, for higher dimensionalities, the adaptive-manifold filter obtains fairly constant denoising performance using the same number of manifolds, showing its independence of dimensionality.*

On a **GTX 280 1GB** GPU, the PL filter has a poor performance due to the lack of L1 cache and atomic instructions, both of which are essential for implementing an efficient parallel hash-table. On this GPU, the PL filter requires from 0.5 to 0.7 seconds to filter a 1-megapixel color image — our AM filter is 10 to 100× faster for $\sigma_r < 0.2$, or up to 200× faster for σ_r up to 1. The GPU implementation of the guided filter by Bauszat et al. [2011] filters a 0.75 megapixel color image in 0.07 sec — our AM filter is 2 to 10× faster for $\sigma_r < 0.2$, or up to 40× faster for σ_r up to 1. All cases consider $\sigma_s \in [1, 128]$.

On a **GTX 580 1.5GB** GPU, the PL filter has much improved performance: from 0.05 to 0.1 seconds to filter a 1-megapixel color image. Our AM filter is 10 to 30× faster for $\sigma_s \in [4, 32]$, filtering the same image in 0.001 to 0.007 sec. Considering $\sigma_s \in [1, 128]$, our filter is 3 to 50× faster. Furthermore, the AM filter can process a 10-megapixel image in 0.02 to 0.07 seconds. For images of this size, PL runs out of memory. GPU performance graphs are shown in the supplementary materials.

Higher-Dimensional Filters Since the signal $f : \mathbb{R}^{d_{\mathcal{S}}} \to \mathbb{R}^{d_{\mathcal{R}}}$ and the adaptive manifolds $\eta_k : \mathbb{R}^{d_{\mathcal{S}}} \to \mathbb{R}^{d_{\mathcal{R}}}$ both define $d_{\mathcal{S}}$-dimensional manifolds in $\mathbb{R}^d$, η_k is able to adapt to f equally well regardless of the range dimensionality $d_{\mathcal{R}}$. Thus, the number of manifolds (K) is *independent of the dimension of the space in which the filter operates* — that is, the complexity $O(dNK)$ of the AM filter is linear in both the number of pixels N and dimensionality d, since K is independent of these two values. This independence is shown in Figure 10, where the AM filter achieves fairly constant PSNR for a wide range of dimensionalities using the same number of manifolds. For the graph shown in Figure 10, $K = 15$.

The complexity of previous filters are either quasilinear in N or quadratic in d: $O(dNlogN)$ for the Gaussian kd-trees [Adams et al. 2009]; $O(d^2N)$ for the permutohedral lattice [Adams et al. 2010]; and $O(d_{\mathcal{R}}^{2.807\cdots}N)$ for the guided filter [He et al. 2010], since it requires the inversion of one $d_{\mathcal{R}} \times d_{\mathcal{R}}$ matrix per pixel. As

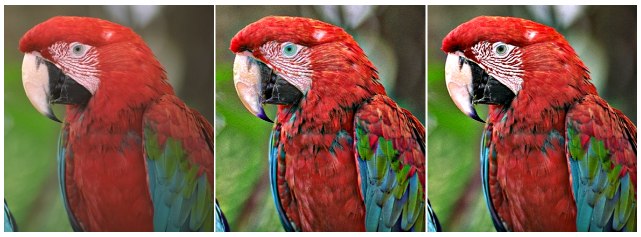


Figure 11: *Example of color detail enhancement. Our Euclidean filter enhances image features differently than a geodesic filter. For instance, note the colors of the parrots' eyes. Choosing between the two results is a subjective choice.*

such, our filter scales significantly better with dimensionality and image size. For example, our C++ CPU implementation of the AM filter applies *non-local-means* [Buades et al. 2005] denoising to a 1 megapixel image, using 25 dimensions, in 3.5 seconds ($K = 15$, $\sigma_s = 8, \sigma_r = 0.2$), while our MATLAB implementation processes the same image in 23 seconds. The C++ CPU implementation of the Gaussian kd-trees (accuracy parameter equal to 1) generates a similar result in 45 seconds, and the C++ CPU implementation of the permutohedral lattice takes 105 seconds. The guided filter has no implementation available for $d_{\mathcal{R}} > 3$.

8 Applications

Our high-dimensional filter can provide real-time feedback for several applications. Next, we describe a few examples. The supplementary materials include results of our filter applied to videos.

5-D color filters can be used to create several effects for images and videos. Figure 8 compares the results produced by our 5-D color filter (b) and the ones generated by a brute-force bilateral filter (c), and by the *guided filter* [He et al. 2010]. Note how our filter smooths the sky, while preserving thin features, such as the lighthouse's handrail. For this example, the guided filter failed to smooth certain parts of the sky, which are highlighted in (d). Figure 11 compares the use of our Euclidean filter (b) to a geodesic filter (c) to perform adaptive contrast enhancement without introducing noticeable haloing artifacts. Such artifacts tend to appear with the use of standard unsharp masking.

Non-local-means filters for denoising [Buades et al. 2005] work by averaging pixels which have similar neighborhoods in the image. The idea is that by using more image features to evaluate the similarity between pixels, one can generate more robust estimators for the noisy input data. This is implemented by replacing the vectors $\hat{p}_i$ from Eq. (1) by the colors of *all* pixels in a $m \times m$ patch around p_i. For an RGB color image, this defines a $(3\,m^2 + 2)$-D space, for which a naïve evaluation of Eq. (1) becomes intractable. For efficiency, one uses PCA to reduce the dimensionality of the resulting feature space ($\mathcal{R}$). Tasdizen [2008] has shown that using only 6 main PCA-computed dimensions actually produces better denoising results than working with the full space. Figure 12 illustrates the use of our approach to implement *non-local-means* filtering.

Filtering with additional information can help to increase the robustness of pixel-correlation estimation. For instance, the recent work of Zhuo et al. [2010] uses an infrared flash to simultaneously capture a crisp infrared image, and a color image obtained under low-lighting conditions (and thus, noisy). Figure 13(a-b) show an infrared-color pair from [Zhuo et al. 2010]. Note the noisy color image (b). Using the *non-local-means* algorithm (in 6-D) on the color data results in noise reduction, but introduces blur (see the horse head on the back of the book in (c)). By incorporating the infrared data as an extra dimension, our filter achieves optimal denoising results for this scene, as shown in (d). The original algorithm of Zhuo et al. uses a geodesic filter for denoising (as opposed to a Euclidean filter), and cannot be effectively used with high-dimensional spaces. Their result exhibits blurring in many regions of the image, such as in the text on the back of the book in (e).

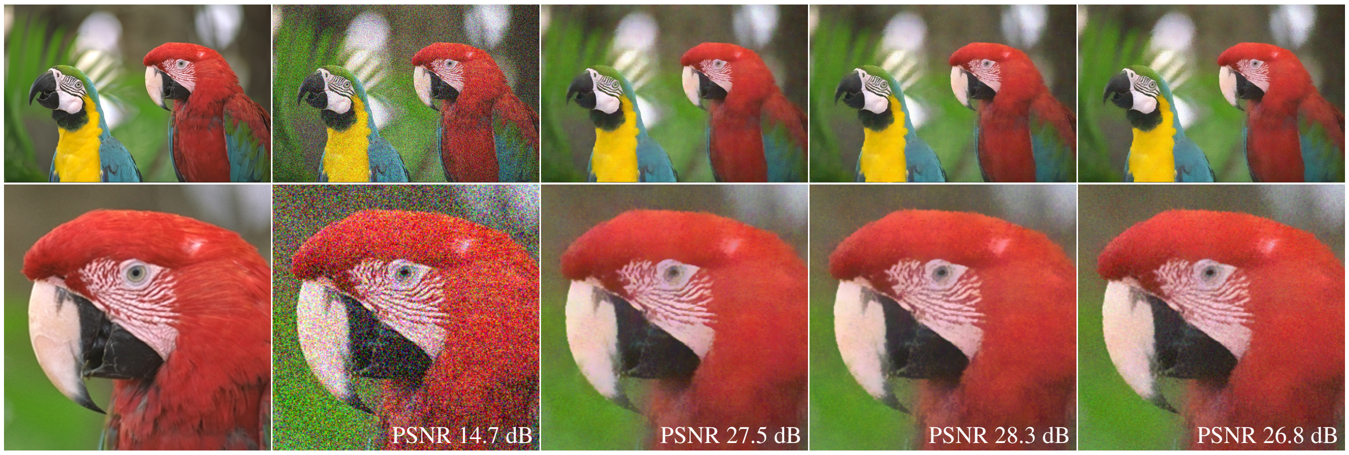
Ray-traced indirect illumination is increasingly making its way into interactive and real-time applications. Despite the advances in algorithms and hardware, sampling all light paths in a scene still requires considerable time. For this reason, modern techniques apply fast and approximate algorithms to denoise undersampled global illumination images [Bauszat et al. 2011], with the goal of obtaining visually pleasing, albeit physically incorrect renderings. Our high-dimensional filter can be effectively used for this task. In the example in Figure 14, the underlying geometric information of the scene is used to help filtering noisy indirect illumination, considerably improving the final rendering (Figure 1(a), which uses $K = 7$). This 8-D filter only considers pixels $p_i = (x_i, y_i)$ that are close in the 2-D image, and have similar positions (X_i, Y_i, Z_i) and normal vectors (n_{ix}, n_{iy}, n_{iz}) in the 3-D scene. In this case, $\hat{p}_i = (x_i, y_i, X_i, Y_i, Z_i, n_{ix}, n_{iy}, n_{iz})$.

A **hybrid Euclidean-geodesic filter** uses Euclidean distances for some dimensions and geodesic distances for others. Let $[\hat{p}_i]_c$ be the c-th coordinate of the point $\hat{p}_i \in \mathcal{S} \times \mathcal{R}$ associated with an input pixel p_i. For each *range* coordinate c for which one wants to perform geodesic filtering, we make $[\hat{\eta}_{ki}]_c = [\hat{p}_i]_c$, $\forall k \, \forall p_i$. For such dimensions, blurring over the adaptive manifolds η_k is equivalent to *blurring over the original signal's manifold*. Figure 15 shows one possible application of such a Euclidean-geodesic filter: performing local tonal adjustments [Lischinski et al. 2006] using additional depth information. In this example, a scribble over the color image in (a) is used to compute a selection mask for pixels with similar colors and depth. A pure Euclidean filter causes the mask to bleed into other statues, as shown in Figure 15(b). On the other hand, using Euclidean distance for color, and geodesic distance for depth, the selection mask is constrained to the desired statue (Figure 15(c)). While this operation could be performed with a two-pass Euclidean filter for color followed by a geodesic filter for depth (or vice-versa), the flexibility of our approach supports a single-pass hybrid Euclidean-geodesic filter.

9 Conclusions and Future Work

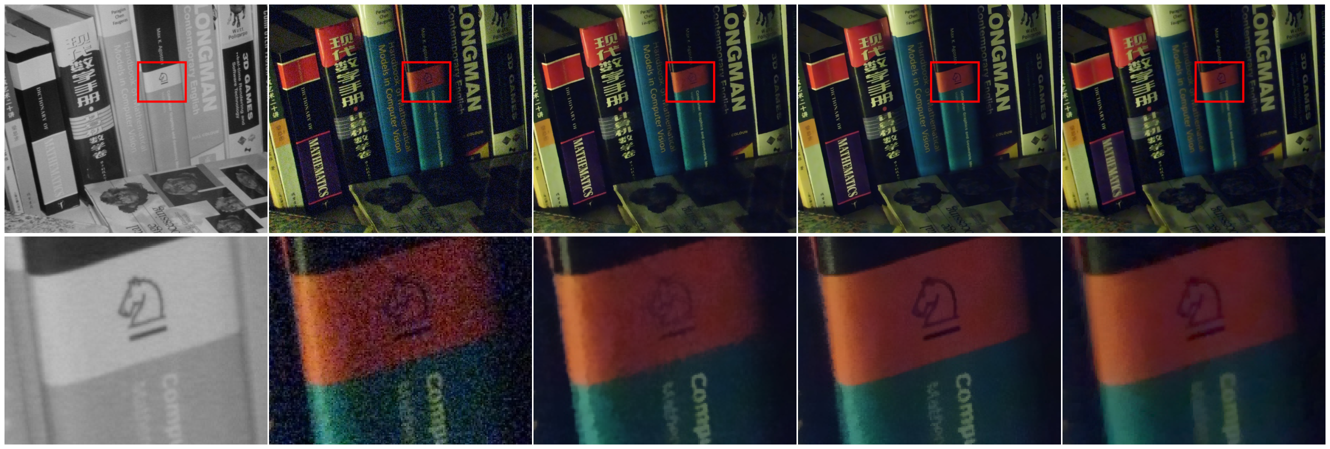
We presented an efficient technique for performing high-dimensional filtering of images and videos in real time. Our filter produces high-quality results and is the first high-dimensional filter with linear cost both in the number of pixels and in the dimension of the space in which the filter operates. We presented a formal derivation for the equations that define the filter, providing a theoretical justification for our method and for its properties. Our filter is faster and requires less memory than previous approaches, being able to process a 10-Megapixel full-color image at 50 fps on modern GPUs. We have also demonstrated the first hybrid Euclidean-geodesic filter that runs in a single pass. We illustrated the effectiveness of our approach performing a variety of image processing tasks such as edge-aware color filtering in 5-D, noise reduction (in 27-D and in 147-D), and detail enhancement.

Our filter (Eq. (4)) is essentially querying the value of a multivariate function by interpolating several scattered points using normalized convolution. Thus, one could replace the Gaussian kernel with any kernel with compact support to get a new kind of filter. It is not clear, however, how to extend our approach to Gaussian functions



(a) Photograph **(b)** Noise std. dev. 0.2 **(c)** Our filter (147-D) **(d)** Our filter (6-D) **(e)** Brute force (6-D)

Figure 12: *Example of non-local means denoising using our filter. For (c) the patch space of* 7×7 *RGB pixel colors was used in full, resulting in a 147-D space* (PCA was not applied). *For (d) the space was reduced to 6-D using PCA, as suggested by Tasdizen [2008]. Our filter works on both 147-D and 6-D using* ***the same number of manifolds*** $K = 15$ *(and* $\sigma_s = 8, \sigma_r = 0.35$*). When compared to a brute-force evaluation of Eq. (1) shown in (e), our filter is slightly more aggressive in reducing noise, due to its outlier-suppression property.*



(a) Infrared (IR) **(b)** Photograph (color) **(c)** Ours 6-D color **(d)** Ours 6-D color + 1-D IR **(e)** Zhuo et al. color + IR

Figure 13: *Example of denoising using additional information from an infrared-color image pair (a) and (b). Using non-local-means on the color data (*7×7*-pixel patches, reduced to 6-D) reduces noise but introduces blur (c). By using the infrared data as an extra dimension, our filter achieves optimal denoising for the scene (d). The original algorithm of Zhuo et al. [2010] uses a geodesic filter for denoising, which does not work effectively for higher dimensions. Their result exhibits blurring in regions such as in the text on back of the books (e).*

with non-diagonal covariance matrices.

One could accelerate the worst-case performance of our filter (large spatial standard deviation) using a Monte Carlo sampling approach: we do not have to project every pixel on the manifolds since the diffusion (term $\phi_1\left(B_{P_k}^T \xi_{kij}\right)$ in Eq. (20)) will blend almost everyone. Thus, it should suffice to just project a subset of the samples.

Limitations When the range standard deviations are small and the number of manifolds K is not adequate (too small), quantization artifacts may appear. These artifacts can be completely removed using some image-space edge-aware filtering approach [Gastal and Oliveira 2011]. For applications such as tone mapping, Euclidean filters may introduce artifacts due to edge-sharpening [Farbman et al. 2008]. However, recent gradient-restoration approaches [Yang et al. 2011] can fix this problem in real-time.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. This work was sponsored by CNPq-Brazil (grants 158666/2010-0, 557814/2010-3, 308936/2010-8, and 480485/2010-0). The Ajax 3d-model is courtesy of Jotero.com. Input image and depth map from Figure 15 courtesy of Bhavsar and Rajagopalan [2010]. Input images from Figure 1(b) and (c) are from publicdomainpictures.net. Figures 6(a), 8(a), 11(a), and 12(a) are from the Kodak Photo CD PCD0992 (http://r0k.us/graphics/kodak/).

References

ADAMS, A., GELFAND, N., DOLSON, J., AND LEVOY, M. 2009. Gaussian kd-trees for fast high-dimensional filtering. In *SIGGRAPH*.

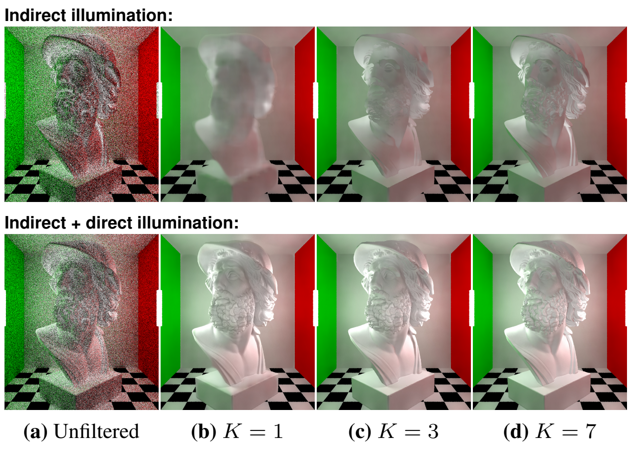


Figure 14: *Path-traced scene was rendered with 1 sample per pixel for indirect illumination, plus 16 light samples for direct illumination. While the unfiltered indirect illumination result in (a) is unusable, applying our filter with as few as $K = 1$ manifold (b-d) can produce high-quality indirect-illumination denoising.*

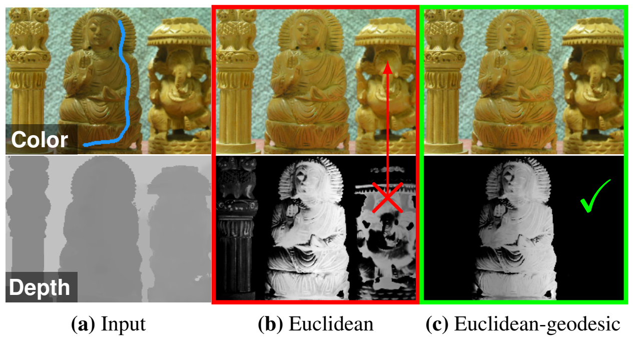


Figure 15: *Hybrid Euclidean-geodesic filter. A user wants to lighten the middle statue, shown by the blue scribble.* (b) *Using a filter with Euclidean response in color and depth, the selection mask bleeds to other statues.* (c) *A filter with Euclidean response in color and geodesic response in depth produces the desired mask.*

ADAMS, A., BAEK, J., AND DAVIS, M. A. 2010. Fast high-dimensional filtering using the permutohedral lattice. *CGF 29*, 2, 753–762.

ADAMS, A. B. 2011. *High-dimensional gaussian filtering for computational photography*. PhD thesis, Stanford University.

ARASARATNAM, I., HAYKIN, S., AND ELLIOTT, R. 2007. Discrete-time nonlinear filtering algorithms using gauss–hermite quadrature. *Proc. of the IEEE 95*, 5, 953–977.

AURICH, V., AND WEULE, J. 1995. Non-linear gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995, 17. DAGM-Symposium*, 538–545.

BAE, S., PARIS, S., AND DURAND, F. 2006. Two-scale tone management for photographic look. *ACM TOG 25*, 3, 637–645.

BARASH, D. 2002. A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE TPAMI 24*, 844–847.

BAUSZAT, P., EISEMANN, M., AND MAGNOR, M. 2011. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum 30*, 4, 1361–1368.

BENNETT, E. P., AND MCMILLAN, L. 2005. Video enhancement using per-pixel virtual exposures. *ACM TOG 24*, 845–852.

BHAVSAR, A. V., AND RAJAGOPALAN, A. N. 2010. Depth estimation and inpainting with an unconstrained camera. In *Proceedings of the British Machine Vision Conference*, 84.1–12.

BUADES, A., COLL, B., AND MOREL, J. 2005. A non-local algorithm for image denoising. In *IEEE CVPR*, vol. 2, 60–65.

CHEN, J., PARIS, S., AND DURAND, F. 2007. Real-time edge-aware image processing with the bilateral grid. *ACM TOG 26*, 3, 103.

CRIMINISI, A., SHARP, T., ROTHER, C., AND P'EREZ, P. 2010. Geodesic image and video editing. *ACM TOG 29*, 5, 134.

DABOV, K., FOI, A., KATKOVNIK, V., AND EGIAZARIAN, K. 2007. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE TIP 16*, 8, 2080–2095.

DERICHE, R., 1993. Recursively implementating the gaussian and its derivatives.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. In *SIGGRAPH '02*, 257–266.

EISEMANN, E., AND DURAND, F. 2004. Flash photography enhancement via intrinsic relighting. In *ACM TOG*, vol. 23, ACM, 673–678.

FARBMAN, Z., FATTAL, R., LISCHINSKI, D., AND SZELISKI, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM TOG 27*, 3, 67.

FATTAL, R., AGRAWALA, M., AND RUSINKIEWICZ, S. 2007. Multiscale shape and detail enhancement from multi-light image collections. *ACM TOG 26*, 51:1–51:9.

FATTAL, R. 2009. Edge-avoiding wavelets and their applications. *ACM TOG 28*, 3, 22.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2010. Shared sampling for real-time alpha matting. *CGF 29*, 2, 575–584.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM TOG 30*, 4, 69:1–69:12. Proceedings of SIGGRAPH 2011.

HE, K., SUN, J., AND TANG, X. 2010. Guided image filtering. In *ECCV*. Springer Berlin / Heidelberg, 1–14.

HE, K., RHEMANN, C., ROTHER, C., TANG, X., AND SUN, J. 2011. A global sampling method for alpha matting. In *CVPR*, IEEE, 2049–2056.

HECKBERT, P. S. 1986. Filtering by repeated integration. *SIGGRAPH Comput. Graph. 20*, 4, 315–321.

KNUTSSON, H., AND WESTIN, C.-F. 1993. Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data. In *CVPR*, 515–523.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM TOG 26*, 96:1–96:5.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM TOG 23*, 689–694.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM TOG 25*, 3, 646–653.

NEHAB, D., MAXIMO, A., LIMA, R. S., AND HOPPE, H. 2011. Gpu-efficient recursive filtering and summed-area tables. *ACM TOG 30*, 176:1–176:12.

NIST, 2011. National Institute of Standards and Technology: Digital library of mathematical functions, August.

PARIS, S., AND DURAND, F. 2009. A fast approximation of the bilateral filter using a signal processing approach. *IJCV 81*, 1, 24–52.

PETSCHNIGG, G., SZELISKI, R., AGRAWALA, M., COHEN, M., HOPPE, H., AND TOYAMA, K. 2004. Digital photography with flash and no-flash image pairs. *ACM TOG 23*, 3, 664–672.

PHAM, T., AND VAN VLIET, L. 2005. Separable bilateral filtering for fast video preprocessing. *IEEE Intl. Conf. on Multimedia and Expo 0*, 4 pp.

PORIKLI, F. 2008. Constant time O(1) bilateral filtering. In *CVPR*, 1–8.

RICHARDT, C., ORR, D., DAVIES, I., CRIMINISI, A., AND DODGSON, N. 2010. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. In *ECCV*. Springer Berlin / Heidelberg, 510–523.

SMITH, S. M., AND BRADY, J. M. 1997. Susan – a new approach to low level image processing. *International journal of computer vision 23*, 1, 45–78.

SOCHEN, N., KIMMEL, R., AND BRUCKSTEIN, A. 2001. Diffusions and confusions in signal and image processing. *Journal of Mathematical Imaging and Vision 14*, 3, 195–209.

TASDIZEN, T. 2008. Principal components for non-local means image denoising. In *ICIP*, IEEE, 1728–1731.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *ICCV*, 839–846.

WEBER, M., MILCH, M., MYSZKOWSKI, K., DMITRIEV, K., ROKITA, P., AND SEIDEL, H. 2004. Spatio-temporal photon density estimation using bilateral filtering. In *CGI*, IEEE, 120–127.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM TOG 25*, 3, 1226.

YANG, C., DURAISWAMI, R., GUMEROV, N., AND DAVIS, L. 2003. Improved fast gauss transform and efficient kernel density estimation. In *ICCV*, IEEE, 664–671.

YANG, Q., TAN, K. H., AND AHUJA, N. 2009. Real-time O(1) bilateral filtering. In *CVPR*, 557–564.

YANG, L., SANDER, P. V., LAWRENCE, J., AND HOPPE, H. 2011. Antialiasing recovery. *ACM TOG 30*, 22:1–22:9.

ZHUO, S., ZHANG, X., MIAO, X., AND SIM, T. 2010. Enhancing low light images using near infrared flash images. In *IEEE ICIP*, IEEE, 2537–2540.

A Formal Derivation of Our Approach

A d-dimensional unit-height Gaussian function can be expressed as the convolution of two d-dimensional Gaussian functions:

$$\phi_{\Sigma_a+\Sigma_b}(t) = C \int_{\mathbb{R}^d} \phi_{\Sigma_a}(t-\tau)\ \phi_{\Sigma_b}(\tau)\ d\tau, \tag{14}$$

where C is a normalization factor. Let $t = x - y$, $\tau = \eta - y$, and $\Sigma_a = \Sigma_b$. Eq. (14) can then be rewritten as:

$$\phi_\Sigma(x-y) = \frac{1}{|\Sigma|^{1/2}} \left(\frac{2}{\pi}\right)^{\frac{d}{2}} \int_{\mathbb{R}^d} \phi_{\frac{\Sigma}{2}}(x-\eta)\ \phi_{\frac{\Sigma}{2}}(\eta-y)\ d\eta, \tag{15}$$

where $|\Sigma|$ is the determinant of the covariance matrix Σ.

Since the Gaussian function is separable over orthogonal directions:

$$\phi_\Sigma(\hat{p}_i - \hat{p}_j) = \phi_{\Sigma_\mathcal{S}}(p_i - p_j)\ \phi_{\Sigma_\mathcal{R}}(f_i - f_j), \tag{16}$$

where $\Sigma_\mathcal{S}$ and $\Sigma_\mathcal{R}$ are the diagonal submatrices of Σ associated with the spatial and range dimensions, respectively. The Gaussian over the range $\mathcal{R}$ in Eq. (16) can be rewritten using Eq. (15) and evaluated numerically (using an approximation to the Gauss-Hermite quadrature rule – see Section A.2) as a weighted sum:

$$\begin{aligned}\phi_{\Sigma_\mathcal{R}}(f_i - f_j) &\propto \int_{\mathbb{R}^{d_\mathcal{R}}} \phi_{\frac{\Sigma_\mathcal{R}}{2}}(f_i-\eta)\, \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta - f_j)\, d\eta \\ &\approx \sum_{k=1}^{K} w_{ki}\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j)\,.\end{aligned} \tag{17}$$

The scaling factor outside the integral was not included as it will cancel out in the division performed in Eq. (1). The K points $\eta_{ki} \in \mathbb{R}^{d_\mathcal{R}}$ are the locations where the integrand is sampled, and *each pixel* p_i *has its own sampling set* $\{\eta_{1i}, \ldots, \eta_{Ki}\}$. In practice, K will be the number of adaptive manifolds, and each pixel p_i has exactly one sampling point η_{ki} on the k-th adaptive manifold. Increasing the number of sampling points (*i.e.*, increasing K) gives a better approximation for the integral. The integration weights w_{ki} are discussed in Section A.2. We can substitute Eq. (16) and (17) in the numerator of Eq. (1):

$$\sum_{p_j \in \mathcal{S}} \phi_\Sigma(\hat{p}_i - \hat{p}_j)\ f_j \tag{18-a}$$

$$\mathrel{\underset{\sim}{\propto}} \sum_{p_j \in \mathcal{S}} \left[\phi_{\Sigma_\mathcal{S}}(p_i - p_j) \sum_{k=1}^{K} w_{ki}\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j) \right] f_j \tag{18-b}$$

$$= \sum_{k=1}^{K} w_{ki} \sum_{p_j \in \mathcal{S}} \phi_{\Sigma_\mathcal{S}}(p_i - p_j)\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j)\ f_j, \tag{18-c}$$

where "$\underset{\sim}{\propto}$" is the approximately proportional relation. Eq. (18-c) was obtained by reordering the summations. The expression for the denominator of Eq. (1) is similar, not including the last term f_j.

Key observation: by a suitable choice of sampling points $\{\eta_{ki}\}$, the inner summation from Eq. (18-c) can be computed in $O(dN)$ time *for all* N pixels p_i. This makes the total cost of Gaussian filtering all pixels $O(dNK)$. Next, we show how to choose $\{\eta_{ki}\}$. All our results were produced with $K \ll N$.

A.1 Choosing the Set of Sampling Points $\{\eta_{ki}\}$

Let $\hat{\eta}_{ki} = (p_i, \eta_{ki})$ be the d-dimensional vector whose coordinates are the concatenation of the ($\mathcal{S}$) coordinates p_i and the ($\mathcal{R}$) coordinates η_{ki}. Using the separability property of the Gaussian to combine the product of $\phi_{\Sigma_\mathcal{S}}$ and $\phi_{\Sigma_\mathcal{R}/2}$ into a single Gaussian ϕ_{Σ_η}, **the inner summation in Eq. (18-c)** can then be rewritten as

$$\Psi_{blur}(\hat{\eta}_{ki}) = \sum_{p_j \in \mathcal{S}} \phi_{\Sigma_\eta}(\hat{\eta}_{ki} - \hat{p}_j)\ f_j,\ \ \Sigma_\eta = \begin{bmatrix} \Sigma_\mathcal{S} & \\ & \Sigma_\mathcal{R}/2 \end{bmatrix}, \tag{19}$$

where $\hat{p}_j = (p_j, f_j)$. Note that Ψ_{blur} *defines a Gaussian filtering (a convolution) on a d-dimensional space.*

Proposition A.1: For each $k = 1 \ldots K$, if the set $\{\hat{\eta}_{ki}\}$ lies on a $d_\mathcal{S}$-dimensional flat (a generalized plane) P_k in d-dimensional space, then $\Psi_{blur}(\hat{\eta}_{ki})$ *can be computed, for all* $\hat{\eta}_{ki}$*, by a Gaussian filtering over* P_k . In other words, Ψ_{blur} can be computed as a Gaussian convolution on a $d_{\mathcal{S}}$-dimensional space.

Proof. Let $B_{P_k} = (b_1 \ldots b_{d_{\mathcal{S}}})$ be any orthonormal basis that spans the flat P_k. Also, let $B^{\perp}_{P_k} = (b_{d_{\mathcal{S}}+1} \ldots b_d)$ be any orthonormal basis which is also orthonormal to B_{P_k}. Together, B_{P_k} and $B^{\perp}_{P_k}$ form an orthonormal basis for $\mathbb{R}^d$. Thus, since the Gaussian function is separable in any orthonormal basis, Eq. (19) can be rewritten as

$$\Psi_{blur}(\hat{\eta}_{ki}) = \sum_{p_j \in \mathcal{S}} \phi_1\left(B^T_{P_k}\,\xi_{kij}\right)\,\underbrace{\phi_1\left(B^{\perp\,T}_{P_k}\,\xi_{kij}\right) f_j}_{\Psi^{P_k}_{splat}(\hat{\eta}_{kj})}, \qquad (20)$$

where $\xi_{kij} = \Sigma_\eta^{-1/2}\,(\hat{\eta}_{ki} - \hat{p}_j)$. This decomposition defines a Gaussian filtering of the term $\Psi^{P_k}_{splat}$ in Eq. (20) over the flat P_k, and corresponds to the **blurring** step in Figure 3 (b). □

The term $\Psi^{P_k}_{splat}(\hat{\eta}_{kj})$ computes a Gaussian distance-weighting of f_j, and corresponds to the **splatting** operation (Figure 3(a)), modeled by Eq. (3). Eq. (18-c) can be re-written as $\sum_{k=1}^{K} w_{ki}\ \Psi_{blur}(\hat{\eta}_{ki})$. This is, after the normalization defined in Eq. (1), the **slicing** operation (Figure 3(c)) modeled by Eq. (4).

The subscript '1' in ϕ_1 (Eq. (20)) indicates that its covariance matrix in Eq. (2) is an identity matrix, and can be disregarded. The product $B^T\,\xi_{kij}$ computes the projection of vector $\xi_{kij} \in \mathbb{R}^d$ onto the basis vector B. The scaling matrix $\Sigma_\eta^{-1/2}$ transforms an anisotropic, axis-aligned Gaussian onto an isotropic Gaussian with identity covariance matrix. This scaling is common in implementations of high-dimensional filters [Chen et al. 2007; Adams et al. 2010; Gastal and Oliveira 2011].

Proposition A.2: For each $k = 1 \ldots K$, if the set $\{\hat{\eta}_{ki}\}$ lies on a $d_{\mathcal{S}}$-dimensional manifold M_k in d-dimensional space, and if M_k is approximately linear in all local neighborhoods of $\mathcal{S}$, then $\Psi_{blur}(\hat{\eta}_{ki})$ *can be approximated, for all* $\hat{\eta}_{ki}$*, by a Gaussian filtering over* M_k*.*

Proof. Due to the compactness of the Gaussian function, over 99% of the value f_i of pixel p_i is "diffused" to pixels p_j whose Mahalanobis distance in space

$$\|p_j - p_i\|_{\Sigma_{\mathcal{S}}} = \sqrt{(p_j - p_i)^T\,\Sigma_{\mathcal{S}}^{-1}\,(p_j - p_i)}$$

is less than 3. Thus, by *Proposition* A.1, if the manifold M_k is approximately linear in a neighborhood of size 3 around each pixel p_i, $\Psi_{blur}(\hat{\eta}_{ki})$ can be well approximated for all $\hat{\eta}_{ki}$ by a Gaussian filtering over M_k. □

Splatting the pixel colors onto the manifold M_k has cost $O(d_{\mathcal{R}})$ (the supplementary materials present further discussion on this). Finally, approaches for performing filtering over manifolds [Sochen et al. 2001] include methods which work in $O(dN)$ time for N d-dimensional points [Criminisi et al. 2010; Gastal and Oliveira 2011]. Thus, since our selection of sampling set $\{\hat{\eta}_{ki}\}$ lies on the manifold M_k, the value $\Psi_{blur}(\hat{\eta}_{ki})$ can be computed for all $\hat{\eta}_{ki}$ in $O(dN)$ time.

A.2 Approximate Gauss-Hermite Quadrature

The Gauss-Hermite quadrature rule [NIST 2011] defines the following approximation for a Gaussian integral:

$$\int_{\mathbb{R}} \phi_{\Sigma_a}(y - x)\ \phi_{\Sigma_b}(x - z)\ dx \approx \sum_{k=1}^{K} w_k\,\phi_{\Sigma_b}(x_k - z)\,, \qquad (21)$$

where x_k are the roots of the Hermite polynomial $H_K(y - x)$, and the weights w_k are approximately Gaussian: $w_k \approx s_K\,\phi_{\Sigma_a}(y - x_k)$, for some constant scaling factor s_K. This rule can be easily extended to the multidimensional integral in Eq. (17) by successive applications [Arasaratnam et al. 2007]. According to this rule, the integral in Eq. (17) can be computed exactly by a weighted sum if the sampling set $\{\eta_{ki}\}$ coincides with the roots of $H_K(f_i - \eta)$, and also if the Gaussian $\phi_{\Sigma_{\mathcal{R}}/2}(\eta_{ki} - f_j)$ is well interpolated by a polynomial of degree at most $2K - 1$ (which is not a big problem since the Gaussian is smooth and has, practically, compact support).

For each p_i, the sampling set $\{\eta_{ki}\}$ computed in Section 4 does not coincide with the roots of $H_K(f_i - \eta)$. Otherwise, the manifolds would be an exact copy of the original signal, and this would break the approximate linearity (among samples η_{ki} from the k-th manifold) required by *Proposition* A.2, strongly impacting the performance of our method. Nonetheless, one can still approximate the integral in Eq. (17) using a Gaussian weighting function evaluated at $\{\eta_{ki}\}$. The error introduced by this approach is expected to be small for a few reasons: (i) by using Gaussian weights one is still respecting how much each sample should contribute to the integral, according to the quadrature rule from Eq. (21); (ii) for non-outlier pixels, the set $\{\eta_{ki}\}$ computed in Section 4 provides a sampling of $\mathcal{R}$ with a density similar to the roots of $H_K(f_i - \eta)$ — *i.e.*, denser in regions close to $f_i \in \mathcal{R}$; (iii) the integrand in Eq. (17) is given by smooth Gaussian functions; and (iv) Eq. (1) defines a Gauss transform in homogeneous coordinates, which (as discussed by Adams [2011]) hides scaling errors in the division by the sum of the weights.

From Eq. (18-c), Eq. (19), and Eq. (21), it follows that the estimator for the filtered value of a pixel is given by Eq. (4), which defines a *normalized convolution interpolator* [Knutsson and Westin 1993].

B Fast Eigenvector Computation

Proposition A.3: The eigenvector v_1 associated with the largest eigenvalue of the $d_{\mathcal{R}} \times d_{\mathcal{R}}$ matrix XX^T, where $X = \mathbf{f}_k - \boldsymbol{\eta}_k$, can be approximated in $O(m\,d_{\mathcal{R}}\,N)$ time using an iterative algorithm with m iterations. This avoids the $O(d^2_{\mathcal{R}}N)$ cost of computing the matrix XX^T, which is very desirable when $m \ll d_{\mathcal{R}}$.

Proof. It can be easily proven that $\lim_{m\to\infty}(XX^T)^m w \propto v_1$ for a random vector w which is not orthogonal to v_1 (see supplementary materials). Thus, since matrix multiplication is associative,

$$(XX^T)^m w = \underbrace{(XX^T \ldots XX^T)}_{m \text{ terms } XX^T} w = (X(X^T \ldots (X\,\underbrace{(X^T w)}_{(a)}) \ldots)),$$

where the term (a) is a $O(d_{\mathcal{R}}N)$ matrix-vector multiplication which results in a vector. Repeating this multiplication until all terms XX^T are exhausted yields a (non-normalized) vector which approximates v_1. The total time complexity is $O(2\,m\,d_{\mathcal{R}}N)$. □

In our experience $m = 1$ generates great results for color bilateral filtering ($d_{\mathcal{R}} = 3$) and for non-local-means denoising ($d_{\mathcal{R}} = 6$), while $m = 2$ to 3 is best for $d_{\mathcal{R}} > 20$. Such a low number of iterations is possible since this algorithm converges quickly when the data in X is highly anisotropic (*i.e.*, some eigenvalues are considerably larger than the others). When the data is more isotropic, pixels are evenly distributed around the manifolds, and thus any vector v_1 will provide a good segmentation for Step 3 of our manifold creation process (Section 4). Finally, when filtering video sequences frame-by-frame, one should set the vector w used to filter the t-th frame equal to v_1 from the previous $(t - 1)$-th frame, to preserve temporal coherency. The first frame is filtered using a random w.