UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

EDUARDO SIMÕES LOPES GASTAL

Efficient High-Dimensional Filtering for Image and Video Processing

Dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Manuel Menezes de Oliveira Neto, Ph.D.
Advisor

Porto Alegre, March 2015

CIP – CATALOGING-IN-PUBLICATION

Gastal, Eduardo Simões Lopes

Efficient High-Dimensional Filtering for Image and Video Processing / Eduardo Simões Lopes Gastal. – Porto Alegre: PPGC da UFRGS,

.

155 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS,

. Advisor: Menezes de Oliveira Neto, Ph.D..

1. High-Dimensional Filtering, Domain Transform, Adaptive Manifolds, Non-Uniformly Sampled Signals, Geodesic Filtering, Euclidean Filtering, Hybrid Filtering. I. de Oliveira Neto, Ph.D., Menezes. II. Título.

ACKNOWLEDGEMENTS

I offer my sincere thank you to my adviser, Manuel Menezes de Oliveira Neto, for his teachings, guidance, and remarkable perseverance during the development of this work. I extend my gratitude to the faculty and staff at the Graduate Program in Computer Science (PPGC-UFRGS), especially to Elisiane da Silveira Ribeiro for her help with administrative tasks, and to Professors Cláudio R. Jung and João L. D. Comba, who shared valuable knowledge during my studies. I also thank Professors Alexandre X. Falcão and Diego Nehab for taking part in my dissertation committee and contributing helpful suggestions to this and future work.

There are no words that properly express my love and gratitude towards my family. My mother Guiomar Lourdes Gastal, my father Francisco de Paula Simões Lopes Gastal, and my godmother Aglae Gastal Giovio Irigoyen taught me to be a human being and showed me the world. My aunts Ana Maria, Marlene, and Rosangela always treated me like their own son and shaped me into the person I am today. My girlfriend Vanessa Cristina da Silva Pires showed me the meaning of companionship and strength. And of course, none of this would have been possible without the care I received, even indirectly, from my grandparents Adelina, Oreste, Graziella, and Francisco. I should also mention the indispensable and unconditional love from my canine friends: Kuki, Hanna, and Hank. You never failed to bring me into a good mood.

This work was funded by CAPES and CNPq (proc. 158666/2010-0, 142889/2010-5).

*“The growing good of the world is partly dependent on unhistoric acts;
and that things are not so ill with you and me as they might have been,
is half owing to the number who lived faithfully a hidden life,
and rest in unvisited tombs.”*

— Mary Ann Evans *a.k.a.* George Eliot

ABSTRACT

Filtering is arguably the single most important operation in image and video processing. In particular, *high-dimensional filters* are a fundamental building block for several applications, having recently received considerable attention from the research community. Unfortunately, naive implementations of such an important class of filters are too slow for many practical uses, specially in light of the ever increasing resolution of digitally captured images. This dissertation describes three novel approaches to efficiently perform high-dimensional filtering: the **domain transform**, the **adaptive manifolds**, and a mathematical formulation for recursive filtering of **non-uniformly sampled signals**.

The domain transform defines an isometry between curves on the 2D image manifold in 5D and the real line. It preserves the geodesic distance between points on these curves, adaptively warping the input signal so that high-dimensional *geodesic* filtering can be efficiently performed in linear time. Its computational cost is not affected by the choice of the filter parameters; and the resulting filters are the first to work on color images at arbitrary scales in real time, without resorting to subsampling or quantization.

The adaptive manifolds compute the filter's response at a reduced set of sampling points, and use these for interpolation at all input pixels, so that high-dimensional *Euclidean* filtering can be efficiently performed in linear time. We show that for a proper choice of sampling points, the total cost of the filtering operation is linear both in the number of pixels and in the dimension 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, providing a sound theoretical justification.

Finally, we introduce a mathematical formulation for linear-time recursive filtering of non-uniformly sampled signals. This formulation enables, for the first time, *geodesic edge-aware* evaluation of *arbitrary* recursive infinite impulse response filters (not only low-pass), which allows practically unlimited control over the shape of the filtering kernel. By providing the ability to experiment with the design and composition of new digital filters, our method has the potential do enable a greater variety of image and video effects.

The high-dimensional filters we propose provide the fastest performance (both on CPU and GPU) for a variety of real-world applications. Thus, our filters are a valuable tool for the image and video processing, computer graphics, computer vision, and computational photography communities.

Keywords: High-Dimensional Filtering, Domain Transform, Adaptive Manifolds, Non-Uniformly Sampled Signals, Geodesic Filtering, Euclidean Filtering, Hybrid Filtering.

Filtragem Eficiente em Altas-Dimensões para Processamento de Imagens e Vídeos

RESUMO

Filtragem é uma das mais importantes operações em processamento de imagens e vídeos. Em particular, *filtros de altas dimensões* são ferramentas fundamentais para diversas aplicações, tendo recebido recentemente significativa atenção de pesquisadores da área. Infelizmente, implementações ingênuas desta importante classe de filtros são demasiadamente lentas para muitos usos práticos, especialmente tendo em vista o aumento contínuo na resolução de imagens capturadas digitalmente. Esta dissertação descreve três novas abordagens para filtragem eficiente em altas dimensões: a **domain transform**, os **adaptive manifolds**, e uma formulação matemática para a aplicação de filtros recursivos em **sinais amostrados não-uniformemente**.

A domain transform, representa o estado-da-arte em termos de algoritmos para filtragem utilizando métrica geodésica. A inovação desta abordagem é a utilização de um procedimento simples de redução de dimensionalidade para implementar eficientemente filtros de alta dimensão. Isto nos permite a primeira demonstração de filtragem com preservação de arestas em tempo real para vídeos coloridos de alta resolução (full HD).

Os adaptive manifolds, representam o estado-da-arte em termos de algoritmos para filtragem utilizando métrica Euclidiana. A inovação desta abordagem é a ideia de subdividir o espaço de alta dimensão em fatias *não-lineares* de mais baixa dimensão, as quais são filtradas independentemente e finalmente interpoladas para obter uma filtragem de alta dimensão com métrica Euclidiana. Com isto obtemos diversos avanços em relação a técnicas anteriores, como filtragem mais rápida e requerendo menos memória, além da derivação do primeiro filtro Euclidiano com custo linear tanto no número de pixels da imagem (ou vídeo) quanto na dimensionalidade do espaço onde o filtro está operando.

Finalmente, introduzimos uma formulação matemática que descreve a aplicação de um filtro recursivo em sinais amostrados de maneira *não-uniforme*. Esta formulação estende a ideia de filtragem geodésica para *filtros recursivos arbitrários* (tanto passa-baixa quanto passa-alta e passa-banda). Esta extensão fornece maior controle sobre as respostas desejadas para os filtros, as quais podem então ser melhor adaptadas para aplicações específicas. Como exemplo, demonstramos—pela primeira vez na literatura—filtros geodésicos com formato Gaussiano, Laplaciana do Gaussiano, Butterworth, e Cauer, dentre outros. Com a possibilidade de se trabalhar com filtros arbitrários, nosso método permite uma nova variedade de efeitos para aplicações em imagens e vídeos.

Palavras-chave: Filtragem em Altas Dimensões, Transformação de Domínio, Variedades Adaptativas, Sinais Amostrados Não-Uniformemente, Filtragem Geodésica e Euclidiana.

LIST OF FIGURES

CONTENTS

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; DORSEY, 2002), *denoising* (BUADES; COLL; MOREL, 2005), *detail manipulation* (BAE; PARIS; DURAND, 2006; FATTAL; AGRAWALA; RUSINKIEWICZ, 2007), *upsampling* (KOPF et al., 2007), *spatio-temporal filtering* (BENNETT; MCMILLAN, 2005; RICHARDT et al., 2010), *photon-map filtering* (WEBER et al., 2004; BAUSZAT; EISEMANN; MAGNOR, 2011), *alpha matting* (GASTAL; OLIVEIRA, 2010; HE et al., 2011), *recoloring* (CHEN; PARIS; DURAND, 2007), and *stylization* (WINNEMöLLER; OLSEN; GOOCH, 2006). Existing methods are able to produce good results in many practical scenarios, and high-dimensional filters are readily available to end users in commercial and open source software (Adobe Systems Inc., 2012a,b; KIMBALL; MATTIS; GIMP Development Team, 2012).

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 (commonly called 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 (see Section 2.2.1). Thus, each filter type provides best results for different applications.

Due to their wide applicability, several high-dimensional and related filters have been proposed. The most popular one is the bilateral filter (AURICH; WEULE, 1995; SMITH; BRADY, 1997; TOMASI; MANDUCHI, 1998), which works by weight-averaging the colors of neighbor pixels based on their distances in image and color space. For 2D RGB images, it operates in a 5D space (BARASH, 2002), and a naïve implementation is too slow for many practical uses. Another popular filter is anisotropic diffusion (PERONA; MALIK, 1990). It is modeled using partial differential equations and implemented as an iterative process, which is usually equally slow. As a result, several techniques have been proposed that either try to accelerate anisotropic diffusion or bilateral filtering, or introduce alternative ways of performing similar filtering operations on images and videos. While they clearly improve performance, these solutions natively only handle grayscale images (DURAND; DORSEY, 2002; CHEN; PARIS; DURAND, 2007; YANG; TAN; AHUJA, 2009), are still not sufficiently fast for real-time applications (PARIS; DURAND, 2009; ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 2010; GREWENIG; WEICKERT; BRUHN, 2010; SUBR; SOLER; DURAND, 2009), restrict filtering to certain scales (FATTAL, 2009), or may introduce artifacts by not using true Euclidean distances (HE; SUN; TANG, 2010).

1.1 Thesis Statement

This dissertation introduces new ways to efficiently perform high-dimensional filtering with both Euclidean and geodesic metrics. We propose several filters that address the main limitations of previous techniques, in addition to providing the fastest performance (on both CPU and GPU) for a variety of real-world applications. This efficiency comes from their *linear cost* in both the number of pixels and the dimensionality of the space in which the filters operate. Thus, the central thesis statement of this research follows:

> *It is possible to perform high-dimensional filtering of images and videos in linear time in both the number of pixels and the dimensionality of the space in which the filters operate. The filter response can be computed with such complexity for all of Euclidean, geodesic, and hybrid Euclidean-geodesic metrics. Moreover, it is possible to use arbitrary recursive filters with the geodesic metric to obtain fine control over the filter's response.*

We demonstrate the validity of these claims as well as the efficiency and versatility of our filters on several *real-time* image and video processing tasks, including color edge-aware smoothing, depth-of-field effects, stylization, recoloring, colorization, detail enhancement, denoising (using up to 147 dimensions), global illumination smoothing, and tone mapping. Some of these applications are illustrated in Figure 1.1.

1.2 Publications

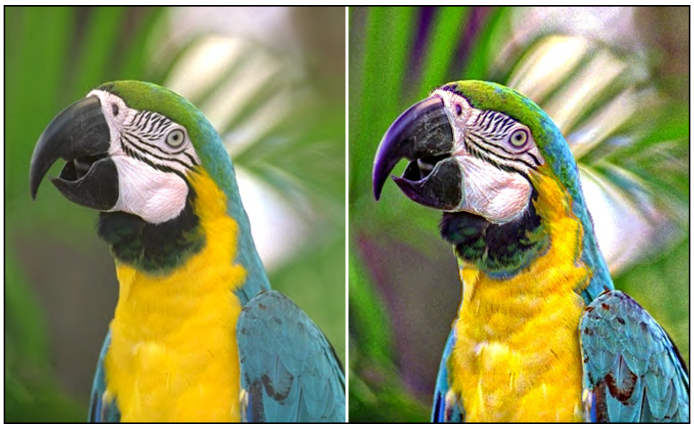
The results presented in this dissertation have been published at ACM SIGGRAPH 2011–2012 (Chapters 3 and 4), and at Eurographics 2015 (Chapter 5). Our filters have already been put to use by researchers at Adobe, NVIDIA, Google, and Disney, in applications such as color grading (BONNEEL et al., 2013), viewfinder editing for digital cameras (BAEK et al., 2013), synthetic defocus for smartphone cameras (BARRON et al., 2015), and enforcing temporal consistency in video effects (LANG et al., 2012). Thus, our filters provide a valuable tool for the image and video processing, computer graphics, computer vision, and computational photography communities.

We provide official implementations of our filters in the Appendix and in our webpages (GASTAL; OLIVEIRA, 2014a,b,c):

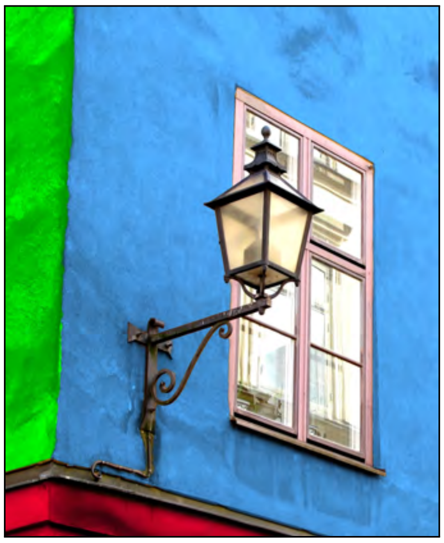
(a) Denoising

(b) Non-photorealism

(c) Stylization


(d) Detail enhancement

(e) Colorization


(f) Recoloring

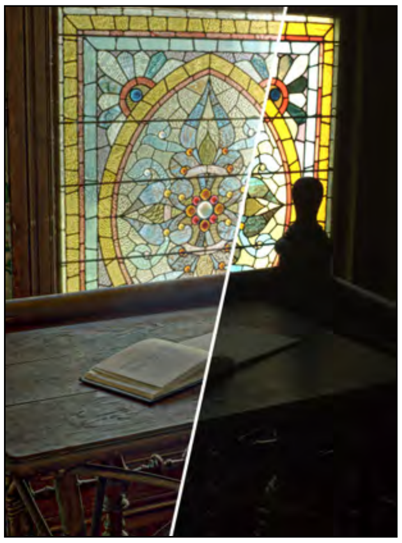

(g) HDR tone mapping

Figure 1.1: Some of the several applications of our high-dimensional filters.

http://inf.ufrgs.br/~eslgastal/DomainTransform,
http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds,
http://inf.ufrgs.br/~eslgastal/NonUniformFiltering.

An unofficial implementation of our filters is also available in the Open Source Computer Vision library (OpenCV, 2014a,b), as of August 2014.

1.3 Overview of our High-Dimensional Filters

1.3.1 Domain Transform for Geodesic Filtering

Chapter 3 presents the *Domain Transform* technique we introduced in (GASTAL; OLIVEIRA, 2011), used to efficiently perform geodesic filtering of images and videos. Its efficiency derives from a key observation: *if the geodesic distances measured on top of a manifold are preserved in a space of lower dimensionality, many translation-invariant filters in this new space will behave as geodesic filters*. Thus, treating an RGB image as a 2D manifold embedded in a 5D space, our transform defines an isometry between curves on this manifold and the real line: it preserves the geodesic distances between points on the curve, adaptively warping the input signal so that 1D geodesic filtering can be efficiently performed in linear time in the number of pixels and dimensionality of the space in which the filter operates. We demonstrate three realizations for our 1D geodesic filters, based on normalized convolution, interpolated convolution, and recursion. These filters have very distinct impulse responses, making each one more appropriate for specific applications. Finally, although our 1D filters cannot be exactly generalized to higher spatial dimensions, we show how to use them to efficiently produce high-quality nD geodesic filters.

Our domain transform approach has several desirable features. First, the use of 1D operations leads to considerable speedups over previous techniques and potential memory savings. For instance, it can filter one megapixel color images in 0.007 seconds on a GTX 280 GPU. Second, its computational cost is not affected by the choice of the filter parameters. Third, it is the first geodesic filter technique capable of working on color images at arbitrary scales in real time, without resorting to subsampling or quantization. Figure 3.1 shows a few examples of geodesic-filtering effects possible with the domain transform.

The **contributions** of our domain-transform filtering technique include:

- A novel approach for efficiently performing high-quality edge-aware filtering of images and videos based on a dimensionality-reduction strategy (Sections 3.1 and 3.2). Our approach leads to filters with several desirable features and significant speedups over previous techniques;

- A technique to perform anisotropic edge-preserving filtering on curves of the 2D image manifold using 1D linear filters. It consists of anisotropically scaling the curve, which is then mapped to the real line using an isometry, followed by the application of a 1D linear filter (Section 3.2);

- A technique to efficiently implement 2D edge-preserving smoothing filters as a sequence of 1D filtering operations (Section 3.3). The resulting 2D filters correctly handle color images and behave as expected even in extreme situations (Section 3.2.3);
- The *first* edge-preserving smoothing filter that simultaneously exhibits the following properties: (i) it supports a continuum of scales; (ii) its processing time is linear in the number of pixels, and is independent of the filter parameters, allowing for real-time computations; (iii) correctly handles color images; and (iv) offers control over the kernel's shape. For this, we show examples of approximate Gaussian and exponential responses (Section 3.4);
- A demonstration that our approach can be used to create a variety of effects for images and videos in real time (Section 5.5).

1.3.2 Adaptive Manifolds for Euclidean Filtering

Chapter 4 presents the *Adaptive Manifolds* technique we introduced in (GASTAL; OLIVEIRA, 2012), used to efficiently perform Euclidean filtering of images and videos. 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 (K typically varies from 3 to 15). Thus, the resulting filter is the *first high-dimensional Euclidean filter with linear cost both in N and in d*. We present a derivation for the equations that define our method, providing a 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; COLL; MOREL, 2005). This kind of versatility has also been described by Adams et al. (ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 2010). However, our filter is faster and requires less memory than previous approaches. For instance, we can "bilateral-like" filter a 10-Megapixel full-color image in real time (50 fps) on a GTX 580 GPU. Furthermore, the flexibility of our approach allows for the first hybrid Euclidean-geodesic filter that runs in a single pass. This allows for a filter with Euclidean response in selected dimensions of the space and geodesic response in the remaining dimensions. Figure 4.1 shows a few examples of Euclidean-filtering effects possible with the adaptive manifolds.

The **contributions** of our adaptive-manifold filtering technique include:

- An efficient approach for performing high-dimensional Euclidean filtering. Our solution produces high-quality results, and is faster and requires less memory than previous approaches (Section 4.3). 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 4.5.2);

- A theoretical justification for the method and for its properties, by means of a formal derivation of the equations that define it (Section 4.2);
- An algorithm for hierarchically computing nonlinear manifolds adapted to an input signal (Section 4.3);
- A mechanism for trading off accuracy and speed, which also lends to a filter with outlier-suppression properties (Section 4.4);
- The first demonstration of a hybrid Euclidean-geodesic filter that runs in a single pass (Section 4.7).

1.3.3 High-Order Recursive Filtering of Non-Uniformly Sampled Signals

Chapter 5 presents a discrete-time mathematical formulation for applying recursive digital filters to non-uniformly sampled signals, which we introduced in (GASTAL; OLIVEIRA, 2015). This formulation enables, for the first time, *geodesic edge-aware* evaluation of *arbitrary* recursive infinite impulse response digital filters (not only low-pass), which allows practically unlimited control over the shape of the filtering kernel. It also presents several desirable features: it preserves the stability of the original filters; is well-conditioned for low-pass, high-pass, and band-pass filters alike; its cost is linear in the number of samples and is not affected by the size of the filter support. Our method is general and works with any non-uniformly sampled signal and any recursive digital filter defined by a difference equation. Since our formulation directly uses the filter coefficients, it works out-of-the-box with existing methodologies for digital filter design. We demonstrate the effectiveness of our approach by filtering non-uniformly sampled signals in various image and video processing tasks.

The **contributions** of our non-uniform filtering formulation include:

- A discrete-time $O(r\,N)$ mathematical formulation for applying arbitrary recursive filters to non-uniformly sampled signals (Section 5.2), where N is the number of samples being filtered and r is the order of the filter;
- Two normalization schemes for filtering non-uniformly sampled signals: one based on piecewise resampling (Section 5.2.5), and the other based on spatially-variant scaling (Section 5.2.6). In edge-aware applications, our infinite impulse response (IIR) normalization schemes provide control over the filter's response to signal discontinuities (*i.e.*, edges). This was previously only possible for finite impulse response (FIR) filters;

- A general technique to obtain linear-time low-pass, high-pass, and band-pass edge-aware filters (Section 5.5.1). Our approach allows one to perform all these filters in real time;
- The first linear-time edge-aware demonstrations of several low/high/band-pass filters, including Gaussian, Laplacian of Gaussian, and Butterworth (Section 5.5);
- A demonstration of uses of non-uniform filtering in various image and video processing applications (Section 5.5), for which we discuss important details, such as common boundary conditions (Section 5.2.7), and symmetric filtering (Section 5.2.8).

2 BACKGROUND ON HIGH-DIMENSIONAL FILTERING

2.1 Notation used in this dissertation

A signal is defined by a function

$$f : \mathcal{S} \subset \mathbb{R}^{d_\mathcal{S}} \to \mathcal{R} \subset \mathbb{R}^{d_\mathcal{R}}$$

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

$$D_N(\mathcal{S}) = \{p_1, \ldots, p_N\}$$

be the set of N samples obtained by sampling $\mathcal{S}$ using a regular grid (a discretization of $\mathcal{S}$). 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.

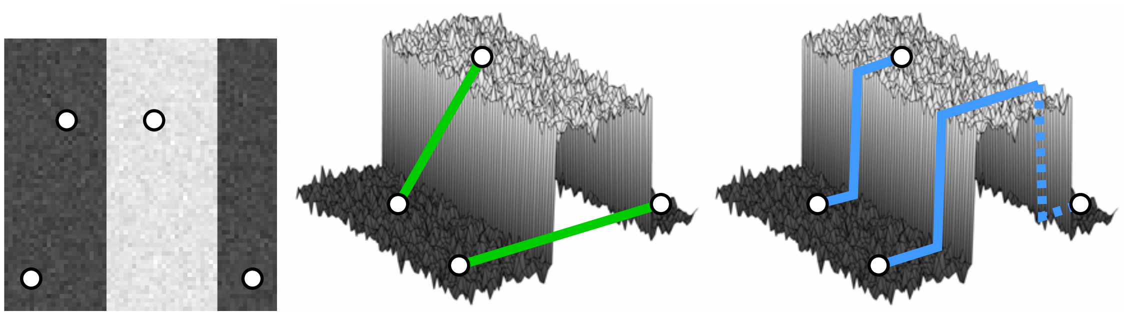
For each pixel $p_i \in \mathcal{S}$, let $\hat{p}_i \in \mathcal{S} \times \mathcal{R}$ be the point in a d-dimensional space ($d = d_\mathcal{S} + d_\mathcal{R}$) with coordinates given by the concatenation of the spatial coordinates $p_i \in \mathcal{S}$ and the range coordinates $f_i \in \mathcal{R}$. 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 $\hat{p}_i = (x_i, y_i, R_i, G_i, B_i)^T$; and in a YUV video, a pixel $p_i = (x_i, y_i, t_i)^T$ with color value $f_i = (Y_i, U_i, V_i)^T$ has $\hat{p}_i = (x_i, y_i, t_i, Y_i, U_i, V_i)^T$. This concatenation of coordinates will also be denoted as $\hat{p}_i = (p_i, f_i)$.

2.2 High-Dimensional Filtering

High-dimensional filtering produces a new set of pixel colors as weighted sums of the colors of the input pixels. The weights of this linear combination are given by a function ϕ called the *filter kernel*.Filtering a signal f with ϕ gives a new signal g. This process is expressed by the discrete high-dimensional convolution (a linear operator) of f and ϕ:

$$g_i = \sum_{p_j \in \mathcal{S}} \phi\left(\Delta(\hat{p}_i, \hat{p}_j)\right) f_j, \tag{2.1}$$

where $\Delta(\hat{p}_i, \hat{p}_j)$ is a distance-measurement function based on some specified metric. For example, the Euclidean distance would be given by $\Delta(\hat{p}_i, \hat{p}_j) = \|\hat{p}_i - \hat{p}_j\|_2$, where $\|\cdot\|_2$ is



(a) Grayscale Image (b) Euclidean Distance (c) Geodesic Distance

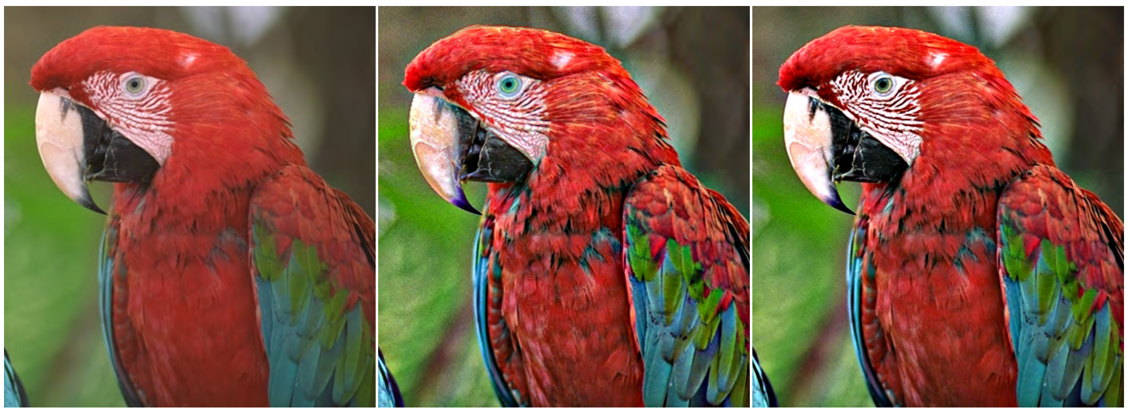
Figure 2.1: Comparison of Euclidean and geodesic distance metrics. The grayscale image in (a) is defined by a function $f : \mathbb{R}^2 \to \mathbb{R}$. It can be interpreted as a surface (2D manifold) embedded in 3D space—a heightfield. This is illustrated in (b) and (c). Note how the pixels marked by white circles in (a) sit on top of the manifold in (b-c). The Euclidean distance shown in (b) is the smallest distance in 3D space: the length of a straight line (shown in green) between the high-dimensional points. The geodesic distance shown in (c) is the smallest distance *on top of the manifold*: the length of a straight line (shown in blue) in a *curved* space (for clarity, the illustration omits the small variations present in the blue line due to the noisy manifold). Each metric provides best results for different types of applications.

the ℓ_2 norm. Furthermore, in this example, ϕ is then the *impulse response* of the filter in Euclidean space. Note that Equation 2.1 defines a high-dimensional *translation-invariant* filter: the weights defined by the kernel ϕ depend *only on the distance* between $\hat{p}_i$ and $\hat{p}_j$. Their absolute positions are irrelevant.

In the following sections, we review the relevant literature for high-dimensional filtering in image and video processing. It's important to note that previous works usually equate high-dimensional filtering with *low-pass* high-dimensional filtering. In this case, the filter kernel is designed to have unit gain at zero frequency, meaning that the DC component (average value) of the signal is preserved by the filter. In practice this is achieved by *normalizing* Equation 2.1 through a division by the sum of the weights:

$$g_i = \sum_{p_j \in \mathcal{S}} \phi\left(\Delta(\hat{p}_i, \hat{p}_j)\right)\, f_j \Bigg/ \sum_{p_j \in \mathcal{S}} \phi\left(\Delta(\hat{p}_i, \hat{p}_j)\right). \tag{2.2}$$

We focus on this normalized equation throughout Chapters 3 and 4. In such cases, similar to previous works, we build *high*-pass filters for detail manipulation by computing differences between the outputs of *low*-pass filters. In Chapter 5, however, we present a generic mathematical formulation capable of working with arbitrary digital filters (low, high and band-pass).



(a) *Photograph* (b) *Our Euclidean filter* (c) *Our geodesic filter*

Figure 2.2: Example of color detail enhancement using our geodesic and Euclidean filters: the Domain Transform (Chapter 3) and the Adaptive Manifolds (Chapter 4), respectively. Each filter enhances image features differently: note the colors of the parrot's eye. The Euclidean filter generates a result in between global and local contrast enhancement, while the result of the geodesic filter is mostly local. Choosing between the two results is a subjective choice.

2.2.1 Euclidean vs Geodesic Filters

The weights for the linear combination defined by the filter are a direct function of the distance between pixels. In other words, *the filter response changes according to the metric chosen for computing the high-dimensional distances between pixels*. In this work we focus on two types of metrics: Euclidean and geodesic.

The main difference between the two groups is the filter behavior near strong discontinuities (commonly called edges) in the signal, as illustrated in Figure 2.1. Note how the Euclidean distance between the two bottom pixels in Figure 2.1(a), marked by the white circles, is much smaller than their geodesic distance. This is mainly due to the fact that these two pixels lie on dark gray areas of the image, but between them lies a vertical band of light gray pixels. Since the geodesic distance respects the structure of the manifold, it is forced to "go over the hill" when connecting the two pixels, as shown in (c). The Euclidean metric does not know about this underlying structure, and simply connects the two pixels with a straight line in 3D space, avoiding the "hill" and computing a smaller distance.

Each metric provides different filtering response and gives best results for different types of applications. This difference is illustrated in Figure 2.2 for a simple detail enhancement example. Other examples include the fact that Euclidean response is better suited for recoloring disjoint elements in an image (CHEN; PARIS; DURAND, 2007), while geodesic response is best for adding colors to grayscale images (LEVIN; LISCHINSKI; WEISS, 2004). These are illustrated in Chapters 3 and 4.

2.2.2 Euclidean Filters

For Euclidean low-pass filters, the weights defined by the kernel decrease with the Euclidean distance. A common choice for ϕ with the Euclidean metric is an axis-aligned Gaussian function:

$$\phi_\Sigma\left(\Delta(\hat{p}_i, \hat{p}_j)\right) = \phi_\Sigma\left(\hat{p}_i - \hat{p}_j\right) = \exp\left(-\frac{1}{2}(\hat{p}_i - \hat{p}_j)^T \Sigma^{-1} (\hat{p}_i - \hat{p}_j)\right), \qquad (2.3)$$

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

If the input signal f is an image, Equations 2.2 and 2.3 describe the standard *bilateral filter* (AURICH; WEULE, 1995; SMITH; BRADY, 1997; TOMASI; MANDUCHI, 1998), which works by weighted-averaging the colors of neighbor pixels based on their (Euclidean) distances in space and range. That is, the bilateral filter only mixes pixels which are close in imagespace *and* have similar colors. For (2D) RGB images, it can be interpreted as operating in a 5D space (BARASH, 2002). Filters of this kind are commonly called *edge-preserving* smoothers, since they manage to remove small scale details from an image while preserving well-defined edges. This is illustrated in Figure 2.3.

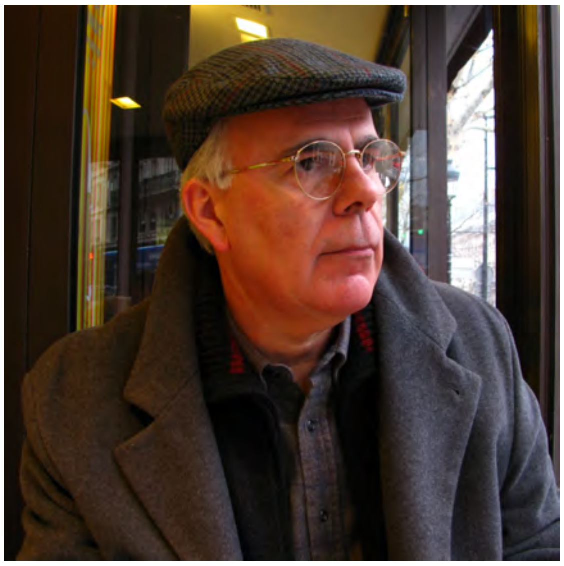
A *joint bilateral filter* (EISEMANN; DURAND, 2004; PETSCHNIGG et al., 2004) is obtained by taking the vectors $\hat{p}_i$ and $\hat{p}_j$ (used for distance computation with ϕ) from some image other than f (which may include depth and normal images (WEBER et al., 2004)). Common applications for this procedure are upsampling sparse data using a guide image (KOPF et al., 2007), colorization (GASTAL; OLIVEIRA, 2011), and image abstraction (ZHANG et al., 2014).

A *non-local-means filter* (BUADES; COLL; MOREL, 2005) is 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; MCMILLAN, 2005). This gives a new notion of distance: pixels are considered close if they have similar local neighborhoods (*i.e.*, structure) around them. This is illustrated in Figure 2.4. By describing such a local neighborhood by an $M \times M$ window around a pixel, a non-local-means filter can then be seen as working on a $(d_\mathcal{R}\, M^2 + d_\mathcal{S})$-dimensional space.

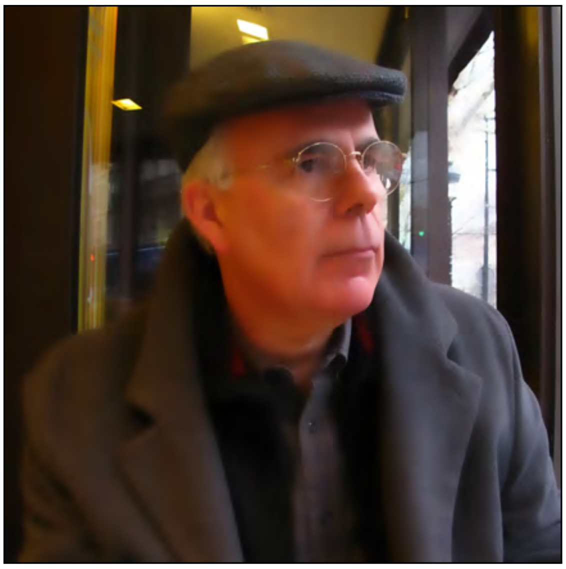
2.2.2.1 Accelerating Euclidean filters

Naïvely evaluating Equation 2.2 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.

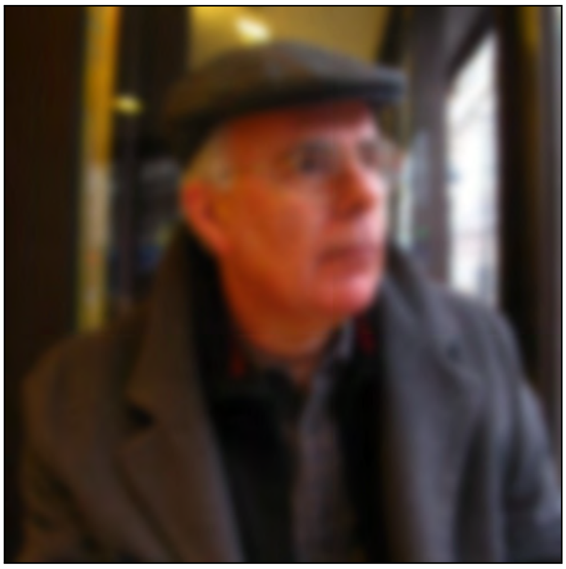
For **grayscale** image filtering, Durand and Dorsey (DURAND; DORSEY, 2002) compute the filter response by linearly interpolating several discretized range values, each filtered with a Gaussian kernel in the frequency domain. Using the Fast Fourier Transform, the best-case time complexity is superlinear $O(N \log N)$ in the number of pixels being filtered. Porikli (PORIKLI, 2008) uses summed-area tables to filter each intensity level using box filters, or polynomial approximations for the Gaussian. This improves the time



(a) Photograph



(b) Bilateral Filtering



(c) Gaussian Filtering

Figure 2.3: A photograph (a) is filtered using the edge-preserving bilateral filter (b). Pixel neighborhoods are computed in 5-D space, composed of the image coordinates and RGB pixel colors. Thus, the bilateral filter only mixes pixels which are similar in color and have similar positions in the image. Note how strong edges and the overall structure of the image are preserved, while small texture variations, such as on the gray coat, are smoothed. For comparison, a standard Gaussian filter (c) completely removes high-frequency details from the image, generating a blurred image.

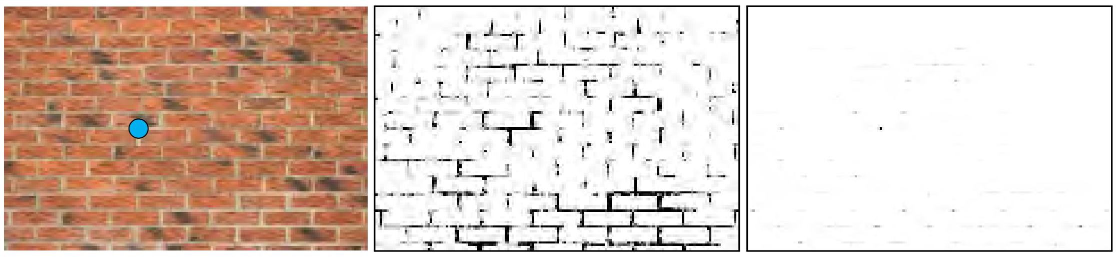

(a) Photograph (b) Bilateral similarity (c) Non-Local Similarity

Figure 2.4: Comparison of pixel similarity as computed by the bilateral filter (b) and the non-local means filter (c). Similarity is computed as the distance in the high-dimensional space between the pixel marked with the blue dot in (a) and all other pixels in the image. Darker values for similarity means the pixels are closer, in the high-dimensional space, to the marked pixel. Note how the bilateral similarity (b) only takes into account pixel colors, while the non-local similarity takes into account the structure around the marked pixel. Thus, in (c) only pixels belonging to "T" joints in the brick wall are considered as similar to the marked pixel.

complexity to linear in the number of pixels. Yang et al. (YANG; TAN; AHUJA, 2009) push the idea further, and avoid representing the entire filtering space. Recursive filters are used to process each discretized range "slice". Time complexity of all these methods grows exponentially with the dimensionality of the range $d_{\mathcal{R}}$, thus, in practice, they can only be efficiently applied to grayscale images (*i.e.*, one-dimensional range).

For **color** image filtering, Paris and Durand (PARIS; DURAND, 2009) represent the entire filtering space using a 5D *bilateral grid* and perform filtering by downsampling, which makes Equation 2.2 tractable for kernels ϕ_Σ with large support. This is possible since the bilateral filter is a high-dimensional low-pass operator, and according to the sampling theorem one can represent this (practically) band-limited signal using a small set of samples (PROAKIS; MANOLAKIS, 2007). Since the full high-dimensional space is represented in the grid structure, sometimes unneeded work is performed in empty regions of the space. A simplified version of this method was shown to perform in real-time on GPUs for three-dimensional (grayscale) bilateral filtering (CHEN; PARIS; DURAND, 2007). Worst-case performance for the bilateral grid is exponential in the dimensionality of the range: $O(N2^{d_{\mathcal{R}}})$. Pham and van Vliet (PHAM; VAN VLIET, 2005) approximate Equation 2.2 for small 2D kernels as two separate 1D kernels (in the imagespace). While this process does not approximate the true 2D bilateral filter, it was successfully applied to video preprocessing tasks such as abstraction. Worst-case performance for this decomposition is still quadratic in the number of pixels.

For **higher-dimensional** filtering, Adams et al. (ADAMS et al., 2009) use a *kd-tree* with efficient Monte-Carlo sampling to sparsely gather the filter response. Querying this data-structure leads to superlinear complexity in the number of pixels being filtered. Adams et al. (ADAMS; BAEK; DAVIS, 2010) use a *permutohedral lattice* to tessellate the high-dimensional space using uniform simplicies, and a hash table to store pixel colors. This effectively avoids performing unneeded work in empty regions of the high-dimensional space. Other approaches, such as the *guided filter* (HE; SUN; TANG, 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; EISEMANN; MAGNOR, 2011), among others. 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.

2.2.3 Geodesic Filters

A popular approach for edge-aware image processing is the anisotropic diffusion (AD) operator (PERONA; MALIK, 1990). It is modeled using a partial differential equation (PDE) which is a modified version of a linear diffusion process (*i.e.*, a Gaussian low-pass). In 1D we have

$$\frac{\partial}{\partial t}u(x,t) = \frac{\partial}{\partial x}\left(s(u,x,t)\,\frac{\partial}{\partial x}u(x,t)\right), \tag{2.4}$$

with initial condition given by the input signal $u(x,0) = f(x)$, and the filtered output given by $g(x) = u(x,t^*)$ for a chosen time $t^* > 0$. $s(u,x,t)$ is the *edge-stopping* function, which should have a small value near strong edges in the signal. To understand the intuition behind Equation 2.4, note that where $s(u,x,t) = 0$ we have $\frac{\partial}{\partial t}u(x,t) = 0$, which means no diffusion is performed (*i.e.*, $u(x,t)$ does not vary in time); and where $s(u,x,t) = 1$ we have $\frac{\partial}{\partial t}u(x,t) = \frac{\partial^2}{\partial x^2}u(x,t)$, which means linear diffusion (Gaussian smoothing) is performed.

The PDE in Equation 2.4 is implemented as an iterative process, which is usually slow. Some approaches have been proposed to improve the speed of anisotropic diffusion (WEICKERT; ROMENY; VIERGEVER, 1998; GREWENIG; WEICKERT; BRUHN, 2010). However, they do so at the cost of accuracy and still hardly achieve interactive performance. Kimmel et al. (KIMMEL; SOCHEN; MALLADI, 1997) generalized many diffusion processes through the use of Beltrami flow.

Farbman et al. (FARBMAN et al., 2008) perform edge-preserving smoothing using a weighted least squares framework. They solve a sparse linear system derived from the following quadratic minimization problem:

$$g = \underset{u}{\text{argmin}}\ (u-f)^2 + \lambda \left(\frac{\partial u}{\partial x}\right)^2 \bigg/ \left(\frac{\partial f}{\partial x}\right).$$

The first term $(u-f)^2$ encourages a solution u close to the input image f, the term $\left(\frac{\partial u}{\partial x}\right)^2$ minimizes the magnitude of the derivatives of the output (*i.e.*, smoothes the result), but only where the input image does not have strong edges (*i.e.*, the term $\cdot \Big/ \left(\frac{\partial f}{\partial x}\right)$). The parameter λ controls the amount of smoothing. The solution of a sparse linear system limits the performance of the technique.

The solution of linear systems has also been employed by Levin et al. (LEVIN; LISCHINSKI; WEISS, 2004) for image colorization, and by Subr et al. (SUBR; SOLER; DURAND, 2009) for multiscale image decomposition. More recently, Fattal (FATTAL, 2009) proposed a new family of edge avoiding wavelets (EAW). This multiscale representation can be quickly computed, but constrains the sizes of the smoothing kernels (in pixels) to powers of two. Criminisi et al. (CRIMINISI et al., 2010) presented a geodesic framework for edge-aware filtering defined for grayscale images that employs quantization of the luma channel.

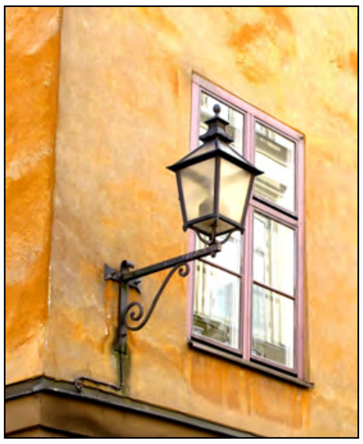
3 DOMAIN TRANSFORM FOR GEODESIC FILTERING

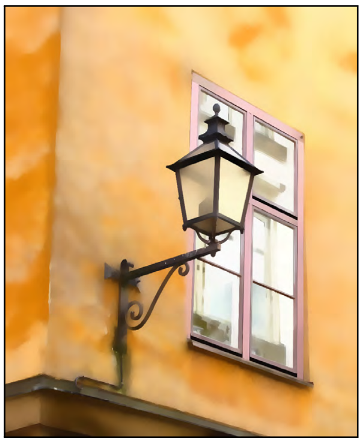
This chapter presents a new approach for efficiently performing edge-preserving geodesic filtering of images and videos that addresses the main limitations of previous techniques. Our approach is based on a novel **domain transform** and its efficiency derives from a key observation: an RGB image is a 2D manifold in a 5D space, and an edge-preserving filter can be defined as a 5D translation-invariant kernel, whose response decreases as the distances among pixels increase in 5D (see Section 2.2). *If these distances are preserved in a space of lower dimensionality, many translation-invariant filters in this new space will also be edge-preserving*. This observation is true since the weights defined by the filter kernel depend only on the distances between pixels, and not their absolute positions in space (Equation 2.1). Thus, one is free to compute new coordinates for the pixels, as long as their positions relative to each other are preserved. By choosing new coordinates in a lower-dimensional space, we are able to significantly improve the time performance of edge-preserving filtering.

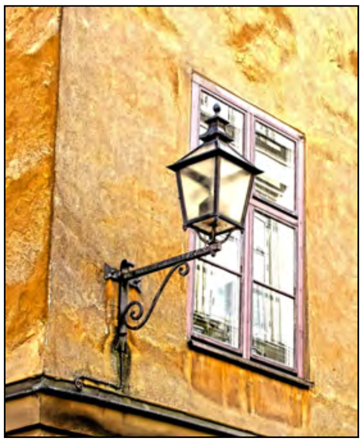
The domain transform defines an isometry between curves on the 2D image manifold and the real line. It preserves the geodesic distances between points on the curve, adaptively warping the input signal so that 1D edge-preserving filtering can be efficiently performed in linear time. We demonstrate three realizations for our 1D geodesic filters, based on normalized convolution, interpolated convolution, and recursion. These filters have very distinct responses, making each one more appropriate for specific applications. Finally, although our 1D filters cannot be exactly generalized to higher dimensions, we show how to use them to efficiently produce high-quality 2D edge-preserving filters.

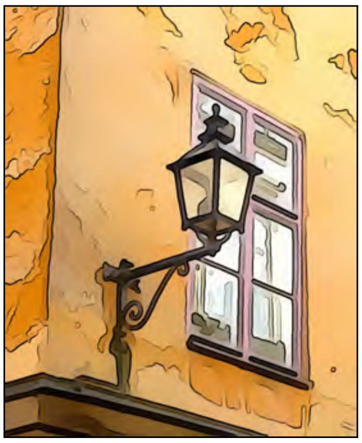
Our approach has several desirable features. First, the use of 1D operations leads to considerable speedups over existing techniques and potential memory savings. For instance, it can filter one-megapixel color images in 0.007 seconds on a GeForce GTX 280 GPU. Second, its computational cost is not affected by the choice of the filter parameters. Third, it is the first edge-preserving technique capable of working on color images at arbitrary scales in real time, without resorting to subsampling or quantization.

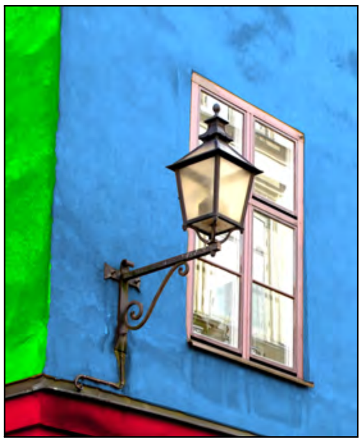
We demonstrate the versatility of our domain transform and edge-preserving geodesic filters on several real-time image and video processing tasks including edge-preserving smoothing, depth-of-field effects, stylization, recoloring, colorization, detail enhance-

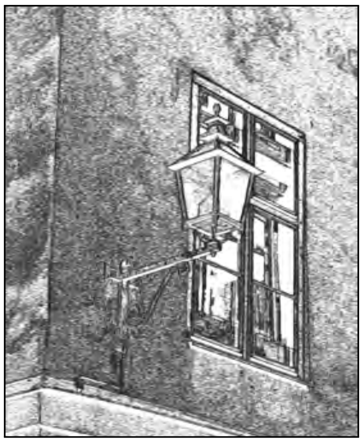

(a) Photograph


(b) Edge-aware smoothing


(c) Detail enhancement


(d) Stylization


(e) Recoloring


(f) Pencil drawing

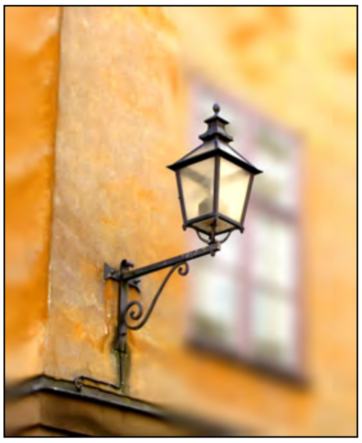

(g) Depth-of-field simulation

Figure 3.1: A variety of effects illustrating the versatility of our domain transform and edge-preserving filters applied to the photograph in *(a)*.

ment, and tone mapping (Section 5.5). Examples of some of these effects can be seen in Figure 3.1, applied to the photograph shown in (a).

3.1 Transform for Edge-Preserving Filtering

Our approach is inspired by the multi-dimensional interpretation of edge-preserving filters (BARASH, 2002). Let $f : \mathcal{S} \subset \mathbb{R}^2 \to \mathbb{R}^3$ be a 2D RGB color image, defining a 2D manifold M_f in $\mathbb{R}^5$ (KIMMEL; SOCHEN; MALLADI, 1997). Thus, $\hat{p}_i = (p_i, f_i) \in M_f$ is a point on this manifold. Let F be an edge-preserving translation-invariant filter kernel in 5D. The image g obtained when filtering f with F can be expressed as the linear combination

$$g_i = \sum_{p_j \in D_N(\mathcal{S})} F\left(\hat{p}_i, \hat{p}_j\right)\, f_j, \tag{3.1}$$

where the kernel F is assumed to be normalized: $\sum_{p_j \in D_N(\mathcal{S})} F(\hat{p}_i, \hat{p}_j) = 1$. As discussed in Section 2.2.2, the bilateral filter kernel (disregarding normalization) is given by a Gaussian function:

$$F\left(\hat{p}_i, \hat{p}_j\right) = \phi_\Sigma\left(\hat{p}_i - \hat{p}_j\right) = \exp\left(-\frac{1}{2}(\hat{p}_i - \hat{p}_j)^T \Sigma^{-1} (\hat{p}_i - \hat{p}_j)\right), \tag{3.2}$$

where Σ is a 5×5 diagonal covariance matrix that controls, for each dimension, how fast the weights decrease with distance. Σ is commonly defined using only two parameters σ_s and σ_r (respectively referred to as the spatial and range standard deviations of the kernel):

$$\Sigma = \begin{bmatrix} \sigma_s^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_s^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_r^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_r^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_r^2 \end{bmatrix}. \tag{3.3}$$

Since the bilateral filter works in 5D space, its naive implementation is too slow for many practical uses.

3.1.1 Problem Statement

Our work addresses the fundamental question of *whether there exists a transformation* $t : \mathbb{R}^5 \to \mathbb{R}^l$*,* $l < 5$*, and a filter kernel* H *defined over* $\mathbb{R}^l$ *that, for any input image* f*, produce an equivalent result as the 5D edge-preserving kernel* F:

$$\sum_{p_j \in D_N(\mathcal{S})} F\left(\hat{p}_i, \hat{p}_j\right)\, f_j = \sum_{p_j \in D_N(\mathcal{S})} H\left(\, t(\hat{p}_i),\, t(\hat{p}_j)\,\right)\, f_j. \tag{3.4}$$

This construction becomes attractive when evaluating t plus H is more efficient than evaluating the original kernel F. In our case, we are interested in replacing the evaluation of a computationally expensive edge-preserving filter defined in 5D with a transformation

t and a lower-dimensional linear filter H, evaluated in $\mathbb{R}^l, l < 5$. While one could try to exactly mimic the response of a specific filter F (*e.g.*, anisotropic diffusion or the bilateral filter), in this work we instead focus on finding a transformation that *maintains the edge-preserving property of the filter*.

3.1.2 Distance-Preserving Transforms

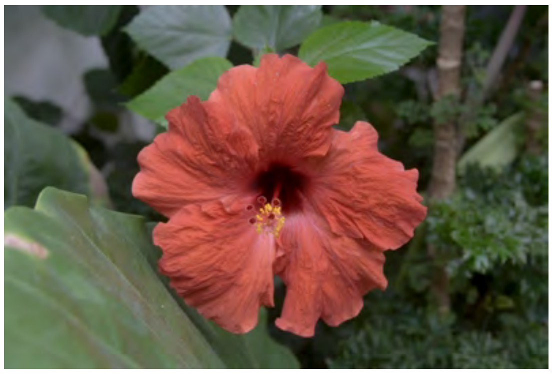
When performing edge-preserving smoothing, the amount of mixing between two pixels should be decrease with their distance, which can be expressed in any metric in the 5D space (FARBMAN; FATTAL; LISCHINSKI, 2010). For example, the bilateral filter uses the ℓ_2 norm (CHEN; PARIS; DURAND, 2007), while Criminisi et al. (CRIMINISI et al., 2010) use the intrinsic (geodesic) distance on the image manifold. If the transformation t preserves the original distances from $\mathbb{R}^5$ in $\mathbb{R}^l$, it will also maintain the edge-preserving property of a filter defined in the lower-dimensional space.

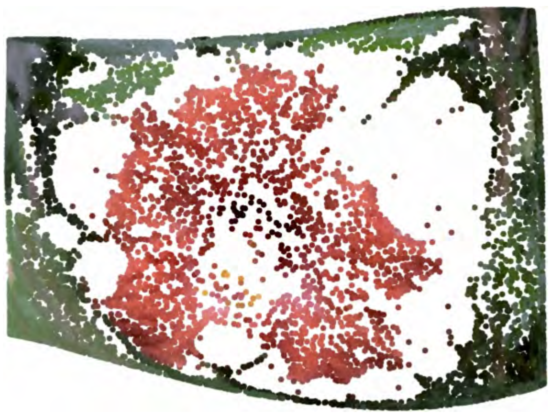
A distance-preserving transformation is known as an *isometry* (O'NEILL, 2006), and finding one is not an easy task. Let us consider the case of mapping a grayscale image to a plane, which involves finding an isometry $t : \mathbb{R}^3 \rightarrow \mathbb{R}^2$. This can be visualized as trying to flatten a heightfield without introducing any metric distortions. Unfortunately, it is known that such mappings do not exist in general (they only exist for surfaces with zero Gaussian curvature) (O'NEILL, 2006), and only approximate solutions can be found.

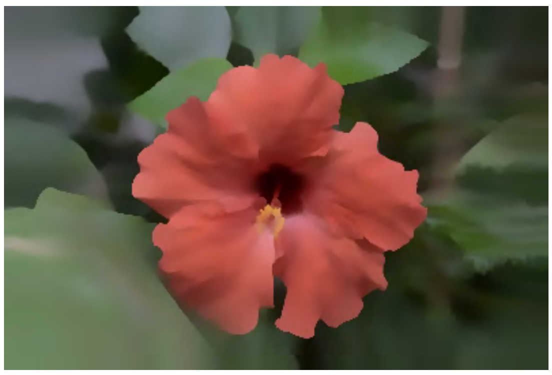
For our purpose of edge-aware filtering, preserving the distances among pixels is essential. Solutions which fail to correctly preserve the relative position between pixels introduce hard to predict and image-dependent errors, as shown in Figure 3.2. Furthermore, existing approaches from the dimensionality-reduction (BELKIN; NIYOGI, 2003) and texture-mapping (LÉVY et al., 2002) literature use optimization methods, which are too slow for our use in real-time edge-preserving filtering. While a solution for 2D and above spatial domains ($d_\mathcal{S} \geq 2$) does not exist in general, Section 3.2 shows that an isometric transform exists for a 1D spatial domain ($d_\mathcal{S} = 1$) *if distances are measured geodesically*. Section 3.3 then shows how this 1D transform can be effectively used to filter 2D color images, as well as higher dimensions: by performing separate passes along each spatial dimension of the signal. Note that this approach works for all dimensionality of the range — *i.e.*, all values of $d_\mathcal{R}$ are treated correctly.

3.2 The Domain Transform

For deriving an isometric 1D transform, let $f : \mathcal{S} \rightarrow \mathbb{R}$, $\mathcal{S} = [0, +\infty) \subset \mathbb{R}$, be a 1D signal, which defines a curve C in $\mathbb{R}^2$ by the graph $(x, f(x))$, for $x \in \mathcal{S}$ (Figure 3.3, left). Our goal is to find a transform $t : \mathbb{R}^2 \rightarrow \mathbb{R}$ which preserves, in $\mathbb{R}$, the original distances between points on C, given by some metric. Thus, let the sampling $D_N(\mathcal{S}) = \{x_1, x_2, \ldots, x_N\}$ of $\mathcal{S}$ be defined as $x_{i+1} = x_i + h$, for some sampling interval h. Note


(a) Input photograph (400x267 pixels)


(b) 5D space reduced to 2D using Laplacian Eigenmaps (BELKIN; NIYOGI, 2003). Only 10% of pixels are shown.


(c) Result produced by a 2D Gaussian filter applied to the domain in (b)

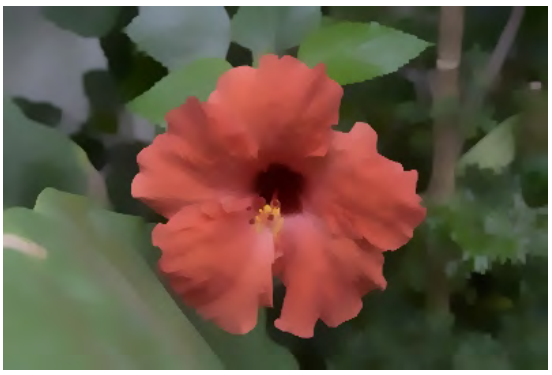

(d) Result produced by our NC filter with $\sigma_s = 8$ and $\sigma_r = 0.4$

Figure 3.2: Dimensionality reduction techniques could be used to find an approximate isometric transformation. However, such approximations lead to artifacts in the filtered image. For this example we used Laplacian Eigenmaps (BELKIN; NIYOGI, 2003) to reduce the dimensionality of the color image in (a) from 5D to 2D. Notice how the red flower got clustered into the middle of the domain, and the background foliage got compressed to the sides. When filtering the dimensionality-reduced image with a 2D Gaussian kernel, as shown in (c), edges in the background are not correctly preserved (the background is mostly blurred), as in the case of the branch and foliage edges on the right of the image. Generating the dimensionality-reduced image shown in (b) took 25 minutes on a MATLAB implementation. Our approach takes 0.015 seconds (using C++) to produce its result, shown in (d), and correctly handles all image edges. Finally, we note that approximate solutions for the transformation produce worse results for more complex images. An animated GIF image, flipping between (c) and (d), can be found at http://inf.ufrgs.br/~eslgastal/DomainTransform/Approximate_2D_Transform_Example/flower_flip.gif. This animation shows how our NC filter correctly preserves the background edges.

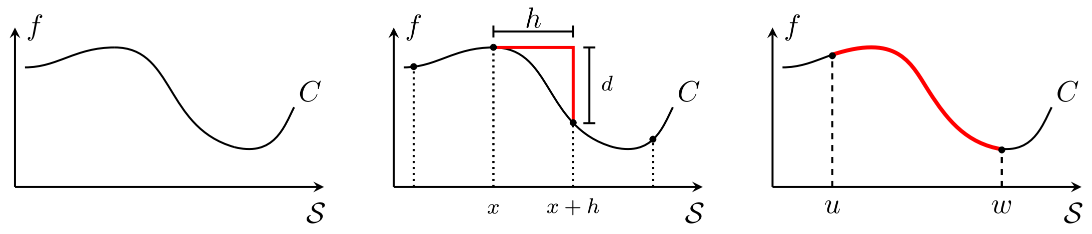


Figure 3.3: *(Left)* Curve C defined by the graph $(x, f(x))$, $x \in \mathcal{S}$. *(Center)* In ℓ_1 norm, $\|(x+h, f(x+h)) - (x, f(x))\|_1 = h + d = h + |f(x+h) - f(x)|$. *(Right)* Arc length of C, from u to w.

that $\{x_i\}$ represent 1D pixel positions, *i.e.* $p_i = (x_i)$. We seek a transform t that satisfies

$$|t(x_i, f_i) - t(x_j, f_j)| = \Delta\left((x_i, f_i), (x_j, f_j)\right), \tag{3.5}$$

where $x_i, x_j \in D_N(\mathcal{S})$, $f_i = f(x_i)$, $|\cdot|$ is the absolute value operator, and $\Delta(\cdot,\cdot)$ is some chosen metric. In this work, we use the nearest-neighbor ℓ_1 norm; thus, t only needs to preserve the distances between neighboring samples x_i and x_{i+1}. As we will soon show, this choice gives rise to the geodesic metric. Finally, let $\mathcal{D}t : \mathbb{R} \to \mathbb{R}$ be defined as $\mathcal{D}t(x) = t(\hat{x}) = t(x, f(x))$. To be isometric, the desired transform must satisfy the following equality (in ℓ_1 norm) (Figure 3.3, center):

$$\mathcal{D}t(x+h) - \mathcal{D}t(x) = h + |f(x+h) - f(x)|, \tag{3.6}$$

which states that the distance between neighboring samples in the new domain ($\mathbb{R}$, on the left-hand-side of the Equation) must equal the ℓ_1 distance between them in the original domain ($\mathbb{R}^2$, on the right-hand-side of the Equation). To avoid the need for the absolute value operator on the left of (3.6), we constrain $\mathcal{D}t$ to be monotonically increasing — *i.e.*, $\mathcal{D}t(x+h) \geq \mathcal{D}t(x)$. Dividing both sides of Equation 3.6 by h and taking the limit as $h \to 0$ we obtain

$$\mathcal{D}t'(x) = 1 + |f'(x)|, \tag{3.7}$$

where $\mathcal{D}t'(x)$ denotes the derivative of $\mathcal{D}t(x)$ with respect to x. Integrating Equation 3.7 on both sides and defining $\mathcal{D}t(0) = 0$, we get

$$\mathcal{D}t(u) = \int_0^u 1 + |f'(x)|\ dx, \quad u \in \mathcal{S}. \tag{3.8}$$

We assume f is everywhere differentiable and thus Equation 3.8 is well defined. In image and video processing this is not a significant constraint since f is sampled (discretized). In this case, derivatives are finite and must be approximated somehow (we use backward differences—see Section 3.4 and the discussion on denoising in Section 3.5.2).

Intuitively, $\mathcal{D}t$ is "unfolding" the curve C defined in $\mathbb{R}^2$ (Figure 3.3, left) into $\mathbb{R}$, while preserving the distances among neighboring samples. Moreover, for any two points u and w in $\mathcal{S}$, $w \geq u$, the distance between them in the new domain is given by

$$\mathcal{D}t(w) - \mathcal{D}t(u) = \int_u^w 1 + |f'(x)|\ dx, \tag{3.9}$$

which is the arc length of curve C in the interval $[u, w]$, under the ℓ_1 norm (Figure 3.3, right). As such, the transformation given by Equation 3.8 *preserves the geodesic distance between all points on the curve*. A similar derivation is possible for the ℓ_2 norm (as shown in Section 3.2.4).

3.2.1 Multichannel Signals

For edge-preserving filtering, it is important to process all channels of the input signal at once, as processing them independently will introduce artifacts around the edges (TOMASI; MANDUCHI, 1998).

A 1D signal $f : \mathcal{S} \subset \mathbb{R} \to \mathcal{R} \subset \mathbb{R}^{d_\mathcal{R}}$ has $d_\mathcal{R}$ "channels":

$$f(x) = \left(f_{[1]}(x),\, f_{[2]}(x),\, \ldots\, ,\, f_{[d_\mathcal{R}]}(x) \right)^T .$$

In the case f is an image, the k-th channel $f_{[k]}$ can be a color channel in some color space (*e.g.*, RGB or CIE Lab), or a more complex representation, such as a diffusion map (FARBMAN; FATTAL; LISCHINSKI, 2010).

The signal f defines a curve C in $\mathbb{R}^{d_\mathcal{R}+1}$. Applying a derivation similar to the one in Section 3.2, one obtains the multichannel transformation:

$$\mathcal{D}t(u) = \int_0^u 1 + \sum_{k=1}^{d_\mathcal{R}} \left| f'_{[k]}(x) \right| \, dx,$$

which can be written more compactly as

$$\mathcal{D}t(u) = \int_0^u 1 + \left\| f'(x) \right\|_1 \, dx, \tag{3.10}$$

(where $\| \cdot \|_1$ is the ℓ_1 norm) since the derivative of a vector-valued function is computed by differentiating its components (*i.e.*, channels):

$$\begin{aligned} \|f'(x)\|_1 &= \left\| \left(f'_{[1]}(x),\, f'_{[2]}(x),\, \ldots\, ,\, f'_{[d_\mathcal{R}]}(x) \right)^T \right\|_1 \\ &= |f'_{[1]}(x)| + |f'_{[2]}(x)| + \ldots + |f'_{[d_\mathcal{R}]}(x)| \\ &= \sum_{k=1}^{d_\mathcal{R}} \left| f'_{[k]}(x) \right| . \end{aligned}$$

Equation 3.10 defines a warping $\mathcal{D}t : \mathcal{S} \to \mathcal{S}_w$ of the signal's 1D spatial domain by the isometric transform $t : \mathbb{R}^{d_\mathcal{R}+1} \to \mathbb{R}$, where

$$\mathcal{D}t(u) = t(\hat{u}) = t(u, f_{[1]}(u), \ldots, f_{[d_\mathcal{R}]}(u)).$$

We call $\mathcal{D}t$ a *domain transform*.

3.2.2 Application to Edge-Preserving Filtering

Equation 3.10 reduces the evaluation domain of the filter from $\mathbb{R}^{d_\mathcal{R}+1}$ to $\mathbb{R}$. Thus, the filter H (see Equation 3.4) is one-dimensional. Since our transformation is isometric, any filter H, whose response decreases with distance at least as fast as F's, will be edge-preserving. Section 3.4 discusses some choices of H. By reducing the dimensionality of the filter from $d_{\mathcal{R}} + 1$ to 1, it may seem that we lost the ability to control its support over the signal's space and range (*i.e.*, control the values of σ_s and σ_r). But, as we show, one can encode the values of σ_s and σ_r in the transformation itself.

Given a 1D signal f and a 1D normalized filtering kernel H (*e.g.*, with unit area), we can define $f_a(u) = f(u/a)$, which stretches/shrinks f by a, and $H_{1/a}(u - \tau) = a\,H(au - \tau)$, which shrinks/stretches H by $1/a$ and renormalizes it to unit area, where τ is a translation. One can confirm that $H_{1/a}$ is normalized:

$$\begin{aligned}
\int_{-\infty}^{+\infty} H_{1/a}(u-\tau)\,du &= \int_{-\infty}^{+\infty} H(au-\tau)\,a\,du \\
(\text{Substitute } w = au-\tau) \quad &= \int_{-\infty}^{+\infty} H(w)\,a\,a^{-1}\,dw \\
&= \int_{-\infty}^{+\infty} H(w)\,dw \\
&\overset{\text{def}}{=} 1,
\end{aligned}$$

since H has unit area by definition.

Representing the convolution operator by $*$, one verifies that

$$\begin{aligned}
(f * H_{1/a})(\tau) &\overset{\text{def}}{=} \int_{-\infty}^{+\infty} f(u)\,H_{1/a}(u-\tau)\,du \\
&= \int_{-\infty}^{+\infty} f_a(au)\,H(au-\tau)\,a\,du \\
(\text{Substitute } au = w) \quad &= \int_{-\infty}^{+\infty} f_a(w)\,H(w-\tau)\,a\,a^{-1}\,dw \\
&= \int_{-\infty}^{\infty} f_a(w)\,H(w-\tau)\,dw \\
&\overset{\text{def}}{=} (f_a * H)(\tau),
\end{aligned}$$

or more compactly:

$$(f * H_{1/a})(\tau) = (f_a * H)(\tau). \tag{3.11}$$

Thus, *under convolution, scaling the filter's support by $1/a$ is equivalent to scaling the signal's support by a (and vice-versa).* This is an important observation, since it shows that the support of the original multi-dimensional kernel F can be completely encoded in 1D. Thus, to encode the filter's support onto the domain transform filtering operation, we:

1. Derive a_i, for each dimension d_i of the signal, from the desired support of filter F over d_i — *i.e.*, one needs to find the value of a_i which makes filtering the scaled signal f_a with filter H equivalent to filtering the original signal f with filter F;

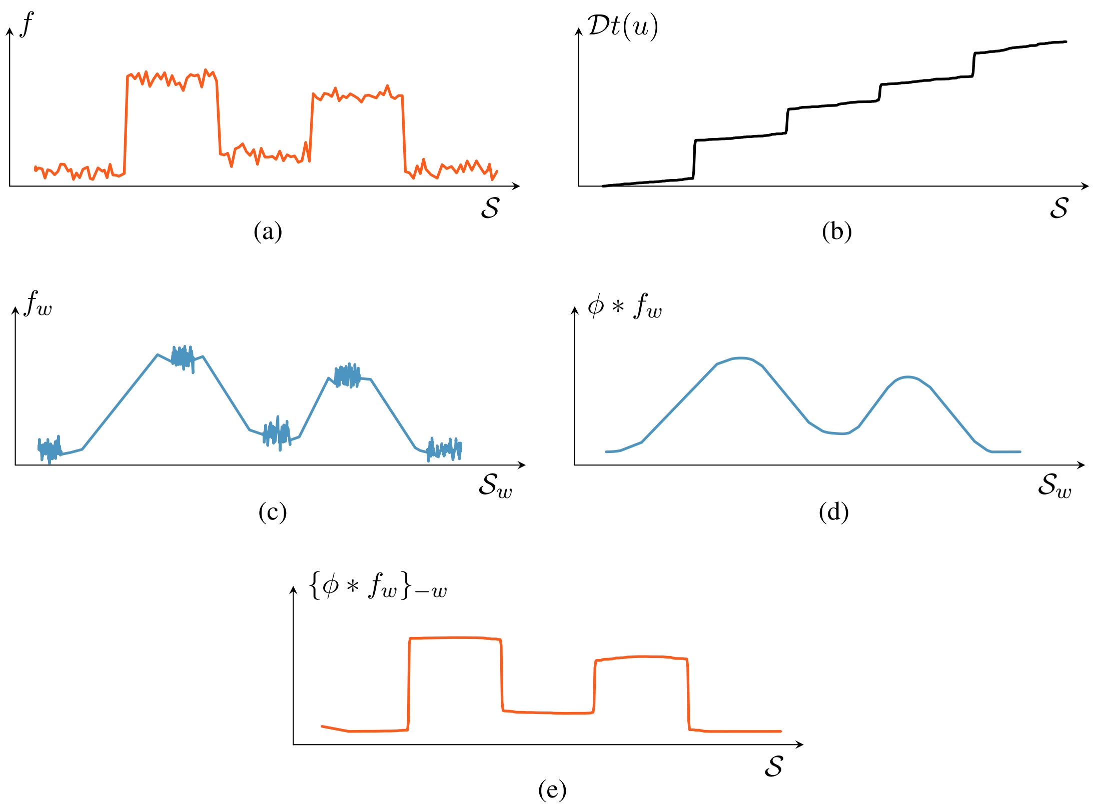


Figure 3.4: 1D edge-preserving filtering using $\mathcal{D}t(u)$. *(a)* Input signal f. *(b)* The domain transform $\mathcal{D}t(u), u \in \mathcal{S}$. *(c)* Signal f plotted in the transformed domain ($\mathcal{S}_w$). *(d)* Signal f filtered in the transformed domain $\mathcal{S}_w$ with a 1D Gaussian ϕ *(e)* and plotted in $\mathcal{S}$.

2. Scale each dimension d_i of the signal by its corresponding a_i, obtaining the scaled signal f_a;
3. Compute and apply the domain transform to the scaled signal f_a;
4. Filter the scaled signal in 1D using H on the new domain;
5. Reverse the domain transform to obtain the resulting filtered signal.

Figure 3.4 illustrates these steps: the use of a domain transform for filtering the 1D signal f, shown in (a) in its original domain $\mathcal{S}$. (b) shows the associated domain transform $\mathcal{D}t(u)$ (computed using Equation 3.12, described below). (c) shows signal f in the transformed domain $\mathcal{S}_w$ or, more compactly, $f_w(\mathcal{D}t(u)) = f(u)$. The result of filtering f with a Gaussian filter H in $\mathcal{S}_w$ is shown in (d). (e) shows the desired filtered signal obtained by reversing $\mathcal{D}t(u)$ for the signal shown in (d). The small-scale variations were eliminated and the strong edges preserved. In practice the parameter σ_s controls how strong an edge needs to be in order for it to get preserved in the filtered image (TOMASI; MANDUCHI, 1998).

The remaining of this section discusses how to compute the appropriate scaling factors and how to use them in scaling the signal. We will refer to the desired variances of the filter F over the signal's spatial domain $\mathcal{S}$ as σ_s^2, and over the signal's range as $\sigma_{r_k}^2$, for all channels $k \in \{1, \ldots, d_{\mathcal{R}}\}$.

Obtaining the scaling factors Since filtering f_a with H is equivalent to filtering f with $H_{1/a}$ (Equation 3.11), we need to find the scaling factors (for each dimension) that make $\mathrm{Var}(H_{1/a}) = \mathrm{Var}(F)$. Thus, let σ_H^2 be the variance of the filter H. Note that σ_H^2 is a free parameter. By the scaling property of variances (LOEVE, 1977), one can derive the scaling factor a_s for the spatial dimensions as:

$$\begin{aligned} & \mathrm{Var}(H_{1/a_s}) = \mathrm{Var}_s(F) \\ \Longrightarrow \quad & \mathrm{Var}(H_{1/a_s}) = \sigma_s^2 \\ \Longrightarrow \quad & \mathrm{Var}(H/a_s) = \sigma_s^2 \\ \Longrightarrow \quad & \mathrm{Var}(H)/a_s^2 = \sigma_s^2 \\ \Longrightarrow \quad & \sigma_H^2/a_s^2 = \sigma_s^2 \\ \Longrightarrow \quad & a_s = \sigma_H/\sigma_s. \end{aligned}$$

A similar derivation yields the scaling factors for the range dimensions:

$$a_{r_k} = \sigma_H/\sigma_{r_k}.$$

Note that the scaling factors 'a' may vary for each dimension in $\mathcal{S} \times \mathcal{R} \subset \mathbb{R}^{1+d_{\mathcal{R}}}$, allowing the definition of anisotropic filtering in 1D. These results are valid for *any* value $\sigma_H > 0$.

Scaling the signal According to Equation 3.11, before it can be filtered by H, the signal f should be scaled by 'a' prior to evaluating the domain transform, resulting in f_a. Scaling the distances in the right-hand side of Equation 3.6 by the appropriate a factors (*i.e.*, $a_s h + a_r\,|f(x+h) - f(x)|$) and carrying out the derivation, one obtains

$$\mathcal{D}t(u) = \int_0^u \frac{\sigma_H}{\sigma_s} + \sum_{k=1}^{d_{\mathcal{R}}} \frac{\sigma_H}{\sigma_{r_k}} \left| f'_{[k]}(x) \right| \, dx.$$

Since σ_H is a free parameter, we let $\sigma_H = \sigma_s$ and obtain our final **domain transform**, where a single value of σ_r has been used for all channels for simplicity:

$$\mathcal{D}t(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \left\| f'(x) \right\|_1 \, dx. \tag{3.12}$$

Filtering the signal in the transformed domain is done through 1D convolution with H. Further details are presented in Section 3.4.

3.2.3 Analysis

This section analyzes the filtering-related properties of our domain transform (Equation 3.12). As $\mathcal{D}t(x)$ is applied to a 1D signal f, its domain is *locally* scaled by

$$\mathcal{D}t'(x) = 1 + \frac{\sigma_s}{\sigma_r} |f'(x)|, \tag{3.13}$$

where the summation over all channels has been omitted for simplicity. By making the substitution $a = \mathcal{D}t'(x)$ in Equation 3.11, one can see that scaling the input signal f by

$\mathcal{D}t'(x)$ is equivalent to scaling the support of the filter H by $1/\mathcal{D}t'(x)$. Thus, the amount of local smoothing introduced by H in the signal at $f(x)$, can be expressed as

$$\text{smoothing}_H(x) \propto \left(\sigma_H \Big/ \mathcal{D}t'(x)\right) = \sigma_s \Bigg/ \left(1 + \frac{\sigma_s}{\sigma_r}\,|f'(x)|\right). \tag{3.14}$$

Using Equation 3.14, we analyze the relationship of H's response with the parameters σ_s and σ_r, as well as with $f(x)$, using the following limits:

$$\begin{aligned}
&\lim_{\sigma_r \to \infty} \text{smoothing}_H(x) = \sigma_s, && \lim_{\sigma_r \to 0} \text{smoothing}_H(x) = 0,\\
&\lim_{\sigma_s \to \infty} \text{smoothing}_H(x) = \frac{\sigma_r}{|f'(x)|}, && \lim_{\sigma_s \to 0} \text{smoothing}_H(x) = 0,\\
&\lim_{|f'(x)| \to \infty} \text{smoothing}_H(x) = 0, && \lim_{|f'(x)| \to 0} \text{smoothing}_H(x) = \sigma_s.
\end{aligned}$$

Relationship to σ_r When σ_r approaches infinity, $\mathcal{D}t(x) = x$, and, as expected, H's response will be no longer edge-preserving, but a smoothing one proportional to σ_s. When σ_r approaches zero, $\mathcal{D}t'(x)$ goes to infinity, and any filter H with compact support will produce a filtered signal identical to the input, as expected.

Relationship to σ_s Interestingly, as σ_s approaches infinity, H does not produce unbounded smoothing in the image. This is exactly what is expected from an edge-preserving filter when σ_r is held constant. Furthermore, the amount of smoothing is inversely proportional to the gradient magnitude of the signal, which is the most commonly used estimator of image edges. Finally, when σ_s approaches zero, no smoothing is performed, as expected.

Relationship to f When the gradient magnitude of the input signal is very large, no smoothing is performed. On the other hand, in regions where the gradient magnitude is not significant, smoothing is performed with the same response of a linear smoothing filter. Note that in both cases, our filter behaves as an edge-preserving one.

3.2.4 The Domain Transform with ℓ_2 norm

For completeness, this section presents the equations for computing a domain transform using the ℓ_2 norm. The ℓ_2 norm domain transform described below is used as a building block for the Euclidean high-dimensional filter we describe in Chapter 4.

By using the ℓ_2 norm on the right-hand-side of Equation 3.5 and carrying out the same derivation, one obtains the ℓ_2 norm domain transform for multichannel signals:

$$\mathcal{D}t(u) = \int_0^u \sqrt{1 + \left(\frac{\sigma_s}{\sigma_r}\right)^2 \sum_{k=1}^{d_{\mathcal{R}}} f'_{[k]}(x)^2}\; dx,$$

which can be written more compactly as

$$\mathcal{D}t(u) = \int_0^u \sqrt{1 + \left(\frac{\sigma_s}{\sigma_r}\right)^2 \left\|f'(x)\right\|_2^2}\; dx. \tag{3.15}$$

For simplicity, we use the ℓ_1 norm domain transform (Equation 3.12) for all the examples and analysis in the remaining of this Chapter 3. However, all of the following discussion is valid for both the ℓ_1 and the ℓ_2 norm domain transforms.

3.2.5 The Multivariate Non-Axis-Aligned Domain Transform with ℓ_p norm

It is similarly easy to derive the equations for a domain transform based on an arbitrary non-diagonal positive-definite covariance matrix Σ. Let $\gamma : \mathbb{R} \to \mathbb{R}^{d_{\mathcal{R}}+1}$ be a parametric curve defined by the input signal $f(x)$:

$$\gamma(x) = \left(x, f_{[1]}(x), \ldots, f_{[d_{\mathcal{R}}]}(x)\right)^T . \tag{3.16}$$

The multivariate non-axis-aligned domain transform defined with distances measured with an ℓ_p norm is

$$\mathcal{D}t_p(u) = \sigma_H \int_0^u \left\|\Sigma^{-1/2}\, \gamma'(x)\right\|_p dx. \tag{3.17}$$

3.3 Filtering 2D Signals

Equation 3.12 defines a domain transform for signals with one-dimensional spatial domain ($d_{\mathcal{S}} = 1$) and arbitrary range dimensionality. Ideally, an inherently two-dimensional transform $\mathcal{D}t(x, y)$ should be used for signals with two-dimensional spatial domain ($d_{\mathcal{S}} = 2$), directly mapping the content at positions (x, y) in the original domain to positions (u, v) in the transformed domain. Unfortunately, as discussed in Section 3.1, $\mathcal{D}t(x, y)$ (*i.e.*, $t : \mathbb{R}^{d_{\mathcal{R}}+2} \to \mathbb{R}^2$) does not exist in general (O'NEILL, 2006), even in geodesic space (one where distances are measured geodesically). Since it is not possible to simultaneously satisfy all the distance requirements in $\mathbb{R}^2$, the use of a space with higher-dimensionality would be needed, implying additional computational and memory costs. To avoid these extra costs, we use our 1D geodesic transform to perform 2D geodesic filtering.

The most common approach for filtering signals with 2D spatial domain using 1D operations is to perform separate passes along each spatial dimension of the signal. For an image, this means performing a (*horizontal*) pass along each image row, and a (*vertical*) pass along each image column (SMITH, 1987; OLIVEIRA; BISHOP; MCALLISTER, 2000). Assuming the horizontal pass is performed first, the vertical pass is applied to the result produced by the vertical one (and vice-versa). This construction is extensively used with standard separable linear filters (DOUGHERTY, 1994) and anisotropic diffusion (WEICKERT; ROMENY; VIERGEVER, 1998), and is also related to the computation of geodesic distances on the image manifold using raster-scan algorithms (CRIMINISI et al., 2010).

One caveat is that filtering a 2D signal using a 1D domain transform is not a separable operation; otherwise, this would be equivalent to performing $\mathcal{D}t(x, y)$ in 2D. Since edge-preserving filters should not propagate information across strong edges, they cannot

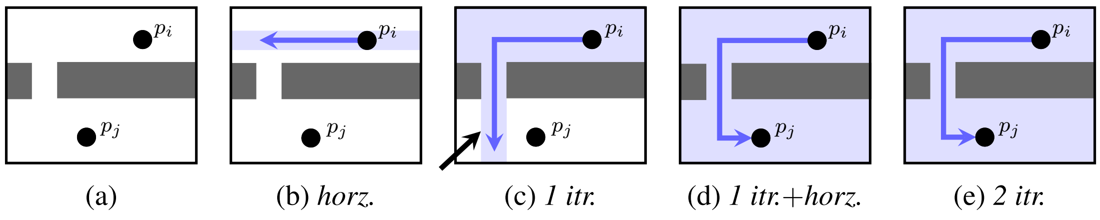


Figure 3.5: 2D edge-preserving smoothing using two-pass 1D filtering. *(a)* p_i and p_j belong to same region. *(b)* One horizontal pass. *(c)* One complete iteration (*i.e.*, horz.+vert. pass). Information from p_i cannot yet reach p_j. *(d)* After an additional horizontal pass. *(e)* Two complete iterations (*i.e.*, horz.+vert.+horz.+vert. passes).

be implemented using a single *iteration* of a two-pass 1D filtering process (we define one iteration to be a horizontal pass followed by a vertical one, or vice-versa). This situation is illustrated in Figure 3.5, where pixels p_i and p_j belong to a same region (represented in white in (a)) and, therefore, should have their information combined. Figure 3.5 (b) shows, in blue, the region reachable from p_i after one horizontal pass; and (c) after one complete iteration (assuming that the horizontal pass is performed first). The region reachable from p_j is analogous. By not reaching the entire white region after one iteration, this process may introduce visual artifacts perceived as "stripes" (indicated by the black arrow in Figure 3.5 (c)). For this example, one additional horizontal pass would be needed to propagate p_i's information to the entire white region, thus eliminating the stripe (Figure 3.5 (d)). Further passes do not alter the result (e).

The required number of horizontal and vertical passes depends on (the geometry of) the image content, and, therefore, is hard to predict. However, we use two key observations to make these artifacts unnoticeable in the filtered images: (i) stripes are only present along the last filtered dimension: a horizontal (*cf.* vertical) step removes stripes introduced by the previous vertical (*cf.* horizontal) step; and (ii) the length of the stripes is proportional to the size of the filter support used in the last pass. Thus, we interleave a sequence of vertical and horizontal passes, such that the two 1D filters used in each iteration (consisting of a vertical filter and a horizontal filter) have a σ value that is half of the one used in the previous iteration. This progressively reduces the extension of the artifacts, making them virtually unnoticeable. In practice, three iterations usually suffice to achieve good results (Section 3.3.1). During a horizontal pass, f' in Equation 3.12 is the partial derivative computed along the rows of image f, while, in a vertical pass, f' represents the partial derivative computed along the image columns.

Since variances (and not standard deviations) add (LOEVE, 1977), care must be taken when computing the σ_H value for each iteration: we must use standard deviations that halve at each step *and* whose squared sum matches the original desired variance σ_H^2. This is achieved by the following expression:

$$\sigma_{H_i} = \sigma_H \sqrt{3} \frac{2^{K-i}}{\sqrt{4^K - 1}}, \tag{3.18}$$

where σ_{H_i} is the standard deviation for the kernel used in the i-th iteration, K is the total number of iterations, and σ_H is the standard deviation of the desired kernel. The image resulting from the i-th iteration is used as input for the $(i+1)$-th iteration. The domain transforms $\mathcal{D}t(x)$ and $\mathcal{D}t(y)$ are computed only once (for the original image) and used with all the different scales of the filter H.

Figures 3.6 (b) and (c) illustrate the results of performing, respectively, one and three 1D edge-preserving filtering iterations on the image shown in (a). Figure 3.6 (d) and (e) compare the face of the statue before and after the filtering operation, and shows that small scale details have been smoothed while the important edges have been preserved. Although our filter is performed as a series of 1D operations along rows and columns, it correctly handles diagonal edges. Figure 3.7 illustrates this property on an example containing several sharp edges at various slopes. The image on the right shows the filtered result obtained using only two iterations of our two-pass 1D filter. The original edges have been faithfully preserved, while the colors have been properly filtered.

The decomposition of a 2D edge-preserving filter as a sequence of 1D filtering operations can be generalized to higher dimensions. One needs to simply add filtering steps along the added dimensions (*e.g.*, along time in a video sequence). Unfortunately, the use of 1D operations causes the filter not to be rotationally invariant: *e.g.*, filtering an image and then rotating it 90 degrees will *not* result in the same output as first rotating it 90 degrees and then filtering it (even if the filter is rotationally symmetric). However, this is also true for other fast edge-preserving filters (FARBMAN et al., 2008; FATTAL, 2009). Finally, some extreme cases do exist. For example, consider a very thin line which is almost, but not exactly, horizontal. Propagating information along a wide extent of this structure would take a considerable number of horizontal + vertical iterations. This problem can be mitigated by adding filtering iterations along non-axis-aligned directions, such as 45-degree diagonals or multiples of some other angle. This is a similar procedure to the one used in algorithms for computing geodesic distance-maps in parametric surfaces (WEBER et al., 2008). Another alternative is to use a Euclidean filter, such as the Adaptive-Manifold filter we describe in Chapter 4. Still, we did not find any visible artifacts caused by this extreme case in the large number of images and videos we filtered with the domain transform (using only horizontal + vertical iterations),

3.3.1 Convergence Analysis

Artifact-free filtered images can be obtained by increasing the number of iterations. Here, we describe an experiment designed to empirically analyze the convergence of the 2D filtering process. For color images with channels in the $[0,1]$ range, ten to twelve iterations are sufficient to cause the mean-square difference between the results of subsequent iterations to fall below a threshold of 10^{-4}, chosen experimentally. The quality of a filtered result obtained after n iterations is evaluated by comparing it to the result obtained for the same image after 15 iterations, which, for practical purposes, can be considered artifact free. The comparison is performed using the Structural Similarity (SSIM) index (WANG et al., 2004). SSIM is an image-quality metric consistent with human perception. Its structural nature makes it appropriate for detecting the "stripe" artifacts discussed in the previous section. Since the SSIM index detects similarity, we use its complement $(1 - SSIM)$ as an error measure.

The graph in Figure 3.8 summarizes the errors measured for various numbers of filtering iterations. These results represent the maximum errors obtained while filtering 31 natural images with various contents. Each curve corresponds to a fixed value of σ_r. For each point along a σ_r curve, we plot the maximum error obtained among all values of $\sigma_s \in \{1, 10, 20, 40, 60, 80, 100, 200, 500, 1000, 3000\}$. The graph shows that the dissimilarity metric decreases quickly with the first three iterations, which represents a good tradeoff between filtering quality and computational time.

3.4 Filtering in the Transformed Domain

Our full MATLAB source code is available in Appendix A.

Given a domain transform $\mathcal{D}t : \mathcal{S} \to \mathcal{S}_w$, the transformed signal $f_w(\mathcal{D}t(x)) = f(x)$ is then filtered using the 1D kernel H. This section discusses alternatives for performing this filtering operation on digital signals, where f_w will likely be non-uniformly sampled.

Chapter 5 analyses in-depth the concept of filtering non-uniformly sampled signals. More specifically, Section 5.5.1 shows how to perform *geodesic edge-aware* evaluation of *arbitrary* digital filters using the domain transform.

3.4.1 Normalized Convolution (NC)

Filtering the non-uniformly sampled signal f_w in $\mathcal{S}_w$ can be seen as filtering a uniformly sampled signal with missing samples (Figure 3.9, left). This scenario has been studied by Knutsson and Westin (KNUTSSON; WESTIN, 1993) in the context of data uncertainty, where they showed that optimal filtering results, in the mean square sense, are obtained by *normalized convolution* (NC). For a uniform discretization $D_N(\mathcal{S})$ of the original domain $\mathcal{S}$, NC describes the filtered value $g(x_i)$ of a sample $x_i \in D_N(\mathcal{S})$ as

$$g(x_i) = \frac{1}{K_i} \sum_{x_j \in D_N(\mathcal{S})} f(x_j)\, H\left(\, t(\hat{x}_i),\, t(\hat{x}_j)\,\right), \tag{3.19}$$

where

$$K_i = \sum_{x_j \in D_N(\mathcal{S})} H(\, t(\hat{x}_i),\, t(\hat{x}_j)\,) \tag{3.20}$$

is a normalization factor for x_i, and $t(\hat{x}_i) = \mathcal{D}t(x_i)$. For N samples and an arbitrary kernel H, the cost of evaluating Equation 3.19 for all x_i is $O(N^2)$. However, as $\mathcal{D}t(x)$ is monotonically increasing (Equation 3.12), we use an efficient *moving-average* approach (DOUGHERTY, 1994) to perform NC with a box filter in $O(N)$ time. The box kernel is defined as

$$H\left(t(\hat{x}_i),\, t(\hat{x}_j)\right) = \left[\, |t(\hat{x}_i) - t(\hat{x}_j)| \le r \,\right], \tag{3.21}$$

where $r = \sigma_H\sqrt{3}$ is the filter radius (see Section 3.4.1.1 for the derivation of r), and $[\cdot]$ is the Iverson bracket: a boolean function that returns 1 when its argument is true, and 0 otherwise. This box kernel has a constant radius in $\mathcal{S}_w$, but a space-varying and non-symmetric radius in $\mathcal{S}$, where its size changes according to the similarity between x_i and its neighborhood in the image manifold M_f (Figure 3.9, right, in blue). This can be interpreted as an estimate of which neighbors belong to the same population as x_i. The box kernel is then a robust estimator of the population mean, with connections to robust anisotropic diffusion (BLACK et al., 1998) and bilateral filtering (DURAND; DORSEY, 2002).

The cost of evaluating Equation 3.19 using the box kernel from Equation 3.21 is linear in the number of samples. We use it for the 1D filtering iterations described in Section 3.3, with σ_{H_i} defined by Equation 3.18. For three iterations, the resulting filter produces an indistinguishable approximation to a Gaussian filter (PSNR > 40) when $\sigma_r = \infty$. Figure 3.13 compares this result to the ones obtained with several other filters.

CPU Implementation Since samples are not uniformly spaced in $\mathcal{S}_w$, the number of samples added to and removed from the kernel window as it slides from one sample to the next is not constant. Thus, performing box filtering in $\mathcal{S}_w$ requires updating K_i, plus one additional memory read per sample to check its domain coordinate. One only needs to perform convolution at positions in $\mathcal{S}_w$ that contain samples, as other positions will not contribute to the filtered image in the (discrete) original domain. Finally, derivatives are estimated using backward differences.

GPU Implementation Our domain transform is highly parallel: each thread calculates the value of $\mathcal{D}t'(x)$ (Equation 3.13) for one sample, and a parallel scan operation performs the integration. For filtering, each thread computes the filtered value of one pixel. To find the first and last pixels inside the current 1D kernel window (Figure 3.9, right, in blue), we perform two binary searches on the transformed domain ($\mathcal{S}_w$) coordinates. Once the first and last pixels under the 1D kernel have been identified, the sum of the colors of all contributing pixels is calculated using a 1D summed area table (per color channel). These tables need to be updated before each horizontal/vertical pass.

3.4.1.1 Derivation of Box Filter radius 'r' from the desired variance σ_H^2

The continuous, normalized box filter kernel is

$$h(x) = \begin{cases} \frac{1}{2r} & \text{for } |x| \le r, \\ 0 & \text{for } |x| > r. \end{cases}$$

The first and second moments of h are, respectively:

$$\begin{aligned} \langle h \rangle &= \int_{-\infty}^{\infty} x\, h(x) &&= \int_{-r}^{r} \frac{x}{2\,r} = 0 \quad \text{and} \\ \langle h^2 \rangle &= \int_{-\infty}^{\infty} x^2\, h(x) &&= \int_{-r}^{r} \frac{x^2}{2\,r} = \frac{r^2}{3}. \end{aligned}$$

The variance of h is then given by

$$\mathrm{Var}(h) = \langle h^2 \rangle - \langle h \rangle^2 = \frac{r^2}{3}.$$

Given the desired variance σ_H^2, which must be equal to $\mathrm{Var}(h)$, we solve for r:

$$\sigma_H^2 = \frac{r^2}{3} \implies |r| = |\sigma_H|\sqrt{3}.$$

The absolute value operators on the right hand side can be discarded since both r and σ_H are always positive.

3.4.2 Interpolated Convolution (IC)

Another option when dealing with irregularly sampled data is to use interpolation for approximating the original continuous function (PIRODDI; PETROU, 2004). Figure 3.9 (center) shows a reconstructed signal l_w obtained by linear interpolation (in $\mathcal{S}_w$) of the samples shown in Figure 3.9 (left). Filtering l_w is performed by continuous convolution:

$$g(x_i) = \int_{\mathcal{S}_w} l_w(x_w)\; H\left(\, t(\hat{x}_i),\, x_w \,\right)\; dx_w, \tag{3.22}$$

where H is a normalized kernel. Interpolated convolution has an interesting interpretation: a *linear* diffusion process working *on the signal*. Figure 3.9 (right) shows this interpretation for a box filter of radius r, where the kernel window is shown in red. This is the same interpretation as the 1D Beltrami flow PDE (SOCHEN; KIMMEL; BRUCKSTEIN, 2001).

Implementation For a box filter, Equation 3.22 can be evaluated for all pixels in $O(N)$ time. This is achieved by a *weighted moving-average* (DOUGHERTY, 1994). The normalized box kernel is given by

$$H\left(\, t(\hat{x}_i),\, x_w \,\right) = \left[\; |t(\hat{x}_i p) - x_w| \leq r \,\right] \big/\, 2\,r, \tag{3.23}$$

where $r = \sigma_H\sqrt{3}$ is the filter radius. Substituting Equation 3.23 in Equation 3.22:

$$g(x_i) = \frac{1}{2\,r} \int_{t(\hat{x}_i)-r}^{t(\hat{x}_i)+r} l_w\left(x_w\right)\; dx_w. \tag{3.24}$$

The linearly-interpolated signal l_w does not need to be uniformly resampled, since the area under its graph can be explicitly computed using the trapezoidal rule.

3.4.3 Recursive Filtering (RF)

For a discrete signal $f[i] = f(x_i)$, non edge-preserving filtering can be performed using a 1st-order recursive filter as

$$g[i] = (1 - a)\ f[i] + a\ g[i-1], \tag{3.25}$$

where $a \in [0, 1]$ is a *feedback coefficient* (SMITH, 2007). This filter has an infinite impulse response (IIR) with exponential decay: an impulse of magnitude m at position i generates a response of magnitude $m\ (1 - a)\ a^{j-i}$ at position $j \geq i$. Note that $j - i$ can be interpreted as the *distance* between samples x_i and x_j, assuming a unitary sampling interval. Based on this observation, a recursive edge-preserving filter can be defined in the transformed domain as

$$g[i] = (1 - a^{d_i})\ f[i] + a^{d_i}\ g[i-1], \tag{3.26}$$

where $d_i = \mathcal{D}t(x_i) - \mathcal{D}t(x_{i-1})$ is the distance between neighbor samples x_i and x_{i-1} in the transformed domain ($\mathcal{S}_w$). With this filter, an impulse of magnitude m at pixel x_i generates a response at pixel $x_j > x_i$ of magnitude

$$m\ (1 - a^{d_i})\ a^{d_{i+1} + \ldots + a^{d_j}} = m\ (1 - a^{d_i})\ a^{\mathcal{D}t(x_j) - \mathcal{D}t(xi)}.$$

As the distances d_i increase, a^{d_i} goes to zero, stopping the propagation chain and, thus, preserving edges. This can be interpreted as a geodesic propagation on the image lattice. The impulse response of Equation 3.26 is not symmetric, since it only depends on previous inputs and outputs (it is a causal filter). A symmetric response is achieved by applying the filter twice: for a 1D signal, Equation 3.26 is performed left-to-right (*cf.* top-to-bottom) and then right-to-left (*cf.* bottom-to-top).

The feedback coefficient of this filter is computed from the desired filter variance as $a = \exp(-\sqrt{2}/\sigma_H)$ (see Section 3.4.3.1 for the derivation). Since $a \in [0, 1]$, the filter is stable (SMITH, 2007), and its implementation in $O(N)$ time is straightforward.

Finally, as mentioned in the beginning of Section 3.4, we note that it is possible to use the domain transform with *arbitrary* recursive digital filters. This topic is dealt with in Chapter 5, more specifically Section 5.5.1.

3.4.3.1 Derivation of RF feedback coefficient 'a' from the desired variance σ_H^2

The continuous equivalent of the recursive kernel is

$$h(x) = (1 - a)\ a^x, \quad x \in [0, +\infty),\ a \in (0, 1);$$

where x represents the distance between samples and a is the feedback coefficient. $h(x)$ is not normalized since

$$\int_0^\infty h(x)\ dx = \left.\frac{(1-a)a^x}{\log(a)}\right|_0^\infty = -\frac{1-a}{\log(a)}.$$

Normalizing h we obtain

$$h(x) = -\log(a)\ a^x.$$

The first and second moments of h are, respectively:

$$\begin{aligned} \langle h \rangle &= -\log(a) \int_0^\infty x\ a^x dx &&= -\frac{1}{\log(a)} \quad \text{and} \\ \langle h^2 \rangle &= -\log(a) \int_0^\infty x^2\ a^x dx &&= \frac{2}{\log(a)^2}. \end{aligned}$$

The variance of h is then given by

$$\mathrm{Var}(h) = \langle h^2 \rangle - \langle h \rangle^2 = \frac{1}{\log(a)^2}.$$

Since the signal is filtered twice with h (left-to-right and right-to-left), the total variance of the filter is $2\ \mathrm{Var}(h)$. Given the desired variance σ_H^2:

$$\sigma_H^2 = 2\ \mathrm{Var}(h) = \frac{2}{\log(a)^2};$$

we solve for a and find

$$a = \exp(-\sqrt{2}/\sigma_H) \quad \text{and} \quad a = \exp(\sqrt{2}/\sigma_H);$$

where the former is our solution since $a \in (0, 1)$.

3.5 Comparison to Other Approaches

We compare our edge-preserving filters based on normalized convolution (NC), interpolated convolution (IC), and recursion (RF) against previous works: brute-force bilateral filter (BF) (TOMASI; MANDUCHI, 1998); anisotropic diffusion (AD) (PERONA; MALIK, 1990); edge-avoiding wavelets (EAW) (FATTAL, 2009); weighted least squares filter (WLS) (FARBMAN et al., 2008), which has been shown to produce good results for tone and detail manipulation; and finally the permutohedral lattice BF (PLBF) (ADAMS et al., 2009) and constant time BF (CTBF) (YANG; TAN; AHUJA, 2009), which are, respectively, the fastest color and grayscale bilateral filter approximations.

3.5.1 Filter Response

Figure 3.10 shows a comparison of the impulse response of our three filters NC, IC and RF (all performed using three iterations) against the impulse response of BF, AD and WLS. The NC and IC filters have Gaussian-like response, similar to AD and BF. In the presence of strong edges, the IC filter behaves similarly to AD. The NC filter has a higher response near strong edges, which is a direct implication of its interpretation as a robust mean: pixels near edges have fewer neighbors in the same population, and will weight their contribution strongly. Finally, our recursive filter (RF) has an exponential impulse response which is completely attenuated by strong edges, like the WLS's response.

The NC filter is ideal for stylization and abstraction, since it accurately smoothes similar image regions while preserving and sharpening relevant edges. For applications where sharpening of edges is not desirable (*e.g.*, tone mapping and detail manipulation), the IC and RF filters produce results of equal quality as the state-of-the-art techniques (FARBMAN et al., 2008; FATTAL, 2009). Finally, for edge-aware interpolation (*e.g.*, colorization and recoloring), the RF filter produces the best results due to its infinite impulse response, which propagates information across the whole image lattice. Section 5.5 illustrates the use of our filters for all these applications.

3.5.2 Smoothing Quality

Figure 3.11 shows a side-by-side comparison of edge-aware smoothing applied to a portion of the photograph shown in Figure 3.1 (a). For small amounts of smoothing, the bilateral filter (b) and our NC filter (c) produce visually similar results. For further smoothing, the bilateral filter may incorrectly mix colors, as observed in the window frame (Figure 3.11 (d)). In contrast, our filter manages to continuously smooth image regions while preserving strong edges, due to its geodesic nature. This effect, illustrated in Figure 3.11 (e), is similar to the results obtained with WLS, shown in (f), and anisotropic diffusion (using (D'ALMEIDA, 2004)), shown in (g). Since the scale of EAW cannot be freely controlled, the technique is not ideal for edge-preserving smoothing. Figure 3.11 (h) shows the result produced by EAW with a maximum decomposition depth of 5 and coefficients for each detail level defined by $0.6^{(5-level)}$, which preserves some high-frequency details. Setting these coefficients to zero results in distracting artifacts.

Our filters converge to standard linear smoothing filters on regions with weak edges, or when the range support σ_r is set to a large value (see Section 3.2.3). This feature is desirable, for instance, in joint filtering for performing depth-of-field effects, as shown in Figure 3.1 (g) and discussed in Section 5.5. Figure 3.13 shows that previous techniques have difficulty to simulate a regular smoothing filter, either because their kernels cannot be explicitly controlled, as in the cases of WLS (b) and EAW (c); or because the downsampling required for performance introduces structural artifacts, as in the case of PLBF (d). In contrast, our filters provide an indistinguishable approximation to a Gaussian (f) for $\sigma_r = \infty$, as shown in (e).

Temporal Coherence The domain transform filters are guaranteed to be temporally coherent as long as the input image derivatives are temporally coherent. This observation is derived directly from Equation 3.10: as long as $f'(x)$ varies smoothly over time, $\mathcal{D}t'(x)$ will also vary smoothly over time. In fact, the domain transform has already been used by other researchers exactly to improve the temporal coherence of video data, such as optical flow, depth, and disparity (LANG et al., 2012). In our project webpage (GASTAL; OLIVEIRA, 2014a) we include video filtering examples where each frame was filtered independently and the results are of high quality and without temporal artifacts.

Denoising One will note that we do not include denoising as an effective application of the domain transform, despite the fact that we have had some success using our filters for removing small to medium amounts of noise. The main issue with noisy images is finding robust estimates for their derivatives, which cannot be done using simple backward differences. One possible solution is pre-filtering the image with a low-pass filter to avoid large oscillations in the image gradient. See (CATTÉ et al., 1992) for a similar approach applied to anisotropic diffusion filtering.

3.5.3 Performance Evaluation

This section reports performance numbers obtained on a 2.8 GHz Quad Core PC with 8 GB of memory and a GeForce GTX 280.

Filtering on CPU We implemented our NC and RF filters on CPU using C++. For the IC filter we have a MATLAB implementation, but its performance in C++ is expected to be similar to NC's. On a single CPU core, the typical runtimes of our NC and RF filters for processing a 1 megapixel color image using three iterations are 0.16 and 0.06 seconds, respectively. Their performances scale linearly with image size, filtering 10 megapixel color images in under 1.6 and 0.6 seconds. On a quad-core CPU, we achieve a 3.3× speedup.

We compare the performance of our edge-aware filters against the fastest filters from previous works: EAW, PLBF and CTBF. The PLBF and our NC and RF filters process all three color channels simultaneously, while CTBF only processes grayscale. Thus, CTBF is actually performing one third of the work done by the other three methods. We measured the reported results on a single CPU core. For PLBF and CTBF we used source code provided by the authors.

The runtimes for both PLBF and CTBF are inversely proportional to the value of σ_r. For σ_r approaching zero, their runtimes are above 10 seconds. The runtimes of our filters are independent of the σ_s and σ_r parameters, and they use no simplifications to improve performance. In our experience, to achieve good edge-preserving smoothing with PLBF or CTBF values of $\sigma_r < 0.15$ should be used. In this range, our filters with three iterations are 5 to 15× faster than these approaches. For small amounts of smoothing, one can obtain good results using two or even one iteration of our filter (Figure 3.8), with speed-ups of 25 to 40× over PLBF for color filtering. According to Fattal (FATTAL, 2009), EAW can smooth a 1 megapixel grayscale image in 0.012 seconds on a 3.0 GHz CPU. Since it generates decompositions at only a few scales, it is not generally applicable for edge-preserving smoothing. WLS takes 3.5 seconds to solve its sparse linear system using a fast CPU implementation.

A graph comparing the performances of these techniques can be seen in Figure 3.12. The PLBF and our NC and RF filters process all three color channels simultaneously, while WLS, EAW and CTBF only process grayscale range. Note however that WLS and EAW can also perform true color filtering, but the available implementations from the respective authors can process grayscale images only. For PLBF and CTBF, the values of

σ_s were chosen so as to minimize their run times. The available implementation of CTBF only produces accurate results for $\sigma_r < 0.45$. This is indicated in Figure 3.12 by the red “x", after which CTBF outputs a black image. EAW and WLS do not explicitly use a σ_r parameter. For our NC and RF filters, we tuned their parameters to obtain similar blurring and edge-preserving results as PLBF and CTBF. Note that our σ_r value is defined for the ℓ_1 norm in $\mathbb{R}^3$, while the σ_r of PLBF and CTBF is defined for the ℓ_2 norm. The reported performances for the EAW and WLS filters were taken directly from (FARBMAN et al., 2008) and (FATTAL, 2009). The EAW filter can only generate smoothing decompositions at a few scales, indicated in the graph as discrete green triangles.

Filtering on GPU We implemented our NC filter on GPU using CUDA. The total time required for filtering a 1 megapixel color image is 0.7 milliseconds for computing the domain transform plus 2 milliseconds for each 2D filtering iteration. This gives a total runtime of approximately 0.007 seconds for three iterations of our filter — a speedup of $23\times$ compared to our one-core CPU implementation. Since our filter scales linearly with the image size, our GPU implementation is able to filter 10 megapixel color images in under 0.07 seconds.

We compare the performance of our GPU filter against the GPU Bilateral Grid (CHEN; PARIS; DURAND, 2007). While their implementation is as fast as ours, it only processes luminance values, which may generate undesired color-ghosting artifacts. Their approach draws its efficiency from downsampling, which is not possible for small spatial and range kernels. The GPU implementation of PLBF can filter a 0.5 megapixel image in 0.1 sec on a GeForce GTX 280 (ADAMS; BAEK; DAVIS, 2010). A GPU implementation of WLS filters a 1 megapixel grayscale image in about 1 second (FARBMAN et al., 2008).

3.6 Real-Time Applications

We show a variety of applications that demonstrate the versatility of our domain transform and filters for image processing. Given its speed, our approach can be performed on-the-fly on high-resolution images and videos. This improved performance provides users with instant feedback when tuning filter parameters.

3.6.1 Detail Manipulation

Edge-preserving filters can be used to decompose image details into several scales, which can be manipulated independently and recombined to produce various effects (FARBMAN et al., 2008; FATTAL, 2009). Let $J_0, \ldots, J_k$ be progressively smoother versions of an image $f = J_0$. Several *detail layers* capturing progressively coarser details are constructed as $D_i = J_i - J_{i+1}$. Figure 3.14 shows an example of fine-scale detail enhancement applied to the flower image in (a). The result in (b) was created by filtering the image in (a) once using our IC filter ($\sigma_s = 20$ and $\sigma_r = 0.08$) and by manipulating the detail layer D_0 using a sigmoid function described by Farbman et al. (FARBMAN et al., 2008). Figure 3.14 (c) shows the result produced by an EAW filter of Fattal (FATTAL, 2009). The images (b) and (c) present similar visual quality. Our filter, however, allows for extra flexibility when decomposing image details, since it can produce a continuum of smoothed images J_i, by varying the values of the parameters σ_s and σ_r.

3.6.2 Tone Mapping

Edge-aware tone mapping avoids haloing and other artifacts introduced in the compression process. Figure 3.15 compares the result of a tone mapping operator implemented using our RF filter (a) and the WLS filter by Farbman et al. (FARBMAN et al., 2008) (b). The quality of these results is similar, but our filter is significantly faster, resulting in the fastest high-quality tone-mapping solution available. The result in Figure 3.15 (a) was obtained by manipulating three detail layers from the HDR image's log-luminance channel. Each layer was obtained in 12 milliseconds using two iterations of our RF filter with: $\sigma_s = 20$ and $\sigma_r = 0.33$ for J_1; $\sigma_s = 50$ and $\sigma_r = 0.67$ for J_2; and $\sigma_s = 100$ and $\sigma_r = 1.34$ for J_3. The compressed luminance channel L_C was obtained as:

$$L_C = 0.12 + \mu + 0.9\ (B - \mu) + 0.3\ D_0 + 0.2\ D_1 + 0.2\ D_2;$$

where B is a linear compression of J_3 to the range $[0, 1]$:

$$B = (J_3 - min(J_3))/(max(J_3) - min(J_3));$$

and μ is the mean value of all pixels in B. J_i and D_i were defined in the previous section.

3.6.3 Stylization

Stylization aims to produce digital imagery with a wide variety of effects not focused on photorealism. Edge-aware filters are ideal for stylization, as they can abstract regions of low-contrast while preserving, or enhancing, high-contrast features. Figure 3.16 illustrates an application of our NC filter to produce abstracted results. Given an input image (top left), the magnitude of the gradient of the filtered image (top right) is superimposed to the filtered image itself to produce high-contrast edges around the most salient features. Another interesting stylization effect can be obtained by assigning to each output pixel a scaled version of the value of the normalization factor K_i from Equation 3.19. This produces a pencil-like non-photorealistic drawing, such as the one shown in Figure 3.1 (f), obtained scaling K_i by 0.11.

3.6.4 Joint Filtering

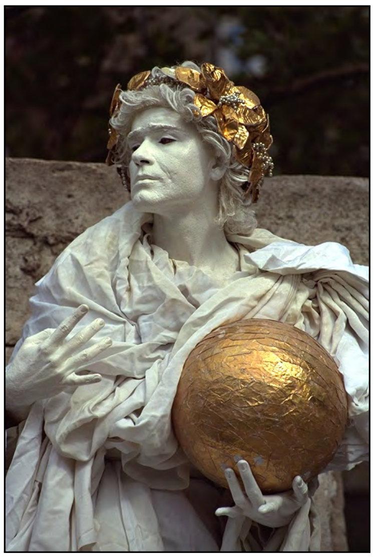
Our approach can also be used for *joint filtering*, where the content of one image is smoothed based on the edge information from a second image. For instance, by using the values of an alpha matte (GASTAL; OLIVEIRA, 2010) as the the image derivatives in Equation 3.12, one can simulate a depth-of-field effect (Figure 3.1 (g)). This example emphasizes why converging to a Gaussian-like response is an important property of our filter. Alpha mattes can also be combined with other maps to create some *localized* or *selective stylization*, as shown in Figure 3.17. In this example, an alpha matte and a structure resembling a Voronoi diagram have been added to define new edges to be preserved (Figure 3.17, bottom left). The resulting filter produces a unique stylized depth-of-field effect. The edges of the diagram were then superimposed to the filtered image to create the result shown on the right. A similar result is shown in Figure 3.1 (d), where the edges to be preserved were identified by a Canny edge detector.

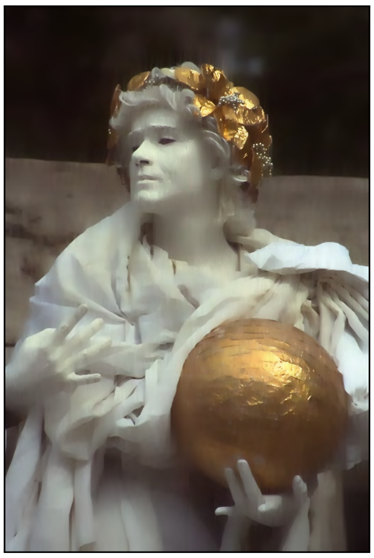
3.6.5 Colorization

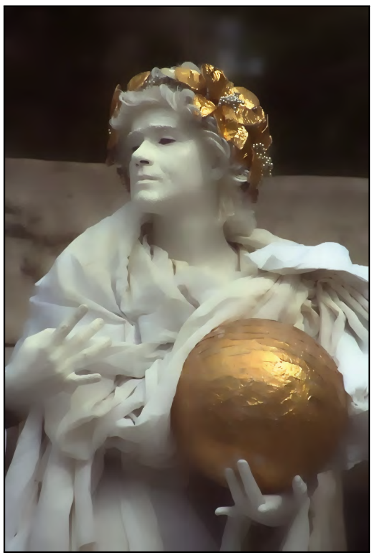
Similar to previous approaches (LEVIN; LISCHINSKI; WEISS, 2004; FATTAL, 2009), we propagate the colors S from user-supplied scribbles by blurring them using the edge information from a grayscale image f (Figure 5.11, left). To keep track of how much color propagates to each pixel, we also blur a normalization function N, which is defined to be one at pixels where scribbles are provided and zero otherwise. Let $\tilde{S}$ and $\tilde{N}$ be the blurred versions of S and N, respectively. The final color of each pixel p is obtained as $\tilde{S}(p_i)/\tilde{N}(p_i)$. This value is combined with the original luminance from the grayscale image to produce the colorized result. Figure 5.11 compares the results obtained with our RF filter ($\sigma_s = 100$ and $\sigma_r = 0.03$) with the ones of Levin et al. (LEVIN; LISCHINSKI; WEISS, 2004). In our experience, good colorization results can be obtained with values of σ_r from 0.01 to 0.1 and $\sigma_s > 100$.

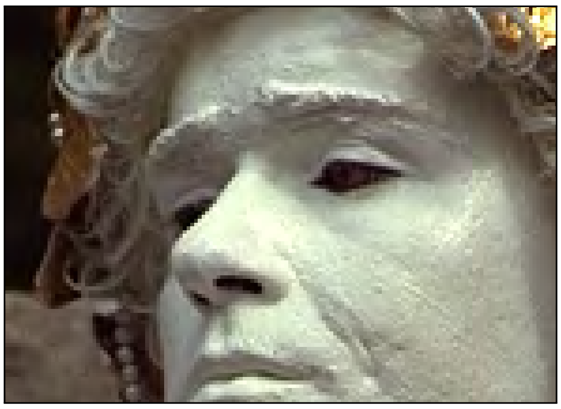
3.6.6 Recoloring

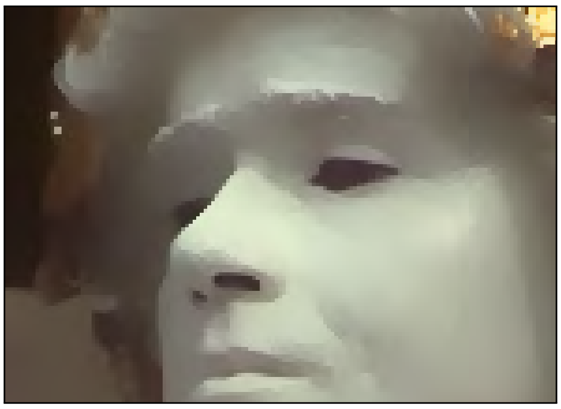
Localized manipulations of image colors is achieved using soft segmentation. Color scribbles define a set of regions of interest, where each region R_i is defined by a color c_i. For each region R_i, an *influence map* (LISCHINSKI et al., 2006) is obtained by blurring its associated normalization function N_{R_i} defined by all scribbles with color c_i. The contribution of R_i for the recoloring of pixel p is defined as $\tilde{N}_{R_i}(p)/\sum_j \tilde{N}_{R_j}(p)$. Figure 3.1 (e) shows a recoloring example obtained using this technique.


(a) Input


(b) 1 itr.


(c) 3 itr.


(d) Details from Input (a)


(e) Details from 3 iterations (c)

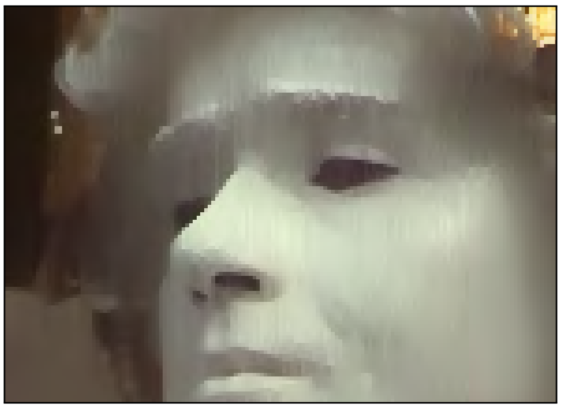

(f) Details from 1 iterations (b)

Figure 3.6: Two-pass 1D filtering ($\sigma_H = \sigma_s = 40$ and $\sigma_r = 0.77$). *(a)* Input image. *(b)* One filtering iteration. *(c)* Three filtering iterations. *(d)* Details from (a). *(e)* Details from (c). The image content has been smoothed while its edges have been preserved. *(f)* Details from (b). An insufficient number of iterations leads to vertical stripes in this example. The large value for σ_r was used to generate an exaggerated smoothing effect, which also ends up smoothing lower-contrast edges. A smaller value for σ_r may be used to preserve more edges in the filtered image.

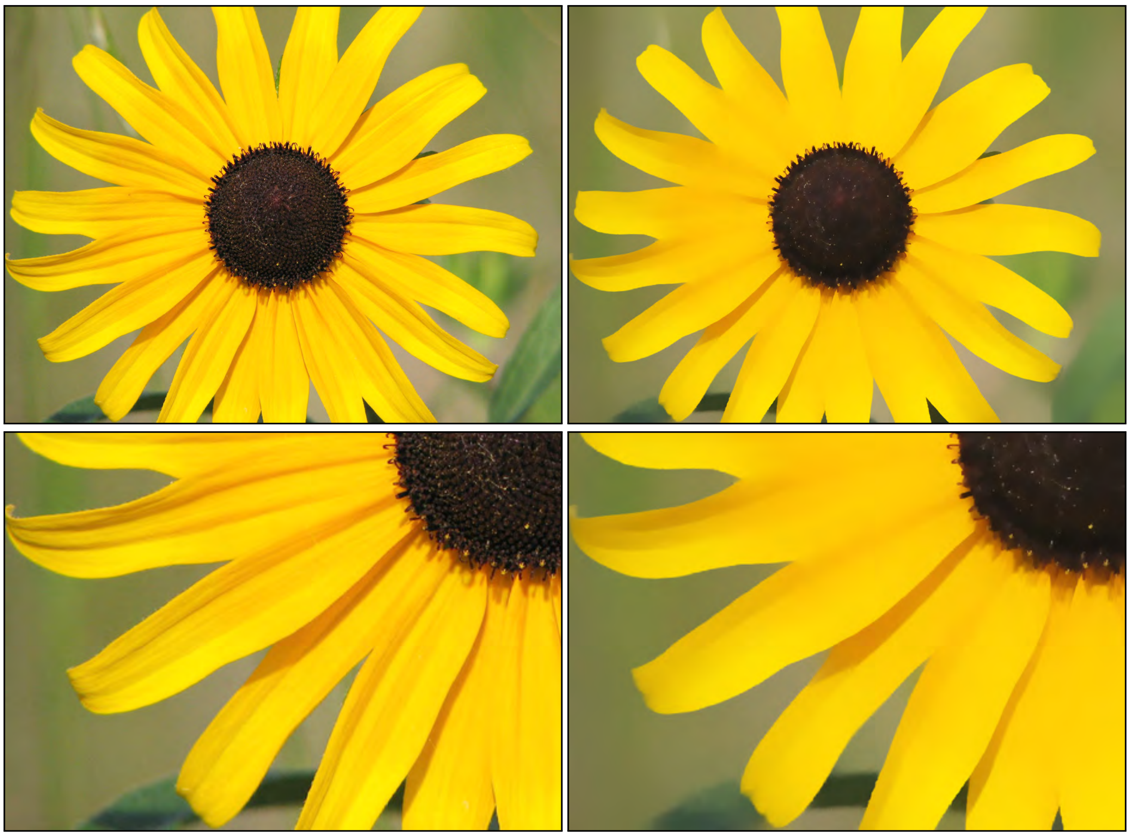


Figure 3.7: Filtering of diagonal edges. *(Top Left)* Input image (1280×960 pixels). *(Top Right)* Filtered image with two iterations of our two-pass 1D filter ($\sigma_H = \sigma_s = 50$ and $\sigma_r = 0.5$). *(Bottom Left)* Detail from the input image. *(Bottom Right)* Detail from the filtered image.

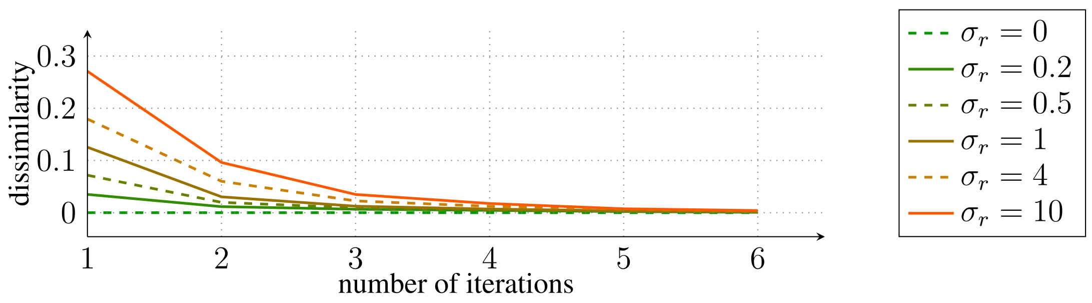


Figure 3.8: Maximum measure of dissimilarity (according to SSIM) between filtered images and their corresponding “ideal” results as a function of number of iterations, for different values of σ_r.

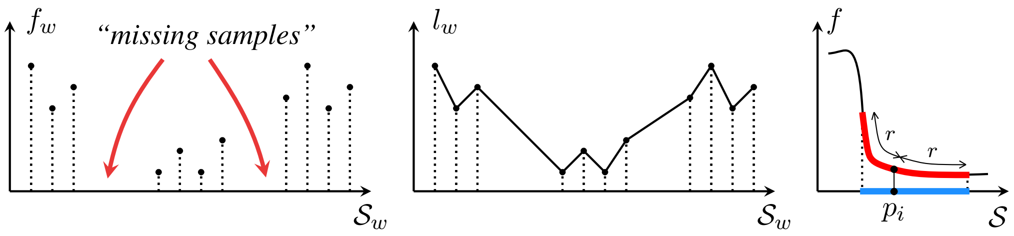


Figure 3.9: Filtering in the transformed domain. *(Left)* Normalized convolution (NC). *(Center)* Interpolated convolution (IC). *(Right)* Their interpretation: NC box kernel in blue, IC box kernel in red.

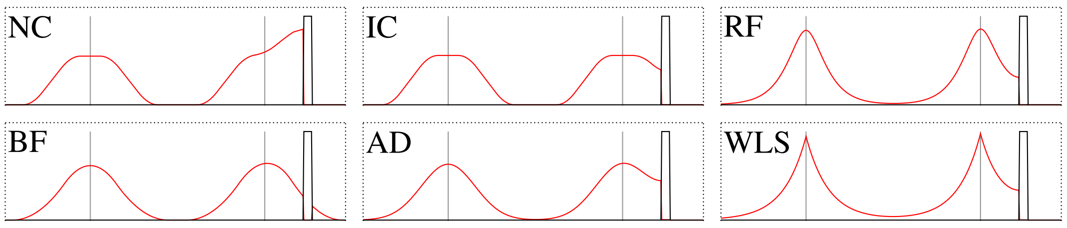
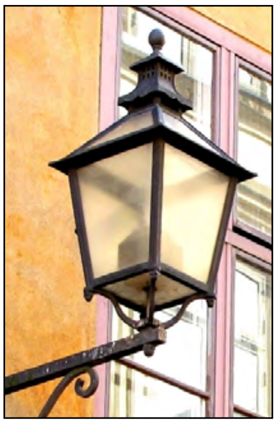
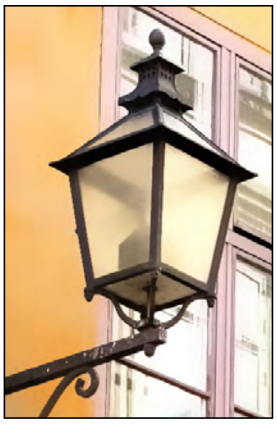
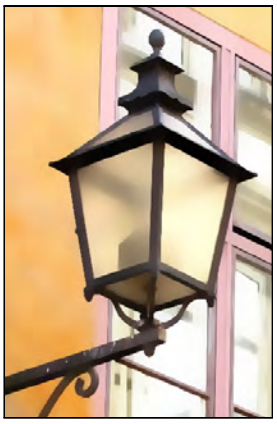
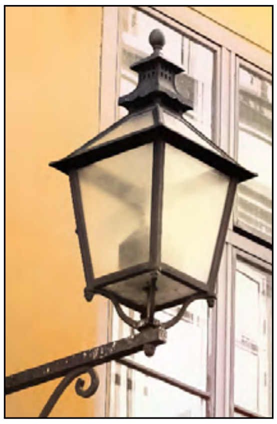


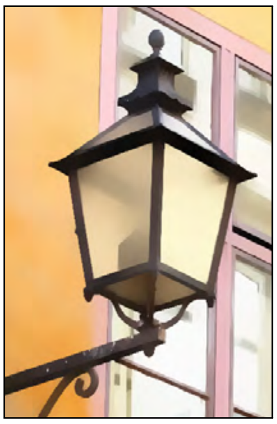
Figure 3.10: Impulse response for various filters at neighborhoods without (left of each graph) and with strong edges (right of each graph). NC: Normalized Convolution; IC: Interpolated Convolution; RF: Recursive Filter; BF: Bilateral Filter; AD: Anisotropic Diffusion; WLS: Weighted Least Squares. The parameters for each filter were selected to achieve similar variance for all impulse responses. For all filters the edge at the right of each graph was set to match the maximum possible variation in the signal, such as to completely stop propagation/diffusion if possible (note that the bilateral filter, which works in Euclidean space, manages to "jump" the edge regardless of its magnitude).

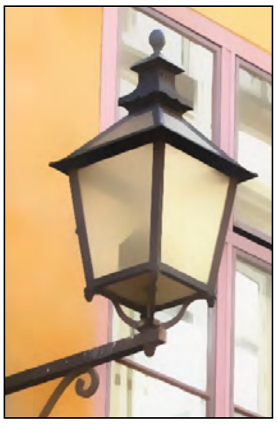

(a) Input

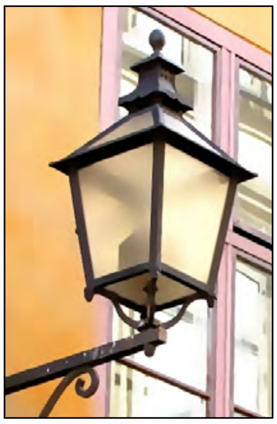

(b) BF, $\sigma_s = 17$


(c) NC, $\sigma_s = 17$


(d) BF, $\sigma_s = 40$


(e) NC, $\sigma_s = 80$


(f) WLS


(g) AD

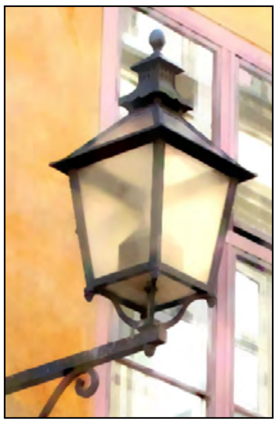

(h) EAW

Figure 3.11: Qualitative comparison of the results of bilateral filters (BF) with $\sigma_r = 0.2$ (*b* and *d*), our normalized convolution filter (NC) with $\sigma_r = 0.8$ (*c* and *e*), weighted least squares filter (WLS) *(f)* with $\lambda = 0.15$ and $\alpha = 2$, anisotropic diffusion (AD) *(g)*, and the edge-avoiding wavelets (EAW) *(h)*.

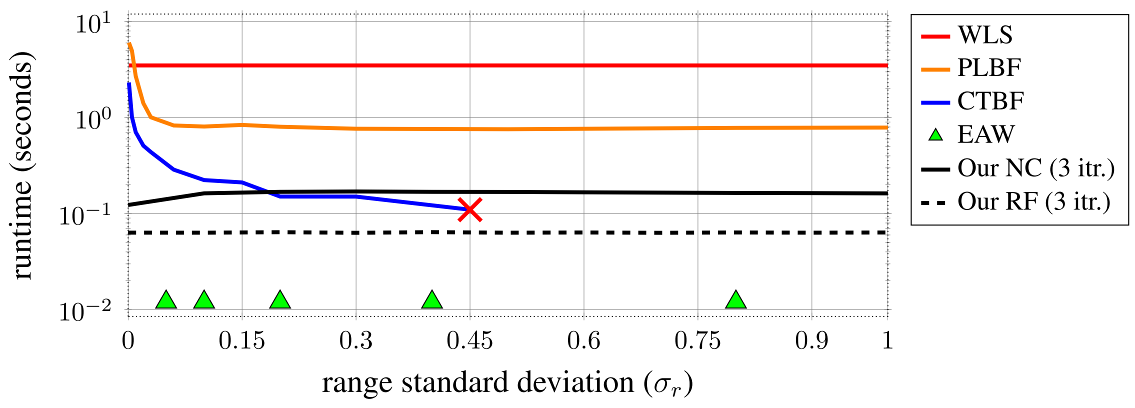
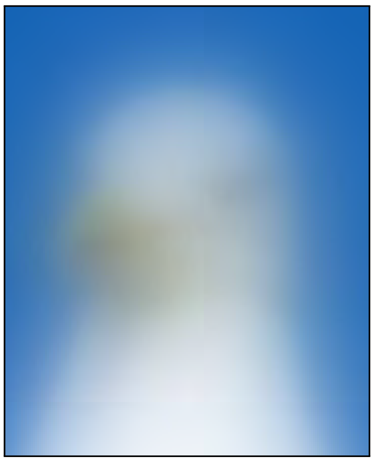
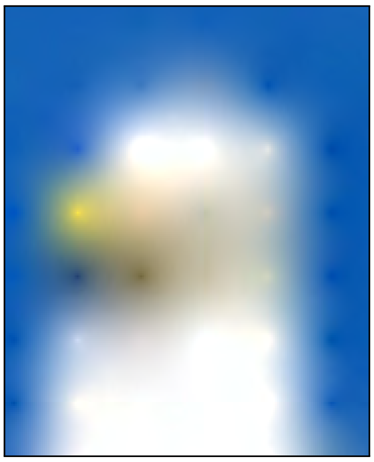
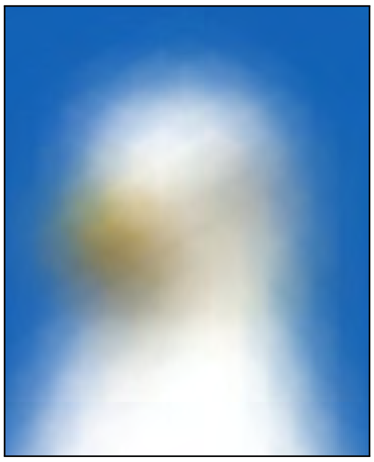


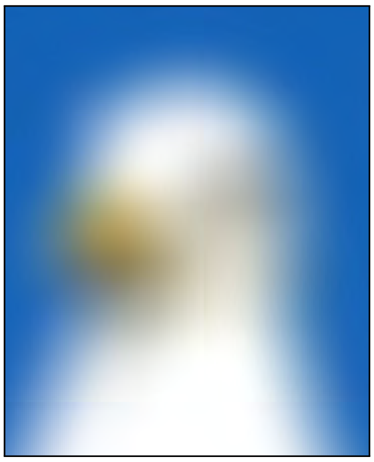
Figure 3.12: CPU performance of various edge-preserving filters for filtering a 1 megapixel image. Note that WLS and EAW do not explicitly use a σ_r parameter, and their reported performances were taken directly from (FARBMAN et al., 2008) and (FATTAL, 2009), respectively. For our NC and RF filters, we tuned their parameters to obtain similar blurring and edge-preserving results as PLBF and CTBF. The reported performances for WLS, EAW and CTBF are for processing grayscale range only.

(a) Input


(b) WLS


(c) EAW


(d) PLBF


(e) Our NC

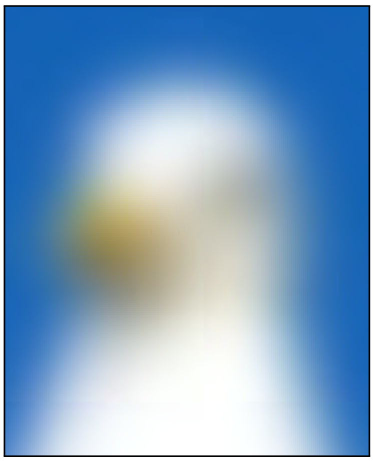
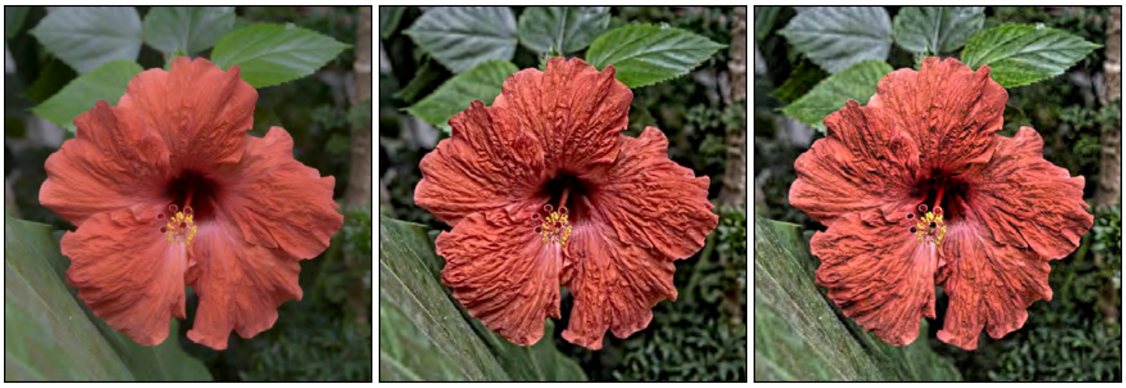
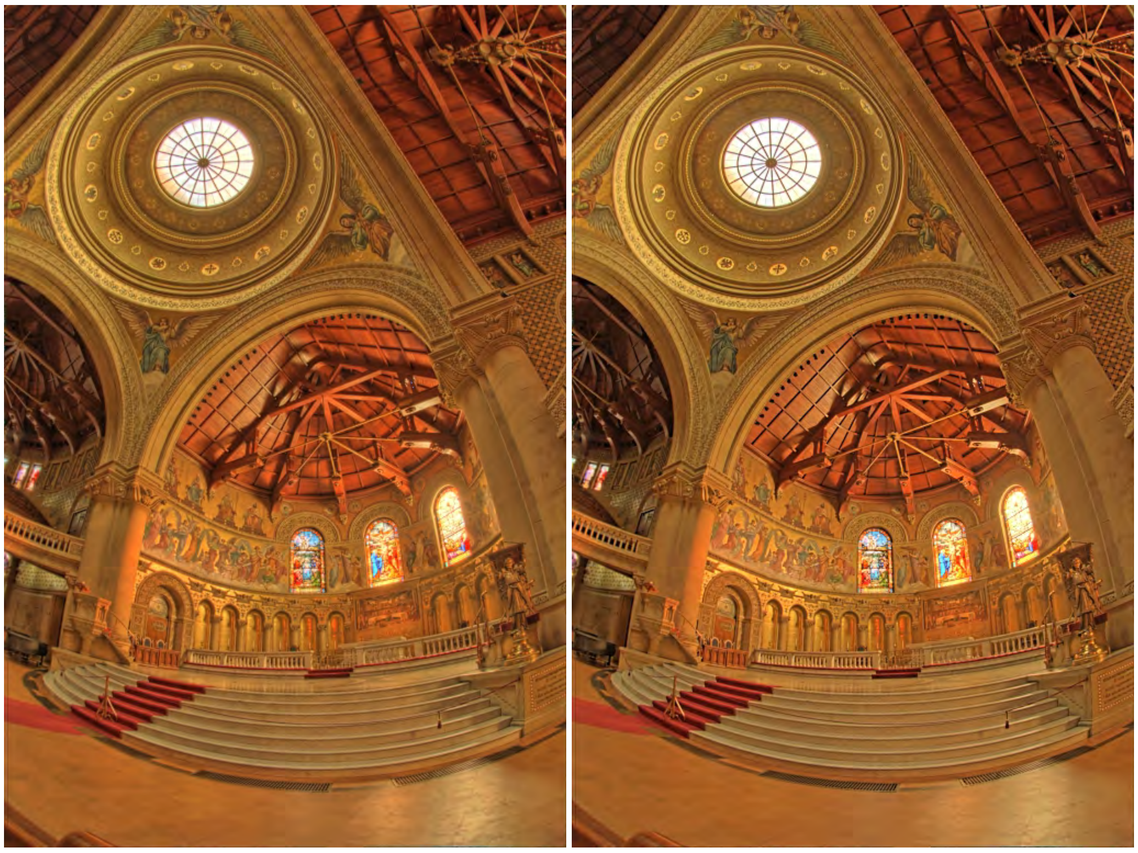

(f) Gaussian

Figure 3.13: Approximating a Gaussian filter. Parameters from all approaches where tuned to best approximate a Gaussian with $\sigma = 15$. For the NC filter, $\sigma_s = 15$, and $\sigma_r = \infty$. Note that WLS and EAW were not designed to approximate Gaussian blur. Best viewed in the electronic version.



(a) Input (b) Ours (IC) (c) EAW

Figure 3.14: Fine detail manipulation. (a) Input image. (b) Our result. J_1 was obtained with the IC filter ($\sigma_s = 20$ and $\sigma_r = 0.08$). (c) EAW result by Fattal (FATTAL, 2009).



(a) Ours (RF) (b) WLS

Figure 3.15: Tone mapping results: (a) using our RC filter and (b) using the WLS filter.

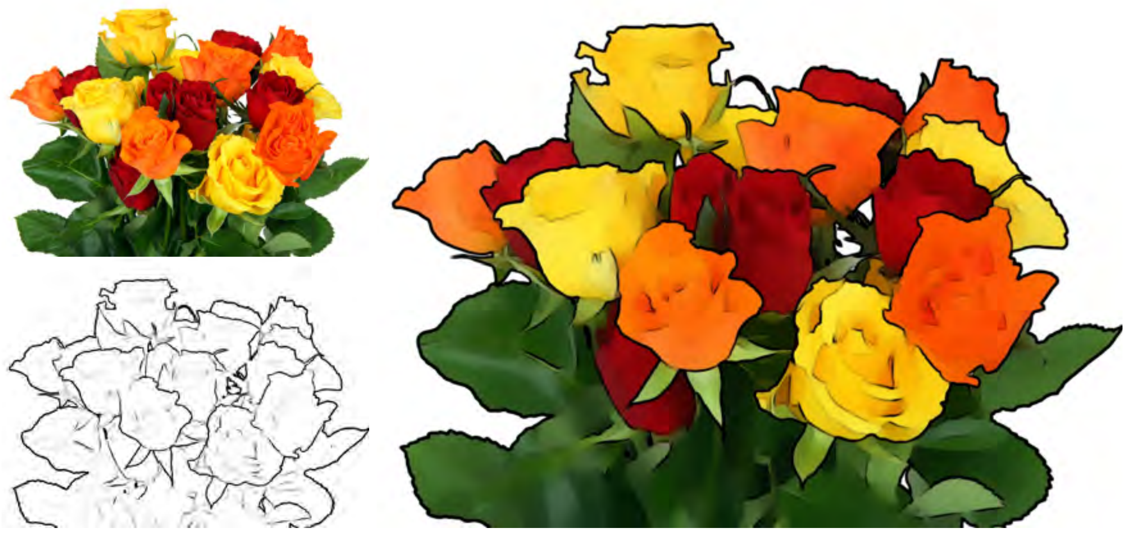


Figure 3.16: High-contrast edges around the most salient features of an edge-aware filtered image, producing a stylized look.

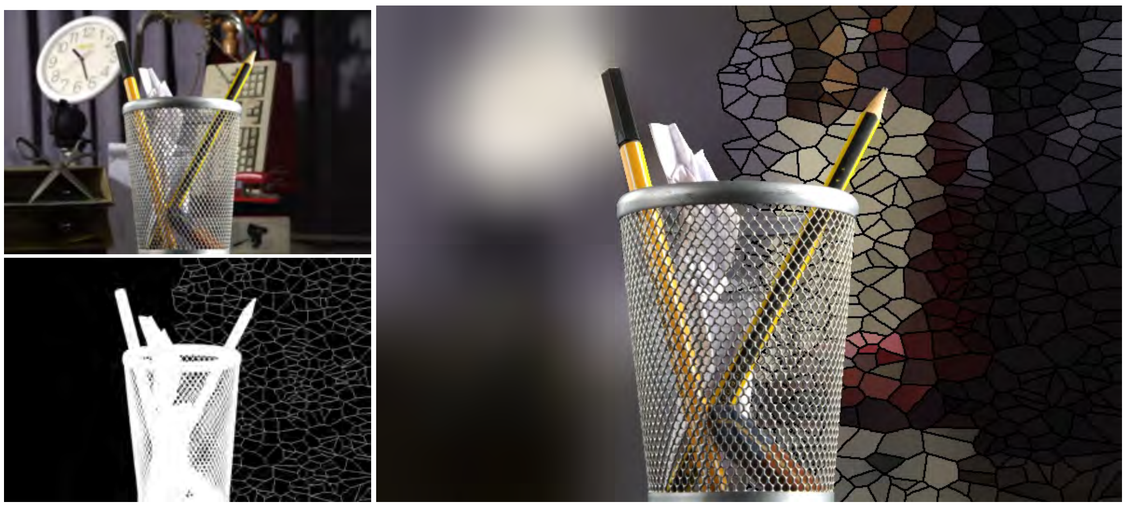
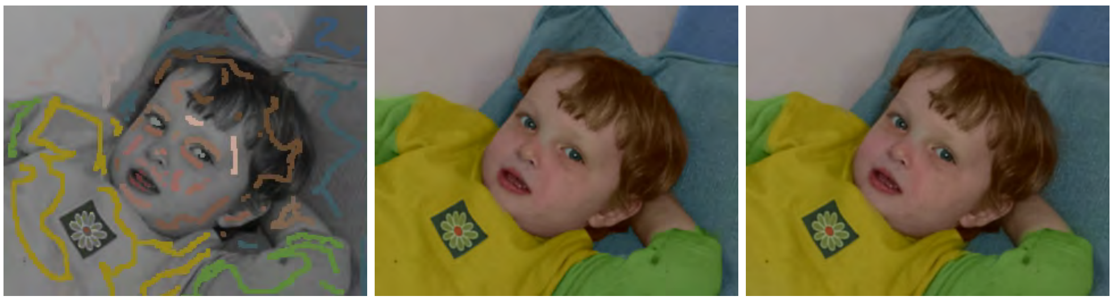


Figure 3.17: Stylized depth-of-field effect. *(Top Left)* Input image. *(Bottom Left)* Alpha matte combined with a Voronoi-like diagram. *(Right)* Result produced by our edge-aware filter using joint filtering. See text for details.



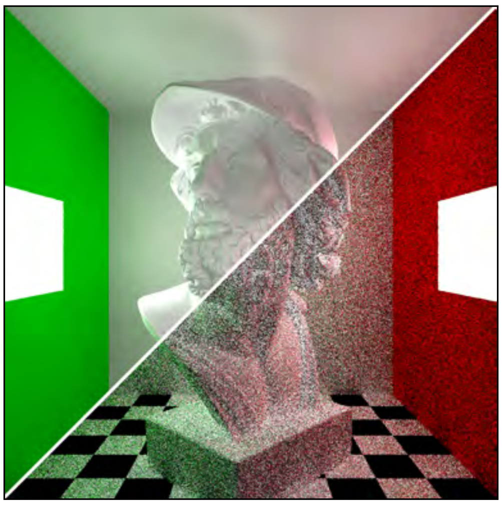
(a) Input (b) Ours (c) Levin et al. (LEVIN; LISCHINSKI; WEISS, 2004)

Figure 3.18: Colorization example. *(a)* Grayscale input image with user scribbles. *(c)* Our result using RF ($\sigma_s = 100$ and $\sigma_r = 0.03$). *(b)* Result of Levin et al. (LEVIN; LISCHINSKI; WEISS, 2004).

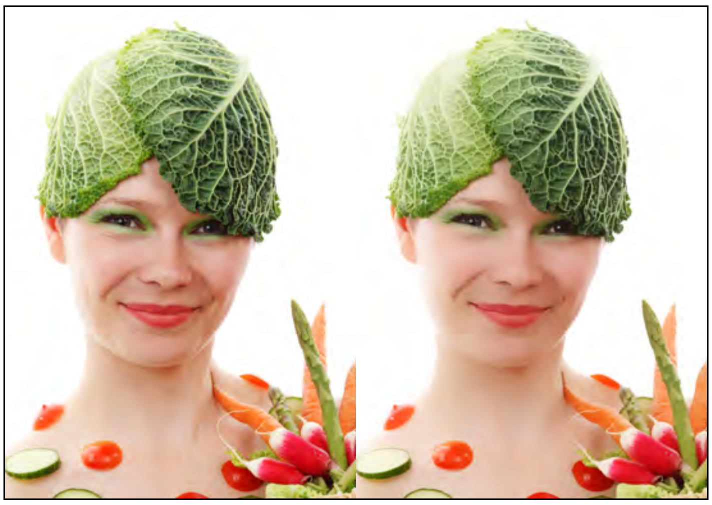
4 ADAPTIVE MANIFOLDS FOR EUCLIDEAN FILTERING

This chapter presents a new approach for efficiently performing high-quality high-dimensional *Euclidean* 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. K controls the visual quality of the resulting filter. For color images, K typically varies from 3 to 15 for high-quality filtering. Thus, the resulting filters are the *first high-dimensional Euclidean 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; COLL; MOREL, 2005). This kind of versatility has also been described by Adams et al. (ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 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.

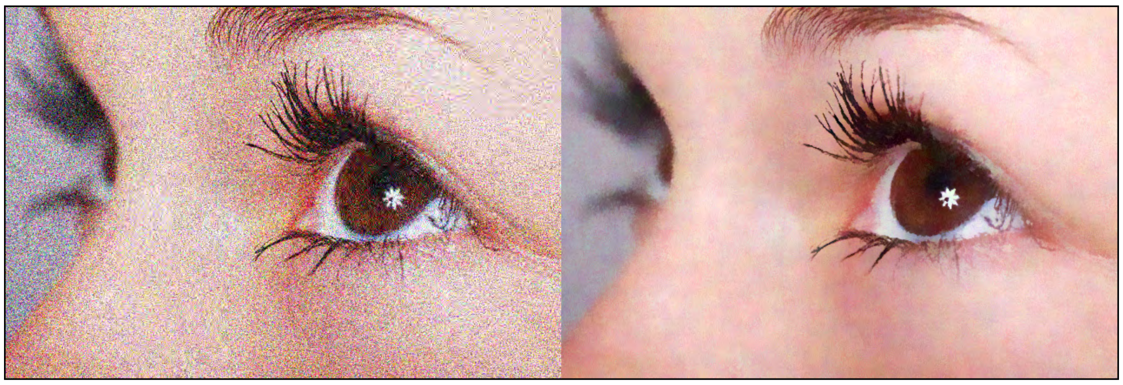
Figure 4.1 shows some filtering examples obtained using our technique. Figure 4.1(a) depicts the result of filtering indirect illumination from an undersampled scene, rendered with path tracing. The filter works in an 8-D space, composed of two spatial dimensions and six range dimensions (3-D scene position and normal vector). Note the quality of the filtered image, despite the highly-noisy input. Figure 4.1(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 4.1(c) shows the result of applying a non-local-means filter to obtain a denoised version (right) of a corrupted image (left). 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.



(a) *Filtering using geometric information*



(b) *Edge-aware smoothing*



(c) *Non-local means denoising*

Figure 4.1: 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.

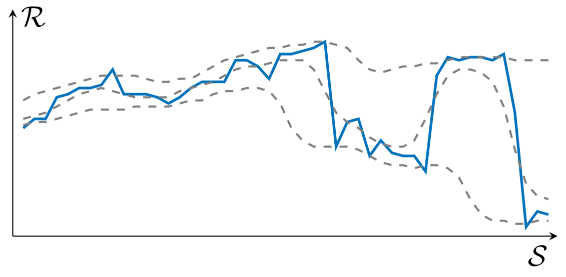
4.1 Adaptive Manifolds

The filtering operation described by Equations 2.2 and 2.3 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 $\hat{p}_i \in \mathcal{S}\times\mathcal{R}$. This approach shows performance gains when $M \ll N$ (PARIS; DURAND, 2009; ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 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; DURAND, 2009; YANG; TAN; AHUJA, 2009; ADAMS; BAEK; DAVIS, 2010). We call such points a *structured set of points*. In this case, using linear-time filtering approaches (DERICHE, 1993; HECKBERT, 1986; YANG; TAN; AHUJA, 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 $O(dNM)$.

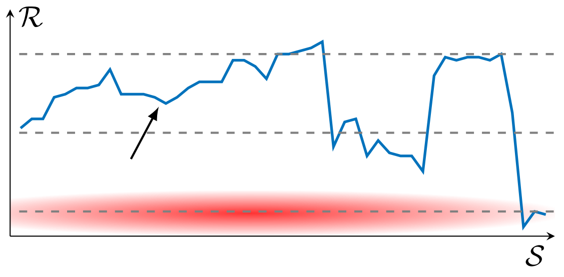
Previous approaches build a structured set of points by laying them onto oriented $d_{\mathcal{S}}$-dimensional flats in $\mathcal{S} \times \mathcal{R}$, either axis-aligned (PARIS; DURAND, 2009; YANG; TAN; AHUJA, 2009) or with a few discrete orientations (ADAMS; BAEK; DAVIS, 2010). These flats define 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 dimensionality of the range domain $d_{\mathcal{R}}$ increases, one needs more flats to enclose the original pixels $\hat{p}_i$ into cells, degrading runtime performance. 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 sparse representations of the high-dimensional space such as hash tables (ADAMS; BAEK; DAVIS, 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, Equation 2.2 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 Equation 2.2, we show (Section 4.2) 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; 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 multilinear or barycentric interpolation. This is a key factor for the performance of our method, and results from the use of normalized convolution;



(a) *1-D Adaptive manifold*



(b) *1-D Standard linear manifold*

Figure 4.2: 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, creating pathways to exchange information among pixels with similar range values. *(b)* Standard linear quantization underrepresents certain regions of the signal (see black arrow), and performs unnecessary work in other regions (in red).

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;

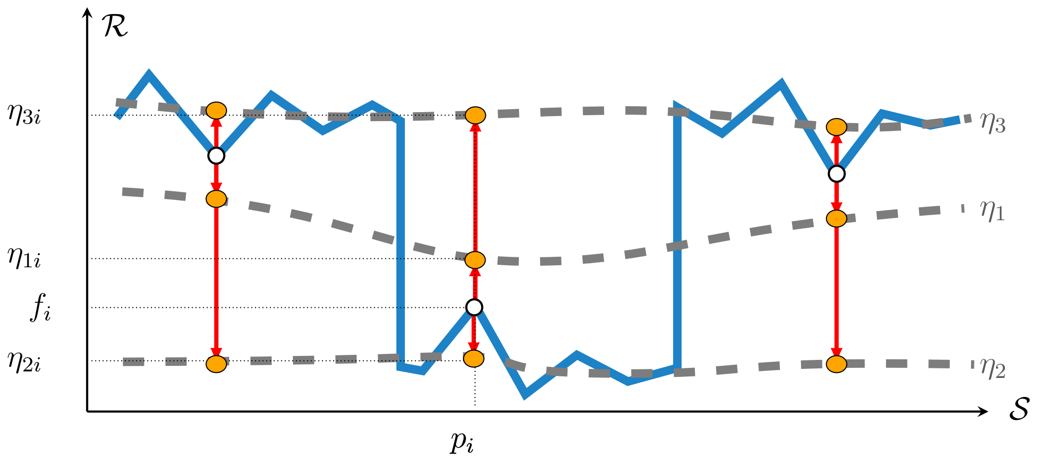
3. *The filter response can be computed in linear time.* This is possible since the sampling points are structured on $d_\mathcal{S}$-dimensional manifolds;

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 (*i.e.*, locally homeomorphic to $\mathbb{R}^{d_S}$), they adapt to the signal equally well, regardless of the range dimensionality $d_\mathcal{R}$.

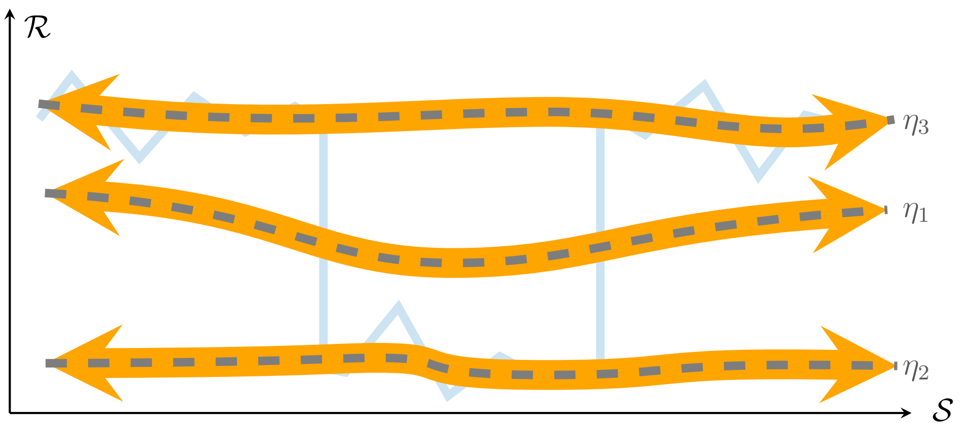
Thus, *our filter is the first high-dimensional Euclidean filter with linear cost both in the number of pixels N and in dimensionality d* (Section 4.5.2). Furthermore, its implementation is faster and requires less memory than previous approaches. Figure 4.2(a) illustrates our approach for a 1-D signal ($d_\mathcal{S} = d_\mathcal{R} = 1$). Our adaptive manifolds (dashed lines) adjust themselves to the underlying signal (in blue) in high-dimensional space (in Figure 4.2(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 the range $d_\mathcal{R}$.

4.1.1 Euclidean Filtering Using Adaptive Manifolds

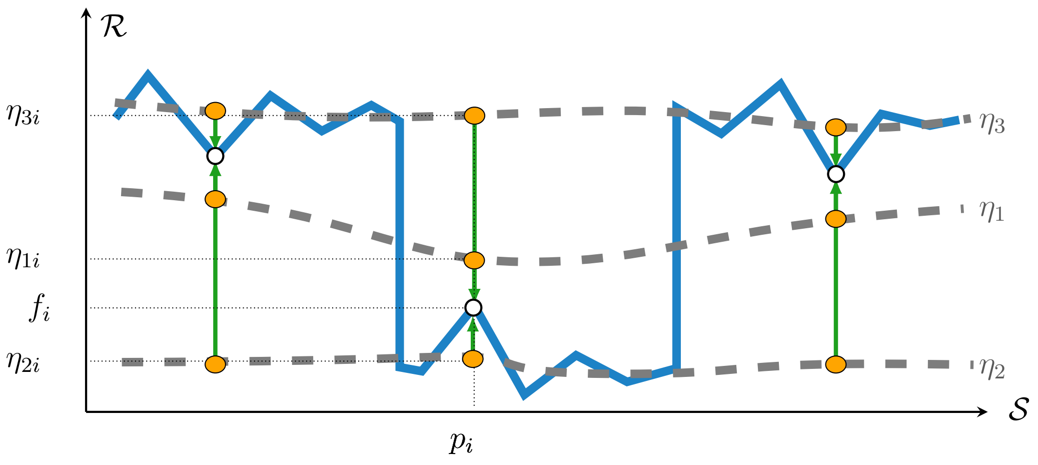
The steps for computing Equation 2.2 with a Gaussian kernel using our nonlinear adaptive manifolds are illustrated in Figure 4.3 for a 1-D signal. To help the readers establish an analogy and compare it with other approaches (*e.g.*, the Gaussian *kd-trees*, the *permutohedral lattice*, and the *5-D bilateral grid*), we use the terminology adopted



(a) *Splatting*



(b) *Blurring*



(c) *Slicing*

Figure 4.3: Steps of our adaptive-manifold filter (in 1-D), using the terminology from Adams et al. (ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 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.

by Adams et al. (ADAMS et al., 2009; ADAMS; BAEK; DAVIS, 2010). Furthermore, to simplify the exposition, this section presents the final form of our equations. Please refer to Section 4.2 for their derivations.

Let K be the total number of adaptive manifolds that will be used to filter a signal f (Section 4.4.1 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 '$\hat{\ }$'). Thus, $\hat{\eta}_{ki} = (p_i, \eta_{ki})$. Section 4.3 discusses how the range coordinates η_{ki} for the sampling points are computed.

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 4.3(a)). The projected values are stored at the sampling points $\hat{\eta}_{ki}$ associated with p_i:

$$\Psi_{splat}(\hat{\eta}_{ki}) = \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_i)\, f_i. \tag{4.1}$$

$\Sigma_\mathcal{R}$ 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 Section 4.2.

Blurring performs Gaussian filtering over each adaptive manifold, mixing the splatted values Ψ_{splat} from all sampling points $\hat{\eta}_{ki}$ (Figure 4.3(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; PARIS; DURAND, 2007; ADAMS; BAEK; DAVIS, 2010; GASTAL; 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 4.3(c)). We gather from the same sampling points $\hat{\eta}_{ki}$ used for splatting p_i (Figure 4.3(a)). Interpolation is done with normalized convolution, described by Knutsson and Westin (KNUTSSON; WESTIN, 1993) in the context of missing data. *g_i is then computed using nearby values known at positions $\hat{\eta}_{ki}$*:

$$g_i = \frac{\sum_{k=1}^{K} w_{ki}\, \Psi_{blur}(\hat{\eta}_{ki})}{\sum_{k=1}^{K} w_{ki}\, \Psi^0_{blur}(\hat{\eta}_{ki})}, \qquad w_{ki} = \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_i). \tag{4.2}$$

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

$$\Psi^0_{splat}(\hat{\eta}_{ki}) = \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_i). \tag{4.3}$$

Contrast Ψ^0_{splat} from Equation 4.3 with Ψ_{splat} from Equation 4.1, and note the absence of the color f_i on the far right of Ψ^0_{splat}.

The next section provides a detailed derivation of these equations and shows that this process indeed approximates the brute-force evaluation of Equation 2.2. Section 4.3 discusses how to compute the sampling points $\hat{\eta}_{ki} = (p_i, \eta_{ki})$ that define the adaptive manifolds.

4.2 Formal Derivation of Our Approach

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

$$\phi_{\Sigma_a+\Sigma_b}(t) = C \int_{\mathbb{R}^d} \phi_{\Sigma_a}(t-\tau)\ \phi_{\Sigma_b}(\tau)\ d\tau, \tag{4.4}$$

where C is a normalization factor. Let $t = x - y$, $\tau = \eta - y$, and $\Sigma_a = \Sigma_b$. Equation 4.4 can then be rewritten as:

$$\phi_\Sigma(x-y) = \frac{1}{|\Sigma|^{1/2}} \left(\frac{2}{\pi}\right)^{\frac{d}{2}} \int_{\mathbb{R}^d} \phi_{\frac{\Sigma}{2}}(x-\eta)\ \phi_{\frac{\Sigma}{2}}(\eta-y)\ d\eta, \tag{4.5}$$

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

Since the Gaussian is separable over orthogonal directions, one can write ϕ_Σ as

$$\phi_\Sigma(\hat{p}_i - \hat{p}_j) = \phi_{\Sigma_\mathcal{S}}(p_i - p_j)\ \phi_{\Sigma_\mathcal{R}}(f_i - f_j)\,, \tag{4.6}$$

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 Equation 4.6 can be rewritten using Equation 4.5 and evaluated numerically (using an approximation to the Gauss-Hermite quadrature rule — see Section 4.2.3) as a weighted sum:

$$\begin{aligned} \phi_{\Sigma_\mathcal{R}}(f_i - f_j) &\propto \int_{\mathbb{R}^{d_\mathcal{R}}} \phi_{\frac{\Sigma_\mathcal{R}}{2}}(f_i - \eta)\, \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta - f_j)\, d\eta \\ &\approx \sum_{k=1}^{K} w_{ki}\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j)\,. \end{aligned} \tag{4.7}$$

The scaling factor outside the integral was not included as it will cancel out in the division performed in Equation 2.2. The K summation 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 K sampling points (*i.e.*, the number of adaptive manifolds) gives a better approximation for the integral. The integration weights w_{ki} are discussed in Section 4.2.3. We can substitute Equations 4.6 and 4.7 in the numerator of Equation 2.2:

$$\sum_{p_j \in \mathcal{S}} \phi_\Sigma(\hat{p}_i - \hat{p}_j)\ f_j \ \overset{\propto}{\sim} \sum_{p_j \in \mathcal{S}} \left[\phi_{\Sigma_\mathcal{S}}(p_i - p_j) \sum_{k=1}^{K} w_{ki}\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j) \right] f_j \tag{4.8a}$$

$$= \sum_{k=1}^{K} w_{ki} \sum_{p_j \in \mathcal{S}} \phi_{\Sigma_\mathcal{S}}(p_i - p_j)\ \phi_{\frac{\Sigma_\mathcal{R}}{2}}(\eta_{ki} - f_j)\ f_j, \tag{4.8b}$$

where "$\underset{\sim}{\propto}$" is the approximately proportional relation. Equation 4.8b was obtained by reordering the summations. The expression for the denominator of Equation 2.2 is similar, not including the last term f_j.

Key observation: by a suitable choice of sampling points $\{\eta_{ki}\}$, the inner summation from Equation 4.8b 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 $K \ll N$.

4.2.1 Choosing the Set of Sampling Points $\{\eta_{ki}\}$

This section shows that, for each $k = 1 \dots K$, if the set of sampling points $\{\hat{\eta}_{ki}\}$ 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, then the inner summation in Equation 4.8b can be computed in $O(dN)$ time *for all* N pixels p_i. This makes the total cost of Gaussian filtering all pixels $O(dNK)$.

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 Equation 4.8b can then be rewritten as

$$\Psi_{blur}(\hat{\eta}_{ki}) = \sum_{p_j \in \mathcal{S}} \phi_{\Sigma_\eta}(\hat{\eta}_{ki} - \hat{p}_j)\ f_j, \quad \Sigma_\eta = \begin{bmatrix} \Sigma_{\mathcal{S}} & \\ & \Sigma_{\mathcal{R}}/2 \end{bmatrix}, \tag{4.9}$$

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

Proposition 4.1: For each $k = 1 \dots K$, if the set of sampling points $\{\hat{\eta}_{ki}\}$ 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 instead of d-dimensional space (note that $d_{\mathcal{S}} < d$ since $d = d_{\mathcal{S}} + d_{\mathcal{R}}$).

Proof. Let $B_{P_k} = (b_1 \dots b_{d_{\mathcal{S}}})$ be any orthonormal basis that spans the flat P_k. Also, let $B_{P_k}^{\perp} = (b_{d_{\mathcal{S}}+1} \dots b_d)$ be any orthonormal basis which is also orthogonal to B_{P_k}. Together, B_{P_k} and $B_{P_k}^{\perp}$ form an orthonormal basis for $\mathbb{R}^d$. Thus, since the Gaussian function is separable in any orthonormal basis, Equation 4.9 can be rewritten as

$$\Psi_{blur}(\hat{\eta}_{ki}) = \sum_{p_j \in \mathcal{S}} \phi_1\left(B_{P_k}^T\, \xi_{kij}\right) \underbrace{\phi_1\left(B_{P_k}^{\perp T}\, \xi_{kij}\right) f_j}_{\Psi_{splat}^{P_k}(\hat{\eta}_{kj})}, \tag{4.10}$$

where

$$\xi_{kij} = \Sigma_\eta^{-1/2}\,(\hat{\eta}_{ki} - \hat{p}_j). \tag{4.11}$$

This decomposition defines a Gaussian filtering of the term $\Psi_{splat}^{P_k}$ in Equation 4.10 over the $d_{\mathcal{S}}$-dimensional flat P_k, and corresponds to the **blurring** step in Figure 4.3 (b). $\square$

The term $\Psi^{P_k}_{splat}(\hat{\eta}_{kj})$ computes a Gaussian distance-weighting of f_j, and corresponds to the **splatting** operation in Figure 4.3(a), modeled by Equation 4.1. Furthermore, Equation 4.8b can be rewritten as

$$\sum_{k=1}^{K} w_{ki} \; \Psi_{blur}(\hat{\eta}_{ki}).$$

This is, after the normalization defined in Equation 2.2, the **slicing** operation modeled by Equation 4.2 and illustrated in Figure 4.3(c).

The subscript '1' of the Gaussian ϕ_1 in Equation 4.10 indicates that its covariance matrix 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 vectors of B. The scaling matrix $\Sigma_\eta^{-1/2}$ in Equation 4.11 transforms an anisotropic, axis-aligned Gaussian onto an isotropic Gaussian with identity covariance matrix (hence, ϕ_1).

Proposition 4.2: For each $k = 1 \ldots K$, if the set of sampling points $\{\hat{\eta}_{ki}\}$ 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 4.1, if the manifold M_k is approximately linear in a neighborhood of size 3 around each pixel p_i (as measured in a space scaled by $\Sigma_\mathcal{S}^{-1}$), $\Psi_{blur}(\hat{\eta}_{ki})$ can be well approximated for all $\hat{\eta}_{ki}$ by a Gaussian filtering over M_k. $\square$

Splatting the pixel colors onto the manifold M_k has cost $O(d_\mathcal{R})$ (Section 4.2.2 presents further discussion on this). Finally, approaches for performing filtering over manifolds (SOCHEN; KIMMEL; BRUCKSTEIN, 2001) include methods which work in $O(dN)$ time for N d-dimensional points (CRIMINISI et al., 2010; GASTAL; OLIVEIRA, 2011). Thus, if one selects a sampling set $\{\hat{\eta}_{ki}\}$ lying on the manifolds M_k, the value $\Psi_{blur}(\hat{\eta}_{ki})$ can be computed for all $\hat{\eta}_{ki}$ in $O(dN)$ time. This makes the total cost of Gaussian filtering all pixels $O(dNK)$.

4.2.2 Splatting onto the Adaptive Manifolds

The term $\Psi^{P_k}_{splat}(\hat{\eta}_{kj})$ in Equation 4.10 defines a Gaussian distance-weighted splatting of a pixel's color f_j onto a flat P_k. Note that the argument of the Gaussian is the distance of $\hat{p}_j$ to the flat P_k (in a space scaled by $\Sigma_\eta^{-1/2}$).

Our approach uses nonlinear manifolds instead of flats, thus, when splatting a pixel's color f_j onto an adaptive manifold M_k, one must compute the distance from $\hat{p}_j$ to M_k. Since this manifold is nonlinear (it is only approximately linear), this can be done in a few ways.

Following the term $\Psi^{P_k}_{splat}(\hat{\eta}_{kj})$ in Equation 4.10, one way to compute the distance to a manifold would be to find the flat P_{kj} which approximates this manifold in the neighborhood around p_j. The Gaussian distance-weighted projection should then be performed using the closest distance between $\hat{p}_j$ and P_{kj}. However, finding this flat would require some sort of linear regression on points sampled from the manifold; or the computation of tangent flats.

Another way of computing this distance is by projecting the point $\hat{p}_j$ onto the manifold M_k along the dimensions of the range $\mathcal{R}$. With this approach, the projection point on the manifold would be, by definition, $\hat{\eta}_{kj}$. Furthermore, since this projection is along $\mathcal{R}$, the complexity of computing the distance from $\hat{p}_j$ to $\hat{\eta}_{kj}$, used for splatting, is $O(d_{\mathcal{R}})$ (remember that $\hat{p}_j$ and $\hat{\eta}_{kj}$ have exactly the same coordinates in $\mathcal{S}$). This is the approach we use to splat onto the manifolds (Equation 4.1). However, notice that at the projection point $\hat{\eta}_{kj}$ on the manifold, the basis which locally spans the manifold (and which defines, in the blurring stage, the direction of blurring) is not orthogonal to the direction of this projection (*i.e.*, not orthogonal to the vector $\hat{\eta}_{kj} - \hat{p}_j$). Thus, this introduces an accuracy penalty, since the Gaussian is only truly separable for orthogonal directions. A similar error appears in other approaches which discretize the high-dimensional space (*i.e.*, it is a sampling error).

4.2.3 Approximate Gauss-Hermite Quadrature

The Gauss-Hermite quadrature rule (NIST, 2011) defines the following approximation for a Gaussian integral:

$$\int_{\mathbb{R}} \phi_{\Sigma_a}(y-x)\ \phi_{\Sigma_b}(x-z)\ dx \approx \sum_{k=1}^{K} w_k\, \phi_{\Sigma_b}(x_k - z)\,, \tag{4.12}$$

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 Equation 4.7 by successive applications (ARASARATNAM; HAYKIN; ELLIOTT, 2007). According to this rule, the integral in Equation 4.7 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 4.3 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 4.2, strongly impacting the performance of our method. Nonetheless, one can still approximate the integral in Equation 4.7 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 Equation 4.12; (ii) for non-outlier pixels, the set $\{\eta_{ki}\}$ computed in Section 4.3 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 samples of the original signal $f_i \in \mathcal{R}$; (iii) the integrand in Equation 4.7 is given by smooth Gaussian functions; and (iv) Equation 2.2 can be seen as a Gauss transform in homogeneous coordinates, which (as discussed by Adams (ADAMS, 2011)) hides scaling errors in the division by the sum of the weights.

From Equation 4.8b, Equation 4.9, and Equation 4.12, it follows that the estimator for the filtered value of a pixel is given by Equation 4.2, which defines a *normalized convolution interpolator* (KNUTSSON; WESTIN, 1993). Next, we discuss how to compute the sampling points $\hat{\eta}_{ki} = (p_i, \eta_{ki})$ that define the adaptive manifolds.

4.3 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 4.1 generates manifolds (*i.e.*, functions η_k) with the following **properties**:

- *They are approximately linear in all local neighborhoods.* As we show in Section 4.2, this is required to obtain a good approximation of Equation 2.2 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 Equation 4.2, producing good estimates for Equation 2.2. It also reduces bias, as most pixels are well represented by the manifolds (see Section 4.4 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 in Section 4.3.1). 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 $\mathbf{f}_1 = (f_1 \dots f_N)$ is a $d_{\mathcal{R}} \times N$ matrix containing all pixel colors f_i, and $\boldsymbol{\eta}_1 = (\eta_{11} \dots \eta_{1N})$ is a $d_{\mathcal{R}} \times N$ matrix containing all manifold values η_{1i} associated with each pixel p_i. Section 4.3.2 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 $dot_i = v_1^T(f_i - \eta_{1i})$:

$$\begin{cases} p_i \in \mathcal{C}_- & \text{if } dot_i < 0, \\ p_i \in \mathcal{C}_+ & \text{otherwise.} \end{cases} \tag{4.13}$$

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 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 $\mathcal{C}_-$:

$$\begin{aligned} \eta_-(p_i) &= \sum_{p_j \in \mathcal{C}_-}^{N} W_-(p_j)\, f_j \Bigg/ \sum_{p_j \in \mathcal{C}_-}^{N} W_-(p_j), \\ W_-(p_j) &= \theta(\eta_{1j} - f_j)\, h_{\Sigma_{\mathcal{S}}}(p_i - p_j). \end{aligned} \tag{4.14}$$

$h_{\Sigma_{\mathcal{S}}}$ 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:

$$\theta(\eta_{1j} - f_j) = 1 - w_{1j}. \tag{4.15}$$

The values w_{1j} are the interpolation weights from Equation 4.2. The construction of the manifold η_+ associated with $\mathcal{C}_+$ is done similarly.

Step 5: Based on the stopping criterion discussed in Section 4.4.1, 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 $p_i \in \mathcal{C}_+$. $\square$

These steps are shown in Algorithm 4.1, and define manifolds in a hierarchical tree (Figure 4.4). Going further down this tree yields manifolds better adapted to local populations: each cluster will contain fewer and more correlated pixels, and Equation 4.14 will perform more robust local mean estimates. Figure 4.5 shows the first three levels of the *manifold tree* computed to filter the image in Figure 4.9(a).

Algorithm 4.1 Computing the Adaptive Manifolds

```
$\eta_1 \leftarrow h_{\Sigma_{\mathcal{S}}} * f$                                        (* First manifold *)
$\mathcal{C}_1 \leftarrow \{p_1, \dots, p_N\}$                    (* 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 Equation 4.14
    (* Recurse if more manifolds are needed *)
    $\mathcal{M}_- \leftarrow \{\eta_-\}$; $\mathcal{M}_+ \leftarrow \{\eta_+\}$;
    if stopping criterion not reached (Section 4.4.1) 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
```

$$\begin{tikzcd}
& & & \eta_1 \arrow[dll, dash] \arrow[drr, dash] & & & \\
& \eta_- \arrow[dl, dash] \arrow[dr, dash] & & & & \eta_+ \arrow[dl, dash] \arrow[dr, dash] & \\
\eta_{--} & & \eta_{-+} & & \eta_{+-} & & \eta_{++} \\
\iddots \quad \ddots & & \iddots \quad \ddots & & \iddots \quad \ddots & & \iddots \quad \ddots
\end{tikzcd}$$

Figure 4.4: Manifold tree constructed using Algorithm 4.1.

4.3.1 Low-Pass Filtering

Section 4.2 shows that a good approximation for Equation 2.2 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. This is proven below. The following analysis is performed for 1-D signals, however the same argument generalizes to arbitrary dimensions due to the separability of the filtering operations. We also assume f is twice differentiable, which is not a significant constraint since f (the input image) is sampled (discretized).

Definition 4.1: A 1-D function (*e.g.*, manifold) $\eta(x)$ is linear if its second derivative (or *curvature*) is zero for all x: $\partial_{xx}\eta = 0$. It is said to be approximately-linear in an interval $[a, b]$ if the total curvature over this interval is less than some small value ϵ:

$$\int_a^b \left|\partial_{xx}\eta\right| dx < \epsilon. \tag{4.16}$$

Proposition 4.3: For any $\epsilon > 0$, any interval $[a, b]$, and any signal f whose derivatives follow natural-image distributions (WEISS; FREEMAN, 2007), there exists a low-pass filter h_σ with standard deviation σ for which the function defined by the convolution $\eta = f * h_\sigma$ is approximately linear according to Equation 4.16.

Proof. The derivative of a convolution can be decomposed as:

$$\partial_{xx}\eta = \partial_{xx}(f * h_\sigma) = (\partial_{xx}f) * h_\sigma. \tag{4.17}$$

This convolution performs local averaging of the values $\partial_{xx}f$, which are realizations of a random variable. Equation 4.17 converges to the population's expected value $E[\partial_{xx}f]$ as more samples are averaged — *i.e.*, as the standard deviation σ of the filter increases. For a particular σ value, the variance of the estimator given by Equation 4.17 is proportional to $\mathrm{Var}[\partial_{xx}f]/\sigma$. Thus, for some proportionality constant γ, one can state a high-confidence bound on the error of the estimator:

$$\left|(\partial_{xx}f) * h_\sigma - E[\partial_{xx}f]\right| < \frac{\gamma}{\sigma}\,\mathrm{Var}[\partial_{xx}f]. \tag{4.18}$$

The output of derivative filters applied to natural images have (non-Gaussian) distributions which peak at, and are symmetric about, zero (WEISS; FREEMAN, 2007), and hence

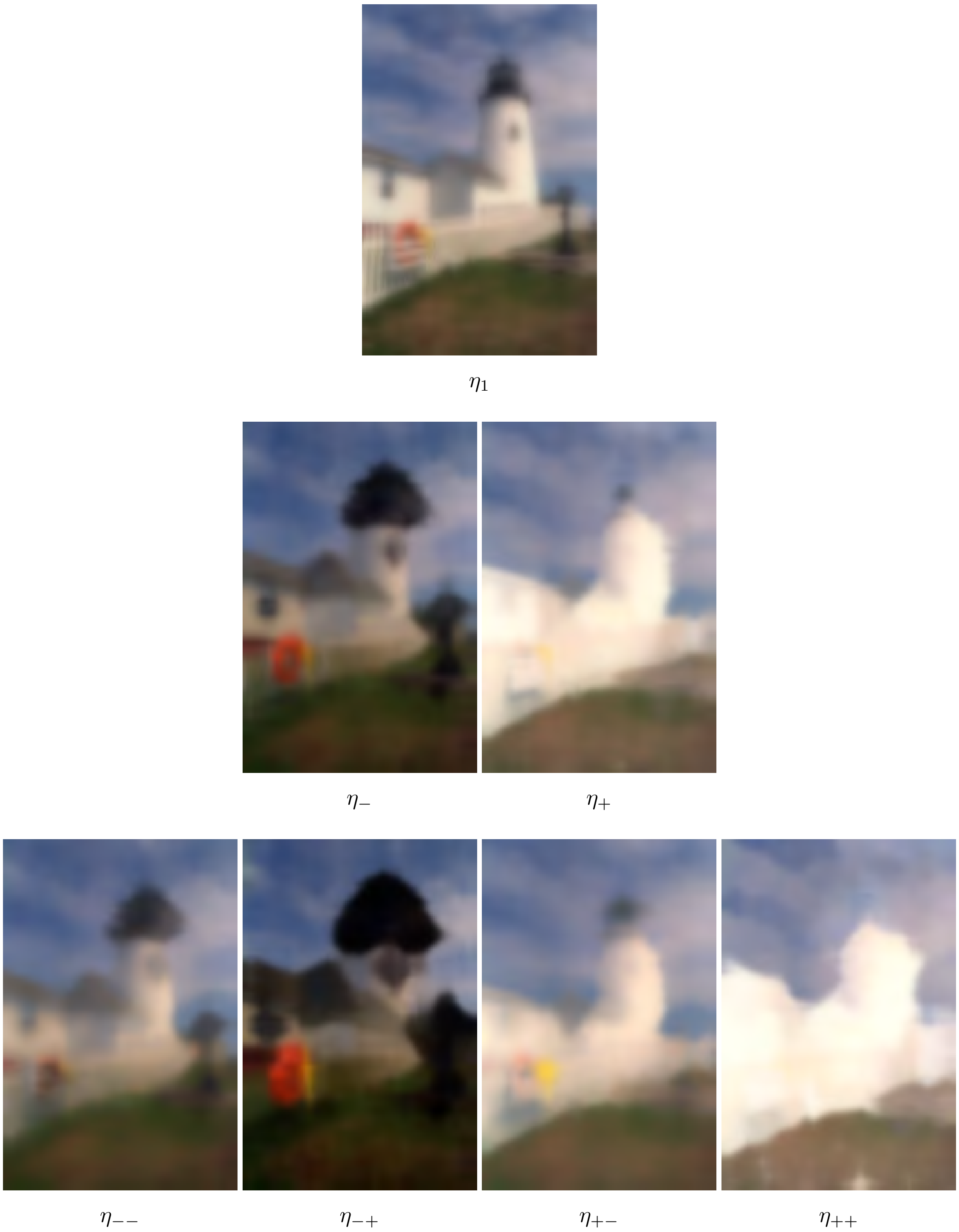


Figure 4.5: The manifolds used by our filter to compute Figure 4.9(b).

$E[\partial_{xx} f] \approx 0$. Plugging this into Equation 4.18 and integrating both sides in $[a, b]$:

$$\int_a^b \left|(\partial_{xx} f) * h_\sigma\right| dx < \int_a^b \frac{\gamma}{\sigma} \operatorname{Var}[\partial_{xx} f]\, dx. \tag{4.19}$$

Evaluating the integral on the right-hand-side yields:

$$\int_a^b \left|(\partial_{xx} f) * h_\sigma\right| dx < \frac{\gamma}{\sigma}(b-a) \operatorname{Var}[\partial_{xx} f]. \tag{4.20}$$

Since the standard deviation σ is a free parameter of the low-pass filter, for any $\epsilon > 0$ and any interval $[a, b]$ one can always find a filter h_σ with standard deviation σ sufficiently large such that the right-hand-side of Equation 4.20 evaluates to less than ϵ, and thus $\eta = f * h_\sigma$ is approximately linear according to Definition 4.1. $\square$

Note that Proposition 4.3 is *not* valid for a general signal f. One particular example would be the signal defined by $f(x) = x^2$, for which a convolution with any symmetric low-pass filter (disregarding boundary conditions) does not alter its curvature $\partial_{xx} x^2 = 2$.

According to our experience, using (in Steps 1 and 4 of our algorithm) a low-pass filter h_{Σ_S} with covariance matrix given by Σ_S (*i.e.*, the spatial variance of ϕ_Σ) produces adaptive manifolds which are sufficiently linear (according to Proposition 4.2) and generate good filtering results. Section 5.4.3 discusses implementation details for this low-pass filter.

4.3.2 Fast Eigenvector Approximation in $O(d_{\mathcal{R}} N)$ Time

Proposition 4.4: 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 $O(d_{\mathcal{R}}^2 N)$ cost of computing the matrix XX^T, which is very desirable when $m \ll d_{\mathcal{R}}$.

Proof. It can be shown 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 Section 4.3.2.1). 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 $O(2\, m\, d_{\mathcal{R}} N)$. $\square$

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 when the range dimensionality is larger than 20 ($d_{\mathcal{R}} > 20$). Such a low number of iterations is possible since this algorithm converges quickly when the vectors in the matrix $X = \mathbf{f}_k - \boldsymbol{\eta}_k$ are highly anisotropic (*i.e.*, when some eigenvalues of XX^T 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 4.3). 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 coherence. The first frame is filtered using a random w.

4.3.2.1 Dominant Eigenvector Computation

Proposition 4.5: For a positive-definite matrix XX^T of size $d_\mathcal{R} \times d_\mathcal{R}$, with eigenvectors $v_1, \ldots, v_{d_\mathcal{R}}$ and eigenvalues $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_{d_\mathcal{R}} > 0$, it is true that

$$\lim_{m\to\infty} (XX^T)^m w \propto v_1;$$

where w is a random vector not orthogonal to v_1. This is the Power Method (DATTA, 2010).

Proof. The eigenvectors $v_1, \ldots, v_{d_\mathcal{R}}$ form an orthonormal basis for $\mathbb{R}^{d_\mathcal{R}}$. The vector w can be expressed on this basis as

$$w = a_1 v_1 \ + \ a_2 v_2 \ + \ \cdots \ + \ a_{d_\mathcal{R}} v_{d_\mathcal{R}},$$

for scalars $a_1, \ldots, a_{d_\mathcal{R}}$. From this it follows that

$$\begin{aligned}
(XX^T)^m w &= (XX^T)^m (a_1 v_1 \ + \ a_2 v_2 \ + \ \cdots \ + \ a_{d_\mathcal{R}} v_{d_\mathcal{R}}) \\
&= a_1 (XX^T)^m v_1 \ + \ a_2 (XX^T)^m v_2 \ + \ \cdots \ + \ a_{d_\mathcal{R}} (XX^T)^m v_{d_\mathcal{R}} \\
&= a_1 \lambda_1^m v_1 \ + \ a_2 \lambda_2^m v_2 \ + \ \cdots \ + \ a_{d_\mathcal{R}} \lambda_{d_\mathcal{R}}^m v_{d_\mathcal{R}} \\
&= a_1 \lambda_1^m \left(v_1 + \frac{a_2}{a_1} \left(\frac{\lambda_2}{\lambda_1} \right)^m v_2 + \cdots + \frac{a_{d_\mathcal{R}}}{a_1} \left(\frac{\lambda_{d_\mathcal{R}}}{\lambda_1} \right)^m v_{d_\mathcal{R}} \right).
\end{aligned}$$

The last line is well defined since w is not orthogonal to v_1 (*i.e.*, $a_1 \neq 0$). Noting that

$$\text{if} \quad (\forall i \neq 1, \lambda_1 > \lambda_i) \quad \text{then} \quad \left(\lim_{m\to\infty} \left(\frac{\lambda_i}{\lambda_1} \right)^m = 0 \right),$$

we have

$$\lim_{m\to\infty} (XX^T)^m \, w \ = \ a_1 \lambda_1^m v_1 \ \propto \ v_1. \ \square$$

Observation: if the largest eigenvalue λ_1 is not unique (*i.e.*, the characteristic polynomial has a multiple root at λ_1, with multiplicity r), $(XX^T)^m w$ will converge to a linear combination of all r eigenvectors $v_1, \ldots, v_r$ associated with λ_1. This combination will be defined by the scalars $a_1, \ldots, a_r$ — *i.e.*, will be defined by the choice of random vector w. For our filter, this means that the pixels are isotropically distributed around the manifolds for the r directions of maximum variation. Thus this combination of eigenvectors will provide a good segmentation for Step 3 of our manifold creation process (Section 4.3).

Note also that using a value of $m + 1$ for the exponent always produces a better approximation for v_1 than using a value of m. In practice, though, we found that as little as $m = 1$ iteration is enough for obtaining good 5D RGB color filtering; and $m = 2$ to $m = 3$ is enough for non-local means denoising.

4.4 Filter Behavior Analysis

It is instructive to analyze the behavior of our filter by contrasting its response with the response of the brute-force filter defined by Equation 2.2, 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 Equation 2.2 *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 Equation 4.2, 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 4.1 do not represent them well. As a result, the values of their colors f_i are highly attenuated during their projections onto the manifolds (Equation 4.1, illustrated in Figure 4.3(a)).

One can handle outlier pixels in **two ways**: *accept* the values computed by Equation 4.2, or *change* the filtered values to better approximate the response of Equation 2.2. 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 4.15 shows an example where our approach was used to implement a *non-local-means* filter (BUADES; COLL; MOREL, 2005). Note how the resulting filter (d) is slightly more aggressive in reducing noise than a conventional *non-local-means* filter (e). 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 Equation 4.2 to better approximate the response of Equation 2.2, 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

$$g_i = \alpha_i\, \tilde{g}_i + (1 - \alpha_i)\, f_i, \quad \alpha_i = \max\nolimits_k\big(\phi_\Sigma(\hat{p}_i - \hat{\eta}_{ki})\big); \tag{4.21}$$

where $\tilde{g}_i$ is the result of Equation 4.2 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 Equation 2.3 is 1), g_i becomes a linear interpolation of $\tilde{g}_i$ and f_i. For an outlier pixel (far away from all manifolds): $\alpha_i \approx 0$ and $g_i \approx f_i$; while for a non-outlier pixel (close to at least one manifold): $\alpha_i \approx 1$ and $g_i \approx \tilde{g}_i$.

4.4.1 Stopping Criteria

According to Proposition 4.2, 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*

		σ_s						
		1	4	8	16	32	64	128
σ_r	0.01	3	3	3	7	15	31	63
	0.10	3	3	3	7	15	31	63
	0.20	3	3	3	7	15	15	31
	0.40	3	3	3	3	7	7	15
	1.00	3	3	3	3	3	3	3

Table 4.1: Several values for number of manifolds (K) computed for RGB color image filtering from Equation 4.22. This table assumes isotropic kernels in space $\mathcal{S}$ and range $\mathcal{R}$, with standard deviations given by σ_s and σ_r, respectively (see first paragraph of Section 4.6).

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 (Equation 4.2) also increases. This means that *fewer manifolds will be needed to estimate Equation 2.2 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 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 4.4), from which the number of manifolds K can be obtained as $K = 2^H - 1$.

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

$$H = \max\big(2, \lceil H_\mathcal{S}\, L_\mathcal{R} \rceil\big), \tag{4.22}$$

where $H_\mathcal{S}$ is a height computed from the spatial standard deviations of the filter, and $L_\mathcal{R}$ is a linear correction computed from the range standard deviations:

$$H_\mathcal{S} = \lfloor \log_2 (\sigma_{\mathcal{S}_{max}}) \rfloor - 1, \quad \text{and} \quad L_\mathcal{R} = 1 - \sigma_{\mathcal{R}_{min}}.$$

The maximum and minimum standard deviations $\sigma_{\mathcal{S}_{max}}$ and $\sigma_{\mathcal{R}_{min}}$ are computed from their respective covariance matrices:

$$\sigma_{\mathcal{S}_{max}} = \sqrt{\max(\Sigma_\mathcal{S})}, \quad \text{and} \quad \sigma_{\mathcal{R}_{min}} = \sqrt{\min(\Sigma_\mathcal{R})}. \tag{4.23}$$

Table 4.1 shows several values for the number of manifolds (K), computed using Equation 4.22 for performing RGB color image filtering. Note how these values follow the guidelines outlined in the beginning of this section: the number of manifolds should *increase* with increasing spatial standard deviation, and should *decrease* with increasing range standard deviation.

Using the tree height given by Equation 4.22 and treating outliers according to Equation 4.21, one achieves good accuracy when compared to brute-force bilateral filtering:

		σ_s						
		1	4	8	16	32	64	128
σ_r	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 4.2: 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. The number of manifolds, K, was computed using the tree height given by Equation 4.22. Section 4.6 defines the parameters σ_s and σ_r.

average PSNR above 40 dB for most combinations of σ_s and σ_r parameters (Table 4.2). Towards the bottom-right of Table 4.2, the adaptive-manifold filter's results diverge from the bilateral filter's results due to our choice of low-pass filter over the manifolds (we use the RF filter from Chapter 3, discussed in Section 5.4.3), 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 4.6). Figure 4.7 provides further examples of the impact, on the filtered images, of varying the values of the parameters σ_s and σ_r.

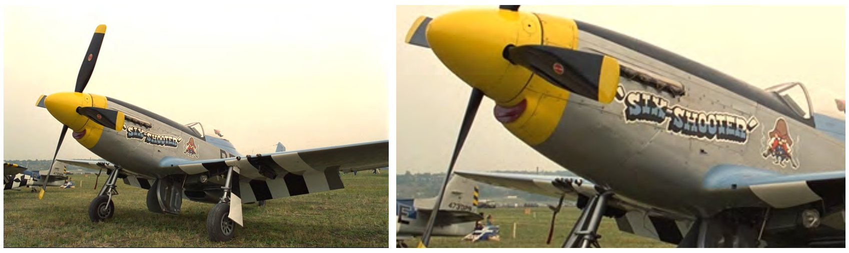
For **natural image denoising** using the non-local-means algorithm (BUADES; COLL; MOREL, 2005), more manifolds are required to deal with the noisy data. Good results are achieved with a small increment to the tree height used for color bilateral filtering:

$$H = 2 + \max\big(2, \lceil H_{\mathcal{S}}\, L_{\mathcal{R}} \rceil\big). \tag{4.24}$$

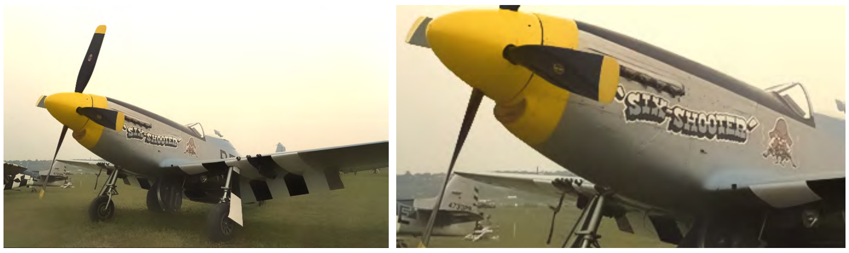
Note that *the number of manifolds is independent of the filter dimensionality* (see Section 4.6.3). In our tests, we achieve consistent results by using the same tree height (Equation 4.24) for denoising in dimensions from 6 to 147 (Figure 4.15). Finally, since outliers will be mostly noise, they should be suppressed (*i.e.*, do not use Equation 4.21).

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 4.1 and Figure 4.17 show an example of noise reduction applied to an image rendered using path tracing, which will be discussed in Section 4.7.

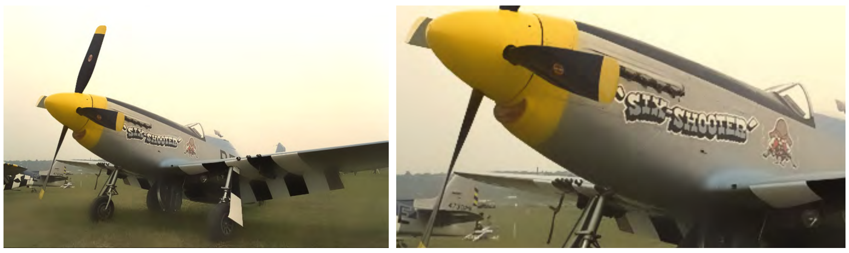
For **other applications**, the number of manifolds can be dynamically computed. A



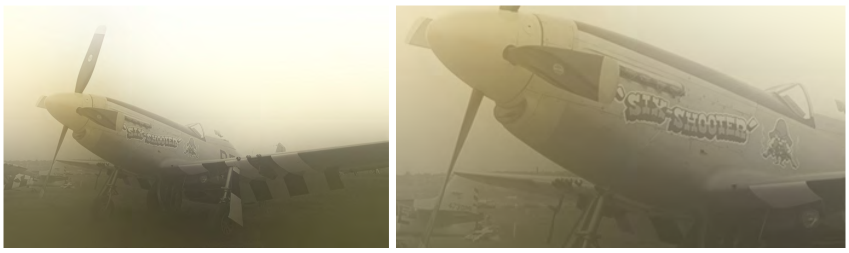
(a) *Input*



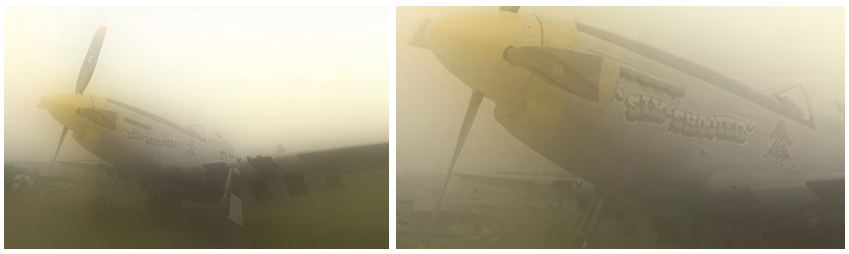
(b) *Bilateral filter* ($\sigma_s = 16,\ \sigma_r = 0.2$)



(c) *Ours* ($\sigma_s = 16,\ \sigma_r = 0.2$)*, PSNR 41 dB vs Bilateral filter*



(d) *Bilateral filter* ($\sigma_s = 64,\ \sigma_r = 1.0$)



(e) *Ours* ($\sigma_s = 64,\ \sigma_r = 1.0$)*, PSNR 37 dB vs Bilateral filter*

Figure 4.6: 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 4.2), 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 5.4.3).

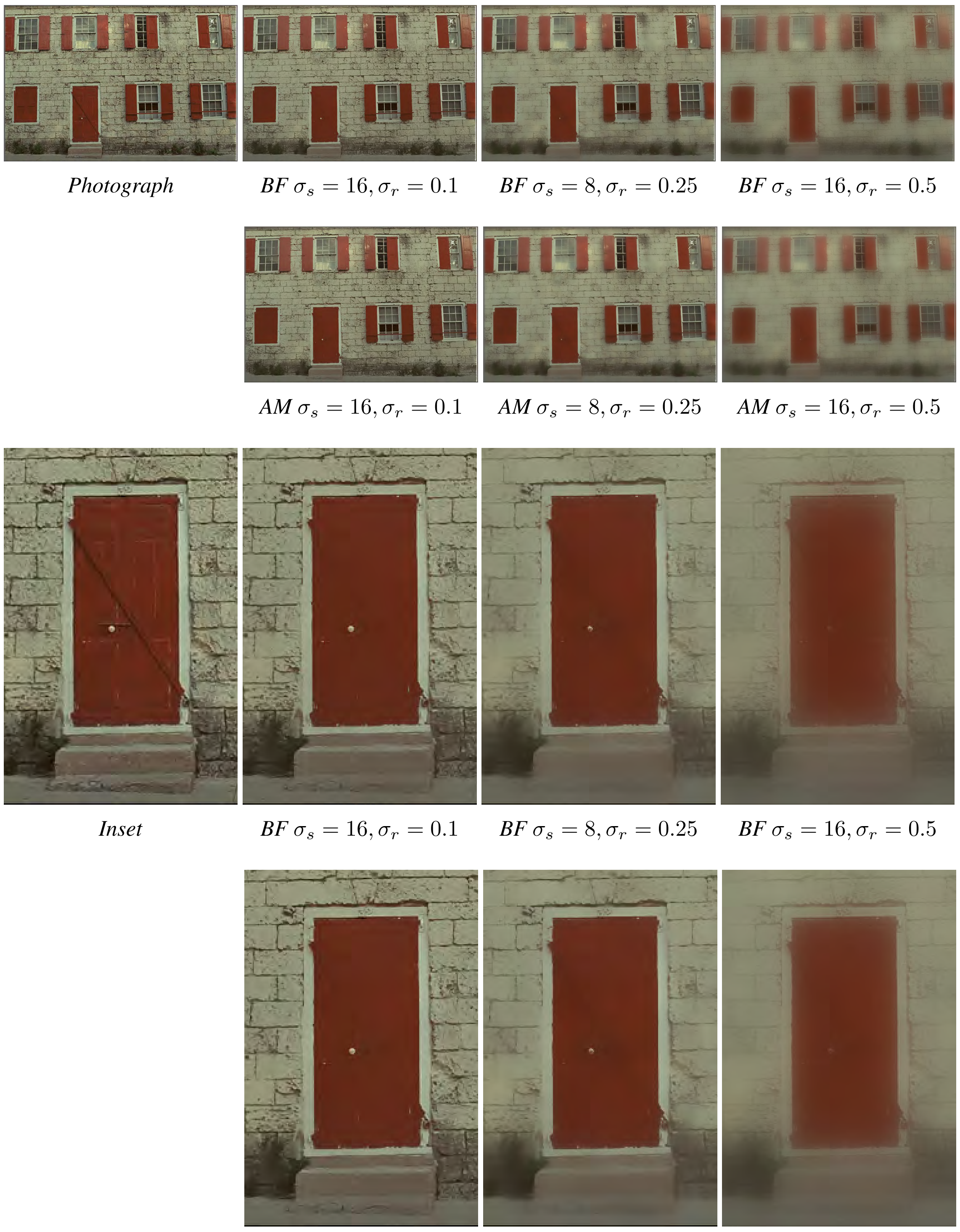

Photograph
BF $\sigma_s = 16, \sigma_r = 0.1$
BF $\sigma_s = 8, \sigma_r = 0.25$
BF $\sigma_s = 16, \sigma_r = 0.5$
AM $\sigma_s = 16, \sigma_r = 0.1$
AM $\sigma_s = 8, \sigma_r = 0.25$
AM $\sigma_s = 16, \sigma_r = 0.5$
Inset
BF $\sigma_s = 16, \sigma_r = 0.1$
BF $\sigma_s = 8, \sigma_r = 0.25$
BF $\sigma_s = 16, \sigma_r = 0.5$
AM $\sigma_s = 16, \sigma_r = 0.1$
AM $\sigma_s = 8, \sigma_r = 0.25$
AM $\sigma_s = 16, \sigma_r = 0.5$

Figure 4.7: Our adaptive-manifold filter (AM) achieves a PSNR above 40 dB when compared to brute-force bilateral filtering (BF), which is practically indistinguishable visual difference.

simple and effective approach would be to traverse the manifold tree (Figure 4.4) 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 4.4 in accordance with the desired filter response.

4.5 Implementation Details

Our full MATLAB source code is available in Appendix B.

For the *low-pass* filter $h_{\Sigma_\mathcal{S}}$ used to compute the adaptive manifolds η_k (see Step 1 of the algorithm in Section 4.3), we use a linear-time recursive filter with exponential decay:

$$out[i] = in[i] + \exp\left(-\sqrt{2}/\sigma_l\right)(out[i-1] - in[i])\,. \tag{4.25}$$

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, Equation 4.25 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 $\{\eta_{ki}\}$ 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 Equation 4.14), they define an "even more" band-limited signal. Thus, one can correctly represent the set $\{\eta_{ki}\}$ using fewer than N samples (N is the original number of samples in f, *i.e.*, the number of pixels in the input image). For the filter in Equation 4.25, one can use $min(N,\, 4\,N/\sigma_l)$ samples (see Section 4.5.1 for the derivation of this bound).

For blurring *over* each manifold η_k (Figure 4.3(b)), we use the recursive filter (RF) we introduced in Section 3.4.3, based on our Domain Transform geodesic-filtering approach. This filter uses the 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 when compared to the brute-force Gaussian bilateral filter for RGB images (Table 4.2). If a more Gaussian-like response is needed, one can use iterations of recursive or box filters (HECKBERT, 1986; GASTAL; 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 the domain-transform 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.

4.5.1 Sampling Rate for Recursive Filter

The recursive filter h described by Equation 4.25, when applied twice (once in each direction) has discrete impulse response (for an impulse at $n = 0$) given by

$$h[n] = -\frac{1}{\sqrt{2}\,\sigma_l}\exp\left(-\frac{\sqrt{2}\,|n|}{\sigma_l}\right), \tag{4.26}$$

where $|n|$ is the absolute value of n. The discrete-time Fourier transform of this filter is given by

$$H[\omega] = \frac{\sinh\left(\frac{\sqrt{2}}{\sigma_l}\right)}{\sqrt{2}\,\sigma_l\left(\cosh\left(\frac{\sqrt{2}}{\sigma_l}\right) - \cos\left(\omega\right)\right)}, \tag{4.27}$$

where $\omega \in [-\pi, \pi]$ is the frequency parameter. Note that H is periodic outside the interval $[-\pi, \pi]$, since the filter is discrete-time. The cut-off frequency ω_c of h (*i.e.*, the frequency above which all frequencies are attenuated by h below a small threshold τ), is obtained solving $H[\omega_c] = \tau$ for ω_c. This yields

$$\omega_c = \text{acos}\left(\cosh\left(\frac{\sqrt{2}}{\sigma_l}\right) - \frac{\sinh\left(\frac{\sqrt{2}}{\sigma_l}\right)}{\sqrt{2}\,\tau\,\sigma_l}\right). \tag{4.28}$$

When this equation is solvable for $\omega_c \in \mathbb{R}$, any signal f filtered with h can be considered bandlimited to the interval $[-\omega_c, \omega_c]$. Thus, in practice, if the original signal f is represented by N samples, the filtered version of f, obtained with the filter h, should be represented by at least $N\,\omega_c/\pi$ samples. In our implementation, we represent this filtered signal with $min(N,\ 4\,N/\sigma_l)$ samples, which is equivalent to a cut-off frequency ω_c obtained with threshold $\tau = 0.0125$.

4.5.2 Complexity Analysis

Since each pixel appears in a single cluster at each level of the binary tree in Figure 4.4, clustering takes $O(d\,N\,\log\,K)$ time, where $d = d_{\mathcal{S}} + d_{\mathcal{R}}$ is the total dimensionality of the space, N is the number of input pixels, and K is the number of adaptive manifolds used for filtering. Computing the manifolds using the filter from Equation 4.25 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 (Equation 4.1) is evaluated for all pixels, and the RF filter is linear-time. Finally, evaluating the summations in Equation 4.2 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 4.1.1) and clustering (Section 4.3). 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 (Equation 4.2); and (iv) N, to store the clusters belonging to the current manifold-tree branch.

4.6 Performance Evaluation

We have implemented three versions of the high-dimensional adaptive-manifold filter, 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 their respective 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 $[0, 1]^3$).

4.6.1 RGB color image filtering on the CPU

Computing the number of manifolds K using Equation 4.22, 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. (ADAMS; BAEK; DAVIS, 2010). Figure 4.8 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. (HE; SUN; TANG, 2010), while generating results indistinguishable from brute-force bilateral filtering (Figure 4.9). 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 4.9(d)). In fact, in some cases, it completely ignores color differences between pixels (Figure 4.10). A real-world application example where this problem becomes noticeable is shown in Figure 4.11, where a noisy path-traced global illumination

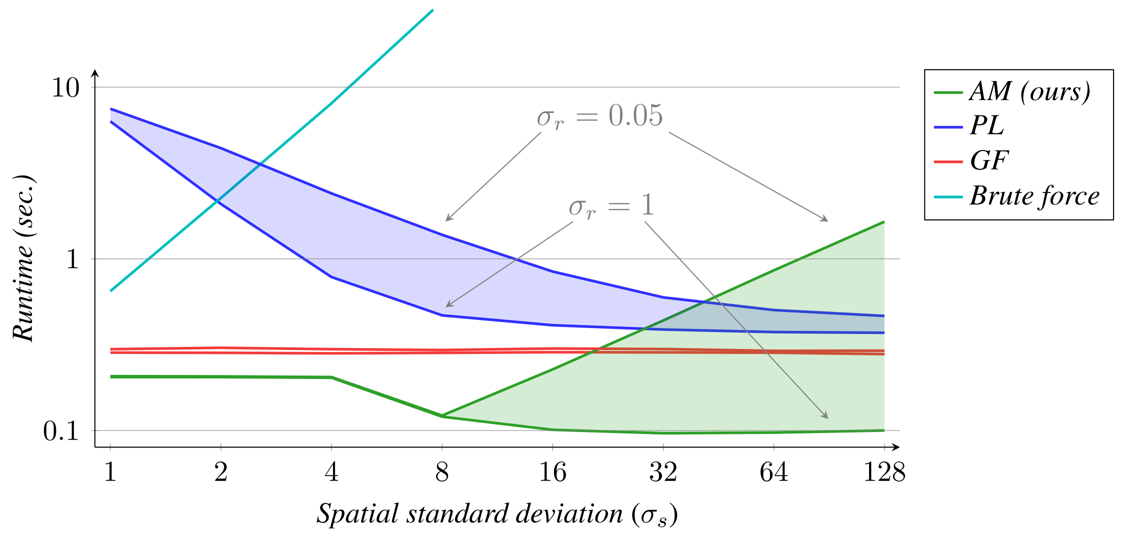


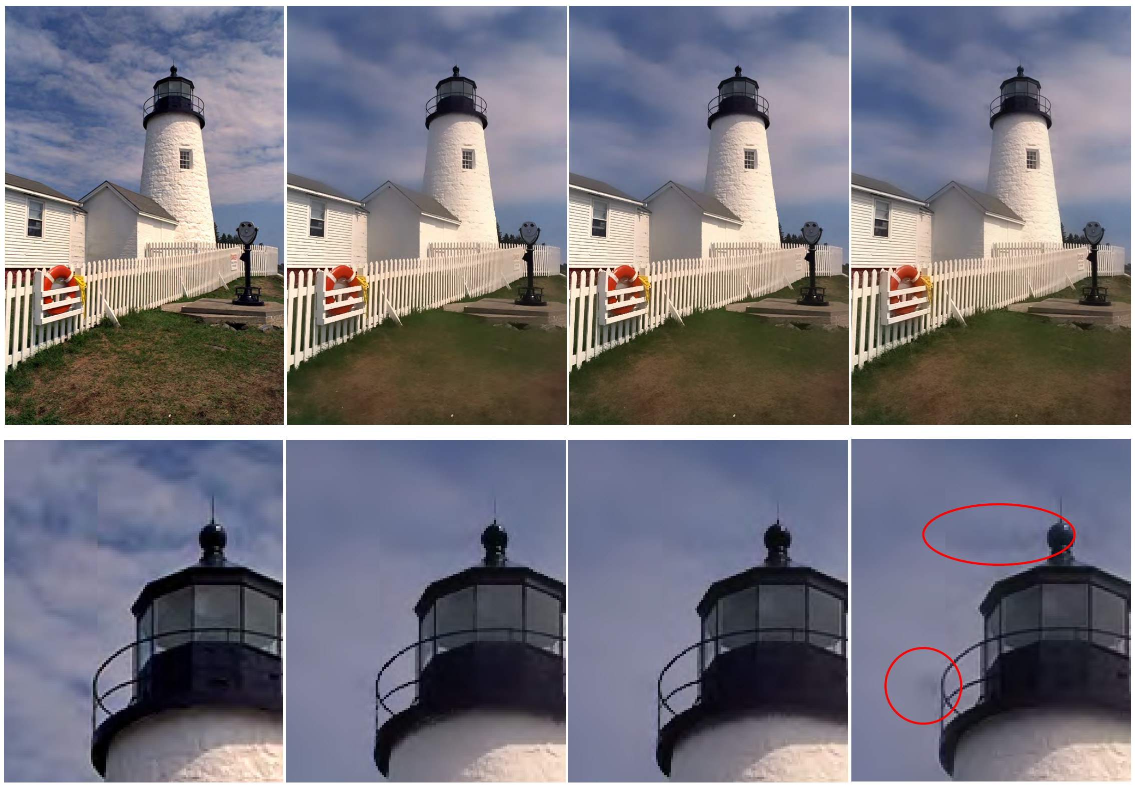
Figure 4.8: ***CPU performance*** of our adaptive-manifold (AM) filter versus the permutohedral lattice (PL) and the guided filter (GF). The vertical axis shows time, in seconds, to filter a 1-megapixel color image. The shaded areas represent performance changes when σ_r varies from 1 (bottom curve) to 0.05 (top curve).

image, shown in (a), was filtered using our adaptive manifolds filter, shown in (b), and using the guided filter, shown in (c). Note how the guided filter introduces haloing artifacts in certain regions of the image. These artifacts are not present in the result produced by our filter.

For a combination of large spatial standard deviations *and* small range standard deviations, our filter is slower than PL (Figure 4.8). 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 Section 4.2), 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. Thus, *our filter provides the fastest alternative for the most common situations*.

4.6.2 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$\times$ 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 small image size. Our approach can filter a 10-megapixel image under 0.04 seconds. Finally, the bottleneck of our GPU implementation are the



(a) *Photograph* (b) *Our AM* (c) *Bilateral (BF)* (d) *Guided (GF)*

Figure 4.9: 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).

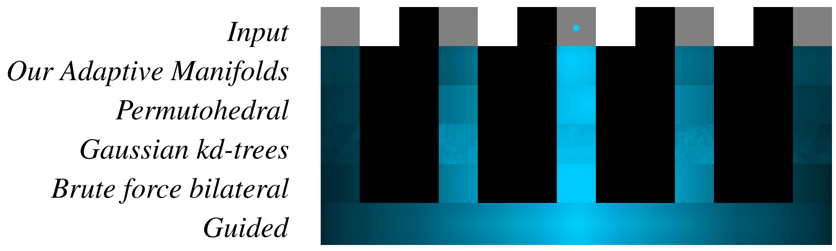
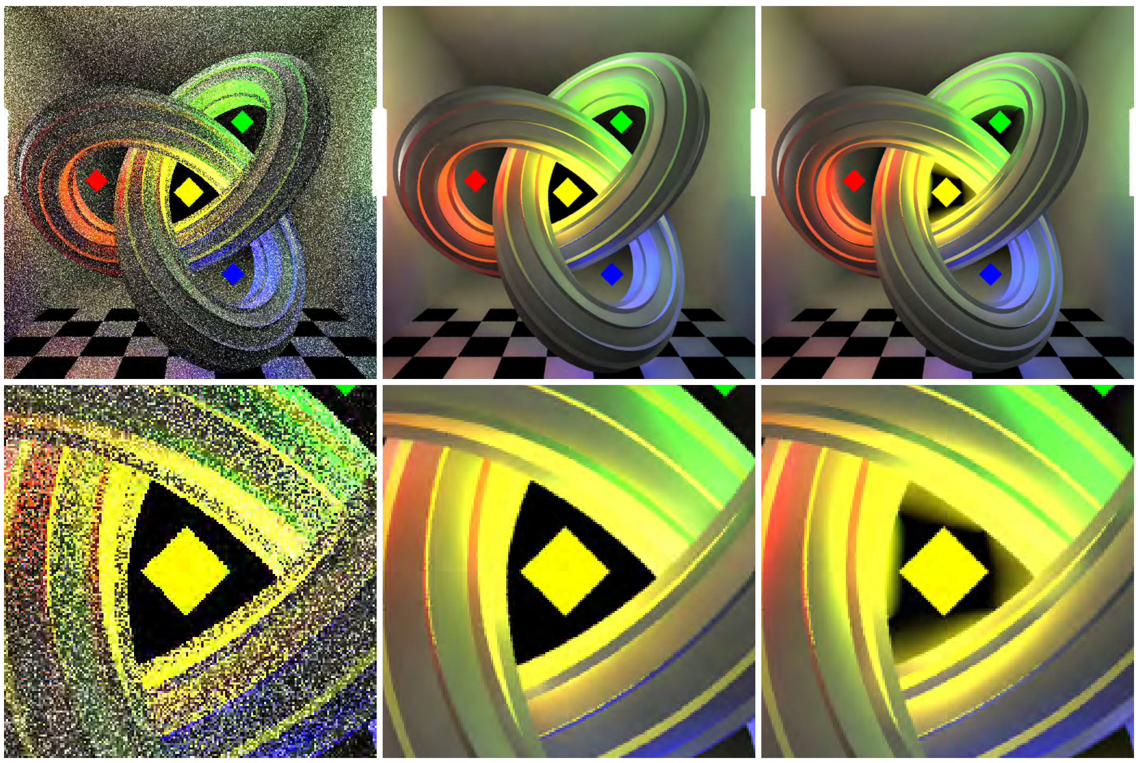


Figure 4.10: Response of filtering the blue impulse in the top image using small $\Sigma_{\mathcal{R}}$ values and the edges from the grayscale image. Our filter and bilateral filters preserve discontinuities from the input signal. The guided filter does not manage to do the same, because the gray shade is halfway between white and black. Black regions in the filtered signal indicate $g_i = 0$.



(d) *Path-traced image* (e) *Our AM filter* (f) *Guided filter*

Figure 4.11: Example of global illumination filtering. The path-traced image in (a) was rendered with 1 sample per pixel for direct and indirect illumination. Our filter in (b) generates a smooth shading from the noisy input by working in a 4-D space composed of 3-D geometric normal vectors and 1-D scene depth. The guided filter in (c) works on the same 4-D space, however it introduces haloing artifacts in certain image regions. This happens since the guided filter does not compute true Euclidean distance between pixels, as discussed in the text.

recursive filters from Section 5.4.3 (*i.e.*, Equation 4.25 and the RF filter). Its performance can be further improved using recent approaches for GPU recursive filtering (NEHAB et al., 2011).

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$\times$ faster for $\sigma_r < 0.2$, or up to 200$\times$ faster for σ_r up to 1. The GPU implementation of the guided filter by Bauszat et al. (BAUSZAT; EISEMANN; MAGNOR, 2011) filters a 0.75 megapixel color image in 0.07 sec — our AM filter is 2 to 10$\times$ faster for $\sigma_r < 0.2$, or up to 40$\times$ faster for σ_r up to 1. All cases consider $\sigma_s \in [1, 128]$. Figure 4.12 shows a performance graph.

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$\times$ 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

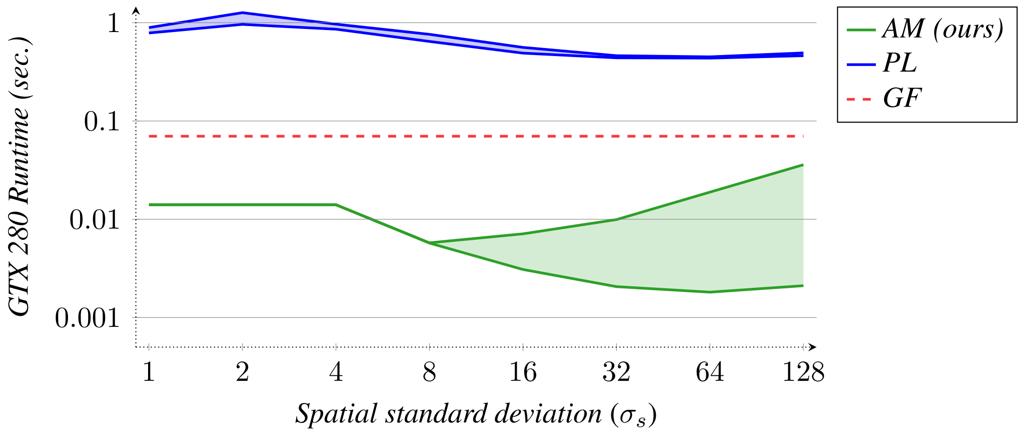


Figure 4.12: ***Performance on a GTX 280 GPU*** of our adaptive-manifold filter (AM) versus the permutohedral lattice (PL), and the guided filter (GF). The vertical axis shows time, in seconds, to filter a 1 Megapixel RGB color image. The shaded areas represent performance changes when σ_r varies from 1 (bottom curve) to 0.05 (top curve). The guided filter performance curve, in dashed red, is based on performance numbers reported by Bauszat et al. (BAUSZAT; EISEMANN; MAGNOR, 2011) on a GTX 285 GPU.

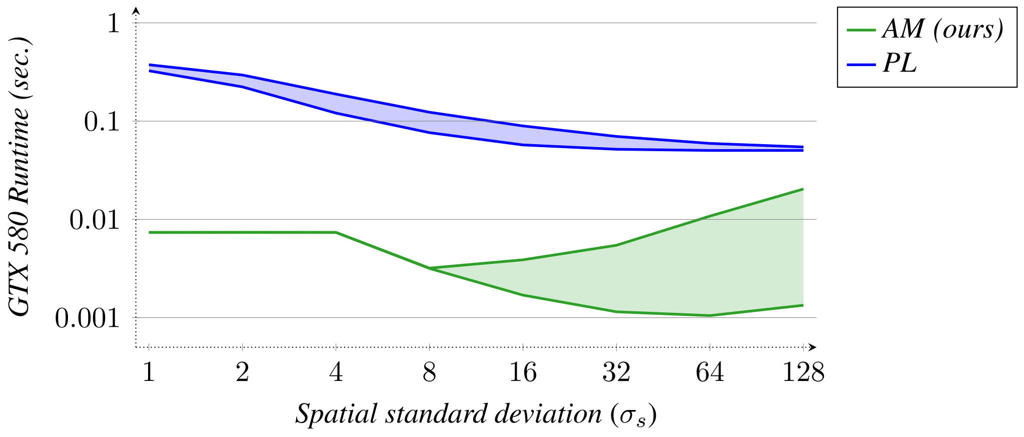


Figure 4.13: ***Performance on a GTX 580 GPU*** of our adaptive-manifold filter (AM) versus the permutohedral-lattice filter (PL). The guided filter (GF) has no performance data available for such a GPU. The vertical axis shows time, in seconds, to filter a 1 Megapixel RGB color image. The shaded areas represent performance changes when σ_r varies from 1 (bottom curve) to 0.05 (top curve).

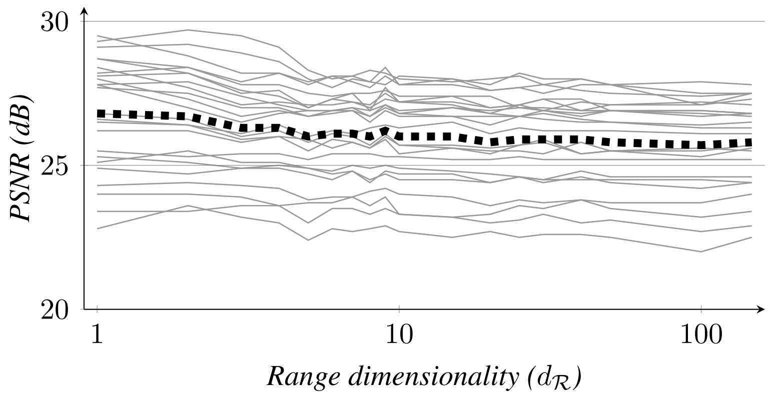


Figure 4.14: PSNR (with respect to ground-truth) for non-local means denoising using the adaptive-manifold filter with different range dimensions, using a fixed number of manifolds ($K = 15, \sigma_s = 8, \sigma_r = 0.35$). *PCA was not applied to the filtering space*. Each gray line represents one of 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 (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.

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. Figure 4.13 shows a performance graph.

4.6.3 Higher-Dimensional Filters

Since the signal $f : \mathbb{R}^{d_{\mathcal{S}}} \rightarrow \mathbb{R}^{d_{\mathcal{R}}}$ and the adaptive manifolds $\eta_k : \mathbb{R}^{d_{\mathcal{S}}} \rightarrow \mathbb{R}^{d_{\mathcal{R}}}$ 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 4.14, 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 4.14, $K = 15$.

The complexity of previous filters available in the literature are either quasilinear in N or quadratic in d: $O(dN \log N)$ for the Gaussian kd-trees (ADAMS et al., 2009); $O(d^2 N)$ for the permutohedral lattice (ADAMS; BAEK; DAVIS, 2010); and $O(d_{\mathcal{R}}^{2.807\cdots} N)$ for the guided filter (HE; SUN; TANG, 2010), since it requires the inversion of one $d_{\mathcal{R}} \times d_{\mathcal{R}}$ matrix per pixel. As 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; COLL; MOREL, 2005) denoising to a 1 megapixel image, using

25 dimensions, in 3.5 seconds ($K = 15$, $\sigma_s = 8, \sigma_r = 0.2$). 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 $d_{\mathcal{R}} > 3$. Our MATLAB implementation of the AM filter processes the same image in 23 seconds.

4.7 Applications

Our high-dimensional filter can provide real-time feedback for several applications. Next, we describe a few examples.

4.7.1 5-D color filters

5-D color filters can be used to create several effects for images and videos. Figure 4.9 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; SUN; TANG, 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 2.2 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.

4.7.2 Non-local-means filters

Non-local-means filters for denoising (BUADES; COLL; MOREL, 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 Equation 2.2 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 Equation 2.2 becomes intractable. For efficiency, one uses PCA to reduce the dimensionality of the resulting feature space ($\mathcal{R}$). Tasdizen (TASDIZEN, 2008) has shown that using only 6 main PCA-computed dimensions actually produces better denoising results than working with the full space. Figure 4.15 illustrates the use of our approach to implement *non-local-means* filtering.

4.7.3 Filtering with additional information

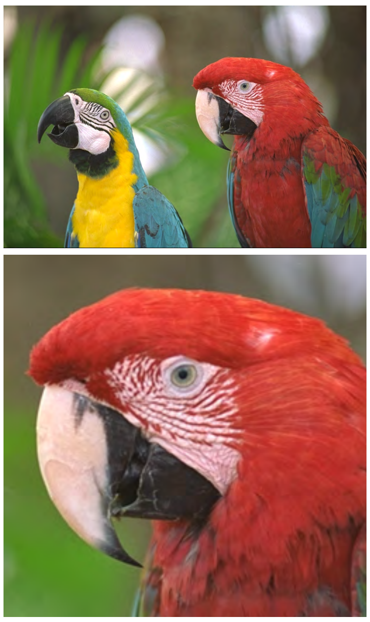
Filtering with additional information can help to increase the robustness of pixel-correlation estimation. For instance, the recent work of Zhuo et al. (ZHUO et al., 2010) uses an infrared flash to simultaneously capture a crisp infrared image, and a color image obtained under low-lighting conditions (and thus, noisy). Figure 4.16(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).

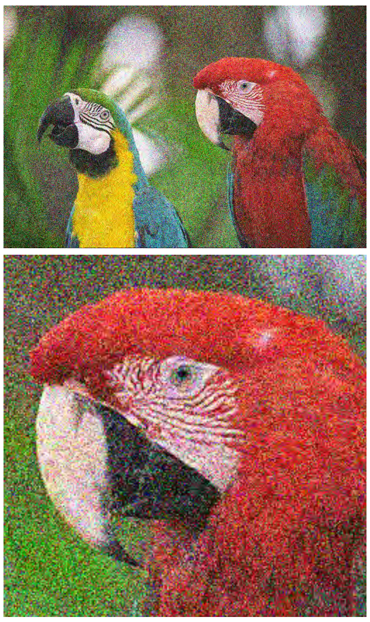
4.7.4 Ray-traced indirect illumination

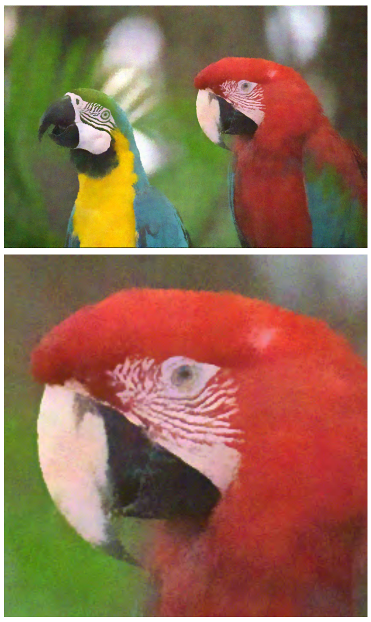
Ray-traced indirect illumination is increasingly making its way into interactive and real-time applications. Despite the advances in algorithms and hardware, sampling all light paths in a scene still requires considerable time. For this reason, modern techniques apply fast and approximate algorithms to denoise undersampled global illumination images (BAUSZAT; EISEMANN; MAGNOR, 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 4.17, the underlying geometric information of the scene is used to help filtering noisy indirect illumination, considerably improving the final rendering (Figure 4.1(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, $\hat{p}_i = (x_i, y_i, X_i, Y_i, Z_i, n_{ix}, n_{iy}, n_{iz})$.

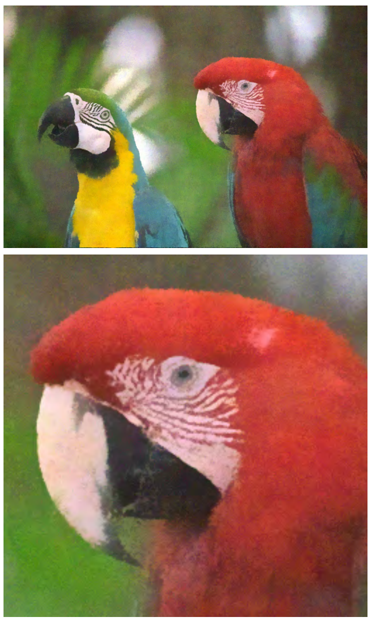
4.7.5 Hybrid Euclidean-geodesic filter

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 4.18 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 4.18(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 4.18(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.


(a) *Photograph*


(b) *Photograph with noise (std. dev. 0.2), PSNR 14.7 dB vs Photograph*


(c) *Our filter (147-D), PSNR 27.5 dB vs Photograph*


(d) *Our filter (6-D), PSNR 28.3 dB vs Photograph*

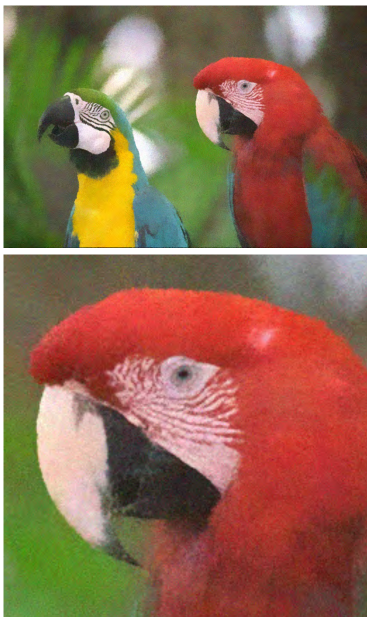
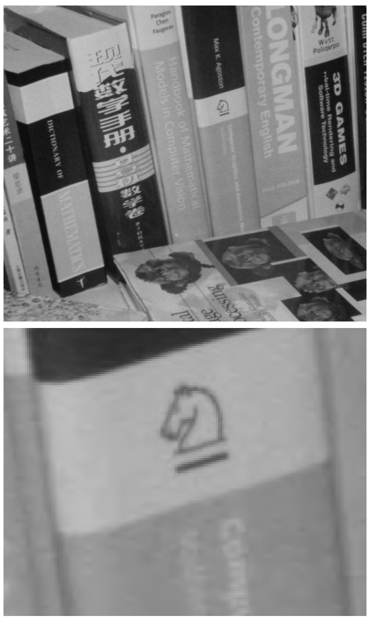
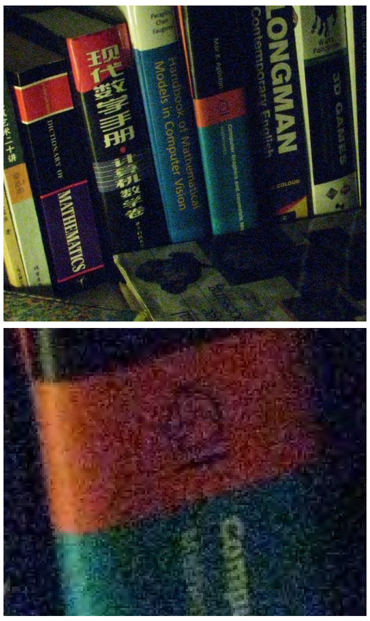

(e) *Brute force (6-D), PSNR 26.8 dB vs Photograph*

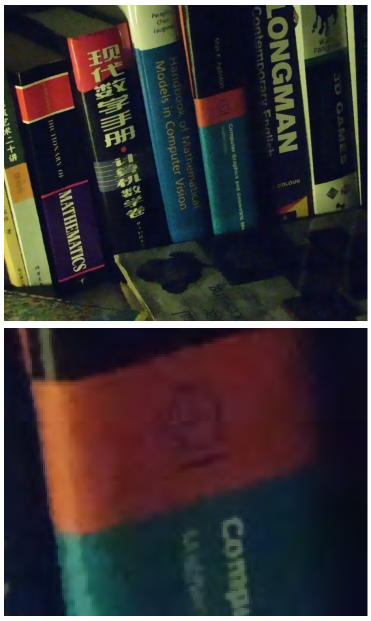
Figure 4.15: Example of non-local means denoising using our filter. For (c) the patch space of 7×7 RGB pixel colors was used in full, resulting in a 147-D space *(PCA was not applied)*. For (d) the space was reduced to 6-D using PCA, as suggested by Tasdizen (TASDIZEN, 2008). Our filter works on both 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 Equation 2.2 shown in (e), our filter is slightly more aggressive in reducing noise, due to its outlier-suppression property.



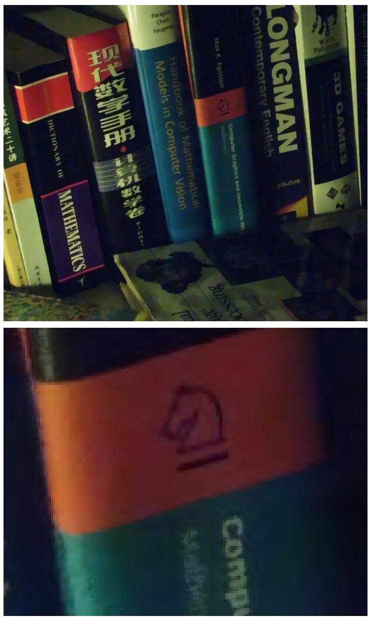
(a) *Infrared (IR)*



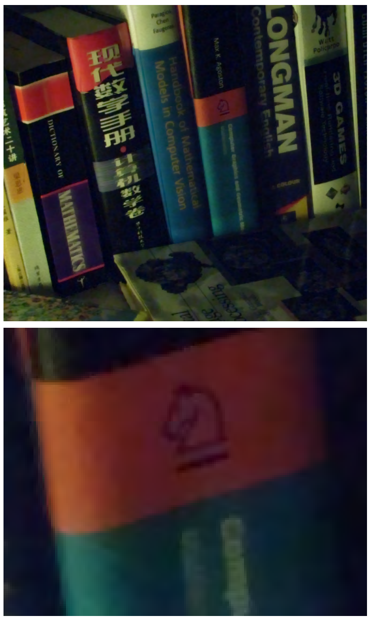
(b) *Photograph (color)*



(c) *Ours 6-D color*



(d) *Ours 6-D color + 1-D IR*



(e) *Zhuo et al. color + IR*

Figure 4.16: 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. (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).

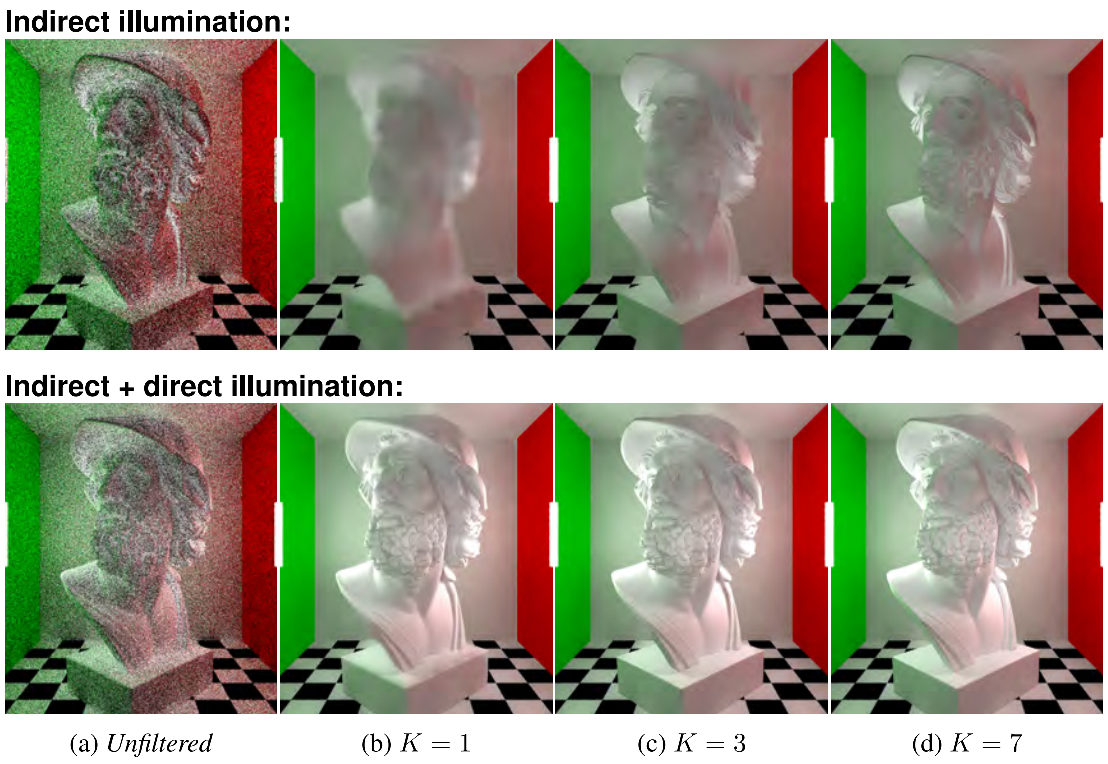


Figure 4.17: Path-traced scene was rendered with 1 sample per pixel for indirect illumination, plus 16 light samples for direct illumination. While the unfiltered indirect illumination result in (a) is unusable, applying our filter with as few as $K = 1$ manifold (b-d) can produce high-quality indirect-illumination denoising.

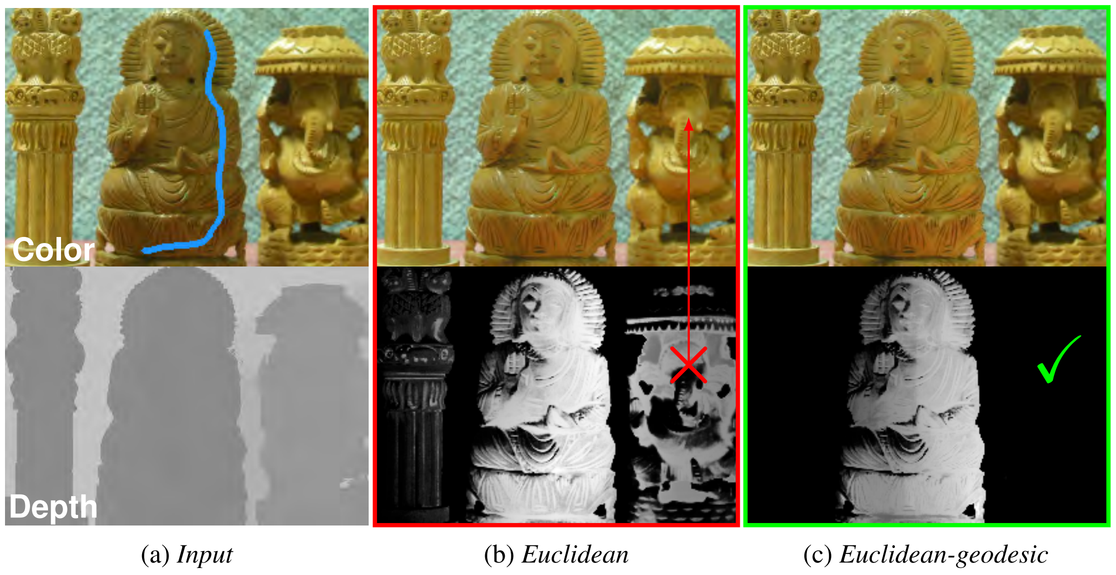


Figure 4.18: Hybrid Euclidean-geodesic filter. A user wants to lighten the middle statue, shown by the blue scribble. *(b)* Using a filter with Euclidean response in color and depth, the selection mask bleeds to other statues. *(c)* A filter with Euclidean response in color and geodesic response in depth produces the desired mask.

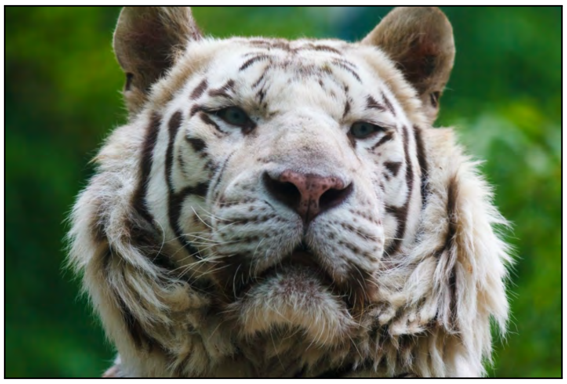
5 HIGH-ORDER RECURSIVE FILTERING OF NON-UNIFORMLY SAMPLED SIGNALS

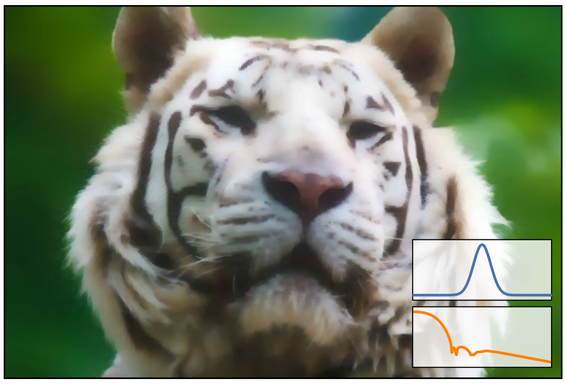
Digital filters are fundamental building blocks for many image and video processing applications. **Recursive digital filters** are particularly important in this context, as they have several desirable features: they can be evaluated in $O(N)$-time for N pixels, being ideal for real-time applications; they can represent infinite impulse response (*i.e.*, the value of a pixel may contribute to the values of the *entire* image or video frame, which is necessary for several applications, such as recoloring and colorization); and their implementation is relatively straightforward, since they are defined by a difference equation (Equation 5.1).

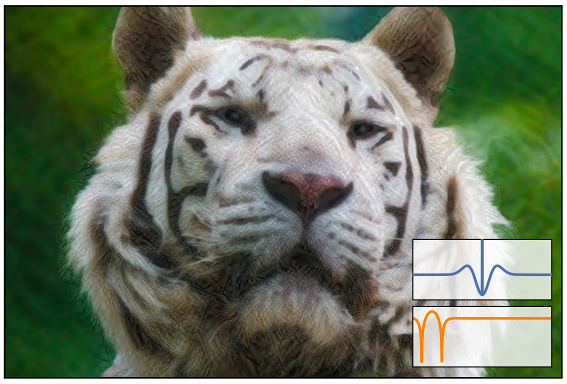
For digital manipulation and processing, continuous signals (often originating from real-world measurements) must be sampled. Traditionally, uniform sampling is preferred, and samples are arranged on a spacetime regular grid (for example, the rows, columns, and frames of a video sequence). However, several applications are better defined using **non-uniform sampling**, such as alias-free signal processing (SHAPIRO; SILVERMAN, 1960), global illumination (JENSEN, 1996), edge-aware image processing (GASTAL; OLIVEIRA, 2011), filtering in asynchronous systems (FESQUET; BIDéGARAY-FESQUET, 2010), particle counting in physics (POULTON; OKSMAN, 2000), among many others (ENG, 2007). The main difficulty is that standard operations for filtering—such as fast Fourier transforms (FFT), convolutions, and recursive filters—are commonly formulated with uniform sampling in mind.

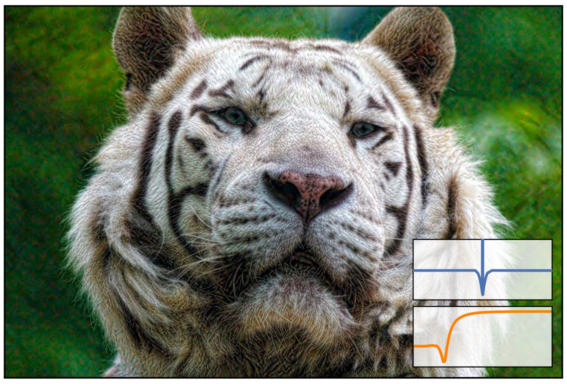
This chapter introduces *a mathematical formulation for applying recursive digital filters to non-uniformly sampled signals*. Our approach is based on simple constructs and provably preserves stability of any digital filter, be it low-pass, high-pass, or band-pass. We also explore relevant aspects and implications to image and video processing applications, such as filter behavior to image edges (Sections 5.2.5, 5.2.6 and 5.4), and common boundary condition specification (Section 5.2.7). The flexibility of our solution allows for the fine-tuning of the filter response for specific applications (Section 5.5).

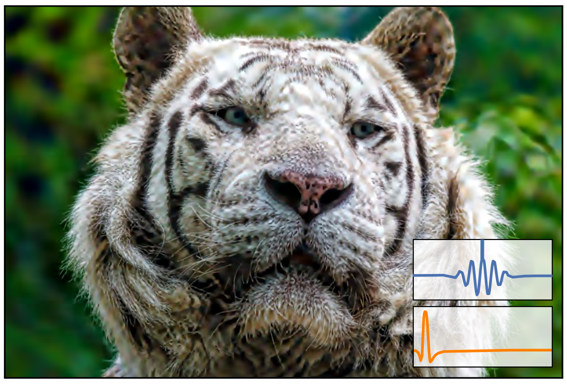
Our method is general and works with any non-uniformly sampled signal and any recursive digital filter defined by a difference equation. Since our formulation works directly with the filter coefficients, it works out-of-the-box with existing methodologies


(a) Photograph


(b) Low-pass Gaussian


(c) Modified Laplacian of Gaussian


(d) High-pass enhancer (Butterworth)


(e) Band-pass enhancer (Butterworth)

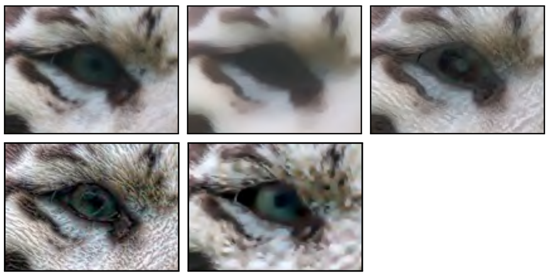

(f) Tiger's right eye details (a)–(e)

Figure 5.1: A variety of high-order recursive filters applied by our method to the photograph in (a). For these examples, non-uniform sampling positions are computed using an edge-aware transform. Thus, the resulting filters preserve the image structure and do not introduce visual artifacts such as halos around objects. The graphs in the insets show the filter's impulse response in blue, and its frequency response (Bode magnitude plot) in orange.

for digital filter design. In particular, we illustrate the usefulness of our formulation by using it to integrate higher-order recursive filtering with recent work on edge-aware transforms (Section 5.5.1). Such an integration allows us to demonstrate, *for the first time ever*, *linear-time edge-aware implementations of arbitrary recursive digital filters*. We show examples of a variety of such filters, including Gaussian, Laplacian of Gaussian, and low/high/band-pass Butterworth and Chebyshev filters (Figure 5.1).

Our solution produces high-quality results in real time, and works on color images at arbitrary filtering scales. The resulting filters have infinite impulse responses, and their computational costs are not affected by the shapes of the filtering kernels. We demonstrate the flexibility and effectiveness of our solution in various image and video applications, including edge-aware color filtering, noise reduction, stylization, and detail enhancement.

5.1 Background on Recursive Filtering

Recursive filters have been extensively studied in the past 50 years, and a variety of mathematical techniques are available for their design and analysis (PROAKIS; MANOLAKIS, 2007). In computer graphics, recursive filtering has been employed in a large number of applications, including interpolation (BLU; THÉVENAZ; UNSER, 1999), temporal coherence (FLEET; LANGLEY, 1995), edge-aware image processing (GASTAL; OLIVEIRA, 2011, 2012; YANG, 2012), and efficient GPU filtering (NEHAB et al., 2011). These filters have several advantages compared to other filtering methods based on brute-force convolution, summed-area-tables, and fast Fourier transforms. For instance, they have linear-time complexity in the number of input samples; infinite impulse responses; and a relatively straightforward implementation.

A causal infinite impulse response (IIR) linear filter is described by a *difference equation* in the spatial/time domain:

$$g[k] = \sum_{i=0}^{Q} n_i\, f[k-i] + \sum_{i=1}^{P} d_i\, g[k-i], \quad k = 0 \ldots N-1; \tag{5.1}$$

where $f[k]$ is the input sequence of length N, $g[k]$ is the output sequence, and $\{n_i, d_i\} \in \mathbb{R}$ are the filter coefficients. P is called the *feedback order* of the filter. A 0th-order filter is simply a finite impulse response (FIR) one. Since the input sequence is finite and only defined for $k = 0 \ldots N-1$, one needs to define the values of $f[-q]$ for $q = 1 \ldots Q$ and $g[-p]$ and for $p = 1 \ldots P$, called the *initial conditions* of the system. Equation 5.1 can be evaluated in $O(N)$ time, and it implements a *causal* filter since its output only depends on previous outputs and current/previous inputs.

The causal system from Equation 5.1 is equivalently described by its *transfer function* $H(z)$ in the z-domain (PROAKIS; MANOLAKIS, 2007):

$$H(z) = \frac{G(z)}{F(z)} = \frac{\sum_{i=0}^{Q} n_i\, z^{-i}}{1 - \sum_{i=1}^{P} d_i\, z^{-i}}, \tag{5.2}$$

where $F(z) = \mathcal{Z}\{f[k]\}$ and $G(z) = \mathcal{Z}\{g[k]\}$ are the z-transforms of the input and output sequences, respectively. The unilateral z-transform of a sequence $x[k]$ is given by

$$X(z) = \mathcal{Z}\left\{x[k]\right\} = \sum_{k=0}^{\infty} x[k]\, z^{-k}. \tag{5.3}$$

The P roots of the denominator in Equation 5.2 are the finite *poles* of the transfer function. The output sequence $g[k]$ can be obtained from $H(z)$ and $F(z)$ by computing the inverse z-transform (on both sides) of $G(z) = H(z)F(z)$. This yields $g[k] = (h*f)[k]$, where $*$ is discrete convolution. $h[k]$ is the *impulse response* of the filter described by Equation 5.1, given by the inverse z-transform of $H(z)$. Its discrete-time Fourier transform $\hat{h}(\omega)$ is obtained from its z-transform as $\hat{h}(\omega) = H(e^{j\,\omega})$, where $j = \sqrt{-1}$. Since the filter is causal, the impulse response is zero for negative indices: $h[k] = 0$ for $k < 0$.

5.1.1 Non-Uniform Sampling

Recursive filtering of non-uniformly sampled signals has been studied by Poulton and Oksman (POULTON; OKSMAN, 2000), and by Fesquet and Bidégaray (FESQUET; BIDéGARAY-FESQUET, 2010). They model the filtering process in the s-domain, related to the continuous spatial/time domain by the Laplace transform. The filter then becomes a continuous differential equation, which can be solved numerically using a variable time-step to represent the non-uniformly sampled output. Essentially, the scheme chosen for the numerical solution defines how one transforms the s-domain (where the filter is modeled in continuous-time) to the z-domain (where the filter is evaluated in discrete-time).

Poulton and Oksman (POULTON; OKSMAN, 2000) approach this problem using the bilinear transform to map between the domains (SMITH, 1997):

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}. \tag{5.4}$$

This is equivalent to solving the associated differential equation using the trapezoidal rule (PROAKIS; MANOLAKIS, 2007). This transform preserves stability and produces good results for several types of filters.

Fesquet and Bidégaray (FESQUET; BIDéGARAY-FESQUET, 2010) review several numerical integration approaches to solve the s-domain differential equation using variable time-steps. They point out that explicit methods such as the 4th-order Runge-Kutta scheme have time-efficient implementations but lack the stability of semi-implicit methods. In some cases, oversampling is needed to ensure stability, which is only possible if one has control over the sampler. They conclude that the semi-implicit bilinear method of (POULTON; OKSMAN, 2000) is possibly the best option in terms of complexity and stability. However, this transform is not defined for systems with poles at $z = -1$, and may be ill-conditioned for systems with poles very close to $z = -1$ (some high pass filters) (MATLAB, 2014a).

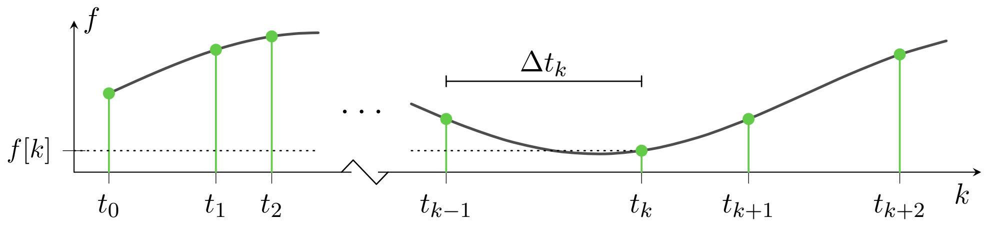


Figure 5.2: Example of a non-uniformly sampled signal.

5.2 Recursive Filtering of Non-Uniformly Sampled Signals

We introduce an alternative formulation for recursive filtering of non-uniformly sampled signals. For this, we extend Equation 5.1 to work directly in the non-uniform discrete domain. As a result, we can directly apply an arbitrary discrete-time filter $H(z)$ to any non-uniform signal. In the upcoming discussions, we focus on properties and applications relevant to image and video processing. For instance, we discuss different normalization schemes (Section 5.2.5 and Section 5.2.6) that may lend to better-suited filters for specific tasks (Section 5.4). We also describe the integration of high-order recursive digital filters with recent work on edge-aware transforms (Section 5.5.1), enabling, for the first time, edge-aware evaluation of a variety of filters, such as those illustrated in Figure 5.1.

Our approach takes as **input** a discrete-time filter defined by its transfer function $H(z)$, an input sequence $f[k]$ of length N, and a set of positive values $\{\Delta t_k\}$ which define the distance (or time delay) between subsequent samples (Figure 5.2). The distance values Δt_k are commonly obtained from real measurements at the time of sampling (FESQUET; BIDéGARAY-FESQUET, 2010), or computed in other ways (GASTAL; OLIVEIRA, 2011). From an initial position t_0, we compute the exact position t_k of the k-th input sample using the recurrence $t_k = t_{k-1} + \Delta t_k$. Our **output** is a sequence $g[k]$ of length N, containing the filtered input values.

5.2.1 The Naive Approach

One might be tempted to directly apply Equation 5.1 to the input sequence $f[k]$, while ignoring the distance values Δt_k. Certainly, this does not produce the desired output, since it treats the sequence $f[k]$ as if it were a uniformly-sampled sequence. The underlying problem lies in the filter $H(z)$ (Equation 5.2), which has a hidden dependency on a *constant* sampling interval T. One can intuitively see this dependency through its discrete-time Fourier transform, obtained from $H(z)$ by letting $z = e^{j\omega}$: note that the frequency parameter $\omega \in [-\pi, \pi]$ is *normalized* relative to the sampling interval T. That is, ω is measured in radians per *sample*. This means that the sequence being filtered should have been sampled using the same interval T, otherwise the response of $H(z)$ cannot be effectively characterized in the frequency domain.

One naive solution for filtering non-uniformly sampled sequences is to perform sample-rate conversion to bring the sampling rate to a constant value. However, this is impractical as it introduces severe overhead to the filtering process: depending on the values of the intervals Δt_k, sample-rate conversion to a constant interval T could require large amounts of memory and time. Another interesting solution is the non-uniform extension of the FFT (MARVASTI, 2001). However, its performance is still superlinear $O(N \log N)$ in the number of pixels, and its implementation somewhat complex.

Our approach, described in the following sections, addresses all of these limitations.

5.2.2 Our Approach

The following sections present the main contribution of our work: a mathematical formulation required to apply an arbitrary recursive filter to non-uniformly sampled signals. We solve this problem by first decomposing a P-th order filter into a set of 1st-order ones (Section 5.2.3), then deriving the equations for the individual 1st-order filters in a non-uniform domain (Section 5.2.4–Section 5.2.8), and finally applying them separately to the input data (Equation 5.7).

5.2.3 Decomposition into 1st-Order Filters

Let $H(z)$ be a P-th order filter whose P poles $b_1, b_2, \ldots, b_P$ are all distinct. Then, through partial-fraction expansion (PROAKIS; MANOLAKIS, 2007), $H(z)$ can be decomposed into a *sum* of P 1st-order filters and one 0th-order FIR filter:

$$H(z) = \sum_{i=1}^{P} \frac{a_i}{1 - b_i\, z^{-1}} + \sum_{i=0}^{Q-P} c_i\, z^{-i}, \quad \{a_i, b_i, c_i\} \in \mathbb{C}. \tag{5.5}$$

The i-th 1st-order filter $H_i(z) = \frac{a_i}{1-b_i\, z^{-1}}$ is described in the spatial domain by the difference equation

$$g_i[k] = a_i\, f[k] + b_i\, g_i[k-1], \tag{5.6}$$

or equivalently by the convolution of the input sequence $f[k]$ with its (causal) impulse response $h_i[k] = a\, b^k$:

$$g_i[k] = (h_i * f)[k].$$

Due to the linearity of Equation 5.3, the original filter $H(z)$ can then be computed in *parallel* in $O(N)$-time as the summed response of all g_i, plus a convolution with the FIR filter:

$$g[k] = \sum_{i=1}^{P} g_i[k] + \sum_{i=0}^{Q-P} c_i\, f[k-i]. \tag{5.7}$$

If $H(z)$ contains a multiple-order pole of order $m > 1$ (*i.e.*, $b_i = b_{i+1} = \ldots = b_{i+m-1}$), its partial-fraction expansion (Equation 5.5) will also contain terms of orders 1 through m:

$$\begin{aligned} &H_i(z) + H_{i+1}(z) + \cdots + H_{i+m-1}(z) \\ &\qquad = \frac{a_{i,1}}{1 - b_i\, z^{-1}} + \frac{a_{i,2}\, z^{-1}}{(1 - b_i\, z^{-1})^2} + \cdots + \frac{a_{i,m}\, z^{-(m-1)}}{(1 - b_i\, z^{-1})^m}. \end{aligned} \tag{5.8}$$

However, any term of order $l = 1 \ldots m$ can be decomposed into a *product* (in the z-domain) of 'l' 1st order terms:

$$\frac{a_{i,l}\ z^{-(l-1)}}{(1 - b_i\ z^{-1})^l} = \frac{a_{i,l}}{1 - b_i\ z^{-1}} \prod_{1}^{l-1} \frac{z^{-1}}{1 - b_i\ z^{-1}}.$$

In the spatial domain, this is the application in *sequence* of 'l' 1st order filters, which is also performed in $O(N)$-time.

5.2.4 1st-Order Filtering in a Non-Uniform Domain

Let $H(z) = a/(1 - b\ z^{-1})$ be a 1st-order filter with coefficients $\{a, b\} \in \mathbb{C}$, which is described in the spatial domain by Equation 5.6. Without loss of generality, from here on we will assume this filter has been designed for a constant and unitary sampling interval $T = 1$. Suppose the input sequence is zero ($f[k] = 0$) for all k between some positive integers k_0 and k_1, where $k_0 < k_1$. Then, if we unroll the recurrence relation in Equation 5.6, the response of the system for $g[k_1]$ can be written as

$$g[k_1] = a\, f[k_1] + b^{k_1 - k_0}\, g[k_0].$$

Note that $k_1 - k_0$ is the spatial distance (or elapsed time) between the k_0-th and k_1-th samples, due to the unitary sampling interval. However, if we take into account the non-uniform distribution of the sequence $f[k]$, the distance $k_1 - k_0$ is incorrect, and should be replaced by $t_{k_1} - t_{k_0}$:

$$g[k_1] = a\, f[k_1] + b^{t_{k_1} - t_{k_0}}\, g[k_0].$$

Thus, let $k_0 = k - 1$ and $k_1 = k$; and note that $\Delta t_k = t_k - t_{k-1}$ (see Figure 5.2). Equation 5.6 is written in a non-uniform domain as

$$g[k] = a\, f[k] + b^{\Delta t_k}\, g[k-1]. \tag{5.9}$$

This equation correctly propagates[1] the value of $g[k-1]$ to $g[k]$ according to the sampling distance Δt_k. However, it fails to preserve the normalization of the filter.

A normalized filter has unit gain at some specified frequency ω_*. For example, a low-pass filter commonly has unit gain at $\omega_* = 0$, which is equivalent to saying that its discrete impulse response should sum to one: $\sum h[k] = 1$. Normalizing a filter is done by scaling its impulse response (in practice, its numerator coefficients $\{n_i\}$ from Equation 5.2) by an appropriate factor γ. To find γ one uses the discrete-time Fourier transform[2], which *assumes a constant sampling interval.* Indeed, non-uniform sampling breaks this assumption, meaning that there does not exist a single scaling factor γ which makes the filter everywhere normalized.

[1]Refer to Section 5.4.3 for a discussion on negative real poles and complex-number outputs.

[2]Let $\omega_* \in [-\pi, \pi]$ be the normalized frequency parameter. The gain $|\gamma|$ at frequency ω_* of a filter $U(z)$ is $|\gamma| = |U(e^{j\,\omega_*})| = |\hat{u}(\omega_*)|$. Thus, $H(z) = U(z)/|\gamma|$ is a filter normalized to unit-gain at ω_*.

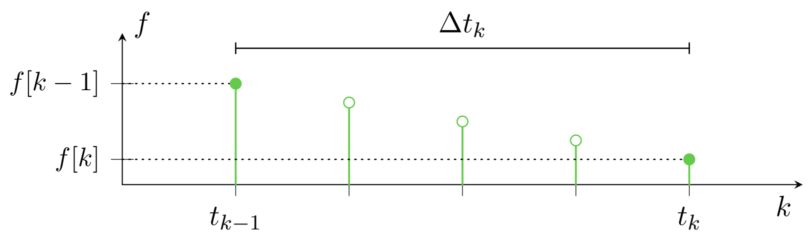


Figure 5.3: Piecewise linear unitary resampling between the $(k-1)$-th and k-th samples. In this example, $\Delta t_k = 4$.

Assuming the input filter $H(z)$ (originally designed for uniformly-sampled signals) is normalized, *we present two ways of correcting Equation 5.9 to preserve normalization when filtering non-uniformly sampled signals*. The first approach is *based on piecewise resampling* (Section 5.2.5), while the second is *based on spatially-variant scaling* (Section 5.2.6). Each approach produces a different response for the filter (see discussion in Section 5.4), while maintaining its defining characteristics. Section 5.2.9 proves the stability of our filtering equations.

5.2.5 Normalization-preserving piecewise resampling

Recall that, in the z-domain, a digital filter is designed and normalized assuming a *constant* sampling interval T (Section 5.2.1). To work with non-uniform sampling, it is impractical to perform sample-rate conversion on the full input sequence due to time and memory costs. In this section, we show how one can perform *piecewise* resampling in a very efficient way. In particular, we show that *it is possible to express this resampling process using a closed-form expression* (*i.e.*, one does not have to actually create and filter new samples, nor store them in memory). Thus, we are able to preserve normalization and maintain the $O(N)$-time performance of the filter, even when dealing with non-uniformly sampled signals.

Without loss of generality, assume $T = 1$. Also assume, for the time being, that the non-uniform distances between samples are positive integers (*i.e.*, $\Delta t_k \in \mathbb{N}$). This restriction will be removed later. To compute the output value $g[k]$, we will use the known previous output value $g[k-1]$ and create new samples between $f[k-1]$ and $f[k]$ to obtain uniform and unitary sampling. This process is illustrated in Figure 5.3, where new input samples (shown as green outlined circles) are linearly interpolated from the actual samples (green circles at times t_{k-1} and t_k). While a linear interpolator does not obtain ideal reconstruction of the underlying continuous signal (PROAKIS; MANOLAKIS, 2007), it is computationally efficient and produces good results for image and video processing (GASTAL; OLIVEIRA, 2011). Other polynomial interpolators such as Catmull-Rom (CATMULL; ROM, 1974) can be used at the expense of additional computation.

Since we are working with a causal filter, the newly interpolated samples will contribute to the value of $g[k]$, but not to $g[k-1]$. As expected, this contribution is simply the convolution of the interpolated samples with the filter's impulse response $h[k]$. Since the convolution result will be added to the value of $g[k]$ (the k-th sample), it is evaluated at position t_k of the domain. The end result is an additional summation term Φ in Equation 5.9:

$$g[k] = a\,f[k] + b^{\Delta t_k}\,g[k-1] + \underbrace{\sum_{i=1}^{\Delta t_k - 1} h[\Delta t_k - i]\,\tilde{f}_k[i]}_{\Phi}\,. \tag{5.10}$$

$\tilde{f}_k[i]$ is the i-th sample interpolated from $f[k-1]$ and $f[k]$:

$$\tilde{f}_k[i] = \frac{i}{\Delta t_k}\,(f[k] - f[k-1]) + f[k-1]. \tag{5.11}$$

Closed-form solution One can evaluate the summation Φ into a closed-form expression by substituting Equation 5.11 and the 1st-order impulse response $h[k] = a\,b^k$ into it:

$$\Phi = \left(\frac{b^{\Delta t_k} - 1}{r_0\,\Delta t_k} - r_1\,b\right) f[k] - \left(\frac{b^{\Delta t_k} - 1}{r_0\,\Delta t_k} - r_1\,b^{\Delta t_k}\right) f[k-1], \tag{5.12}$$

where $r_0 = (b-1)^2/(a\,b)$ and $r_1 = a/(b-1)$. This formula can be evaluated in *constant time* regardless of the number of new interpolated samples. Furthermore, despite our initial assumption, Equation 5.12 works correctly for non-integer values of the non-uniform distances between samples (*i.e.*, $\Delta t_k \in \mathbb{R}$).

5.2.6 Renormalization by spatially-variant scaling

We can avoid the need for reconstructing the underlying continuous signal through spatially-variant scaling. By unrolling the recurrence in Equation 5.9, one obtains a representation of the filtering process as a brute-force convolution:

$$g[k] = \sum_{n=-\infty}^{k} a\,b^{t_k - t_n} f[n]. \tag{5.13}$$

Equation 5.13 can be *interpreted* as a (causal) linear *spatially-variant* system acting on a uniform sequence $f[k]$ and producing another uniform sequence $g[k]$. The frequency characteristics of this system are spatially-variant since its impulse response is spatially variant. Therefore, such a system can only be normalized to unit gain using a spatially-variant scaling factor γ_k. In this way, the sequence resulting from Equation 5.9 is used to build a normalized output sequence $g'[k]$ as

$$g'[k] = g[k]/\,|\gamma_k|\,, \quad k = 0 \ldots N-1. \tag{5.14}$$

The computation of the values $\{\gamma_k\}$ depends on how one interprets the infinite sum in Equation 5.13, as it references input samples $f[k]$ for negative indices k. Two interpretations exist:

Interpretation #1: *Input samples $f[k]$ do not exist for $k < 0$ and, thus, it makes no sense to reference their values.* This is common when working with time-varying signals, such as videos, and filtering along time. Thus, to reference only valid data, the convolution in Equation 5.13 should start at zero:

$$g[k] = \sum_{n=0}^{k} a\, b^{t_k - t_n} f[n]. \tag{5.15}$$

The gain of this system for frequency ω_* is measured by its response to a complex sinusoid oscillating at ω_*. Thus, when processing the k-th sample, the gain $|\gamma_k|$ at frequency ω_* for the filter in Equation 5.15 is

$$|\gamma_k| = \left| \sum_{n=0}^{k} a\, b^{t_k - t_n}\, e^{-j\,\omega_*\, n} \right|. \tag{5.16}$$

Algorithm detail Computing γ_k for all k directly from the summation in Equation 5.16 results in quadratic $O(N^2)$-time complexity. Linear $O(N)$-time performance can be obtained by expressing the value of γ_k in terms of the previous value γ_{k-1}. This results in the recurrence relation

$$|\gamma_k| = \left| a\, e^{-j\,\omega_*\, k} + b^{\Delta t_k}\, \gamma_{k-1} \right|, \quad k = 1 \ldots N-1, \tag{5.17}$$

with initial value $\gamma_0 = a$.
Interpretation #2: *Input samples $f[k]$ exist for $k < 0$ and their values are defined by application-specific initial (or boundary) conditions*. This is common when working with signals defined in space, such as images and 3D volumes. Section 5.2.7 discusses some choices of boundary conditions. For this case, we have no option but to work with an infinite sum to compute the gain:

$$|\gamma_k| = \left| \sum_{n=-\infty}^{k} a\, b^{t_k - t_n}\, e^{-j\,\omega_*\, n} \right|. \tag{5.18}$$

Luckily, the recurrence relation in Equation 5.17 is still valid for this infinite sum, but with a different initial value γ_0. To compute this new γ_0, one can arbitrarily choose the sampling positions t_k for $k < 0$ (as part of the definition of the boundary conditions). The obvious choice is a unitary and uniform sampling, which implies that $t_k = t_0 + k$ for $k < 0$. Given this choice and Equation 5.18, one obtains

$$\gamma_0 = \sum_{n=-\infty}^{0} a\, b^{-n}\, e^{-j\,\omega_*\, n} = \frac{a}{1 - b\, e^{j\,\omega_*}}, \quad |b| < 1. \tag{5.19}$$

The convergence condition $|b| < 1$ is always true for any stable 1st-order filter since b is a pole of its transfer function (PROAKIS; MANOLAKIS, 2007).
Note on normalization: When implementing a P-th order filter, its 1st-order component filters (obtained through partial-fraction expansion in Section 5.2.3) should be normalized using the gain of the original P-th order filter (*i.e., they should **not** be normalized separately*). Thus, let $H(z)$ be a P-th order filter from Equation 5.5. Its gain (and

normalization factor) $|\gamma_k|$ at frequency ω_*, evaluated at k, is given by combining the gains $|\gamma_{i,k}|$ $(i = 1 \ldots P)$ of all its composing 1st-order filters:

$$|\gamma_k| = \left| \sum_{i=1}^{P} \gamma_{i,k} + \sum_{n=0}^{Q-P} c_n \, e^{-j\,\omega_*\,n} \right| .$$

The rightmost summation represents the gain contribution of the 0th-order term in Equation 5.5. Each $\gamma_{i,k}$ is computed replacing the i-th filter coefficients $\{a_i, b_i\}$ into Equation 5.17, and choosing an initial value $\gamma_{i,0}$ according to Interpretation #1 or #2.

If $H(z)$ contains a multiple-order pole b_i of order m, the gain contribution of the terms of orders 1 through m (Equation 5.8) is given by the sum

$$\left| \sum_{l=1}^{m} a_{i,l} \, e^{-j\,\omega_*\,(l-1)} \, \left(\gamma^*_{i,k}\right)^l \right| .$$

$|\gamma^*_{i,k}|$ is the gain for the filter $\frac{1}{1-b_i\,z^{-1}}$, computed by the substitutions $a \to 1$ and $b \to b_i$ into Equation 5.17.

5.2.7 Initial Conditions

To compute the value $g[0]$ of the first output sample for the filters in Equation 5.10 and Equation 5.14, one needs to define the values of $f[-1]$ and $g[-1]$: the initial (or *boundary*) conditions of the system. A *relaxed* initial condition is obtained by setting both values to zero: $f[-1] = g[-1] = 0$. However, when filtering images and videos, one frequently *replicates the initial input sample* (*i.e.*, $f[-1] = f[0]$). The corresponding initial value $g[-1]$ is found by assuming a constant output sequence for $k < 0$, where all its samples have a constant value β. This constant is found by solving the system's difference equation: defining a uniform and unitary sampling for $k < 0$, the 1st-order system's equation $g[-1] = a\,f[-1] + b\,g[-2]$ becomes $\beta = a\,f[0] + b\,\beta$. Solving for β gives

$$g[-1] = \beta = \frac{a}{1-b}\,f[0].$$

5.2.8 Non-Causal and Symmetric Filters

For image and video processing, one is usually interested in filters with *non-causal* response. That is, filters for which the output value of a pixel p depends on the values of *pixels to left and pixels to right* of p (see Section 5.5.1.2 on how we define 2D filters). A non-causal symmetric response is usually achieved by applying the filter in two passes: a causal (left-to-right) pass and an anti-causal (right-to-left) pass. This combination can be done either in cascade (VAN VLIET; YOUNG; VERBEEK, 1998; GASTAL; OLIVEIRA, 2011) or in parallel (DERICHE, 1993; HALE, 2006).

Applying causal and anti-causal filters in cascade has one disadvantage: when filtering a sequence of finite length, the output values of samples close to the boundary may be significantly incorrect (HALE, 2006). This problem can be mitigated by padding the

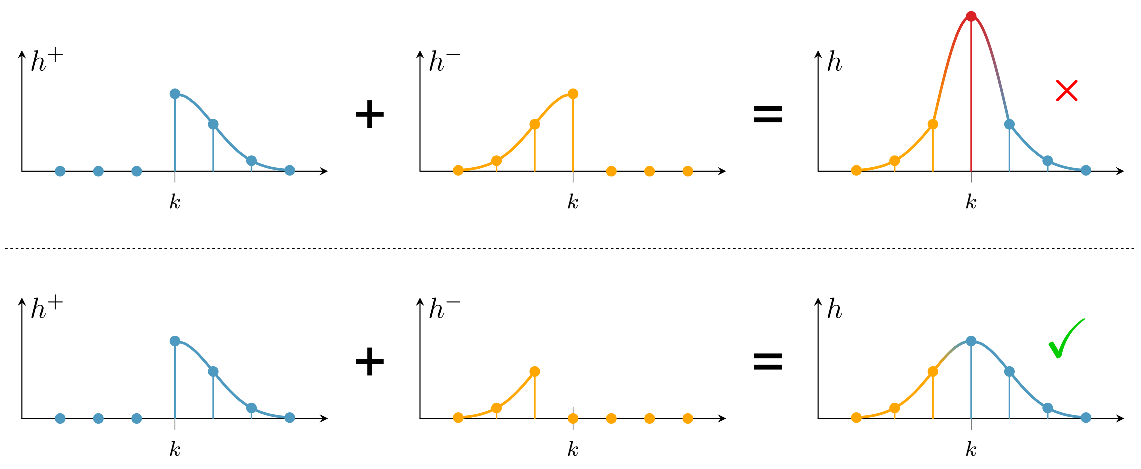


Figure 5.4: *(Top)* Central sample counted by both causal h^+ and “mirrored”’anti-causal h^- filters, resulting in an incorrect impulse response $h = h^+ + h^-$. *(Bottom)* Central sample counted only by the causal h^+ filter, resulting in a correct impulse response $h = h^+ + h^-$.

intermediate sequence with additional samples. This solution is very effective for filters whose kernels quickly approach zero, but otherwise may become a bottleneck. For example, a Gaussian kernel with very large standard deviation may require padding with an excessive number of additional samples (HALE, 2006). In these situations, one should prefer a parallel composition of causal and anti-causal filters.

In parallel, the outputs of the causal and anti-causal filters are simply summed to generate the final output. Since image processing filters are commonly symmetric, it is possible to design the anti-causal filter by “mirroring” the response of the causal one. However, when performing this procedure, one must be careful not to count the central sample twice (DERICHE, 1993); otherwise, the resulting impulse response may be incorrect (Figure 5.4, top). A simple way to avoid this problem is to include the central sample only on the causal pass (Figure 5.4, bottom). For a 1st order filter with causal transfer function given by $H^+(z) = \frac{a}{1-b\,z^{-1}}$, the respective “mirrored” anti-causal transfer function which does not count the central sample is $H^-(z) = \frac{a\,b\,z}{1-b\,z}$, and its corresponding difference equation is given by:

$$g^-[k] = a\;b\;f[k+1] + b\;g^-[k+1]. \tag{5.20}$$

For non-uniform domains, Equation 5.20 should be rewritten as

$$g^-[k] = a\;b^{\Delta t_k}\;f[k+1] + b^{\Delta t_k}\;g^-[k+1], \tag{5.21}$$

and it must be normalized as described in Section 5.2.5 and Section 5.2.6.

5.2.9 Proof of Stability

A necessary and sufficient condition for a digital filter to be stable is that all its poles lie inside the unit circle $|z| < 1$ in the z domain (PROAKIS; MANOLAKIS, 2007). The analytical stability of Eqs. 5.10 and 5.14 is proven below.

Proposition 5.1: For all real $\Delta t_k > 0$ and complex $a, b \neq 0$; if a filter $H(z) = \frac{a}{1 - b\,z^{-1}}$ is stable (*i.e.*, $|b| < 1$), the filter $H'(z)$ derived from $H(z)$ in the way defined by Equation 5.10 is also stable.

Proof. Replace the summation in Equation 5.10 by the closed-form formula from Equation 5.12. The z-domain transfer function of the resulting difference equation has the form

$$H'(z) = \frac{(a + R_0) - R_1\, z^{-1}}{1 - b^{\Delta t_k}\, z^{-1}}.$$

The single pole of this equation is $b^{\Delta t_k}$. For all $\Delta t_k > 0$, we have that $|b^{\Delta t_k}| < 1$ since $|b| < 1$. Thus, $b^{\Delta t_k}$ lies inside the unit circle, and the filter defined by Equation 5.10 is stable. □

In the same way one can easily show the analytical stability of Equation 5.9 and consequently Equation 5.14, since the value of $|\gamma_k|$ is non-zero[3] for all k. Numerically, singularities in the sampling rate ($\Delta t_k \to 0$) may lead to instabilities, since the pole $b^{\Delta t_k}$ gets too close to the unit circle. However, for the applications shown in the paper, we did not experience any numerical issues. Nonetheless, we recommend using 64-bit floating point precision for computations.

5.3 Designing Digital Filters

Our method can be used to filter any non-uniformly sampled signal using any recursive digital filter defined by a difference equation. Since our formulation uses the filter coefficients, it directly works with existing methodologies for IIR digital filter design. For example, both MATLAB (MATLAB, 2014b) and the open-source SciPy library (JONES et al., 2001–) provide routines that compute the coefficients of well-known filters such as Butterworth, Chebyshev, and Cauer. They also implement the partial-fraction expansion described in Section 5.2.3 through the routine `residuez()`.

It is also easy to create new filters by combining and modifying existing ones. For example, the high-pass enhancer filter from Figure 5.1(d) was created by combining a scaled high-pass Butterworth filter with an all-pass filter: $2\,H_{high}(z) + 1$; the filter from Figure 5.1(c) was similarly created from a band-pass Laplacian of Gaussian (LoG): $1 - 2.5\,H_{LoG}(z)$.

Recursive digital filters can also be designed to approximate *in linear-time* other IIR filters which are commonly superlinear in time. For example, Deriche (DERICHE, 1993) and van Vliet et al. (VAN VLIET; YOUNG; VERBEEK, 1998) show how to approximate a Gaussian filter and its derivatives, and Young et al. (YOUNG; VAN VLIET; GINKEL, 2002) implement recursive Gabor filtering.

[3] $|\gamma_k|$ is the gain of the filter for a chosen frequency ω_*, and it makes no sense to choose any ω_* where $|\gamma_k| = 0$; *i.e.*, it makes no sense to normalize a filter to unit gain at a frequency where the gain was originally zero.

To illustrate the use of our approach to obtain non-uniform filtering equations, the next Section shows the derivation of a recursive non-uniform $O(N)$-time Gaussian filter with normalization preserved by piecewise resampling.

5.3.1 Derivation of an $O(N)$-time, non-uniform Gaussian filter with normalization preserved by piecewise resampling.

5.3.1.1 Uniform Recursive Gaussian Filtering

Deriche (DERICHE, 1993) gives the following approximation to the positive region ($x \geq 0$) of a unit-height Gaussian of standard deviation σ:

$$u^+(x) = \text{Re}\left\{\alpha_0 \exp\left(-\frac{\lambda_0}{\sigma}x\right) + \alpha_1 \exp\left(-\frac{\lambda_1}{\sigma}x\right)\right\}, \tag{5.22}$$

where Re $\{\cdot\}$ denotes the real part of a complex number, and

$$\begin{aligned} \alpha_0 &= \quad 1.6800 + 3.7350j, & \lambda_0 &= 1.783 + 0.6318j, \\ \alpha_1 &= -0.6803 + 0.2598j, & \lambda_1 &= 1.723 + 1.9970j. \end{aligned}$$

The symmetric kernel is built by combining the positive half u^+ with the negative one $u^-(x) = u^+(-x)$, yielding an undistinguishable approximation to a Gaussian (mean squared error under 2.5×10^{-8}):

$$e^{\frac{-x^2}{2\sigma^2}} \approx u(x) = \begin{cases} u^+(x) & x \geq 0, \\ u^-(x) & x < 0. \end{cases}$$

In his work, Deriche implements a filter with kernel $u(x)$ by expanding the complex exponentials in Equation 5.22 into their composing sines and cosines, and extracting the real part. This yields a causal 4th-order recursive system for u^+ and an anti-causal one for u^-.

Note that this recursive Gaussian filter, as described by Deriche, only works for uniformly sampled signals. Next, we use our mathematical formulation to generalize the filter to work in non-uniform domains.

5.3.1.2 Non-Uniform Recursive Gaussian Filtering

Different from Deriche, we work directly with the complex exponentials in Equation 5.22, and extract the real part after filtering. The causal and anti-causal (complex) filter transfer functions are, respectively,

$$\begin{aligned} U^+(z) &= \frac{\alpha_0}{1 - e^{-\lambda_0/\sigma}\, z^{-1}} + \frac{\alpha_1}{1 - e^{-\lambda_1/\sigma}\, z^{-1}}, \quad \text{and} \\ U^-(z) &= \frac{\alpha_0\, e^{-\lambda_0/\sigma}\, z}{1 - e^{-\lambda_0/\sigma}\, z} + \frac{\alpha_1\, e^{-\lambda_1/\sigma}\, z}{1 - e^{-\lambda_1/\sigma}\, z}; \end{aligned}$$

which are already decomposed into 1st-order filters. Note that U^- is designed to ignore the central sample, as described in Section 5.2.8.

Since the Gaussian is a low-pass filter, it should be normalized to unit gain at zero-frequency ($\omega_* = 0$), which is equivalent to saying its kernel should have unit-area. However, the kernel defined by Equation 5.22, and implemented by $U(z) = U^+(z) + U^-(z)$, is not unit-area. A unit-area filter $H(z)$ is obtained as $H(z) = U(z)/|\gamma|$ where $|\gamma|$ is the gain of filter $U(z)$ at zero frequency, given by:

$$|\gamma| = |U(e^{j\,\omega_*})|\Big|_{\omega_*=0} = \alpha_0\,\frac{1+e^{-\lambda_0/\sigma}}{1-e^{-\lambda_0/\sigma}} + \alpha_1\,\frac{1+e^{-\lambda_1/\sigma}}{1-e^{-\lambda_1/\sigma}}.$$

Using our methodology described in Section 5.2.2, the difference equation which implements our recursive non-uniform $O(N)$-time Gaussian filter with normalization preserved by piecewise resampling is:

$$g[k] = \sum_{i=0}^{1} \text{Re}\left\{g_i^+[k] + g_i^-[k]\right\},$$

where

$$\begin{aligned}
a_i &= \alpha_i/|\gamma|,\\
b_i &= e^{-\lambda_i/\sigma},\\
g_i^+[k] &= a_i\,f[k] + b_i^{\Delta t_k}\,g_i^+[k-1] + \Phi_{k-1,k}\left(\Delta t_k\right),\\
g_i^-[k] &= a_i\,b_i^{\Delta t_{k+1}}\,f[k+1] + b_i^{\Delta t_{k+1}}\,g_i^-[k+1] + \Phi_{k+1,k}\left(\Delta t_{k+1}\right),\\
\Phi_{j,k}(\delta) &= \left(\frac{b_i^\delta - 1}{r_0\,\delta} - r_1\,b_i\right) f[k] - \left(\frac{b_i^\delta - 1}{r_0\,\delta} - r_1\,b_i^\delta\right) f[j].
\end{aligned}$$

Relaxed boundary condition is obtained by setting out-of-bound values to zero: $f[-1] = f[N] = g_i^+[-1] = g_i^-[N] = 0$. Alternatively, replicated-boundary condition is given by:

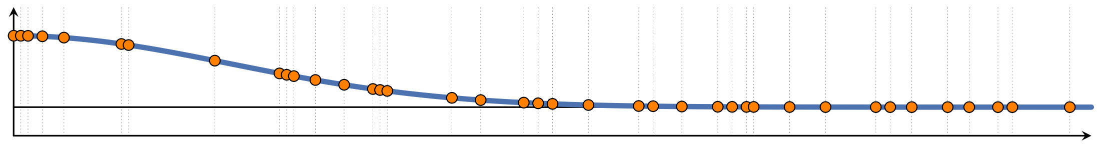
$$\begin{aligned}
f[-1] &= f[0], & g_i^+[-1] &= \frac{a_i}{1-b_i}\,f[0],\\
f[N] &= f[N-1], & g_i^-[N] &= \frac{a_i\,b_i}{1-b_i}\,f[N-1].
\end{aligned}$$

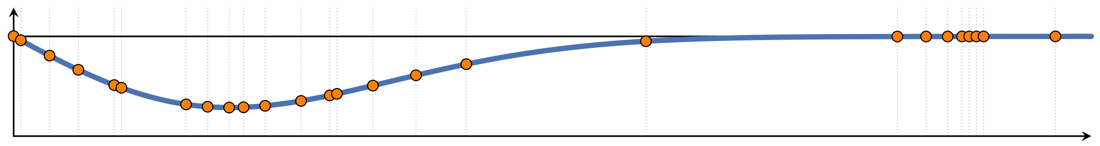
5.4 Evaluation and Discussion

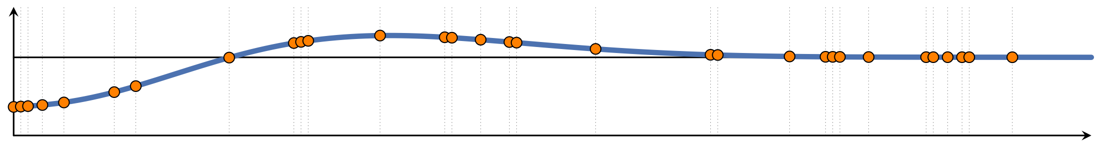
Appendix C includes an implementation of our method, and our webpage http://inf.ufrgs.br/~eslgastal/NonUniformFiltering has various examples with full source code of using it to process synthetic data, as well as several images and video.

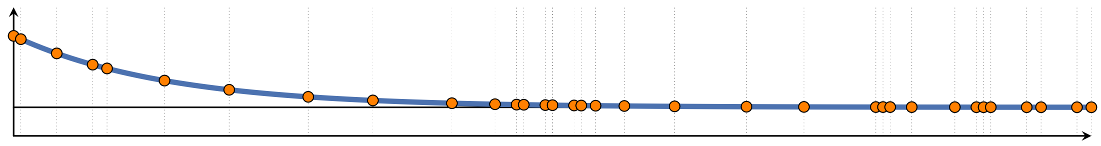
5.4.1 Accuracy

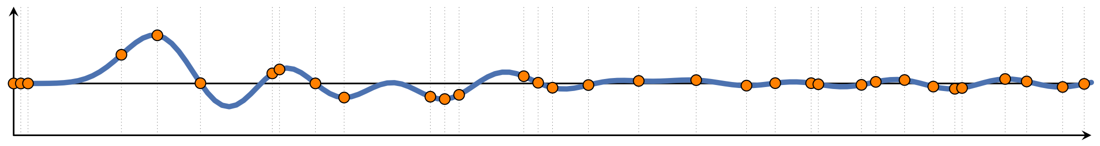
Figure 5.5 illustrates the accuracy of our approach when computing the impulse response of several IIR filters using non-uniform sampling. The filters are defined by their coefficients in the z-domain. Each plot shows the corresponding analytical ground-truth impulse response (solid blue line), together with the output samples (orange dots) obtained by filtering a non-uniformly sampled impulse. The sampling positions (indicated

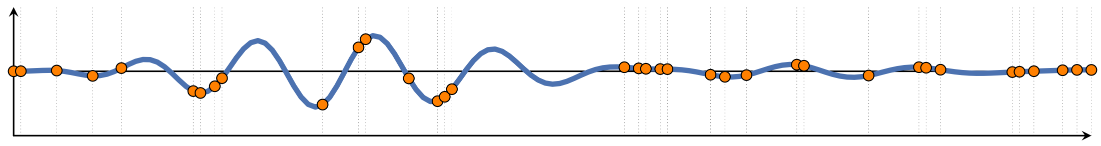

(a) Gaussian (4-th order), PSNR 316.0 dB


(b) Gaussian 1st derivative (4-th order), PSNR 250.9 dB


(c) Laplacian of Gaussian (4-th order), PSNR 288.4 dB


(d) Decaying exponential (1-st order), PSNR 302.9 dB


(e) Chebyshev Type I low-pass (8-th order), PSNR 308.9 dB


(f) Butterworth band-pass (8-th order), PSNR 304.4 dB

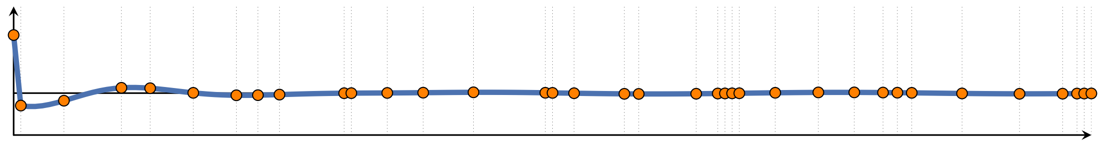
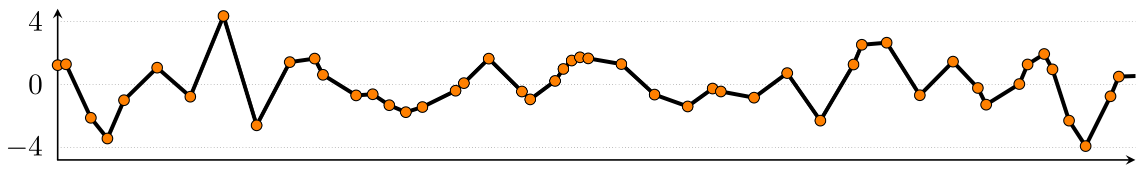
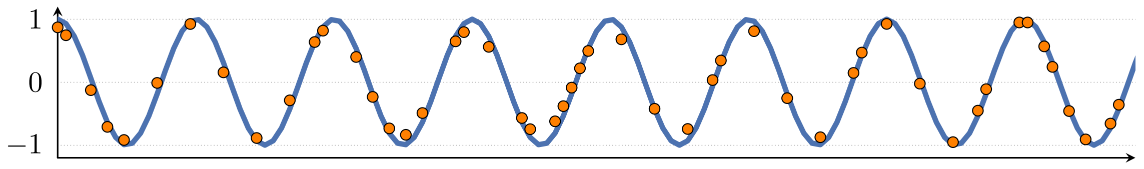

(g) Cauer high-pass (8-th order), PSNR 320.0 dB

Figure 5.5: Accuracy of our approach when filtering an impulse with several IIR filters using non-uniform sampling. The solid blue lines are the analytical ground-truth impulse responses. The small orange dots are the output samples.

by vertical dotted lines) were generated randomly. The impulse was represented by a 1 at the origin, followed by 0's at the sampling positions.



(a) Noisy non-uniformly sampled input signal



(b) De-noised non-uniform signal generated by our method

Figure 5.6: A noisy non-uniformly sampled sinusoid in (a) is filtered by the band-pass Butterworth filter from Figure 5.5(f) using our approach. The filtered samples are shown in (b), superimposed on the original noiseless signal (in blue).

Figure 5.5 shows that our results are numerically accurate and visually indistinguishable from ground-truth, with PSNR consistently above 250 dB (note that a PSNR above 40 dB is already considered indistinguishable visual difference in image processing applications). Furthermore, since the impulse response uniquely characterizes the filter, this experiment guarantees the accuracy of our approach in filtering general non-impulse signals. This conclusion is illustrated in Figure 5.6, where we use the band-pass filter from Figure 5.5(f) to denoise a non-uniformly sampled signal.

5.4.2 Performance

We implemented our approach in C++. Filtering one million samples using a 1st-order filter and 128-bit complex floating point precision takes 0.007 seconds on a single core of an i7 3.6 GHz CPU. This performance scales linearly with the number of samples. It also scales linearly with the order of the filter, which for common applications is rarely larger than 10. All the effects shown in Figure 5.1 were generated using 4th-order filters.

Our approach is highly parallelizable. A high-order filter is decomposed as a sum of *independent* 1st-order filters which can be computed in parallel (Section 5.2.3). Each 1st-order filter can also be parallelized internally using the approach described in (NEHAB et al., 2011).

5.4.3 Implementation Details

The latest CPUs have extremely fast instructions for evaluating the exponentiations in Eqs. 5.10 and 5.14. Our C++ code calls `std::pow(`b`,`Δt_k`)` directly. For older CPUs, one can use precomputed tables to further improve filtering times. Other constants dependent on the filter coefficients, such as r_0 and r_1, should be precomputed outside the main filtering loop.

Image and video processing applications use filters that take real inputs and produce real outputs (*i.e.*, $f, g \in \mathbb{R}$). One property of real filters is that any complex coefficient in its partial-fraction expansion must have a complex-conjugate pair (PROAKIS; MANOLAKIS, 2007). Thus, in practice, we only have to compute the filter response for one coefficient in each complex-conjugate pair, multiply the result by two, and drop the imaginary part.

One important observation is that Equation 5.9 may output a complex number even when working with only real inputs (more specifically, when working with a real input sequence and a real filter of odd-order). For example, imagine a simple 1st-order high-pass filter given by $H(z) = 0.1/(1 + 0.9\,z^{-1})$, evaluated in a non-uniformly sampled domain where $\Delta t_k = 0.5$ for all k. From Equation 5.9, the term $b^{\Delta t_k}$ results in a complex number since the pole $b = -0.9$ is negative and the sampling interval $\Delta t_k = 0.5$ is not an integer: $b^{\Delta t_k} = (-0.9)^{0.5} = 0.9^{0.5} j$. The correct thing to do in such situations is to drop the imaginary part of the output filtered sequence after the sequence has been filtered. This is equivalent to adding a complex-conjugate pole to the filter such that its Fourier transform returns to a conjugate-symmetric form (and thus its impulse response will be real-valued, as desired):

$$\text{Re}\,\{h[k]\} = \frac{1}{2}\Big(H(z) + H^*(z^*)\Big). \tag{5.23}$$

This procedure is required because expressions of the form $H(z) = 1/(1 - b\,z^{-1})$, for $-1 \leq b < 0$, $b \in \mathbb{R}$, are actually a simplified form of the filter's z-transform. Observe that the corresponding discrete impulse response is the sampling of a cosine oscillating at the highest possible representable frequency (*i.e.*, π):

$$\begin{aligned} h[k] &= b^k && (5.24)\\ &= \frac{1}{2}\Big(b^k + b^k\Big) && (5.25)\\ &= \frac{1}{2}\Big((|b|\,e^{-j\pi})^k + (|b|\,e^{j\pi})^k\Big) && (5.26)\\ &= |b|^k \cos(\pi\,k). && (5.27) \end{aligned}$$

An important property used in this derivation is the fact that $e^{j\pi} = e^{-j\pi} = -1$. That is, $e^{j\pi}$ and $e^{-j\pi}$ are complex-conjugate numbers which coincide on the complex plane. Since the continuous form of the filter in Equation 5.27 is real-valued for all k, so should be its discrete form, *even in a non-uniformly sampled domain*. From Equation 5.26 one can easily derive the filter's non-simplified z-transform, which contains a *pair of complex-conjugate poles*, and thus corresponds to a real-valued impulse response:

$$\frac{1}{2}\left(\frac{1}{1 - |b|\,e^{-j\pi}\,z^{-1}} + \frac{1}{1 - |b|\,e^{j\pi}\,z^{-1}}\right).$$

One can easily see that this same result can be obtained from Equation 5.23, which explains the intuition behind it.

5.4.4 Other Approaches

The continuous-space method described by Poulton and Oksman (POULTON; OKSMAN, 2000) may be used for filtering non-uniformly sampled signals. However, their method requires mapping between discrete and continuous space using the bilinear transform, which can become ill-conditioned for very large or small sampling intervals (BRUSCHETTA, 2011), especially in high-pass filters (BRUSCHETTA; PICCI; SACCON, 2014). This has a direct impact on the accuracy and quality of the filter. Furthermore, their approach does not allow control over normalization schemes (Section 5.2.5 and Section 5.2.6), and their work does not explore details which become important when filtering images and videos, such as boundary conditions (Section 5.2.7) and the construction of symmetric filters (Section 5.2.8).

In Chapter 3 we describe a simple 1st-order decaying-exponential low-pass filter which works in a non-uniform domain. It can be shown that such a filter is the simplest special case of our more general method described in the current chapter: the filter from Equation 3.26 can be obtained from our general equations by (i) using a zero-order-hold $(f_k^{zoh}[i] = f[k])$ instead of the linear interpolator in Equation 5.10; and by (ii) noticing that the coefficients of a normalized 1st-order low-pass real filter *must* satisfy $a = 1 - b$. This results in the filtering equation $g[k] = (1 - b^{\Delta t_k})\, f[k] + b^{\Delta t_k}\, g[k-1]$. Additionally, the formulation described in Section 3.4.3 applies causal/anticausal filters in cascade, and does not lend to a truly symmetric filter in non-uniform domains. Our formulation using filters in parallel, described in Section 5.2.8, addresses this limitation.

Finally, our IIR normalization schemes are related to the FIR convolution operators defined in Chapter 3: our spatially-variant scaling provides the same response as normalized convolution (KNUTSSON; WESTIN, 1993), and our piecewise resampling generates the same response as interpolated convolution. Thus, in edge-aware applications, our IIR normalization schemes provide control over the filter's response to the edges (see the plots in Figure 5.7).

5.5 Applications

This Section demonstrates the usefulness of our formulation to various tasks in image and video processing. In particular, we show how to integrate high-order recursive filtering with recent work on edge-aware transforms. Thus, we demonstrate *the first linear-time edge-aware implementations of several recursive digital filters, including Gaussian, Butterworth, and other general low/high/band-pass filters*.

5.5.1 General Edge-Aware Filtering

An *edge-aware* filter transforms the content of an image while taking into account its structure. For example, an edge-aware smoothing filter can remove low-contrast variations in the image while preserving the high-contrast edges; and an edge-aware enhance-

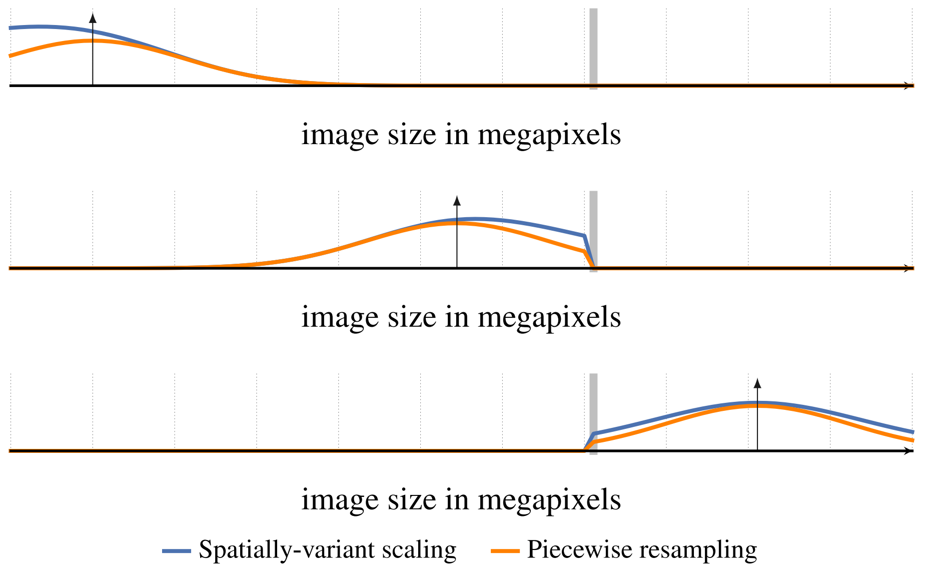


Figure 5.7: An impulse (upward-pointing arrow) travels from the left to the right of the domain. This domain contains a simulated discontinuity close to its center, shown as a vertical gray line (in a real signal, like an image, this could be an edge from an object—see Section 5.5.1). The impulse is filtered using our approach to deal with the discontinuity, and a Gaussian kernel. We normalize the filter either by spatially-variant scaling (impulse response shown in blue), or by piecewise resampling (impulse response shown in orange). Note how each normalization scheme results in a different response to the discontinuity and boundary in the domain. Thus, in edge-aware applications we are able to control the filter's response to the edges in the signal (GASTAL; OLIVEIRA, 2011).

ment filter can increase local contrast without introducing visual artifacts such as halos around objects. Due to these properties, edge-aware filters are important components of several image and video processing applications (DURAND; DORSEY, 2002; LEVIN; LISCHINSKI; WEISS, 2004; LISCHINSKI et al., 2006; FATTAL, 2009; FARBMAN; FATTAL; LISCHINSKI, 2010).

In Chapter 3 we showed how any filtering kernel can be made edge-aware by adaptively warping the input sequence using a *domain transform*. Conceptually, we warp the input image (signal) along orthogonal 1-D manifolds while preserving the distances among neighboring pixels, as measured in higher-dimensional spaces. In such a warped domain, pixels (samples) are *non-uniformly spaced*. Applying a linear filter in this warped domain and then reversing the warp results in an edge-aware filter of the original samples. This process is illustrated in Figure 3.4 for a low-pass filter applied to a 1-D signal. In practice, there is no need to explicitly warp and un-warp the signal, and the entire operation is performed on-the-fly in a single step.

The domain transform is fast and lends to good results. However, in Chapter 3 we only demonstrated its solution (in linear time) on two simple filters: an iterated box filter and a recursive 1st-order decaying-exponential filter (both low-pass filters). *Using our* *formulation of non-uniform filtering, we are able to generalize the domain transform to work on recursive filters of any order, and in linear time*, which allows practically unlimited control over the shape of the filtering kernel. In other words, with our generalization we can *transform any recursive linear filter h* (described by Equation 5.1) *into a recursive edge-aware filter h_{EA}* (described by either Eq. 5.10 or Equation 5.14). The resulting filter is non-linear, and maintains the characteristics of the original filter. Thus, for instance, if h is a low-pass filter, h_{EA} will be a low-pass edge-aware filter. Furthermore, since h_{EA} is also described by a difference equation, it will filter an input sequence of length N in $O(N)$ time.

Next we review the domain transform and show how it integrates with our method. Section 5.5 shows various examples of applications that use this integration.

5.5.1.1 Review of the Domain Transform

Assuming a unitary sampling interval along the rows (or columns) of an image, the imagespace distance between two samples $f[k]$ and $f[k+\delta]$ is δ, for $\delta \in \mathbb{N}$. Using a domain transform $\mathcal{D}t(k)$, the *warped-space* distance between the same samples is $\mathcal{D}t(k+\delta) - \mathcal{D}t(k)$. By definition (see Equation 5.28 below), the domain transform is a monotonically increasing function (*i.e.*, $\mathcal{D}t(k+\delta) - \mathcal{D}t(k) \geq \delta$).

For filtering we chose to use the ℓ_2 norm. Approximating the derivatives in the ℓ_2 domain transform (Equation 3.15) using backward differences, we obtain a (discrete) domain transform given by

$$\mathcal{D}t(k) = \sum_{i=1}^{k} \sqrt{1 + \left(\frac{\sigma_s}{\sigma_r}\right)^2 \sum_{c=1}^{d_{\mathcal{R}}} \left(f_{[c]}[i] - f_{[c]}[i-1]\right)^2}. \tag{5.28}$$

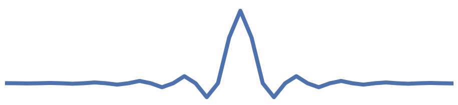
Here, $f_{[c]}[i]$ is the i-th element in the sequence of N samples obtained from the c-th *channel* of the signal, from a total of '$d_{\mathcal{R}}$' channels (for example, an RGB image has $d_{\mathcal{R}} = 3$ channels: red, green, and blue). σ_s and σ_r are parameters of the edge-aware filter, as described in Chapter 3. σ_s controls the imagespace size of the filter kernel, and σ_r controls its range size (*i.e.*, how strongly edges affect the resulting filter).

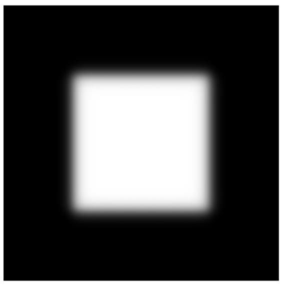
5.5.1.2 Using the Domain Transform with Our Method

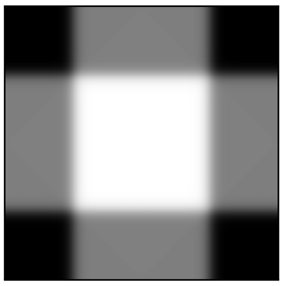
Equation 5.28 defines new non-uniform positions for each sample in the spatial domain. Consequently, the warped-space distance between adjacent samples $f[k-1]$ and $f[k]$ is given by

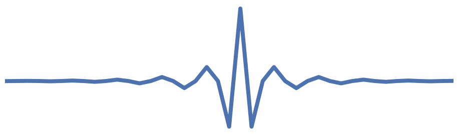
$$\Delta t_k = \sqrt{1 + \left(\frac{\sigma_s}{\sigma_r}\right)^2 \sum_{c=1}^{d} \left(f_{[c]}[k] - f_{[c]}[k-1]\right)^2} \tag{5.29}$$

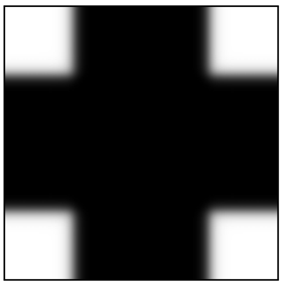
The values $\{\Delta t_k\}$ can be precomputed for all $k = 0 \ldots N-1$, and then substituted into the filtering equations (Eqs. 5.10 and/or 5.14) for evaluating the filter. In this way, we


(a) Low-pass filter


(b) 2D DFT, sequence


(c) 2D DFT, parallel


(d) High-pass filter


(e) 2D DFT, sequence

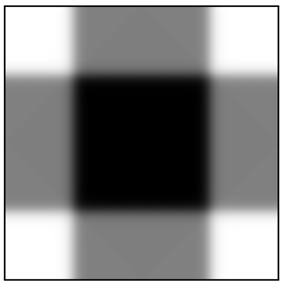
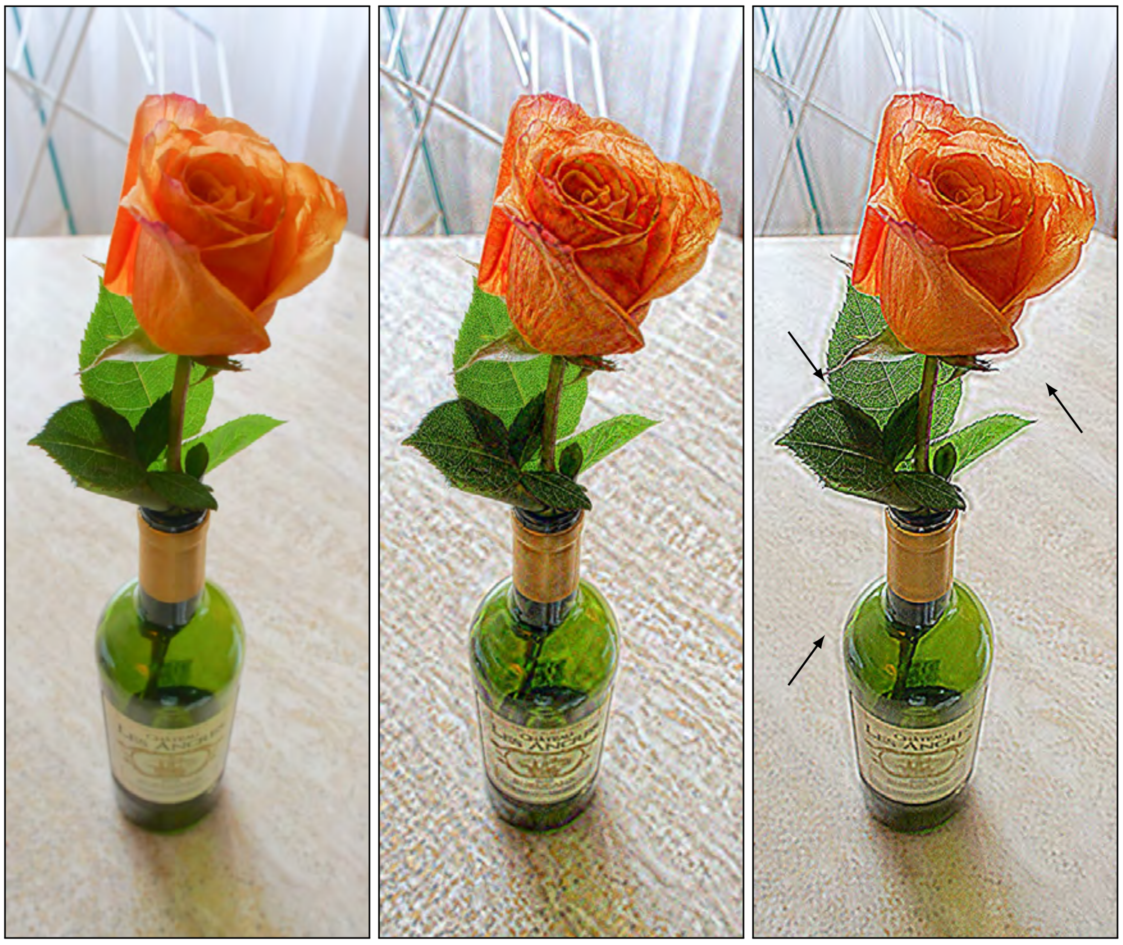

(f) 2D DFT, parallel

Figure 5.8: The 1D low-pass filter with impulse response shown in (a) may be applied to the rows and columns of an image to obtain a 2D filter. The corresponding horizontal and vertical passes may be combined either in sequence or in parallel, each option resulting in a filter with different 2D frequency response. This is illustrated by the 2D discrete Fourier transforms (DFT) shown in (b) and (c), where white represents a gain of one and black a gain of zero, and the zero-frequency has been shifted to the center of the images. It is clear that applying the filter from (a) in sequence removes a greater amount of high-frequencies from the signal. Thus, this low-pass filter is better applied in sequence, as in (b). The opposite is true for high-pass filters. The high-pass filter from (d) is better applied in parallel, as in (f), since a greater amount of high-frequencies are preserved.

obtain an *edge-aware* implementation of arbitrary recursive filters. For example, using the non-uniform Gaussian filter derived in Section 5.3.1, we obtain a $O(N)$-time edge-aware Gaussian.

Filtering 2D Signals As described in Section 3.3, we filter 2D images by performing a horizontal pass along each image row, and a vertical pass along each image column. How the horizontal and vertical passes are combined depend on the desired frequency response of the filter. Low-pass filters are better applied in *sequence*: assuming the horizontal pass is performed first, the vertical pass is applied to the result produced by the horizontal one. High and band-pass filters are usually better applied in *parallel*: the horizontal and vertical passes are performed independently, and their result added at the end. This suggestion is simply a design choice: each option (sequence/parallel) will result in a filter with different 2D frequency response. In our specific case, we apply low-pass filters in sequence since a greater amount of high-frequencies are “removed” from the signal, and we apply high-



(a) Photograph (b) Non-uniform (c) Uniform

Figure 5.9: Detail enhancement using a high-pass filter. Non-uniform sampling (b) avoids the common halo artifacts (black arrows) in traditional uniform sampling (c).

pass filters in parallel since a greater amount of high-frequencies are preserved in the signal (see Figure 5.8). This is the strategy we used for filtering the images shown in the paper. Filtering higher-dimensional signals such as 3D volumes is performed analogously.

5.5.2 Detail Manipulation

Our formulation for non-uniform recursive digital filters enables for the first time the direct application of general filters in edge-aware applications. For example, one can perform general frequency-domain manipulations without introducing artifacts such as halos around objects, as shown in Figure 5.9. This is possible due to the non-uniform sampling of the image pixels defined by Equation 5.29.

Figure 5.1 shows several examples of high-order IIR filters used to manipulate the details of the photograph shown in (a). In (b), a low-pass Gaussian smoothes small variations while preserving large-scale features. For the image shown in (c), we used a modified band-stop Laplacian of Gaussian to create a stylized look for the image. For the result shown in (d), we used a high-pass Butterworth to enhance fine details in the tiger's fur and whiskers. The image in (e) was obtained with a band-pass Butterworth to improve local contrast by enhancing medium-scale details. For (e), an edge-aware low-pass post-filter was applied to obtain the final result. This is necessary because 2D filtering using the domain transform sometimes introduces axis-aligned artifacts in the filtered image. Our webpage shows these filters applied to many other images and video.

While edge-aware detail manipulation has been performed by previous approaches (FATTAL; AGRAWALA; RUSINKIEWICZ, 2007; FARBMAN et al., 2008; PARIS; HASINOFF; KAUTZ, 2011), all of them work by computing differences between the outputs of a fixed type of low-pass filter. By providing the ability to experiment with the design and composition of new digital filters, our method has the potential do enable a greater variety of effects.

5.5.3 Localized Editing

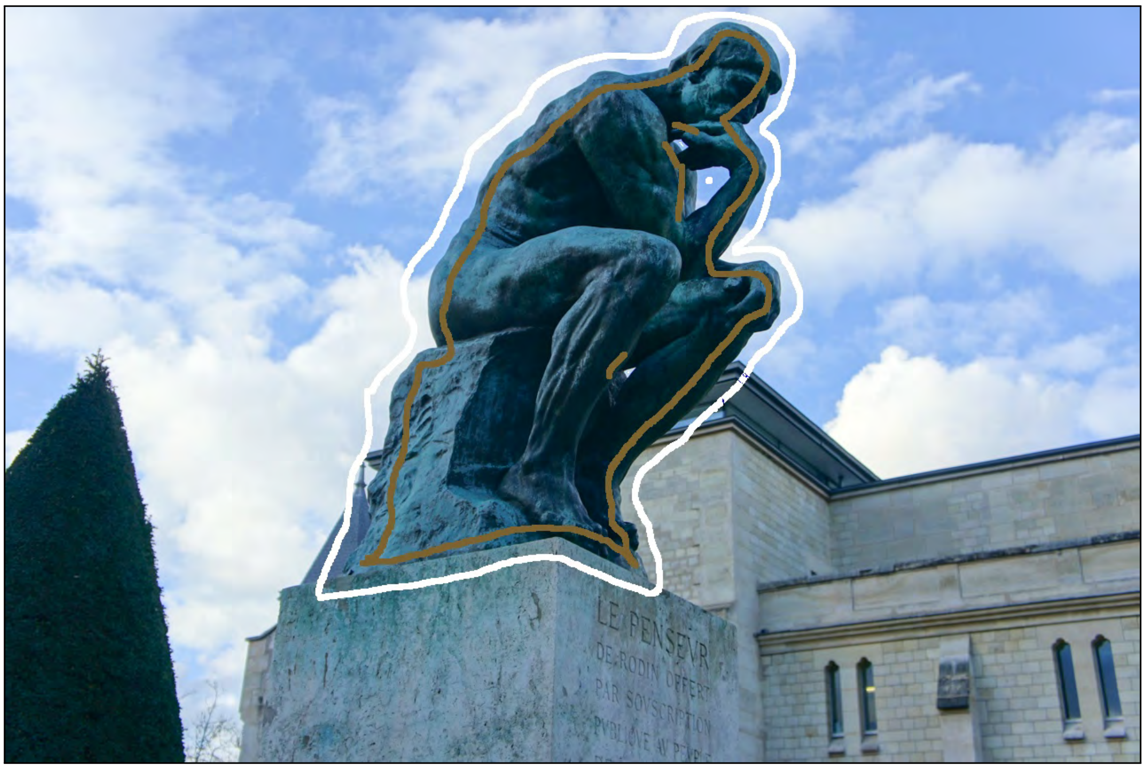
Filtering in non-uniform edge-aware domains can also be used for localized manipulation of pixel colors. In Figure 5.10(a), color scribbles define two regions of interest in the underlying photograph. For each region, we generate an influence map using our low-pass non-uniform Gaussian filter (see (LISCHINSKI et al., 2006) for details). The influence map for the region of interest is then normalized by the sum of influence maps for all regions, which defines a soft-segmentation mask. This mask is used to restrict recoloring to certain parts of the image.

5.5.4 Data-aware Interpolation

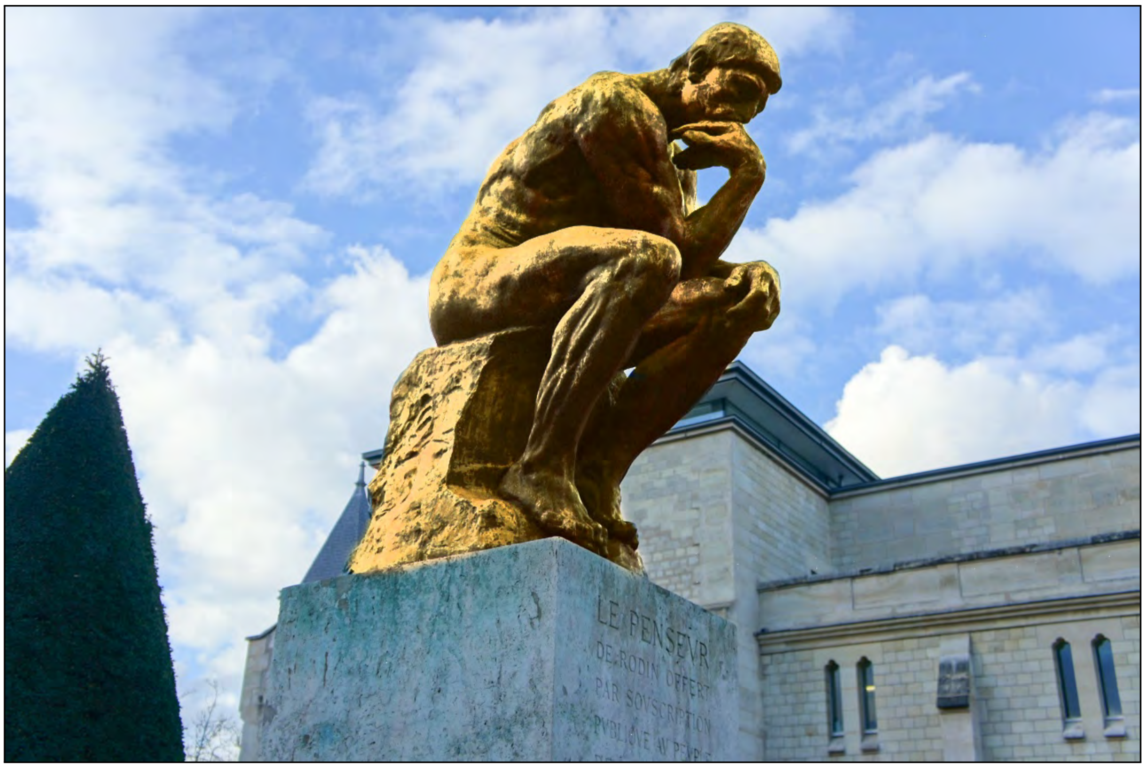
Propagating sparse data across the imagespace also benefits from an edge-aware operator (LEVIN; LISCHINSKI; WEISS, 2004). For example, in Figure 5.11 our low-pass non-uniform Gaussian filter is used to propagate the color of a small set of pixels, shown in (c), to the whole image. This generates the full-color image shown in (d). The non-uniform domain is defined by the domain transform applied to the lightness image in (b).

5.5.5 Denoising

By grouping pixels based on high-dimensional neighborhoods, we can define a fast and simple denoising algorithm, as illustrated in Figure 5.12. We cluster pixels from the noisy photo in (a) based on their proximity on the high dimensional non-local means space (BUADES; COLL; MOREL, 2005). For this example, we generate 30 disjoint clusters using k-means, which are color-coded in (c) for visualization. The pixels belonging to the same cluster define a *non-uniformly sampled signal* in the imagespace. We apply a non-uniform Gaussian filter only to the pixels belonging to the same clusters, averaging-out the zero-mean noise. This is followed by a second non-uniform edge-aware Butterworth low-pass filter on the hole image (adhering to the edges of (a)), with the goal of removing quantization borders which originate from the discrete clusters. The resulting denoised photograph is shown in (b).



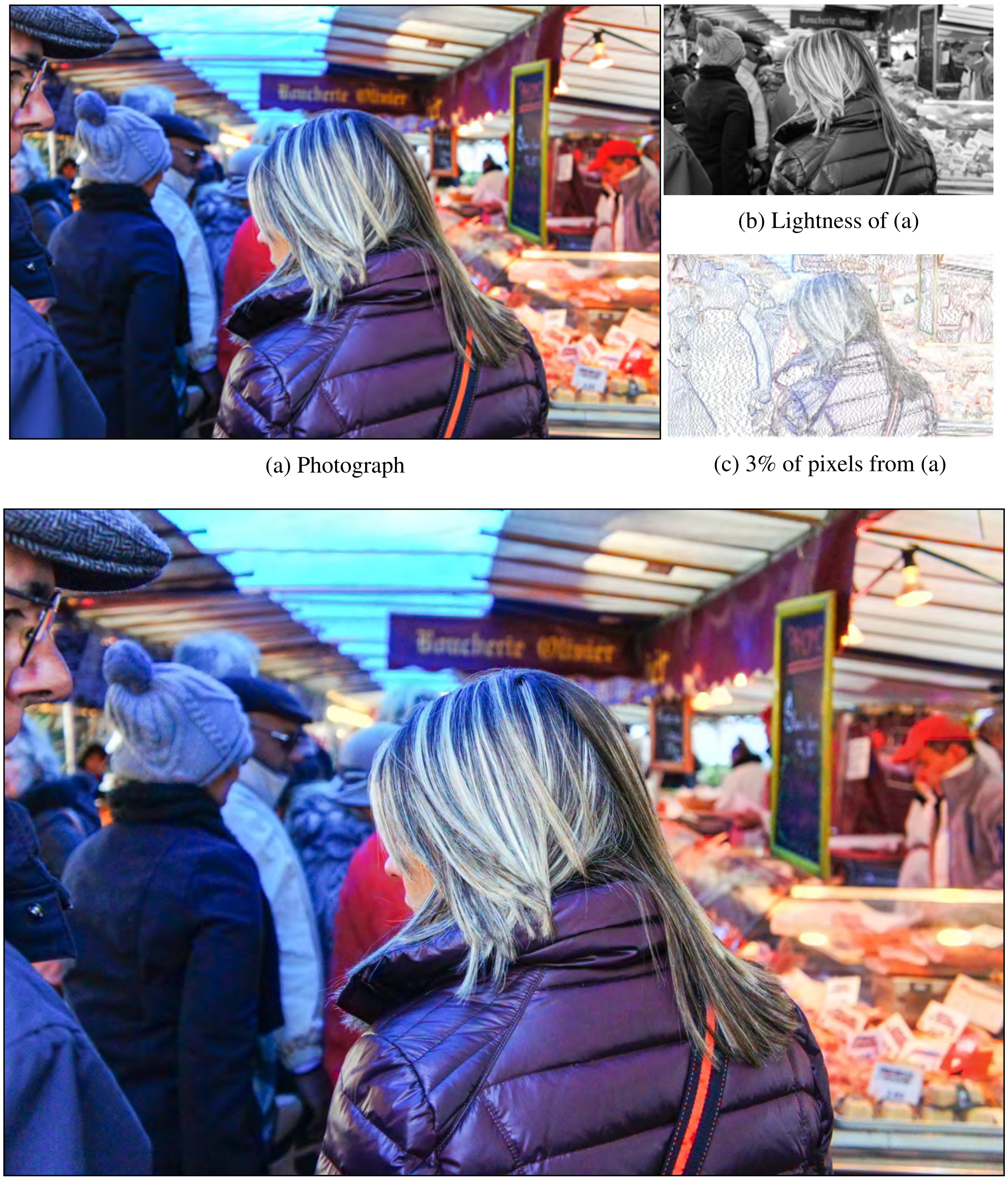
(a) Photograph with scribbles



(b) Edited output

Figure 5.10: Turning bronze into gold using our approach. See the text for details.

Using our formulation, for an image with N pixels, filtering together only pixels belonging to the same clusters is done in $O(N)$ time *for all clusters*. That is, the time



(a) Photograph

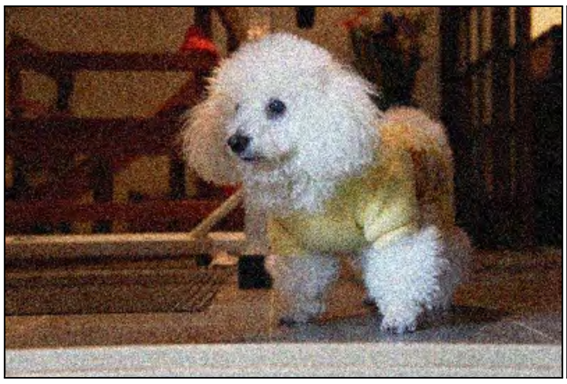
(b) Lightness of (a)

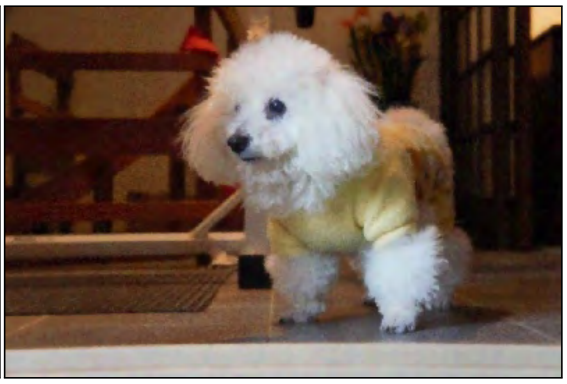
(c) 3% of pixels from (a)

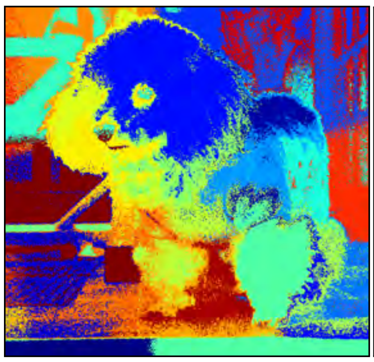
(d) Reconstructed from (b), (c)

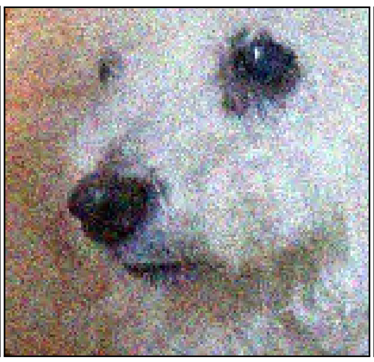
Figure 5.11: Example of data-aware interpolation using non-uniform filtering. A full-color image is reconstructed from the lightness channel and only 3% of the pixels from the original image, shown in (a). The pixels in (c) were importance-sampled using the gradient magnitude of the lightness channel. PSNR of (d) vs (a) is 31.17 dB.

complexity is independent of the number of clusters. Without our formulation, for K clusters, one would have to separate the image into K uniformly-sampled N-pixel images for filtering, which would result in $O(N\,K)$ complexity.


(a) Noisy photograph


(b) Denoised


(c) Clustering


(d) Detail from (a)

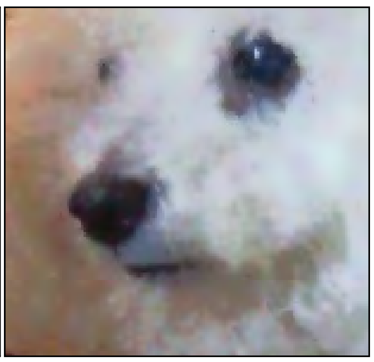
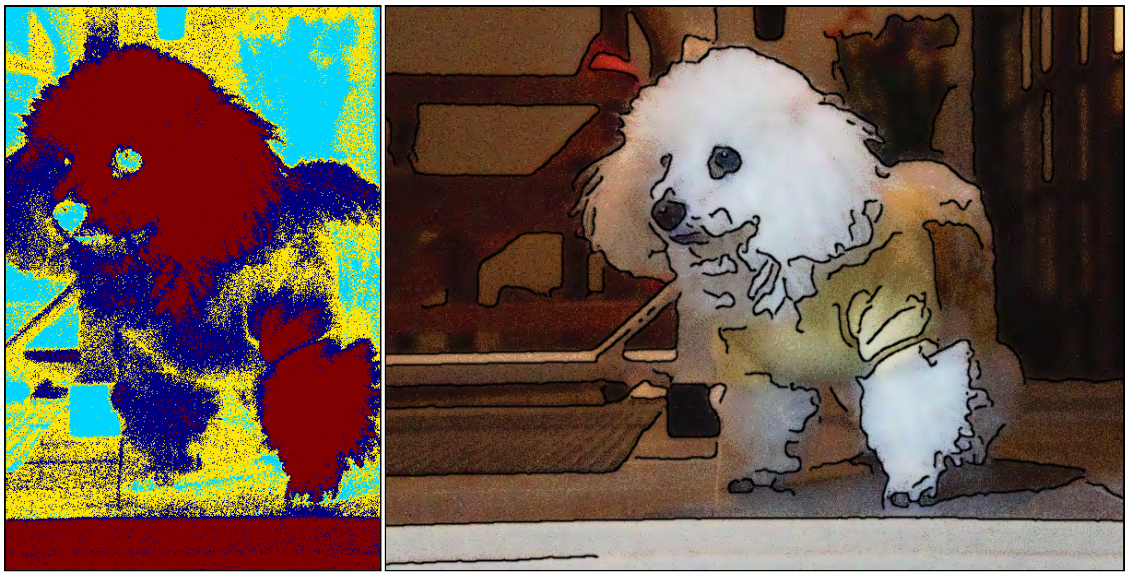

(e) Detail from (b)

Figure 5.12: Denoising using non-uniformly sampled pixel groups defined by k-means clustering. See text for details.

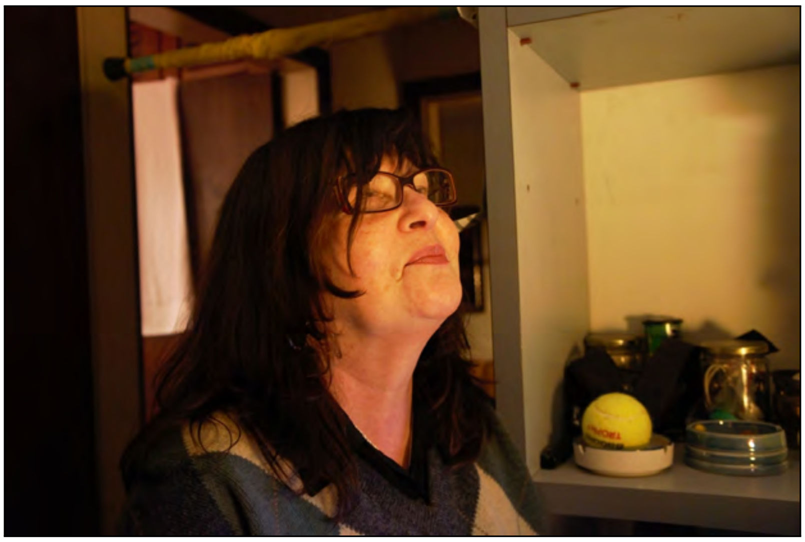
5.5.6 Stylization

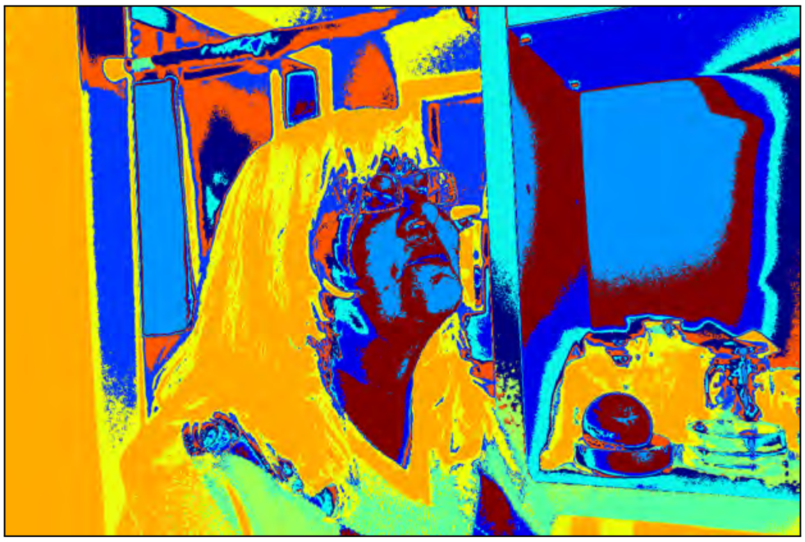
The same idea behind the denoising algorithm above can be used for stylization. In Figure 5.13(a), we cluster pixels based only on their RGB-proximity. Filtering only pixels (of the input image) belonging to the same clusters with a non-uniform Gaussian, and then superimposing edges computed using the Canny algorithm applied to the filtered image, one obtains a soft cartoon-like look (b). This procedure is also illustrated in Figure 5.14 for another photograph.



(a) Clustering (b) Stylized

Figure 5.13: Stylization using non-uniform filtering.


(a) Photograph


(b) Clustering

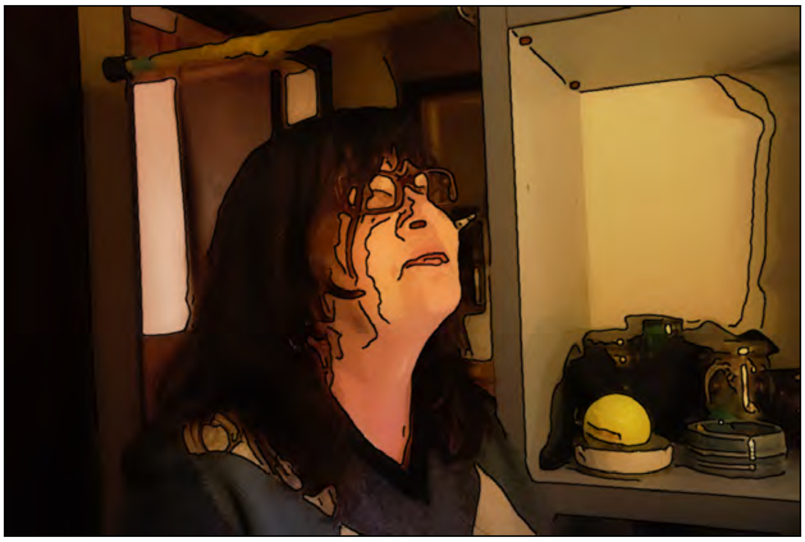

(c) Stylized

Figure 5.14: Another example of stylization using non-uniform filtering.

6 CONCLUSIONS AND FUTURE WORK

This dissertation described two novel approaches for high-dimensional filtering. The **domain transform** was presented in Chapter 3 to efficiently perform high-dimensional geodesic filtering of images and videos. It is the *first* filter of such kind that simultaneously exhibits the following properties: (i) it supports a continuum of scales; (ii) its processing time is linear in the number of pixels, and is independent of the filter parameters, allowing for real-time computations; (iii) correctly handles color images; and (iv) offers control over the kernel's shape. The **adaptive manifolds** were presented in Chapter 4 to efficiently perform high-dimensional Euclidean filtering of images and videos. It is the *first* filter of such kind with linear cost both in the number of pixels and in the dimensionality of the space in which the filter operates.

In Chapter 5 we presented a discrete-time mathematical formulation for applying recursive digital filters to non-uniformly sampled signals. Our method is general and works with any non-uniformly sampled signal and any recursive digital filter defined by a difference equation. Together with the domain transform, we demonstrated the use of our non-uniform formulation to achieve general edge-aware filtering for *arbitrary* digital filters. By providing the ability to experiment with the design and composition of new digital filters, our method has the potential do enable a greater variety of image and video effects.

The filters presented in this dissertation provide the fastest performance (both on CPU and GPU) for a variety of real-world applications. This efficiency come from their *linear cost* in both the number of pixels and the dimensionality of the space in which the filters operate. We demonstrated the versatility of the domain transform and adaptive manifolds on several real-time image and video processing tasks including color edge-aware smoothing, depth-of-field effects, stylization, recoloring, colorization, detail enhancement, denoising (using up to 147 dimensions), global illumination smoothing, and tone mapping. Thus, our filters provide a valuable tool for the image and video processing, computer graphics, computer vision, and computational photography communities.

6.1 Future Work

Some possible areas for further exploration include extending our filters to non-structured data, investigating applications of our single-pass Euclidean-geodesic filter, improving the worst-case complexity of the adaptive-manifold filter, and analysing the manifold-tree structure for processing videos.

6.1.1 Extension of our filters to non-structured data

Non-structured data, such as 3D meshes and point clouds, could also benefit from our fast filters. Such an extension could be used to denoise 3D models while preserving well-defined edges, or enhance the models by increasing the magnitude of small-scale details. One could even perform efficient, approximate n-body simulations for games: since the gravitational potential satisfies Poisson's equation, this potential can be found by convolving its corresponding Green's function with a signal defined by impulses at the masses' locations in space.

6.1.2 Exploration of our single-pass Euclidean-geodesic filter

In Section 4.7.5, we introduce the first single-pass Euclidean-geodesic filter, which uses Euclidean distances for some dimensions and geodesic distances for others. We intend to further explore the properties and applications of single-pass Euclidean-geodesic filters. For example, in Section 4.7.5, we use such a filter to compute a selection mask for local tone adjustments in images. However, this operation could be performed in two passes: an Euclidean filter followed by a geodesic filter. Thus, one of our goals is to find applications which can be performed using a single-pass Euclidean-geodesic filter, but are not possible with iterations of independent Euclidean and geodesic filters.

6.1.3 Improvement of the worst-case performance of the adaptive-manifold filter

When the standard deviation of the filter kernel over the space dimensions is large, the adaptive manifolds become linear almost everywhere, losing their ability to adapt to the signal. To compensate for this, more manifolds are needed to guarantee a sufficient sampling of the high-dimensional space, which affects performance. Performance in this case could be improved by using some kind of Monte Carlo sampling: 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 Equation 4.10) will blend almost everyone. Thus, it should suffice to just project a subset of the samples. Performance could also be improved by building inter-correlated manifolds that allow sub-linear nearest-neighbor searches to find the closest manifolds to a sample. If this is possible, the time complexity of our algorithm could potentially be reduced to $O(dN \log K)$.

6.1.4 Analysis of the adaptive-manifold tree for processing videos

Due to the high correlation between frames in a video sequence, one could obtain performance improvements when filtering videos by recycling manifolds across frames. One example would be computing the root manifold (η_1) only once every k frames, and interpolating the value of η_1 for all $k - 1$ frames in-between. This should work since η_1 is computed by low-pass filtering, and such low frequencies are expected to vary slowly across time (in most situations). A better approach could be to dynamically compute the value of k based on content change across video frames.

REFERENCES

ADAMS, A. B. **High-dimensional gaussian filtering for computational photography**. 2011. Tese (Doutorado em Ciência da Computação) — Stanford University.

ADAMS, A.; BAEK, J.; DAVIS, M. A. Fast high-dimensional filtering using the permutohedral lattice. **CGF**, [S.l.], v.29, n.2, p.753–762, 2010.

ADAMS, A. et al. Gaussian KD-Trees for Fast High-Dimensional Filtering. In: SIGGRAPH. **Proceedings...** [S.l.: s.n.], 2009.

Adobe Systems Inc. **Photoshop CS6**. Computer software.

Adobe Systems Inc. **After Effects CS6**. Computer software.

ARASARATNAM, I.; HAYKIN, S.; ELLIOTT, R. Discrete-time nonlinear filtering algorithms using Gauss–Hermite quadrature. **Proc. of the IEEE**, [S.l.], v.95, n.5, p.953–977, 2007.

AURICH, V.; WEULE, J. Non-Linear Gaussian Filters Performing Edge Preserving Diffusion. In: MUSTERERKENNUNG 1995, 17. DAGM-SYMPOSIUM, London, UK. **Proceedings...** [S.l.: s.n.], 1995. p.538–545.

BAE, S.; PARIS, S.; DURAND, F. Two-scale tone management for photographic look. **ACM TOG**, New York, NY, USA, v.25, n.3, p.637–645, 2006.

BAEK, J. et al. WYSIWYG Computational Photography via Viewfinder Editing. **ACM Trans. Graph.**, New York, NY, USA, v.32, n.6, p.198:1–198:10, Nov. 2013.

BARASH, D. A Fundamental Relationship between Bilateral Filtering, Adaptive Smoothing, and the Nonlinear Diffusion Equation. **IEEE TPAMI**, [S.l.], v.24, p.844–847, 2002.

BARRON, J. T. et al. Fast Bilateral-Space Stereo for Synthetic Defocus. In: CVPR. **Proceedings...** [S.l.: s.n.], 2015. p.4466–4474.

BAUSZAT, P.; EISEMANN, M.; MAGNOR, M. Guided Image Filtering for Interactive High-quality Global Illumination. **Computer Graphics Forum**, [S.l.], v.30, n.4, p.1361–1368, 2011.

BELKIN, M.; NIYOGI, P. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. **Neural Computation**, Cambridge, MA, USA, v.15, n.6, p.1373–1396, June 2003.

BENNETT, E. P.; MCMILLAN, L. Video enhancement using per-pixel virtual exposures. **ACM TOG**, New York, NY, USA, v.24, p.845–852, 2005.

BEZANSON, J. et al. Julia: A fast dynamic language for technical computing. **CoRR**, [S.l.], v.abs/1209.5145, 2012.

BLACK, M. et al. Robust anisotropic diffusion. **IEEE TIP**, [S.l.], v.7, n.3, p.421–432, 1998.

BLU, T.; THÉVENAZ, P.; UNSER, M. Generalized Interpolation: higher quality at no additional cost. In: IEEE ICIP. **Proceedings...** [S.l.: s.n.], 1999. p.667–671.

BONNEEL, N. et al. Example-based Video Color Grading. **ACM TOG**, New York, NY, USA, v.32, n.4, p.39:1–39:12, July 2013.

BRUSCHETTA, M. **A variational integrators approach to second order modeling and identification of linear mechanical systems**. 2011. Tese (Doutorado em Ciência da Computação) — School in Information Engineering - Università Degli Studi Di Padova.

BRUSCHETTA, M.; PICCI, G.; SACCON, A. A variational integrators approach to second order modeling and identification of linear mechanical systems. **Automatica**, [S.l.], v.50, n.3, p.727 – 736, 2014.

BUADES, A.; COLL, B.; MOREL, J. **A non-local algorithm for image denoising**. 2005. 60–65p. v.2.

CATMULL, E.; ROM, R. A class of local interpolating splines. **Computr aided geometric design**, [S.l.], v.74, p.317–326, 1974.

CATTÉ, F. et al. Image selective smoothing and edge detection by nonlinear diffusion. **SIAM Journal on Numerical analysis**, [S.l.], v.29, n.1, p.182–193, 1992.

CHEN, J.; PARIS, S.; DURAND, F. Real-time edge-aware image processing with the bilateral grid. **ACM TOG**, [S.l.], v.26, n.3, p.103, 2007.

CRIMINISI, A. et al. Geodesic image and video editing. **ACM TOG**, New York, NY, USA, v.29, n.5, p.134, 2010.

DABOV, K. et al. Image denoising by sparse 3-D transform-domain collaborative filtering. **IEEE TIP**, [S.l.], v.16, n.8, p.2080–2095, 2007.

D'ALMEIDA, F. **Nonlinear Diffusion Toolbox**. Retrieved October 2014, from http://www.mathworks.com/matlabcentral/fileexchange/3710-nonlinear-diffusion-toolbox.

DATTA, B. **Numerical Linear Algebra and Applications, 2nd Edition**. [S.l.]: Society for Industrial and Applied Mathematics (SIAM), 2010.

DERICHE, R. **Recursively implementating the Gaussian and its derivatives**. 1993.

DOUGHERTY, E. **Digital Image Processing Methods**. [S.l.]: CRC Press, 1994. (Optical engineering).

DURAND, F.; DORSEY, J. Fast bilateral filtering for the display of high-dynamic-range images. In: SIGGRAPH '02. **Proceedings...** [S.l.: s.n.], 2002. p.257–266.

EISEMANN, E.; DURAND, F. Flash photography enhancement via intrinsic relighting. In: ACM TOG. **Proceedings...** [S.l.: s.n.], 2004. v.23, n.3, p.673–678.

ENG, F. **Non-Uniform Sampling in Statistical Signal Processing**. 2007. Tese (Doutorado em Ciência da Computação) — Linköping universitet.

FARBMAN, Z. et al. Edge-preserving decompositions for multi-scale tone and detail manipulation. **ACM TOG**, [S.l.], v.27, n.3, p.67, 2008.

FARBMAN, Z.; FATTAL, R.; LISCHINSKI, D. Diffusion maps for edge-aware image editing. **ACM TOG**, [S.l.], v.29, n.6, p.145, 2010.

FATTAL, R. Edge-avoiding wavelets and their applications. **ACM TOG**, New York, NY, USA, v.28, n.3, p.22, 2009.

FATTAL, R.; AGRAWALA, M.; RUSINKIEWICZ, S. Multiscale shape and detail enhancement from multi-light image collections. **ACM TOG**, New York, NY, USA, v.26, p.51:1–51:9, 2007.

FESQUET, L.; BIDéGARAY-FESQUET, B. IIR digital filtering of non-uniformly sampled signals via state representation. **Signal Processing**, [S.l.], v.90, n.10, p.2811 – 2821, 2010.

FLEET, D. J.; LANGLEY, K. Recursive Filters for Optical Flow. **IEEE TPAMI**, Washington, DC, USA, v.17, n.1, p.61–67, Jan. 1995.

GASTAL, E. S. L.; OLIVEIRA, M. M. Shared Sampling for Real-Time Alpha Matting. **Computer Graphics Forum**, [S.l.], v.29, n.2, p.575–584, 2010. Proceedings of Eurographics 2010.

GASTAL, E. S. L.; OLIVEIRA, M. M. Domain Transform for Edge-Aware Image and Video Processing. **ACM TOG**, [S.l.], v.30, n.4, p.69:1–69:12, 2011. Proceedings of SIGGRAPH 2011.

GASTAL, E. S. L.; OLIVEIRA, M. M. Adaptive Manifolds for Real-Time High-Dimensional Filtering. **ACM TOG**, [S.l.], v.31, n.4, p.33:1–33:13, 2012. Proceedings of SIGGRAPH 2012.

GASTAL, E. S. L.; OLIVEIRA, M. M. **Domain Transform for Edge-Aware Image and Video Processing — Project Webpage**. Retrieved May 2015, from http://inf.ufrgs.br/~eslgastal/DomainTransform.

GASTAL, E. S. L.; OLIVEIRA, M. M. **Adaptive Manifolds for Real-Time High-Dimensional Filtering — Project Webpage**. Retrieved May 2015, from http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds.

GASTAL, E. S. L.; OLIVEIRA, M. M. **High-Order Recursive Filtering of Non-Uniformly Sampled Signals for Image and Video Processing — Project Webpage**. Retrieved May 2015, from http://inf.ufrgs.br/~eslgastal/NonUniformFiltering.

GASTAL, E. S. L.; OLIVEIRA, M. M. High-Order Recursive Filtering of Non-Uniformly Sampled Signals for Image and Video Processing. **Computer Graphics Forum**, [S.l.], v.34, n.2, p.81–93, 2015. Proceedings of Eurographics 2015.

GREWENIG, S.; WEICKERT, J.; BRUHN, A. From box filtering to fast explicit diffusion. **Pattern Recognition**, [S.l.], p.533–542, 2010.

HALE, D. Recursive gaussian filters. **CWP Report 546**, [S.l.], 2006.

HE, K. et al. A global sampling method for alpha matting. In: CVPR. **Proceedings...** [S.l.: s.n.], 2011. p.2049–2056.

HE, K.; SUN, J.; TANG, X. Guided Image Filtering. In: **ECCV**. [S.l.]: Springer Berlin / Heidelberg, 2010. p.1–14.

HECKBERT, P. S. Filtering by repeated integration. **SIGGRAPH Comput. Graph.**, New York, NY, USA, v.20, n.4, p.315–321, 1986.

JENSEN, H. W. Global Illumination using Photon Maps. In: PUEYO, X.; SCHRÃ¶DER, P. (Ed.). **Rendering Techniques**. [S.l.]: Springer, 1996. p.21–30. (Eurographics).

JONES, E. et al. **SciPy**: open source scientific tools for Python — scipy.signal documentation. Retrieved October 2014, http://docs.scipy.org/doc/scipy/reference/signal.html#filter-design.

KIMBALL, S.; MATTIS, P.; GIMP Development Team. **GNU Image Manipulation Program**. Computer software.

KIMMEL, R.; SOCHEN, N. A.; MALLADI, R. From High Energy Physics to Low Level Vision. In: SCALE-SPACE THEORY IN COMPUTER VISION. **Proceedings...** Springer-Verlag, 1997. p.236–247.

KNUTSSON, H.; WESTIN, C.-F. Normalized and Differential Convolution: methods for interpolation and filtering of incomplete and uncertain data. In: CVPR, New York City, USA. **Proceedings...** [S.l.: s.n.], 1993. p.515–523.

KOPF, J. et al. Joint bilateral upsampling. **ACM TOG**, New York, NY, USA, v.26, p.96:1–96:5, 2007.

LANG, M. et al. Practical temporal consistency for image-based graphics applications. **ACM Trans. Graph.**, New York, NY, USA, v.31, n.4, p.34:1–34:8, July 2012.

LEVIN, A.; LISCHINSKI, D.; WEISS, Y. Colorization using optimization. **ACM TOG**, [S.l.], v.23, p.689–694, 2004.

LÉVY, B. et al. Least squares conformal maps for automatic texture atlas generation. In: ACM SIGGRAPH. **Proceedings...** [S.l.: s.n.], 2002. p.362–371.

LISCHINSKI, D. et al. Interactive local adjustment of tonal values. **ACM TOG**, New York, NY, USA, v.25, n.3, p.646–653, 2006.

LOEVE, M. **Probability Theory**. [S.l.]: Springer, 1977. v.1.

MARVASTI, F. **Nonuniform sampling**: theory and practice. [S.l.]: Springer, 2001. v.1.

MATLAB. **version 8.3 (R2014a) — `d2c` documentation**. Retrieved October 2014, `http://www.mathworks.com/help/control/ref/d2c.html`.

MATLAB. **version 8.3 (R2014a) — Digital Filter Design documentation**. Retrieved October 2014, `http://www.mathworks.com/help/signal/digital-filter-design.html`.

NEHAB, D. et al. GPU-efficient recursive filtering and summed-area tables. **ACM TOG**, New York, NY, USA, v.30, p.176:1–176:12, 2011.

NIST. **National Institute of Standards and Technology**: digital library of mathematical functions. 2011.

OLIVEIRA, M. M.; BISHOP, G.; MCALLISTER, D. Relief Texture Mapping. In: ACM SIGGRAPH. **Proceedings...** [S.l.: s.n.], 2000. p.359–368.

O'NEILL, B. **Elementary Differential Geometry**. [S.l.]: AP, 2006.

OpenCV. **Open Source Computer Vision library, version 3.0 — Extended Image Processing documentation**. Retrieved October 2014, from http://docs.opencv.org/trunk/modules/ximgproc/doc/ximgproc.html.

OpenCV. **Open Source Computer Vision library, version 3.0 — Computational Photography, Non-Photorealistic Rendering documentation**. Retrieved October 2014, from http://docs.opencv.org/trunk/modules/photo/doc/npr.html.

PARIS, S.; DURAND, F. A fast approximation of the bilateral filter using a signal processing approach. **IJCV**, [S.l.], v.81, n.1, p.24–52, 2009.

PARIS, S.; HASINOFF, S. W.; KAUTZ, J. Local Laplacian Filters: edge-aware image processing with a laplacian pyramid. **ACM TOG**, New York, NY, USA, v.30, n.4, p.68:1–68:12, 2011.

PERONA, P.; MALIK, J. Scale-Space and Edge Detection Using Anisotropic Diffusion. **IEEE TPAMI**, Washington, DC, USA, v.12, n.7, p.629–639, 1990.

PETSCHNIGG, G. et al. Digital photography with flash and no-flash image pairs. **ACM TOG**, [S.l.], v.23, n.3, p.664–672, 2004.

PHAM, T.; VAN VLIET, L. Separable bilateral filtering for fast video preprocessing. **IEEE Intl. Conf. on Multimedia and Expo**, Los Alamitos, CA, USA, v.0, p.4 pp, 2005.

PIRODDI, R.; PETROU, M. Analysis of irregularly sampled data: a review. **Advances in Imaging and Electron Physics**, [S.l.], v.132, p.109–165, 2004.

PORIKLI, F. Constant time O(1) bilateral filtering. In: CVPR. **Proceedings...** [S.l.: s.n.], 2008. p.1–8.

POULTON, D.; OKSMAN, J. **Digital Filters for Non-Uniformly Sampled Signals**. 2000. 421–424p.

PROAKIS, J. G.; MANOLAKIS, D. K. **Digital Signal Processing**: principles, algorithms, and applications. [S.l.]: Pearson Education India, 2007.

RICHARDT, C. et al. Real-Time Spatiotemporal Stereo Matching Using the Dual-Cross-Bilateral Grid. In: **ECCV**. [S.l.]: Springer Berlin / Heidelberg, 2010. p.510–523.

SHAPIRO, H.; SILVERMAN, R. Alias-Free Sampling of Random Noise. **Journal of the Society for Industrial and Applied Mathematics**, [S.l.], v.8, n.2, p.225–248, 1960.

SMITH, A. R. **Planar 2-Pass Texture Mapping and Warping**. 1987. 263-272p.

SMITH, J. O. **Introduction to Digital Filters with Audio Applications**. [S.l.]: W3K Publishing, 2007.

SMITH, S. M.; BRADY, J. M. SUSAN – A new approach to low level image processing. **International journal of computer vision**, [S.l.], v.23, n.1, p.45–78, 1997.

SMITH, S. W. **The Scientist & Engineer's Guide to Digital Signal Processing**. [S.l.]: California Technical Pub, 1997.

SOCHEN, N.; KIMMEL, R.; BRUCKSTEIN, A. Diffusions and confusions in signal and image processing. **Journal of Mathematical Imaging and Vision**, [S.l.], v.14, n.3, p.195–209, 2001.

SUBR, K.; SOLER, C.; DURAND, F. Edge-preserving multiscale image decomposition based on local extrema. **ACM TOG**, [S.l.], v.28, p.147:1–147:9, 2009.

TASDIZEN, T. Principal components for non-local means image denoising. In: ICIP. **Proceedings...** [S.l.: s.n.], 2008. p.1728–1731.

TOMASI, C.; MANDUCHI, R. Bilateral Filtering for Gray and Color Images. In: ICCV. **Proceedings...** [S.l.: s.n.], 1998. p.839–846.

VAN VLIET, L. J.; YOUNG, I. T.; VERBEEK, P. W. Recursive Gaussian derivative filters. In: ICPR. **Proceedings...** [S.l.: s.n.], 1998. v.1, p.509–514.

WANG, Z. et al. Image quality assessment: from error visibility to structural similarity. **IEEE Trans. on Image Processing**, [S.l.], v.13, n.4, p.600–612, 2004.

WEBER, M. et al. Spatio-temporal photon density estimation using bilateral filtering. In: CGI. **Proceedings...** [S.l.: s.n.], 2004. p.120–127.

WEBER, O. et al. Parallel Algorithms for Approximation of Distance Maps on Parametric Surfaces. **ACM Trans. Graph.**, New York, NY, USA, v.27, n.4, p.104:1–104:16, Nov. 2008.

WEICKERT, J.; ROMENY, B.; VIERGEVER, M. Efficient and reliable schemes for nonlinear diffusion filtering. **IEEE TIP**, [S.l.], v.7, n.3, p.398–410, 1998.

WEISS, Y.; FREEMAN, W. T. What makes a good model of natural images? **CVPR**, [S.l.], v.3, n.8, p.1–8, 2007.

WINNEMöLLER, H.; OLSEN, S. C.; GOOCH, B. Real-time video abstraction. **ACM TOG**, [S.l.], v.25, n.3, p.1226, 2006.

YANG, C. et al. Improved fast gauss transform and efficient kernel density estimation. In: ICCV. **Proceedings...** [S.l.: s.n.], 2003. p.664–671.

YANG, Q. Recursive Bilateral Filtering. In: **ECCV 2012**. [S.l.: s.n.], 2012. v.7572, p.399–413.

YANG, Q.; TAN, K. H.; AHUJA, N. Real-time O(1) Bilateral Filtering. In: CVPR. **Proceedings...** [S.l.: s.n.], 2009. p.557–564.

YOUNG, I.; VAN VLIET, L.; GINKEL, R. van. Recursive Gabor filtering. **IEEE Trans. on Signal Processing**, [S.l.], v.50, n.11, p.2798–2805, Nov 2002.

ZHANG, Q. et al. Rolling Guidance Filter. In: FLEET, D. et al. (Ed.). **Computer Vision – ECCV 2014**. [S.l.]: Springer International Publishing, 2014. p.815–830. (Lecture Notes in Computer Science, v.8691).

ZHUO, S. et al. Enhancing low light images using near infrared flash images. In: IEEE ICIP. **Proceedings...** [S.l.: s.n.], 2010. p.2537–2540.

APPENDIX A DOMAIN TRANSFORM SOURCE CODE

The following sections include our MATLAB implementation of the Normalized Convolution (NC), Interpolated Convolution (IC) and Recursive Filter (RF) based on the domain transform. These filters are described in Section 3.4. Please refer to our webpage at http://inf.ufrgs.br/~eslgastal/DomainTransform for up-to-date source, examples, and instructions on using this code.

A.1 Normalized Convolution filter

```
%  NC  Domain transform normalized convolution edge-preserving filter.
%
%  F = NC(img, sigma_s, sigma_r, num_iterations, joint_image)
%
%  Parameters:
%    img             Input image to be filtered.
%    sigma_s         Filter spatial standard deviation.
%    sigma_r         Filter range standard deviation.
%    num_iterations  Number of iterations to perform (default: 3).
%    joint_image     Optional image for joint filtering.
%
%
%
%  This is the reference implementation of the domain transform NC filter
%  described in the paper:
%
%    Domain Transform for Edge-Aware Image and Video Processing
%    Eduardo S. L. Gastal  and  Manuel M. Oliveira
%    ACM Transactions on Graphics. Volume 30 (2011), Number 4.
%    Proceedings of SIGGRAPH 2011, Article 69.
%
%  Please refer to the publication above if you use this software. For an
%  up-to-date version go to:
%
%             http://inf.ufrgs.br/~eslgastal/DomainTransform/
%
%
%  THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESSED OR IMPLIED WARRANTIES
%  OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
%  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
%  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
%  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
%  OUT OF OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN
%  THIS SOFTWARE.
%
%  Version 1.0 - August 2011.

function F = NC(img, sigma_s, sigma_r, num_iterations, joint_image)

    I = double(img);

    if ~exist('num_iterations', 'var')
```

```
        num_iterations = 3;
    end

    if exist('joint_image', 'var') && ~isempty(joint_image)
        J = double(joint_image);

        if (size(I,1) ~= size(J,1)) || (size(I,2) ~= size(J,2))
            error('Input and joint images must have equal width and height.');
        end
    else
        J = I;
    end

    [h w num_joint_channels] = size(J);

    %% Compute the domain transform
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Estimate horizontal and vertical partial derivatives using finite
    % differences.
    dIcdx = diff(J, 1, 2);
    dIcdy = diff(J, 1, 1);

    dIdx = zeros(h,w);
    dIdy = zeros(h,w);

    % Compute the l1-norm distance of neighbor pixels.
    for c = 1:num_joint_channels
        dIdx(:,2:end) = dIdx(:,2:end) + abs( dIcdx(:,:,c) );
        dIdy(2:end,:) = dIdy(2:end,:) + abs( dIcdy(:,:,c) );
    end

    % Compute the derivatives of the horizontal and vertical domain transforms.
    dHdx = (1 + sigma_s/sigma_r * dIdx);
    dVdy = (1 + sigma_s/sigma_r * dIdy);

    % Integrate the domain transforms.
    ct_H = cumsum(dHdx, 2);
    ct_V = cumsum(dVdy, 1);

    % The vertical pass is performed using a transposed image.
    ct_V = ct_V';

    %% Perform the filtering.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = num_iterations;
    F = I;

    sigma_H = sigma_s;

    for i = 0:num_iterations - 1

        % Compute the sigma value for this iteration
        sigma_H_i = sigma_H * sqrt(3) * 2^(N - (i + 1)) / sqrt(4^N - 1);

        % Compute the radius of the box filter with the desired variance.
        box_radius = sqrt(3) * sigma_H_i;

        F = TransformedDomainBoxFilter_Horizontal(F, ct_H, box_radius);
        F = image_transpose(F);

        F = TransformedDomainBoxFilter_Horizontal(F, ct_V, box_radius);
        F = image_transpose(F);

    end

    F = cast(F, class(img));

end

%% Box filter normalized convolution in the transformed domain.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function F = TransformedDomainBoxFilter_Horizontal(I, xform_domain_position, box_radius)
```

```

    [h w num_channels] = size(I);

    % Compute the lower and upper limits of the box kernel in the transformed domain.
    l_pos = xform_domain_position - box_radius;
    u_pos = xform_domain_position + box_radius;

    % Find the indices of the pixels associated with the lower and upper limits
    % of the box kernel.
    %
    % This loop is much faster in a compiled language.  If you are using a
    % MATLAB version which supports the 'parallel for' construct, you can
    % improve performance by replacing the following 'for' by a 'parfor'.

    l_idx = zeros(size(xform_domain_position));
    u_idx = zeros(size(xform_domain_position));

    for row = 1:h
        xform_domain_pos_row = [xform_domain_position(row,:) inf];

        l_pos_row = l_pos(row,:);
        u_pos_row = u_pos(row,:);

        local_l_idx = zeros(1, w);
        local_u_idx = zeros(1, w);

        local_l_idx(1) = find(xform_domain_pos_row > l_pos_row(1), 1, 'first');
        local_u_idx(1) = find(xform_domain_pos_row > u_pos_row(1), 1, 'first');

        for col = 2:w
            local_l_idx(col) = local_l_idx(col-1) + ...
                find(xform_domain_pos_row(local_l_idx(col-1):end) > l_pos_row(col), 1, 'first') - 1;

            local_u_idx(col) = local_u_idx(col-1) + ...
                find(xform_domain_pos_row(local_u_idx(col-1):end) > u_pos_row(col), 1, 'first') - 1;
        end

        l_idx(row,:) = local_l_idx;
        u_idx(row,:) = local_u_idx;
    end

    % Compute the box filter using a summed area table.
    SAT             = zeros([h w+1 num_channels]);
    SAT(:,2:end,:) = cumsum(I, 2);
    row_indices    = repmat((1:h)', 1, w);
    F              = zeros(size(I));

    for c = 1:num_channels
        a = sub2ind(size(SAT), row_indices, l_idx, repmat(c, h, w));
        b = sub2ind(size(SAT), row_indices, u_idx, repmat(c, h, w));
        F(:,:,c) = (SAT(b) - SAT(a)) ./ (u_idx - l_idx);
    end

end

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function T = image_transpose(I)

    [h w num_channels] = size(I);

    T = zeros([w h num_channels], class(I));

    for c = 1:num_channels
        T(:,:,c) = I(:,:,c)';
    end

end
```

A.2 Interpolated Convolution filter

```
%  IC  Domain transform interpolated convolution edge-preserving filter.
%
%  F = IC(img, sigma_s, sigma_r, num_iterations, joint_image)
%
%  Parameters:
%    img             Input image to be filtered.
%    sigma_s         Filter spatial standard deviation.
%    sigma_r         Filter range standard deviation.
%    num_iterations  Number of iterations to perform (default: 3).
%    joint_image     Optional image for joint filtering.
%
%
%
%  This is the reference implementation of the domain transform IC filter
%  described in the paper:
%
%    Domain Transform for Edge-Aware Image and Video Processing
%    Eduardo S. L. Gastal  and  Manuel M. Oliveira
%    ACM Transactions on Graphics. Volume 30 (2011), Number 4.
%    Proceedings of SIGGRAPH 2011, Article 69.
%
%  Please refer to the publication above if you use this software. For an
%  up-to-date version go to:
%
%              http://inf.ufrgs.br/~eslgastal/DomainTransform/
%
%
%  THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESSED OR IMPLIED WARRANTIES
%  OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
%  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
%  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
%  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
%  OUT OF OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN
%  THIS SOFTWARE.
%
%  Version 1.0 - August 2011.

function F = IC(img, sigma_s, sigma_r, num_iterations, joint_image)

    I = double(img);

    if ~exist('num_iterations', 'var')
        num_iterations = 3;
    end

    if exist('joint_image', 'var') && ~isempty(joint_image)
        J = double(joint_image);

        if (size(I,1) ~= size(J,1)) || (size(I,2) ~= size(J,2))
            error('Input and joint images must have equal width and height.');
        end
    else
        J = I;
    end

    [h w num_joint_channels] = size(J);

    %% Compute the domain transform
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Estimate horizontal and vertical partial derivatives using finite
    % differences.
    dIcdx = diff(J, 1, 2);
    dIcdy = diff(J, 1, 1);

    dIdx = zeros(h,w);
    dIdy = zeros(h,w);

    % Compute the l1-norm distance of neighbor pixels.
    for c = 1:num_joint_channels
        dIdx(:,2:end) = dIdx(:,2:end) + abs( dIcdx(:,:,c) );
        dIdy(2:end,:) = dIdy(2:end,:) + abs( dIcdy(:,:,c) );
    end

    % Compute the derivatives of the horizontal and vertical domain transforms.
    dHdx = (1 + sigma_s/sigma_r * dIdx);
    dVdy = (1 + sigma_s/sigma_r * dIdy);

    % Integrate the domain transforms.
    ct_H = cumsum(dHdx, 2);
    ct_V = cumsum(dVdy, 1);

    % The vertical pass is performed using a transposed image.
    ct_V = ct_V';

    %% Perform the filtering.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = num_iterations;
    F = I;

    sigma_H = sigma_s;

    for i = 0:num_iterations - 1

        % Compute the sigma value for this iteration
        sigma_H_i = sigma_H * sqrt(3) * 2^(N - (i + 1)) / sqrt(4^N - 1);

        % Compute the radius of the box filter with the desired variance.
        box_radius = sqrt(3) * sigma_H_i;

        F = TransformedDomainBoxFilter_Horizontal(F, ct_H, box_radius);
        F = image_transpose(F);

        F = TransformedDomainBoxFilter_Horizontal(F, ct_V, box_radius);
        F = image_transpose(F);

    end

    F = cast(F, class(img));

end

%% Box filter interpolated convolution in the transformed domain.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function F = TransformedDomainBoxFilter_Horizontal(I, xform_domain_position, box_radius)

    [h w num_channels] = size(I);

    % Compute the lower and upper limits of the box kernel in the transformed domain.
    l_pos = xform_domain_position - box_radius;
    u_pos = xform_domain_position + box_radius;

    % Find the indices of the pixels associated with the lower and upper limits
    % of the box kernel.
    %
    % This loop is much faster in a compiled language.  If you are using a
    % MATLAB version which supports the 'parallel for' construct, you can
    % improve performance by replacing the following 'for' by a 'parfor'.

    l_idx = zeros(size(xform_domain_position));
    u_idx = zeros(size(xform_domain_position));

    for row = 1:h
        xform_domain_pos_row = [xform_domain_position(row,:) inf];

        l_pos_row = l_pos(row,:);
        u_pos_row = u_pos(row,:);

        local_l_idx = zeros(1, w);
        local_u_idx = zeros(1, w);

        local_l_idx(1) = find(xform_domain_pos_row > l_pos_row(1), 1, 'first');
        local_u_idx(1) = find(xform_domain_pos_row > u_pos_row(1), 1, 'first');

        for col = 2:w
            local_l_idx(col) = local_l_idx(col-1) + ...
                find(xform_domain_pos_row(local_l_idx(col-1):end) > l_pos_row(col), 1, 'first') - 1;

            local_u_idx(col) = local_u_idx(col-1) + ...
                find(xform_domain_pos_row(local_u_idx(col-1):end) > u_pos_row(col), 1, 'first') - 1;
        end

        l_idx(row,:) = local_l_idx;
        u_idx(row,:) = local_u_idx;
    end

    % Compute the box filter using a summed area table. This SAT is built using
    % the area under the graph (in the transformed domain) of the interpolated
    % signal. We use linear interpolation and compute the area using the
    % trapezoidal rule.

    areas = bsxfun(@times, ...
                0.5 .* (I(:,2:end,:) + I(:,1:end-1,:)), ...
                xform_domain_position(:,2:end,:) - xform_domain_position(:,1:end-1,:) ...
            );

    SAT             = zeros([h w num_channels]);
    SAT(:,2:end,:) = cumsum(areas, 2);
    row_indices     = repmat((1:h)', 1, w);
    F               = zeros(size(I));

    I                      = padarray(I,                     [0 1 0], 'replicate');
    SAT                    = padarray(SAT,                   [0 1 0]);
    xform_domain_position = padarray(xform_domain_position, [0 1 0], 'replicate');

    % Pixel values outside the bounds of the image are assumed to equal the
    % nearest pixel border value.
    xform_domain_position(:,1)   = xform_domain_position(:,1)   - 1.2 * box_radius;
    xform_domain_position(:,end) = xform_domain_position(:,end) + 1.2 * box_radius;

    l_idx = l_idx + 1;

    for c = 1:num_channels

        l1_c = sub2ind(size(SAT), row_indices, l_idx, repmat(c, h, w));
        u0_c = sub2ind(size(SAT), row_indices, u_idx, repmat(c, h, w));

        l0_c = sub2ind(size(SAT), row_indices, l_idx - 1, repmat(c, h, w));
        u1_c = sub2ind(size(SAT), row_indices, u_idx + 1, repmat(c, h, w));

        l1 = sub2ind(size(SAT), row_indices, l_idx);
        u0 = sub2ind(size(SAT), row_indices, u_idx);

        l0 = sub2ind(size(SAT), row_indices, l_idx - 1);
        u1 = sub2ind(size(SAT), row_indices, u_idx + 1);

        % Full (center) areas.
        C = SAT(u0_c) - SAT(l1_c);

        % Left fractional areas.
        alpha = (l_pos - xform_domain_position(l0)) ...
             ./ (xform_domain_position(l1) - xform_domain_position(l0));
        yi    = I(l0_c) + alpha .* ( I(l1_c) - I(l0_c) );
        L     = 0.5 .* (yi + I(l1_c)) .* (1-alpha) ...
             .* (xform_domain_position(l1) - xform_domain_position(l0));

        % Right fractional areas.
        alpha = (u_pos - xform_domain_position(u0)) ...
             ./ (xform_domain_position(u1) - xform_domain_position(u0));
        yi    = I(u0_c) + alpha .* ( I(u1_c) - I(u0_c) );
        R     = 0.5 .* (yi + I(u0_c)) .* (alpha) ...
             .* (xform_domain_position(u1) - xform_domain_position(u0));

        F(:,:,c) = (L + C + R) / (2 * box_radius);

    end

```

```
end

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function T = image_transpose(I)

    [h w num_channels] = size(I);

    T = zeros([w h num_channels], class(I));

    for c = 1:num_channels
        T(:,:,c) = I(:,:,c)';
    end

end
```

A.3 Recursive filter

```
%  RF  Domain transform recursive edge-preserving filter.
%
%  F = RF(img, sigma_s, sigma_r, num_iterations, joint_image)
%
%  Parameters:
%    img             Input image to be filtered.
%    sigma_s         Filter spatial standard deviation.
%    sigma_r         Filter range standard deviation.
%    num_iterations  Number of iterations to perform (default: 3).
%    joint_image     Optional image for joint filtering.
%
%
%
%  This is the reference implementation of the domain transform RF filter
%  described in the paper:
%
%    Domain Transform for Edge-Aware Image and Video Processing
%    Eduardo S. L. Gastal  and  Manuel M. Oliveira
%    ACM Transactions on Graphics. Volume 30 (2011), Number 4.
%    Proceedings of SIGGRAPH 2011, Article 69.
%
%  Please refer to the publication above if you use this software. For an
%  up-to-date version go to:
%
%             http://inf.ufrgs.br/~eslgastal/DomainTransform/
%
%
%  THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESSED OR IMPLIED WARRANTIES
%  OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
%  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
%  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
%  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
%  OUT OF OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN
%  THIS SOFTWARE.
%
%  Version 1.0 - August 2011.

function F = RF(img, sigma_s, sigma_r, num_iterations, joint_image)

    I = double(img);

    if ~exist('num_iterations', 'var')
        num_iterations = 3;
    end

    if exist('joint_image', 'var') && ~isempty(joint_image)
        J = double(joint_image);

        if (size(I,1) ~= size(J,1)) || (size(I,2) ~= size(J,2))
            error('Input and joint images must have equal width and height.');
        end
    else
        J = I;
    end

    [h w num_joint_channels] = size(J);

    %% Compute the domain transform
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Estimate horizontal and vertical partial derivatives using finite
    % differences.
    dIcdx = diff(J, 1, 2);
    dIcdy = diff(J, 1, 1);

    dIdx = zeros(h,w);
    dIdy = zeros(h,w);

    % Compute the l1-norm distance of neighbor pixels.
    for c = 1:num_joint_channels
        dIdx(:,2:end) = dIdx(:,2:end) + abs( dIcdx(:,:,c) );
        dIdy(2:end,:) = dIdy(2:end,:) + abs( dIcdy(:,:,c) );
    end

    % Compute the derivatives of the horizontal and vertical domain transforms.
    dHdx = (1 + sigma_s/sigma_r * dIdx);
    dVdy = (1 + sigma_s/sigma_r * dIdy);

    % We do not integrate the domain transforms since our recursive filter
    % uses the derivatives directly.
    %ct_H = cumsum(dHdx, 2);
    %ct_V = cumsum(dVdy, 1);

    % The vertical pass is performed using a transposed image.
    dVdy = dVdy';

    %% Perform the filtering.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = num_iterations;
    F = I;

    sigma_H = sigma_s;

    for i = 0:num_iterations - 1

        % Compute the sigma value for this iteration
        sigma_H_i = sigma_H * sqrt(3) * 2^(N - (i + 1)) / sqrt(4^N - 1);

        F = TransformedDomainRecursiveFilter_Horizontal(F, dHdx, sigma_H_i);
        F = image_transpose(F);

        F = TransformedDomainRecursiveFilter_Horizontal(F, dVdy, sigma_H_i);
        F = image_transpose(F);

    end

    F = cast(F, class(img));

end

%% Recursive filter.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function F = TransformedDomainRecursiveFilter_Horizontal(I, D, sigma)

    % Feedback coefficient (Appendix of our paper).
    a = exp(-sqrt(2) / sigma);

    F = I;
    V = a.^D;

    [h w num_channels] = size(I);

    % Left -> Right filter.
    for i = 2:w
        for c = 1:num_channels
            F(:,i,c) = F(:,i,c) + V(:,i) .* ( F(:,i - 1,c) - F(:,i,c) );
        end
    end

    % Right -> Left filter.
    for i = w-1:-1:1
        for c = 1:num_channels
            F(:,i,c) = F(:,i,c) + V(:,i+1) .* ( F(:,i + 1,c) - F(:,i,c) );
        end
    end

end

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function T = image_transpose(I)

    [h w num_channels] = size(I);

    T = zeros([w h num_channels], class(I));

    for c = 1:num_channels
        T(:,:,c) = I(:,:,c)';
    end

end
```

A.4 Example Usage

```
I = imread('statue.png');
I = im2double(I);

%% Edge-preserving smoothing example
sigma_s = 60;
sigma_r = 0.4;

% Filter using normalized convolution.
F_nc = NC(I, sigma_s, sigma_r);

% Filter using interpolated convolution.
F_ic = IC(I, sigma_s, sigma_r);

% Filter using the recursive filter.
F_rf = RF(I, sigma_s, sigma_r);

% Show results.
figure, imshow(I); title('Input photograph');
figure, imshow(F_nc); title('Normalized convolution');
figure, imshow(F_ic); title('Interpolated convolution');
figure, imshow(F_rf); title('Recursive filter');
```

APPENDIX B ADAPTIVE MANIFOLDS SOURCE CODE

B.1 Main Filtering Loop

This is the main MATLAB implementation of our adaptive-manifold high-dimensional filter described in Chapter 4. To use this code you must download the `keep.m` utility from http://www.mathworks.com/matlabcentral/fileexchange/181-keep. Please refer to our webpage at http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds for up-to-date source, examples, and instructions on using this code.

```
%  ADAPTIVE_MANIFOLD_FILTER  High-dimensional filtering using adaptive manifolds
%
%  Parameters:
%    f                    Input image to be filtered.
%    sigma_s              Filter spatial standard deviation.
%    sigma_r              Filter range standard deviation.
%
%  Optional parameters:
%    tree_height          Height of the manifold tree (default: automatically computed).
%    f_joint              Image for joint filtering.
%    num_pca_iterations   Number of iterations to computed the eigenvector v1 (default: 1)
%
%  Output:
%    g                    Adaptive-manifold filter response adjusted for outliers.
%    tilde_g              Adaptive-manifold filter response NOT adjusted for outliers.
%
%
%
%  This code is part of the reference implementation of the adaptive-manifold
%  high-dimensional filter described in the paper:
%
%    Adaptive Manifolds for Real-Time High-Dimensional Filtering
%    Eduardo S. L. Gastal  and  Manuel M. Oliveira
%    ACM Transactions on Graphics. Volume 31 (2012), Number 4.
%    Proceedings of SIGGRAPH 2012, Article 33.
%
%  Please refer to the publication above if you use this software. For an
%  up-to-date version go to:
%
%             http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds/
%
%
%  THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESSED OR IMPLIED WARRANTIES
%  OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
%  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
%  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
%  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
%  OUT OF OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN
%  THIS SOFTWARE.
%
%  Version 1.0 - January 2012.

function [g tilde_g] = adaptive_manifold_filter(f, sigma_s, sigma_r, tree_height,...
                                                f_joint, num_pca_iterations)

f = im2double(f);

% Use the center pixel as seed to random number generation.
rand('seed', f(round(end/2),round(end/2),1) );

global sum_w_ki_Psi_blur;
global sum_w_ki_Psi_blur_0;

sum_w_ki_Psi_blur   = zeros(size(f));
sum_w_ki_Psi_blur_0 = zeros(size(f,1),size(f,2));

global min_pixel_dist_to_manifold_squared;

min_pixel_dist_to_manifold_squared = inf(size(f,1),size(f,2));

% If the tree_height was not specified, compute it using the equation of our paper.
if ~exist('tree_height','var') || isempty(tree_height)
    tree_height = compute_manifold_tree_height(sigma_s, sigma_r);
end

% If no joint signal was specified, use the original signal
if ~exist('f_joint','var')
    f_joint = f;
else
    f_joint = im2double(f_joint);
end

% By default we use only one iteration to compute the eigenvector v1 (Appendix B)
if ~exist('num_pca_iterations','var')
    num_pca_iterations = 1;
end

% Display a progress bar
global waitbar_handle;
global tree_nodes_visited;
tree_nodes_visited = 0;
waitbar_handle = waitbar(0, ['Filtering with ' num2str(2^tree_height - 1) ...
                             ' Adaptive Manifolds in ' num2str(2 + size(f_joint,3)) '-D Space...']);

% Algorithm 1, Step 1: compute the first manifold by low-pass filtering.
eta_1     = h_filter(f_joint, sigma_s);
cluster_1 = true(size(f,1),size(f,2));

current_tree_level = 1;

build_manifolds_and_perform_filtering(...
       f ...
     , f_joint ...
     , eta_1 ...
     , cluster_1 ...
     , sigma_s ...
     , sigma_r ...
     , current_tree_level ...
     , tree_height ...
     , num_pca_iterations ...
 );

% Compute the filter response by normalized convolution
tilde_g = bsxfun(@rdivide, sum_w_ki_Psi_blur, sum_w_ki_Psi_blur_0);

% Adjust the filter response for outlier pixels
alpha = exp( -0.5 .* min_pixel_dist_to_manifold_squared ./ sigma_r ./ sigma_r );
g     = f + bsxfun(@times, alpha, tilde_g - f);

% Close progressbar
delete(waitbar_handle);

end

function build_manifolds_and_perform_filtering(...
       f ...
     , f_joint ...
     , eta_k ...
     , cluster_k ...
     , sigma_s ...
     , sigma_r ...
     , current_tree_level ...
     , tree_height ...
     , num_pca_iterations ...
)

    % Dividing the covariance matrix by 2 is equivalent to dividing
    % the standard deviations by sqrt(2).
    sigma_r_over_sqrt_2 = sigma_r / sqrt(2);

    %% Compute downsampling factor
    floor_to_power_of_two = @(r) 2^floor(log2(r));
    df = min(sigma_s / 4, 256 * sigma_r);
    df = floor_to_power_of_two(df);
    df = max(1, df);

    [h_image w_image dR_image] = size(f);
    [h_joint w_joint dR_joint] = size(f_joint);

    downsample = @(x) imresize(x, 1/df, 'bilinear');
    upsample   = @(x) imresize(x, [h_image w_image], 'bilinear');

    %% Splatting: project the pixel values onto the current manifold eta_k

    phi = @(x_squared, sigma) exp( -0.5 .* x_squared ./ sigma / sigma );

    if size(eta_k,1) == size(f_joint,1)
        X = f_joint - eta_k;
        eta_k = downsample(eta_k);
    else
        X = f_joint - upsample(eta_k);
    end

    % Project pixel colors onto the manifold
    pixel_dist_to_manifold_squared = sum( X.^2, 3 );
    gaussian_distance_weights      = phi(pixel_dist_to_manifold_squared, sigma_r_over_sqrt_2);
    Psi_splat                      = bsxfun(@times, gaussian_distance_weights, f);
    Psi_splat_0                    = gaussian_distance_weights;

    % Save min distance to later perform adjustment of outliers
    global min_pixel_dist_to_manifold_squared;
    min_pixel_dist_to_manifold_squared = min(min_pixel_dist_to_manifold_squared,...
                                             pixel_dist_to_manifold_squared);

    %% Blurring: perform filtering over the current manifold eta_k

    blurred_projected_values = RF_filter(...
          downsample(cat(3, Psi_splat, Psi_splat_0)) ...
        , eta_k ...
        , sigma_s / df ...
        , sigma_r_over_sqrt_2 ...
    );

    w_ki_Psi_blur   = blurred_projected_values(:,:,1:end-1);
    w_ki_Psi_blur_0 = blurred_projected_values(:,:,end);

    %% Slicing: gather blurred values from the manifold

    global sum_w_ki_Psi_blur;
    global sum_w_ki_Psi_blur_0;

    % Since we perform splatting and slicing at the same points over the manifolds,
    % the interpolation weights are equal to the gaussian weights used for splatting.
    w_ki = gaussian_distance_weights;

    sum_w_ki_Psi_blur   = sum_w_ki_Psi_blur   + bsxfun(@times, w_ki, upsample(w_ki_Psi_blur  ));
    sum_w_ki_Psi_blur_0 = sum_w_ki_Psi_blur_0 + bsxfun(@times, w_ki, upsample(w_ki_Psi_blur_0));

    %% Compute two new manifolds eta_minus and eta_plus

    % Update progressbar
    global waitbar_handle;
    global tree_nodes_visited;
    tree_nodes_visited = tree_nodes_visited + 1;
    waitbar(tree_nodes_visited / (2^tree_height - 1), waitbar_handle);

    % Test stopping criterion
    if current_tree_level < tree_height

        % Algorithm 1, Step 2: compute the eigenvector v1
        X  = reshape(X, [h_joint*w_joint dR_joint]);
        rand_vec = rand(1,size(X,2)) - 0.5;
        v1 = compute_eigenvector(X(cluster_k(:),:), num_pca_iterations, rand_vec);

        % Algorithm 1, Step 3: Segment pixels into two clusters
        dot = reshape(X * v1', [h_joint w_joint]);
        cluster_minus = logical((dot <  0) & cluster_k);
        cluster_plus  = logical((dot >= 0) & cluster_k);

        % Algorithm 1, Step 4: Compute new manifolds by weighted low-pass filtering
        theta = 1 - w_ki;

        eta_minus = bsxfun(@rdivide ...
            , h_filter(downsample(bsxfun(@times, cluster_minus .* theta, f_joint)), sigma_s / df) ...
            , h_filter(downsample(               cluster_minus .* theta          ), sigma_s / df));

        eta_plus = bsxfun(@rdivide ...
            , h_filter(downsample(bsxfun(@times, cluster_plus .* theta, f_joint)), sigma_s / df) ...
            , h_filter(downsample(               cluster_plus .* theta          ), sigma_s / df));

        % Only keep required data in memory before recursing
        keep f f_joint eta_minus eta_plus cluster_minus cluster_plus sigma_s sigma_r ...
             current_tree_level tree_height num_pca_iterations

        % Algorithm 1, Step 5: recursively build more manifolds.
        build_manifolds_and_perform_filtering(f, f_joint, eta_minus, cluster_minus, sigma_s, ...
                                              sigma_r, current_tree_level + 1, tree_height, ...
                                              num_pca_iterations);

        keep f f_joint eta_plus cluster_plus sigma_s sigma_r current_tree_level ...
             tree_height num_pca_iterations

        build_manifolds_and_perform_filtering(f, f_joint, eta_plus, cluster_plus,  sigma_s, ...
                                              sigma_r, current_tree_level + 1, tree_height, ...
                                              num_pca_iterations);
    end

end

% This function implements a O(dR N) algorithm to compute the eigenvector v1
% used for segmentation. See Appendix B.
function p = compute_eigenvector(X, num_pca_iterations, rand_vec)

p = rand_vec;

for i = 1:num_pca_iterations

    dots = sum( bsxfun(@times, p, X), 2 );
    t = bsxfun(@times, dots, X);
    t = sum(t, 1);
    p = t;

end

p = p / norm(p);

end
```

B.2 Auxiliary Functions

B.2.1 `compute_manifold_tree_height`

This code computes the manifold tree height for RGB color image filtering.

```
function [Height K] = compute_manifold_tree_height(sigma_s, sigma_r)

Hs     = floor(log2(sigma_s)) - 1;
Lr     = 1 - sigma_r;

Height = max(2, ceil(Hs .* Lr));
K      = 2.^Height - 1;

end
```

B.2.2 `h_filter`

This is the low-pass filter 'h' we use for generating the adaptive manifolds.

```
function g = h_filter(f, sigma)

[h w num_channels] = size(f);

g = f;
g = h_filter_horizontal(g, sigma);
g = image_transpose(g);
g = h_filter_horizontal(g, sigma);
g = image_transpose(g);

end

function g = h_filter_horizontal(f, sigma)

a = exp(-sqrt(2) / sigma);

g = f;
[h w nc] = size(f);

for i = 2:w
    for c = 1:nc
        g(:,i,c) = g(:,i,c) + a .* ( g(:,i - 1,c) - g(:,i,c) );
    end
end

for i = w-1:-1:1
    for c = 1:nc
        g(:,i,c) = g(:,i,c) + a .* ( g(:,i + 1,c) - g(:,i,c) );
    end
end

end

function T = image_transpose(I)

[h w num_channels] = size(I);

T = zeros([w h num_channels], class(I));

for c = 1:num_channels
        T(:,:,c) = I(:,:,c)';
end

end
```

B.2.3 `RF_filter`

This is the implementation of the Domain Transform Recursive Filter of Chapter 3 modified to use an ℓ_2-norm when blurring over the adaptive manifolds. See Section 3.2.4.

```matlab
%  This code is part of the reference implementation of the adaptive-manifold
%  high-dimensional filter described in the paper:
%
%    Adaptive Manifolds for Real-Time High-Dimensional Filtering
%    Eduardo S. L. Gastal  and  Manuel M. Oliveira
%    ACM Transactions on Graphics. Volume 31 (2012), Number 4.
%    Proceedings of SIGGRAPH 2012, Article 33.
%
%  Please refer to the publication above if you use this software. For an
%  up-to-date version go to:
%
%             http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds/
%
%
%  THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESSED OR IMPLIED WARRANTIES
%  OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
%  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
%  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
%  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
%  OUT OF OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN
%  THIS SOFTWARE.
%
%  Version 1.0 - January 2012.

function F = RF_filter(img, joint_image, sigma_s, sigma_r)

I = double(img);

if exist('joint_image', 'var') && ~isempty(joint_image)
    J = double(joint_image);

        if (size(I,1) ~= size(J,1)) || (size(I,2) ~= size(J,2))
                error('Input and joint images must have equal width and height.');
        end
else
    J = I;
end

[h w num_joint_channels] = size(J);

dIcdx = diff(J, 1, 2);
dIcdy = diff(J, 1, 1);

dIdx = zeros(h,w);
dIdy = zeros(h,w);

for c = 1:num_joint_channels
    dIdx(:,2:end) = dIdx(:,2:end) + ( dIcdx(:,:,c) ).^2;
    dIdy(2:end,:) = dIdy(2:end,:) + ( dIcdy(:,:,c) ).^2;
end

sigma_H = sigma_s;

dHdx = sqrt((sigma_H/sigma_s).^2 + (sigma_H/sigma_r).^2 * dIdx);
dVdy = sqrt((sigma_H/sigma_s).^2 + (sigma_H/sigma_r).^2 * dIdy);

dVdy = dVdy';

N = 1;
F = I;

for i = 0:N - 1

    sigma_H_i = sigma_H * sqrt(3) * 2^(N - (i + 1)) / sqrt(4^N - 1);

    F = TransformedDomainRecursiveFilter_Horizontal(F, dHdx, sigma_H_i);
    F = image_transpose(F);

    F = TransformedDomainRecursiveFilter_Horizontal(F, dVdy, sigma_H_i);
    F = image_transpose(F);

end
```

```
F = cast(F, class(img));

end


function F = TransformedDomainRecursiveFilter_Horizontal(I, D, sigma)

a = exp(-sqrt(2) / sigma);

F = I;
V = a.^D;

[h w num_channels] = size(I);

for i = 2:w
    for c = 1:num_channels
        F(:,i,c) = F(:,i,c) + V(:,i) .* ( F(:,i - 1,c) - F(:,i,c) );
    end
end

for i = w-1:-1:1
    for c = 1:num_channels
        F(:,i,c) = F(:,i,c) + V(:,i+1) .* ( F(:,i + 1,c) - F(:,i,c) );
    end
end

end

function T = image_transpose(I)

[h w num_channels] = size(I);

T = zeros([w h num_channels], class(I));

for c = 1:num_channels
        T(:,:,c) = I(:,:,c)';
end

end
```

APPENDIX C NON-UNIFORM FILTERING SOURCE CODE

This source code implements the core of our discrete-time formulation for applying arbitrary recursive digital filters to non-uniformly sampled signals. The language is julia (BEZANSON et al., 2012). Please refer to our webpage at http://inf.ufrgs.br/~eslgastal/NonUniformFiltering for full source, examples, and instructions on using this code.

```
# This function implements our formulation which applies a 1st-order digital
# filter to a non-uniformly sampled signal with normalization preserved by
# piecewise resampling.
#
# The applied digital filter depends on the Direction parameter as shown
# on the table below:
#
#       Causal           |  AnticausalInSeries   |  AnticausalInParallel
#                        |                       |
#            a           |           a           |           a*b*z
# H(z) = ----------      |  H(z) = ---------     |   H(z) = ---------
#        1 - b*z^-1      |         1 - b*z       |          1 - b*z
#                        |                       |
#
ours_1storder!{
    TData <: Real,
    TCoef <: Number,
    Direction <: FilteringDirection,
    BoundaryCond <: BoundaryCondition
}(
    # Outputs
    odata :: Ptr{TData}, # Output samples. The result is **added** to the odata
                         # array, so it must be zeroed before calling this
                         # function.
    # Inputs
    idata :: Ptr{TData}, # Input samples
    dt    :: Ptr{TData}, # Sampling position deltas
    a     :: TCoef, # Filter numerator coefficient
    b     :: TCoef, # Filter pole, must satisfy abs(b) < 1
    N     :: Int64, # Number of input samples
          :: Type{Direction}, # One of: Causal, AnticausalInSeries, AnticausalInParallel
          :: Type{BoundaryCond} # One of: Relaxed, Replicated
) = @inbounds begin

    # Some precomputed values from
    const one_over_r0 = (one(TCoef) / ( (b - one(TCoef))*(b - one(TCoef)) / (a*b) )) :: TCoef
    const r1 = (a / (b - one(TCoef))) :: TCoef

    # Compute initial conditions
    if Direction === Causal || Direction === AnticausalInSeries
        const beta = (a / (one(TCoef) - b)) :: TCoef;
    else
        const beta = (b * a / (one(TCoef) - b)) :: TCoef;
    end

    if BoundaryCond === Relaxed
        f_k_minus_1 = zero(TData) :: TData
        g_k_minus_1 = zero(TCoef) :: TCoef
    else
        f_k_minus_1 = idata[Direction === Causal ? 1 : N] :: TData
        g_k_minus_1 = beta * convert(TCoef, f_k_minus_1) :: TCoef
    end

    if Direction === Causal
        datarange = 1:1:N
    else
        datarange = N:-1:1
    end

    # Main filtering loop
    for i = datarange
        if Direction === Causal
            const dt_k = dt[i] :: TData
        else
            if i+1 <= N # Avoid reading out-of-bounds value
                const dt_k = dt[i+1] :: TData
            else
                const dt_k = one(TData) :: TData
            end
        end

        const b_exp_dt = (b ^ dt_k) :: TCoef
        const f_k = idata[i] :: TData
        # The following two lines implement
        const Q_k = (b_exp_dt - one(TCoef)) * one_over_r0 * (one(TCoef) / dt_k) :: TCoef
        const P_k = ((Q_k - r1*b)*f_k - (Q_k - r1*b_exp_dt)*f_k_minus_1) :: TCoef

        if Direction === Causal || Direction === AnticausalInSeries
            const g_k = a*f_k + b_exp_dt*g_k_minus_1 + P_k :: TCoef
        else
            const g_k = a*b_exp_dt*f_k_minus_1 + b_exp_dt*g_k_minus_1 + P_k :: TCoef
        end

        # Drop the imaginary part and write output sample
        odata[i] += real(g_k) :: TData

        f_k_minus_1 = f_k :: TData
        g_k_minus_1 = g_k :: TCoef
    end
end
```