An Optimal Nonorthogonal Separation of the Anisotropic Gaussian Convolution Filter

Christoph H. Lampert and Oliver Wirjadi, *Student Member, IEEE*

***Abstract*—We give an analytical and geometrical treatment of what it means to separate a Gaussian kernel along arbitrary axes in $\mathbb{R}^n$, and we present a separation scheme that allows us to efficiently implement anisotropic Gaussian convolution filters for data of arbitrary dimensionality. Based on our previous analysis we show that this scheme is optimal with regard to the number of memory accesses and interpolation operations needed. The proposed method relies on nonorthogonal convolution axes and works completely in image space. Thus, it avoids the need for a fast Fourier transform (FFT)-subroutine. Depending on the accuracy and speed requirements, different interpolation schemes and methods to implement the one-dimensional Gaussian (finite impulse response and infinite impulse response) can be integrated. Special emphasis is put on analyzing the performance and accuracy of the new method. In particular, we show that without any special optimization of the source code, it can perform anisotropic Gaussian filtering faster than methods relying on the FFT.**

***Index Terms*—Convolution, feature extraction, filtering, multidimensional digital filters.**

I. Introduction

A. Anisotropic Gaussian Filtering

GAUSSIAN convolution filters are frequently used tools in 1-D and multidimensional signal processing. The exponential decay of their kernels in the signal domain, as well as in the frequency domain, and their strict positivity make them very well behaved low-pass filters. 1-D and isotropic multidimensional Gaussians are completely described by a single variance parameter. This makes them easy to handle analytically and simple and fast to implement. But in multidimensional applications, like image processing, Gaussian filters can have different shapes and more free parameters.

From the signal and image processing point of view, anisotropic Gaussians are much more interesting, because these additional parameters encode information about orientation as well as scale. During the last years a tendency towards anisotropic filtering has become visible in many areas. Often, one can rely on local image information, like gradient directions, to build anisotropic filters, allowing smoothing and de-noising while preserving edges [1], [2]. Wang *et al.* propose the use of local covariance estimates to build anisotropic kernels for adaptive video segmentation based on the mean shift procedure [3]. This relies on the fact that from a probabilistic view anisotropic Gaussians can express correlations between different signal directions, while isotropic Gaussians are based on the often unrealistic assumption that all quantities involved are statistically independent. Sometimes, local orientation information can be deduced from an imaging system: diffusion tensor MRI (DT-MRI) computes a tensor describing water diffusion in each voxel. This diffusion process correlates with the physical structure of the imaged object [4]. The *diffusion tensor* contains all necessary information for applying anisotropic Gaussian filters of the type investigated in this report. But even if local orientation is not known *a priori*, anisotropic filters can often be useful. By arranging many such filters with different orientations and variances in the form of a filter bank, local orientations of line and surface structures can be deduced—see [5]–[8]. This approach is often referred to as *orientation space theory* [9], in analogy to the *scale space theory* of isotropic Gaussians [10]. Detecting and enhancing structure using orientation spaces is of particular interest in 3-D imaging, e.g., medical imaging, where no established feature extraction method like the classical 2-D Canny edge detector is available.

Anisotropic filtering is also applied in other areas, including hardware design and visualization. All modern graphics processing units (GPUs) contain dedicated units for anisotropic filtering, allowing visualization effects like adaptive motion blur to be applied in real time [11].

We believe that if today anisotropic Gaussian filters have not become a standard tool in image and signal processing in the same way that isotropic ones have, then this is because there is no established way to efficiently implement them. It is well known that isotropic Gaussian filters can always be separated, i.e., their n-dimensional convolution integral can be implemented as a sequence of n 1-D convolutions along the coordinate axes. For general anisotropic Gaussians, this is not possible. Although they can be separated along their main axes, decomposing the n-dimensional integral into n 1-D ones, the directions of integration then are rotated with respect to the coordinate grid, making this form of implementation cumbersome and slow. Instead, it is in most cases faster to utilize the

Manuscript received September 22, 2005; revised January 26, 2006. The work of C. H. Lampert was supported in part by the German Federal Ministry of Education and Research (BMBF) under Project IPeT (01 IW D03) and in part by the Stiftung Rheinland-Pfalz für Innovation under project BIVaD (15202-386261/737). The work of O. Wirjadi was supported by the Rheinland-Pfalz cluster of excellence "Dependable adaptive systems and mathematical modeling." The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Giovanni Ramponi.

C. H. Lampert is with the Image Understanding and Pattern Recognition Group at the German Research Center for Artificial Intelligence (DFKI) GmbH, 67663 Kaiserslautern, Germany (e-mail: chl@iupr.net).

O. Wirjadi is with the Models and Algorithms in Image Processing Group, Fraunhofer ITWM, 67663 Kaiserslautern, Germany (e-mail: oliver.wirjadi@itwm.fraunhofer.de).

Color versions of Figs. 4, 5, 6, and 9 are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2006.877501

Fourier Convolution Theorem to do the filtering: the signal is Fourier transformed, multiplied with the transformed convolution kernel, and transformed back using the inverse Fourier transform. This requires more memory and computational effort than in the isotropic case and usually makes the inclusion of a mathematical library providing the fast Fourier transform (FFT) necessary. Further, since the FFT is a global operation, the whole image is processed at once and local filtering of just a small window of the image is not as easy as when working directly with the image data. Other drawbacks are that the FFT cannot cope with missing sample data and requires a CPU capable of performing floating-point operations.

Freeman and Adelson showed that it is not possible to construct a steerable basis for the rotated Gaussians [12], and even for an approximate solution as found by Perona in 1992 [13], the basis turned out to be too large for practical use. Also, the basis functions are not separable themselves and thus cannot be implemented in an efficient way.

As late as the year 2002, Geusebroek and Smeulders presented an efficient scheme to separate anisotropic Gaussians in $\mathbb{R}^2$ [14], [15]: their idea was to keep one direction of convolution fixed (and usually axis-aligned), but to allow the other directions to vary depending on the parameters of the Gaussian. The method does not yield orthogonal filtering directions, but allows for an efficient implementation. Using the recursive approximation scheme for 1-D-Gaussian convolutions, as proposed by Young [16], the result is an algorithm that is faster than the Fourier-based one, achieving $O(1)$ complexity per pixel.

Based on this approach, in 2005, Wirjadi and Breuel gave an approximate formula in a special case in $\mathbb{R}^3$ [17]. They fix two directions to be axis aligned, while the third varies with the Gaussian parameters. This again allows for a fast implementation, but the approximation is not good enough to be used in practice for many filter directions.

B. The New Contribution

The goal of this paper is to fill the gap that still exists for dimensions larger than 2. We develop an intuitive, geometrically motivated theory of what it means to separate a Gaussian kernel along arbitrary axes in $\mathbb{R}^n$. From these considerations, we derive a separation scheme for anisotropic Gaussians which is optimal in terms of the number of memory accesses and interpolation operations required.

To allow its application in real-life problems, we concretize the result for the most important cases in image processing: in $\mathbb{R}^2$, the result turns out to be identical to the separation by Geusebroek and Smeulders, showing the optimality of their result in this case. For Gaussians in $\mathbb{R}^3$ with two identical covariance values, we give explicit formulas, parameterized by the variances and major axes of the Gaussians. In the general case, a simple numerical method based on the Cholesky decomposition is derived.

To further demonstrate the proposed method, we have implemented it in plain C and have applied it to a number of typical situations in image processing of 2-D and 3-D data. This makes it possible to present visual output as well as thorough results on accuracy and efficiency.

The rest of the paper is structured as follows:

Section II studies the general form of how to separate the Gaussian convolution kernel. Section III presents two examples of such separations, the classical one based on the singular value decomposition and the one favored in this paper, relying on a triangular factorization of Cholesky type. In Section IV, the latter is calculated explicitly in $\mathbb{R}^2$ and the most frequently needed cases of $\mathbb{R}^3$. In $\mathbb{R}^n$, a numerical way to calculate the separation coefficients is explained. Also, a geometrical interpretation of the separations is given, which sheds light on how and why the algorithm works the way it does. Section II deals with the issues of implementing the theoretical results, in particular the question of how to discretize and interpolate the continuous operators. In Sections VI and VII, runtime and accuracy of the proposed filter setup are examined, comparing them to the FFT-based approach. Section VIII presents some applications of the algorithm to real life problems. Finally, Section IX summarizes the results achieved and discusses current limitations and possible extensions and improvements of the method.

II. Gaussian Convolution Integral

A. Factorization of the Gaussian

The steps to factorize an isotropic Gaussian along the coordinate axes are classical. In some areas, e.g., statistical pattern recognition, the separation of possibly anisotropic Gaussians along their main orthogonal axes is a standard procedure, usually in the setup of the "whitening transform," see e.g., [18, Ch. 2]. In this section, however, we will study the more general question of if and how a Gaussian convolution filter can be separated along arbitrary, possibly nonorthogonal axes in $\mathbb{R}^n$. The main result is the following.

> *For any decomposition $\Sigma = VDV^t$ of the covariance matrix Σ into square matrices D and V, where D is diagonal and positive and V has determinant 1, there is a separation of the n-dimensional Gaussian into 1-D Gaussians, where the separation directions are given by the column vectors of V.*

In Section III, we will see that many such decompositions exist, with or without orthogonal directions, and we will study their properties with regard to an efficient implementation of the Gaussian filter.

To prove the result itself, we first fix some notations. Unless specified otherwise, all calculations will take place in $\mathbb{R}^n$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ are n-dimensional coordinate vectors with respect to the standard Euclidean coordinate system. By $\mathrm{Fun}(X, Y)$, we denote the space of functions from some set X to some set Y. The anisotropic Gaussian filter kernel $g(\mathbf{x}) \in \mathrm{Fun}(\mathbb{R}^n, \mathbb{R})$ with mean 0 and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ then has the form

$$g(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}\mathbf{x}^t\Sigma^{-1}\mathbf{x}\right\} \tag{1}$$

where $|\Sigma|$ is the determinant of Σ. The goal is now to find a separation, i.e., to write g as a product of 1-D Gaussians. Assume for the moment that a decomposition $\Sigma = VDV^t$ as described

above is given. Then $\Sigma^{-1} = V^{-t}D^{-1}V^{-1}$, and we can rewrite the Gaussian to

$$= \frac{\exp\left\{-\frac{1}{2}\mathbf{x}^t(V^{-t}D^{-1}V^{-1})\mathbf{x}\right\}}{(2\pi)^{n/2}|D|^{1/2}} \tag{2}$$

where we also use the fact that $|D| = |\Sigma|$, which follows from $|V| = 1$. Rewriting the numerator yields

$$= \frac{\exp\left\{-\frac{1}{2}(V^{-1}\mathbf{x})^tD^{-1}(V^{-1}\mathbf{x})\right\}}{(2\pi)^{n/2}|D|^{1/2}}. \tag{3}$$

After a linear change of coordinates from $\mathbf{x}$ to $\mathbf{v} = (v_1, \ldots, v_n)$ with $\mathbf{v} := V^{-1}\mathbf{x}$, this becomes

$$= \frac{1}{(2\pi)^{n/2}|D|^{1/2}} \exp\left\{-\frac{1}{2}\mathbf{v}^tD^{-1}\mathbf{v}\right\}. \tag{4}$$

D is known to be a diagonal matrix with positive entries, which we will denote by $d_1^2, \ldots, d_n^2$, such that $|D|^{1/2} = d_1 \cdot \ldots \cdot d_n$. It follows that D^{-1} is diagonal with positive entries as well, namely $D^{-1} = \operatorname{diag}\left(\left(1/d_1^2\right), \ldots, \left(1/d_n^2\right)\right)$. Therefore, the matrix product is in fact just a weighted sum of squares

$$= \frac{1}{(2\pi)^{n/2}d_1 \cdot \ldots \cdot d_n} \exp\left\{-\frac{1}{2}\sum_{i=1}^{n}\frac{v_i^2}{d_i^2}\right\} \tag{5}$$

which we can split up using the addition theorem for the exponential function

$$= \frac{1}{\sqrt{2\pi}d_1} \exp\left(-\frac{1}{2}\frac{v_1^2}{d_1^2}\right) \cdot \ldots \cdot \frac{1}{\sqrt{2\pi}d_n} \exp\left(-\frac{1}{2}\frac{v_n^2}{d_n^2}\right). \tag{6}$$

From this, we see that using the v-coordinate vector gives us the desired product structure

$$= g_1(v_1) \cdot \ldots \cdot g_n(v_n) \tag{7}$$

where each

$$g_i(v_i) := \frac{1}{\sqrt{2\pi}d_i} \exp\left(-\frac{1}{2}\frac{v_i^2}{d_i^2}\right) \tag{8}$$

is an ordinary 1-D Gaussian of mean 0 and variance d_i^2.

B. Separating the Convolution Integral

To see how this factorization of the Gaussian kernel gives rise to a separation of the Gaussian filter, we study what happens to a convolution integral when applying the aforementioned change of coordinates.

For a general function $f \in \operatorname{Fun}(\mathbb{R}^n, \mathbb{R})$, the convolution with the Gaussian $g \in \operatorname{Fun}(\mathbb{R}^n, \mathbb{R})$ is defined as

$$(f * g)(\mathbf{x}) := \int_{\mathbb{R}}\int_{\mathbb{R}} \cdots \int_{\mathbb{R}} f(\mathbf{y})g(\mathbf{x}-\mathbf{y})dy_1dy_2 \ldots dy_n. \tag{9}$$

Changing the coordinates as before, from $\mathbf{x}$ to $\mathbf{u} := V^{-1}\mathbf{x}$ and from $\mathbf{y}$ to $\mathbf{v} := V^{-1}\mathbf{y}$, we obtain

$$= \int_{\mathbb{R}} \cdots \int_{\mathbb{R}}\int_{\mathbb{R}} f(V\mathbf{v})g\left(V\mathbf{u} - V\mathbf{v}\right)|V^{-1}|dv_1dv_2 \ldots dv_n \tag{10}$$

where the factor $|V^{-1}|$ enters from the rules for coordinate changes in integration theory. Using the separation formula (7) and $|V| = 1$, we can rewrite this as

$$= \int_{\mathbb{R}} g_n(u_n - v_n) \int_{\mathbb{R}} g_{n-1}(u_{n-1} - v_{n-1}) \ldots \int_{\mathbb{R}} g_1(u_1 - v_1)f(V\mathbf{v})dv_1dv_2 \ldots dv_n. \tag{11}$$

This is the standard form of a separated convolution integral. To better understand what the formula means in practice, we additionally split up the matrix-vector product $V\mathbf{v} = \sum_i v_i\mathbf{v}^i$, such that

$$= \int_{\mathbb{R}} g_n(u_n - v_n) \int_{\mathbb{R}} g_{n-1}(u_{n-1} - v_{n-1}) \cdots \int_{\mathbb{R}} g_1(u_1 - v_1)f\left(\sum_i v_i\mathbf{v}^i\right)dv_1dv_2 \ldots dv_n \tag{12}$$

where the $\mathbf{v}^i$ are the columns of the matrix V. Integration over v_i therefore means convolution along the direction $\mathbf{v}^i$. This can be written very compactly using the notation of *directional convolutions*

$$= g_n *_{\mathbf{v}^n} \ldots g_2 *_{\mathbf{v}^2} g_1 *_{\mathbf{v}^1} f \tag{13}$$

where the *directional convolution operator* $*_{\mathbf{v}} : \operatorname{Fun}(\mathbb{R}, \mathbb{R}) \times \operatorname{Fun}(\mathbb{R}^n, \mathbb{R}) \to \operatorname{Fun}(\mathbb{R}^n, \mathbb{R})$ is defined by

$$(g *_{\mathbf{v}} f)(\mathbf{x}) := \int_{-\infty}^{\infty} g(\lambda)f(\mathbf{x} - \lambda\mathbf{v})d\lambda. \tag{14}$$

Note that the integration is 1-D, even though the direction is specified by a vector $\mathbf{v} \in \mathbb{R}^n$.

III. Symmetric Factorizations of Σ

The result of the previous section was that any anisotropic Gaussian is separable if we find a decomposition of the covariance matrix of a form $\Sigma = VDV^t$. We call this a *symmetric factorization*. For symmetric and positive definite matrices Σ—as covariance matrices always are—these can be constructed using elementary matrix operations, see [19, Ch.1]. In fact, there are even many of those with different properties, two of which we will review here: the *Singular Value Decomposition (SVD)* and the *triangular factorization of Cholesky type*. The result will be that the latter is more suitable for implementing the Gaussian filter in a discrete setting.

A. SVD

In its general form, the SVD of a symmetric matrix A is $A = VDV^t$, where V is a rotation matrix and D is diagonal, thus qualifying the SVD as a symmetric factorization. Because V is a rotation matrix, its column vectors are orthogonal to each other and each of length 1. They point in the directions of the axes of the hyper-ellipsoid described by $\{\mathbf{x} : \mathbf{x}^t\Sigma^{-1}\mathbf{x} = 1\}$. Using the previous result that we can separate the Gaussian along the columns of V, this reproves the well-known fact that an anisotropic Gaussian is always separable along its major axes.

However, this approach has major problems for practical purposes, because for a general Gaussian, all axes lie rotated in $\mathbb{R}^n$. In a discrete implementation of (12), $\mathbf{v}$ does not have integer entries and $\mathbf{x} - \lambda\mathbf{v}$ does not lie on the sampling grid. While this is no problem from the mathematical point of view, in practice we have to use interpolation in all n dimensions during each of the n integration steps. This makes the computations slow and numerical errors can occur and accumulate.

B. Triangular Factorization of Cholesky Type

To reduce the need for interpolation, we analyzed other decompositions to find a factorization $\Sigma = VDV^t$ that gives rise to an algorithm requiring fewer interpolation steps. First, we study for which choices of direction vector $\mathbf{v}$ the convolution operator $*_{\mathbf{v}}$ has a fast and numerically well-behaved implementation.

- The simplest case is when $\mathbf{v}$ is a standard unit vector of the Euclidean coordinate system, e.g., $\mathbf{v} = (1, 0, 0, \ldots, 0)$. This results in an axis parallel convolution and no interpolation is necessary.
- If $\mathbf{v}$ lies in a plane spanned by two Euclidean coordinate axes, e.g., $\mathbf{v} = (v_1, v_2, 0, \ldots, 0)$, then only a 2-D-interpolation step is necessary per sample. If further $v_1 = 1$ or $v_2 = 1$, interpolation is necessary only 1-dimensionally, because with respect to the other directions the target locations lie on the sampling grid.
- For a general vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, the discrete convolution has to perform an n-dimensional interpolation for each sample point. But, if one of the v_i is 1, the interpolation dimension is at least reduced to $n - 1$.

The directions resulting from a general SVD fall under the worst case described above, each requiring n-dimensional interpolation. Also, none of their vector components can be of integer value, since the length of the direction vectors is known to be 1. Instead, we design a factorization with as many zero components in the direction vectors $\mathbf{v}^i$ as possible. Whenever an entry cannot be made 0, we at least try to make it 1.

Since Σ is a symmetric $n \times n$-matrix, it has $(n(n+1)/2)$ degrees of freedom. Any factorization into V and D will need to have at least as many degrees of freedom. n of those go into diagonal entries of D. Therefore, at least $(n(n-1)/2)$ entries of V will have to remain free. The $(n(n+1)/2)$ other entries of V we can make either 0 or 1, while adhering the condition that V must have determinant 1.

One decomposition that realizes this dimensional analysis bound is the triangular factorization of Cholesky type, see [20, Ch. I]. Starting from the Cholesky decomposition $\Sigma = UU^t$ with upper triangular U, we set $D := \operatorname{diag}(u_{11}^2, \ldots, u_{nn}^2)$, where the u_{ii} are the diagonal entries of U, and $V := D^{-1/2}U$. Then it follows that $\Sigma = VDV^t$ with V an *upper triangular matrix with unit diagonal*

$$V = \begin{pmatrix} 1 & v_{1,2} & v_{1,3} & \cdots & v_{1,n} \\ & 1 & v_{2,3} & \cdots & v_{2,n} \\ & & \ddots & & \vdots \\ & & & 1 & v_{n-1,n} \\ & & & & 1 \end{pmatrix}. \tag{15}$$

In the implementation of (12), this means that the convolution along $\mathbf{v}^1$ can be calculated without need for interpolation. Along $\mathbf{v}^2$, 1-D interpolation in x_1-direction is necessary, etc., until for $\mathbf{v}^n$, $(n-1)$-dimensional interpolation in $x_1, \ldots, x_{n-1}$ has to be performed for each sample. There is never any interpolation required in the x_n variable. Each of the convolution steps requires fewer interpolation steps than any one in the SVD-based separation scheme.

This choice of V in (15) is optimal for the conditions defined above because it contains exactly $(n(n-1)/2)$ free variables, its diagonal contains ones, thus making its determinant 1, and all remaining entries are 0.

C. Calculation of the Triangular Factorization

After having seen that the triangular factorization has the potential to give rise to an efficient separation of the anisotropic Gaussian, we examine how to compute it for a given Σ. In parallel to the most important applications, we study the cases of $n = 2$, $n = 3$, and arbitrary n separately.

1) Explicit Formulas in $\mathbb{R}^2$: The triangular factorization can easily be calculated in closed form by writing down the 2×2 matrices involved

$$V = \begin{pmatrix} 1 & v_{1,2} \\ 0 & 1 \end{pmatrix} \text{ and } D = \begin{pmatrix} d_1^2 & 0 \\ 0 & d_2^2 \end{pmatrix} \tag{16}$$

where $v_{1,2}$, d_1^2, and d_2^2 are the unknown parameters. From this, we obtain

$$VDV^t = \begin{pmatrix} d_1^2 + d_2^2 v_{1,2}^2 & d_2^2 v_{1,2} \\ d_2^2 v_{1,2} & d_2^2 \end{pmatrix}. \tag{17}$$

For any positive definite symmetric matrix

$$\Sigma = \begin{pmatrix} s_{1,1} & s_{1,2} \\ s_{1,2} & s_{2,2} \end{pmatrix} \tag{18}$$

we can rewrite the factorization equation $\Sigma = VDV^t$ component wise, and by solving for the unknowns, we obtain

$$v_{1,2} = \frac{s_{1,2}}{s_{2,2}}, \quad d_1^2 = s_{1,1} - \frac{s_{1,2}^2}{s_{2,2}}, \quad d_2^2 = s_{2,2}. \tag{19}$$

Because Σ is positive definite, it is ensured that $s_{2,2} > 0$ and $s_{1,1}s_{2,2} - s_{1,2}^2 > 0$. Therefore, all expressions are well defined. We see that

$$d_1^2 d_2^2 = s_{1,1}s_{2,2} - s_{1,2}^2 \tag{20}$$

which is the explicit formulation in $\mathbb{R}^2$ of the fact that $|D| = |\Sigma|$.

2) Explicit Formulas in $\mathbb{R}^3$: For 3×3 matrices, the triangular factorization can be calculated explicitly in the same way, but one has to deal with more unknowns which complicates the formulas. We write $\Sigma = (s_{i,j})_{i,j=1,2,3}$ with $s_{i,j} = s_{j,i}$, $D = \operatorname{diag}(d_1^2, d_2^2, d_3^2)$ and $V = (v_{i,j})_{i,j=1,2,3}$ with $v_{i,i} = 1$ and $v_{i,j} = 0$ for $i > j$. Now as before, solving the system of equations deriving from the individual components of $\Sigma = VDV^t$ yields

$$d_1^2 = s_{1,1} - \frac{\left(s_{1,2}s_{3,3} - s_{1,3}s_{2,3}\right)^2}{s_{3,3}\left(s_{2,2}s_{3,3} - s_{2,3}^2\right)} - \frac{s_{1,3}}{s_{3,3}} \tag{21}$$

$$d_2^2 = s_{2,2} - \frac{s_{2,3}^2}{s_{3,3}}, \quad d_3^2 = s_{3,3} \tag{22}$$

$$v_{1,2} = \frac{s_{1,2}s_{3,3} - s_{1,3}s_{2,3}}{s_{2,2}s_{3,3} - s_{2,3}^2}, \quad v_{1,3} = \frac{s_{1,3}}{s_{3,3}}$$

$$v_{2,3} = \frac{s_{1,2}}{s_{3,3}}. \tag{23}$$

Note that instead of using (21), it is often more convenient to utilize that $d_1^2 d_2^2 d_3^2 = |\Sigma|$.

3) Numerical Factorization: In higher dimensions, it is still possible but not practical to derive a closed formula for the factorization matrices depending on the entries of Σ. Instead, V and D can be efficiently calculated using numerical methods. Since this has to be done only once for the filter kernel and not for each pixel, the computational effort is negligible. Some numerical libraries provide a dedicated routine for calculating the triangular factorization; otherwise, it is possible to obtain it from the Cholesky decomposition as described in Section III-B

D. Geometrical Interpretation

To understand how the triangular factorization really works, it is useful to look at it from a geometrical point of view. Finding a separation of the Gaussian can be thought of in the following way: we perform a linear transformation (namely V^{-1}) on the signal, filter with Gaussians along the coordinate axes, and transform the signal back using the transform V. From this, it is seen that the directions of convolution in (12) are the V-transformations of the Euclidean coordinate axes.

This is better visualized geometrically in 2-D using ellipses. Ellipses are the contour lines of Gaussians, and each ellipse uniquely corresponds to a Gaussian kernel and vice versa, if we fix the contour ellipse to lie at half of the Gaussian's maximum value.

1) SVD: It is well known that each ellipse can be made axis-parallel by rotating it. This is how the SVD-based method works: the above mentioned linear transform is a rotation, mapping the ellipses' main axes onto the coordinate axes. After rotating back, this yields convolution directions which are orthogonal to each other and rotated with respect to the Euclidean frame. See Fig. 1(b) for an illustration.

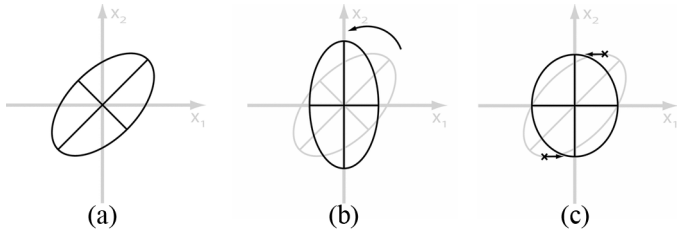

Fig. 1. (a) Any ellipse in the 2-D plane can be transformed into an axis-aligned ellipse by using either a (b) rotation or (c) shear.

2) triangular factorization: Each ellipse can also be made axis-parallel by a shear of the x_2-axis, keeping the x_1-axis fixed—see Fig. 1(c). This means that after transforming the signal back, one convolution direction still is the x_1-axis itself, while the other is a sheared version of the x_2-axis, intersecting the original ellipsis in its highest point.

IV. Anisotropic Gaussian Filtering in Image and Signal Processing

In practice, Σ is often not given explicitly, but implicitly by specifying a number of directions in which the Gaussian filtering should take place, and covariance values that give the filter strength. The covariance matrix is then obtained as $\Sigma = R^t S R$, where the direction vectors form the columns of R, and S is a diagonal matrix containing the variance values. In most cases, the directions are not given as vectors, but from a number of rotation angles. In this section, we give formulas for the parameters $\mathbf{v}^i$ and d_i^2, directly parameterized by such rotation angles.

Although we have derived the theory in arbitrary dimension, in this section, we restrict ourselves to 2-D and special cases of 3-D filtering. Other cases are of little practical interest in image processing and we believe that deriving a (complicated) parameterization for them would not yield any additional insight.

A. Parameterizations for Image Processing in 2-D

At first, we study Gaussians in $\mathbb{R}^2$ with coordinates (x_1, x_2). A Gaussian is then uniquely determined by an angle θ, specifying the major filtering direction, and the two variances σ_1^2, σ_2^2. Its covariance matrix then is $\Sigma = R_\theta^t S R_\theta$ with

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \quad S = \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}. \tag{24}$$

After multiplication, we obtain as a general parameterization

$$\Sigma = \begin{pmatrix} \sigma_1^2\cos^2\theta + \sigma_2^2\sin^2\theta & \left(\sigma_2^2 - \sigma_1^2\right)\cos\theta\sin\theta \\ \left(\sigma_2^2 - \sigma_1^2\right)\cos\theta\sin\theta & \sigma_1^2\sin^2\theta + \sigma_2^2\cos^2\theta \end{pmatrix}. \tag{25}$$

Using (19) leads to Σ's triangular factor matrices

$$V = \begin{pmatrix} 1 & \frac{(\sigma_2^2-\sigma_1^2)\cos\theta\sin\theta}{\sigma_1^2\sin^2\theta+\sigma_2^2\cos^2\theta} \\ 0 & 1 \end{pmatrix} \tag{26}$$

$$D = \begin{pmatrix} \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2\cos^2\theta+\sigma_2^2\sin^2\theta} & 0 \\ 0 & \sigma_1^2\cos^2\theta+\sigma_2^2\sin^2\theta. \end{pmatrix}. \tag{27}$$

Introducing φ to denote the angle between V's column vectors, i.e., $\varphi = \measuredangle(\mathbf{v}^1, \mathbf{v}^2)$ or

$$\tan\varphi = \frac{\sigma_1^2\sin^2\theta+\sigma_2^2\cos^2\theta}{(\sigma_2^2-\sigma_1^2)\cos\theta\sin\theta} \tag{28}$$

we have separated the Gaussian into a convolution along the x_1-axis and a convolution with direction vector $\mathbf{v}^2 = (\cot\varphi, 1)^t$. The corresponding Gaussian variances are

$$d_1^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2\cos^2\theta+\sigma_2^2\sin^2\theta} \tag{29}$$

$$d_2^2 = \sigma_1^2\cos^2\theta+\sigma_2^2\sin^2\theta. \tag{30}$$

This is exactly the result from [15], showing that the n-dimensional theory is in fact a generalization of the 2-D work by Geusebroek *et al.*

B. Parameterizations for Image Processing in 3-D

In $\mathbb{R}^3$ with coordinates (x_1, x_2, x_3), several ways of specifying the Gaussian from angles are possible. Typically, the construction itself can be thought of as starting with an axis aligned anisotropic Gaussian and rotating it in $\mathbb{R}^3$ to its target orientation. Our method of choice is based on Euler angles—see [21]. The total rotation matrix is constructed from at most three elementary rotations about the x_1-axis, x_3-axis and again the x_1-axis, i.e.,

$$R(\psi,\theta,\varphi) = R_{x_1}(\psi)R_{x_3}(\theta)R_{x_1}(\varphi) \tag{31}$$

where R_{x_1} and R_{x_3} denote the usual rotation matrices about the x_1 and x_3 axes, and $\psi \in [0,\pi[, \theta \in [0,(\pi/2)]$ and $\varphi \in [0,\pi[$ are the corresponding rotation angles. It is now possible to follow the same track as in $\mathbb{R}^2$: calculate $\Sigma = R^tSR$ explicitly and solve the system of equations that result from $\Sigma = VDV^t$ by using (21)–(23).

However, the resulting formulas get very long and are not instructive. We will instead only treat a special case which is of most importance in 3-D image processing: when detecting lower dimensional structures in 3-D data, the anisotropic Gaussian is typically chosen with $\sigma_2 = \sigma_3$. Filtering with $\sigma_1 > \sigma_2$ then means to filter in form of a prolate rotational ellipsoid, which corresponds to objects that have mainly 1-D extent. Filtering with $\sigma_1 < \sigma_2$ turns the Gaussian into the shape of an oblate rotational ellipsoid, which is useful when searching for two-dimensional objects.

Choosing $\sigma_2 = \sigma_3$ has the additional advantage that the Gaussian is rotationally invariant to its first axis. With Euler angles, this means that the first rotation in x_1-direction can be dropped from (31), and Σ ends up depending only on four parameters: two rotation angles θ and φ and two variances σ_1 and σ_2—see (32), shown at the bottom of the page. After elementary but tedious computation, we obtain the parameterization

$$d_1^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_2^2\cos^2\theta+\sigma_1^2\sin^2\theta} \tag{33}$$

$$d_2^2 = \frac{\sigma_2^2\left(\sigma_2^2\cos^2\theta+\sigma_1^2\sin^2\theta\right)}{\sigma_2^2-(\sigma_2^2-\sigma_1^2)\sin^2\varphi\sin^2\theta} \tag{34}$$

$$d_3^2 = \sigma_2^2-\left(\sigma_2^2-\sigma_1^2\right)\sin^2\varphi\sin^2\theta \tag{35}$$

$$v_{1,2} = \frac{\left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\cos\theta\sin\theta}{\sigma_2^2-(\sigma_2^2-\sigma_1^2)\sin^2\theta} \tag{36}$$

$$v_{1,3} = \frac{-\left(\sigma_2^2-\sigma_1^2\right)\sin\varphi\cos\theta\sin\theta}{\sigma_2^2-(\sigma_2^2-\sigma_1^2)\sin^2\varphi\sin^2\theta} \tag{37}$$

$$v_{2,3} = \frac{\left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\cos\theta\sin\theta}{\sigma_2^2-(\sigma_2^2-\sigma_1^2)\sin^2\varphi\sin^2\theta}. \tag{38}$$

When checking the special cases $\sigma_1 = \sigma_2 = \sigma_3$ (isotropic Gaussian), or $\theta = k\pi$ with $k \in \mathbb{Z}$ (axis aligned Gaussian), one sees that the triangular factorization results in V being the identity matrix and $D = \mathrm{diag}(\sigma_1, \sigma_2, \sigma_2)$. Thus, it is not necessary to treat them as exceptions.

V. Discretization and Implementation

The results from the previous sections tell us how to factorize Gaussians and we have proposed a specific factorization in Section III-B, with properties which we expect to be useful for implementation. Before implementing the separated filter, the following issues need to be resolved.

A. Discrete Directional Convolution Operator

We need a discrete version of the theory. Therefore, we will from now on study discrete signals and filter masks. To clearly indicate this, we will use capital letters for those, instead of small letters in the continuous case, e.g., the Gaussian will be denoted by $G(\mathbf{x})$ instead of $g(\mathbf{x})$.

$$\Sigma = \begin{pmatrix} \sigma_1^2+\left(\sigma_2^2-\sigma_1^2\right)\sin^2\theta & \left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\cos\theta\sin\theta & -\left(\sigma_2^2-\sigma_1^2\right)\sin\varphi\cos\theta\sin\theta \\ \left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\cos\theta\sin\theta & \sigma_2^2-\left(\sigma_2^2-\sigma_1^2\right)\cos^2\varphi\sin^2\theta & \left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\sin\varphi\sin^2\theta \\ -\left(\sigma_2^2-\sigma_1^2\right)\sin\varphi\cos\theta\sin\theta & \left(\sigma_2^2-\sigma_1^2\right)\cos\varphi\sin\varphi\sin^2\theta & \sigma_2^2-\left(\sigma_2^2-\sigma_1^2\right)\sin^2\varphi\sin^2\theta \end{pmatrix} \tag{32}$$

For a discrete signal on a grid, the convolution integral (11) turns into a sum

$$(G_i *_{\mathbf{v}} F)(\mathbf{x}) = \sum_{k\in\mathbb{Z}} G_i(k)F(\mathbf{x} - k\mathbf{v}) \tag{39}$$

where the coefficients G_i are derived from the 1-D Gaussian by sampling at the grid points. With this, the discrete anisotropic Gaussian filtering (12) can be written as

$$(G * F) = G_1 *_{\mathbf{v}^1} G_2 *_{\mathbf{v}^2} \ldots G_n *_{\mathbf{v}^n} F. \tag{40}$$

B. Interpolation

In (39), the direction vector $\mathbf{v}$ does not need to contain only integer valued entries, so $\mathbf{x} - k\mathbf{v}$ does not, in general, lie on the sampling grid. To obtain a value for F at this position, interpolation from neighboring sample points becomes necessary, see, e.g., [22, Ch.9].

To explain the interpolation needed when using the triangular factorization from Section III-B, we study the case of $\mathbb{R}^3$. The general case can easily be extrapolated from this. The convolution is then given by

$$G * F = G_3 *_{\mathbf{v}^3} G_2 *_{\mathbf{v}^2} G_1 *_{\mathbf{v}^1} F. \tag{41}$$

Consider the first convolution step $F_1 := G_1 *_{\mathbf{v}^1} F$. Its direction is always given by $\mathbf{v}^1 = (1,0,0)^t$, so memory access is only necessary in unit steps along x_1. Thus, having compact memory blocks and no need for interpolation, computation of F_1 is fast and accurate.

Except when the Gaussian is axis-aligned, all subsequent directions will require interpolation orthogonal to one of the coordinate axes. We will analyze the second directional convolution, along $\mathbf{v}_2 = (v_{1,2}, 1, 0)^t$, more closely

$$\begin{aligned} G_2 *_{\mathbf{v}^2} F_1 &= \sum_{k\in\mathbb{Z}} G_2(k) F_1\left(\mathbf{x} - k\,(v_{1,2}, \mathbf{1}, 0)^t\right) \\ &= \sum_{k\in\mathbb{Z}} G_2(k) F_1(x_1 - kv_{1,2}, x_2 - k, x_3). \end{aligned} \tag{42}$$

This shows that we need to interpolate along direction x_1 only. Note that this is a direct result of the choice of factorization in Section III-B. Each following directional convolution operation will need one additional interpolation direction. In 3-D, we need at most two (orthogonal) interpolation directions: let x_1, x_2, x_3 denote integer valued coordinates and $0 \le \delta_1, \delta_2 < 1$ offsets from the voxel origins. Then, we have the following cases, as is also illustrated in Fig. 2.

(a) $F(x_1, x_2, x_3)$ lies on the grid, no interpolation necessary.
(b) $F(x_1 + \delta_1, x_2, x_3)$ needs interpolation in x_1-direction.
(c) $F(x_1 + \delta_1, x_2 + \delta_2, x_3)$ needs interpolation in the x_1, x_2-plane.

The interpolations can be calculated using nearest neighbor, linear/bilinear, or higher order schemes, see, e.g., [23, Ch. 8]. Performing the same analysis in $\mathbb{R}^n$, we find that up to $n-1$ interpolation directions are needed per step with interpolation in x_n never being necessary.

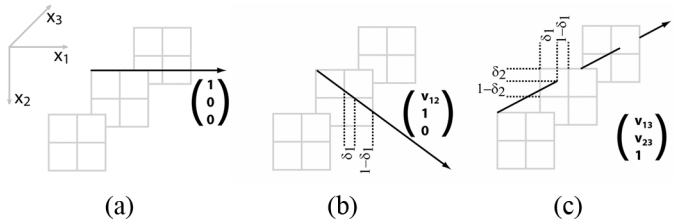

Fig. 2. In 3-D, implementing the proposed separation results in three directional convolution operations, requiring (a) no interpolation, (b) 1-D, or (c) 2-D interpolation.

C. Finite and Infinite Impulse Response Filtering

The proposed decomposition requires computation of n 1-D convolutions. Here we discuss different ways for performing these operations.

Apart from filtering in Fourier space by use of the convolution theorem, finite impulse response (FIR) filtering in image space is the usual way of implementing a convolution filter. There, the convolution sum in (39) is truncated

$$(G_i *_{\mathbf{v}} F)(\mathbf{x}) = \sum_{k=-K}^{K} G_i(k)F(\mathbf{x} - k\mathbf{v}) \tag{43}$$

where K is chosen in a way such that only terms remain in the sum where $G_i(k)$ differs significantly from 0, usually a constant multiple of the standard deviation of the Gaussian. When doing so, runtime increases with σ. This drawback can be resolved by using the infinite impulse response (IIR) implementation of Gaussian filters proposed in [16]. Therein, the authors derive a recursive filtering scheme based on a rational approximation of the Gaussian function.

The original recursive filter is defined by a pair of filters acting on a 1-D-signal. To support directional filtering, we modify these to perform a 1-D-convolution along a direction vector $\mathbf{v}$ within a higher dimensional grid. The resulting filter uses $\mathbf{v}$ as the unit step, analogously to (43). The directional convolution operator is then decomposed into a *forward* and *backward* filter pair

$$\begin{aligned} w(\mathbf{x}) &= BF(\mathbf{x}) + \frac{1}{b_0}\sum_{k=1}^{3} b_k w(\mathbf{x} - k\mathbf{v}) \\ (G *_{\mathbf{v}} F)(\mathbf{x}) &= Bw(\mathbf{x}) \\ &\quad + \frac{1}{b_0}\sum_{k=1}^{3} b_k (G *_{\mathbf{v}} F)(\mathbf{x} + k\mathbf{v}). \end{aligned} \tag{44}$$

See [16] and [24] for details on the filter coefficients b_0, b_1, b_2, b_3, and B and [25] for boundary conditions which need to be applied to correct distortions. Because only the values of the coefficients in (44) depend on σ, but their number does not, the runtime of this implementation of the Gaussian filter is independent of σ.

D. Implementation

There are numerous ways of implementing the discrete directional convolution operator from (39) The first and most obvious one is a direct implementation of the FIR filter in (43), and we will refer to it as the "naive" implementation: for each pixel or voxel in an image evaluate the convolution sum along a direction $\mathbf{v}$.

Another way of interpreting the directional convolution is to cast parallel rays in direction $\mathbf{v}$ through an image, which we call the "line buffer" implementation. Convolutions can then be computed along these lines, which is more efficient than evaluating the convolution at each image location separately. Also, the fast recursive IIR filter described in Section V-C can be applied.

Note that the convolution operator was defined at a fixed image location $\mathbf{x}$. For a general direction $\mathbf{v}$, the lines along which we want to calculate the convolutions will not lie on the sampling grid of a given image. To get an estimate of the resulting value at $\mathbf{x}$, we need to interpolate from lines around that location. In essence, this results in interpolations for both reading and writing data from the image. As above, this means that one has to perform up to $n-1$ dimensional interpolations, both for extracting values along lines and for writing values back to the image.

The third and last type of implementation that we discuss here is the "geometric" variant. We outlined in Section III-D that the matrix V in the proposed separation corresponds to a shear. Thus, the directional convolution of the whole image is readily implemented by performing a shear in the image using V^{-1}, convolving along the coordinate axes using the variance parameters derived in Section IV, and then performing the inverse shear using V.

Implementation of this is straight forward in 2-D. In the 3-D case, the following observation can lead to an efficient implementation: given the triangular matrix V with unit diagonal, the total operation can be separated into first a shear in x_1 and then a shear in x_2 direction. Shearing in x_3 is never necessary. This also implies that for this implementation, only 1-D interpolations will be required in both x_1 and x_2. The convolutions along the coordinate axes can again be calculated using the recursive IIR filter. Higher dimensional data can be treated similarly.

VI. Results

For showing the practical use of the proposed separation scheme, we performed several experiments on speed and accuracy for two and three-dimensional data. In this section, we describe the setup and explain the results of our experiments. To separate facts from opinion, our interpretation of the data will be presented in the subsequent Section VII.

For the tests, we implemented the directional convolution operator in C using the naive, line buffer and geometric variants. For the recursive IIR implementation of the 1-D Gaussian, we used the source code that is publicly available at the homepage of J. M. Geusebroek.[1] The hardware platform was a PC with 2.2 GHz Athlon64 3200+ CPU and 1 GB of RAM, using the Intel icc 9.0 compiler under a 32-bit GNU/Linux Operating System.

[1]http://www.science.uva.nl/~mark/

A. Accuracy

The first aspect is the accuracy of the separation method compared to an exact Gaussian convolution. In 1-D, FIR and recursive IIR filters are known to be good approximations to 1-D-Gaussian filters [16]. In 2-D, the triangular factorization coincides with the separation by Geusebroek *et al.*, who analyzed it and found it to be a close approximation of the true Gaussian [15]. Here, we will therefore focus on the accuracy of 3-D filtering.

To do so, we study the filter's 3-D impulse response. The impulse response is the result of filtering a delta signal. It is characteristic for each linear filter, because it yields a reconstruction of the filter kernel. In our case, we can determine the accuracy of our filter implementation by comparing its impulse response with the corresponding true Gaussian. Unless indicated otherwise, the accuracy results reported are for the geometric method with float data.

Experiments were performed in the same way as in [15] for 2-D. For a $65 \times 65 \times 65$ image, different values for σ_1 and σ_2 were fixed and σ_3 was set to σ_2. Then, φ and θ were varied in steps of 5°, each time calculating the square root of the sum of squared differences between the impulse response and the true Gaussian. By this choice of image size, the error sum is determined using almost the same number of image elements as in [15]. The maximum over all rotations of these L^2 error values is shown in Table 3, once with bilinear interpolation and once with nearest neighbor interpolation. The linear interpolation shows a smaller error, especially for small variance parameters, but in both cases, the errors are of comparable magnitude as reported for the 2-D case.

To get an intuitive understanding of the filter accuracy, and to see how the error is distributed over the image, the impulse response was evaluated in graphical form, shedding light on two questions: Does the three-step approach of the geometric method (shearing, axis-aligned filtering, inverse shearing) result in the right filtering direction? And, what impact does the method of interpolation have on the filters' accuracy?

Fig. 4 shows orthogonal cross sections along the main coordinate planes in an exemplary case ($\sigma_1 = 10, \sigma_2 = \sigma_3 = 5, \theta = 30°$ and $\varphi = 60°$). In Fig. 4(a), linear interpolation was used. The contour lines run close to the correct ones. Fig. 4(b) shows the same situation for nearest neighbor interpolation. The contour lines still have the right elliptic shape, but they show some defects on a local scale. The curves in Fig. 5 show 1-D-profiles that are obtained from the impulse response with linear interpolation with two coordinates fixed and the third varying. The top row of images shows this at the image center and the bottom one at a distance of 10 voxels. In both cases, the impulse response is close to the Gaussian, having the correct width and correctly localized peaks. Off the center, there are some small deviations in the filter strength. These will be discussed in Section VII.

B. Speed

The main motivation for deriving a separable anisotropic Gaussian convolution filter is to achieve a faster implementation than those previously available. We experimentally evaluated the speed of all proposed methods (naive, line buffer,

geometric) using bilinear and nearest neighbor (NN) interpolation on 2-D and 3-D images of single precision floating point and of 8-bit integer type. In the 2-D case, we also included the implementation of the anisotropic Gaussian filter that is available from J. M. Geusebroek's homepage. It integrates a linear interpolation step into the filter loop and works only for floating point images.

So far, the method of choice for anisotropic Gaussian filtering is convolution in the Fourier transform (FT) domain. Therefore, we compare all performance results to the speed of anisotropic Gaussian filtering based on the FFT. As FFT-implementation, we used the *fftw v3* library[2] in single precision mode. This state-of-the-art library is highly optimized, including hand-optimized SIMD assembler code. It is widely recognized as the best implementation of the discrete FT currently available. Single precision with real input data is its fastest mode of operation.

We measured the runtime of all routines on images with side lengths $N = 100,130,\ldots,4990$ for 2-D and $N = 50,55,\ldots,450$ for 3-D. For each measurement, the average runtime of 30 runs was used after excluding the top and bottom 10% as outliers. For the FFT, measurements contain the forward and backward transforms as well as pointwise multiplication with the transformed Gaussian kernel. A full Fourier transform step for the Gaussian can be avoided by pre-computing its transformed values and is therefore not included in the timing measurements.

The geometric implementation was used for the detailed runtime plots, because it turned out to be fastest in the experiments.

The results of the evaluation of filter runtime are presented in Fig. 6. It shows the absolute performance (measured as the total runtime) and the filter throughput (measured as the number of image elements processed per second). We plot the results of the geometric method with floating point data and linear interpolation, comparing it to the FFT. The speed of an isotropic filter is plotted for reference.

The geometric method and the isotropic filter show an almost smooth runtime curve for the range of image sizes. Some peaks of lower performance are visible at $N = 2560$, 3520, 4000, 4480, 4960 for 2-D, and at $N = 160$ and $N = 320$ for 3-D. Our interpretation of those outliers will be given in the discussion in Section VII. Apart from the smallest image sizes, where the throughput is higher, an almost constant number of pixels is filtered per second. The FFT shows a completely different behavior. It is generally slower than the separated routines and in particular, its runtime varies very strongly and non monotonously with image size. This will be analyzed in the discussion as well.

For the remaining methods, we present their performance in a condensed form: for each image size, we calculated the relative performance of all implementations compared to the reference implementation (isotropic, float data). Their average values and standard deviations over all image sizes are presented in Table VII. The filter implementation by Geusebroek can only handle floating point data. To measure its speed on 8-bit images, we therefore measured the time needed for a conversion to float format, for the filtering itself and for a conversion back. For the naive implementation, it is not possible to give a representative value of average runtime, since it depend not only on the image size but also on the variance parameters.

The table shows that all implementations of the Gaussian filter that rely on the nonorthogonal separation are faster than FFT-based filtering. The geometric method is the fastest in the 3-D case. In 2-D images, the routine by Geusebroek is fastest. In all cases, the difference between linear and nearest neighbor interpolation is small, and filtering images in 8-bit data format is often faster than in floating point representation.

VII. Discussion of Results

The results from Section VI can be summed up into two main statements.

A. *Calculating the anisotropic Gaussian using the separation scheme is faster than the FFT-based approach, often even by a large factor.*

B. *The resulting filter is a good approximation of the actual Gaussian.*

In the rest of this section, both statements will be discussed in more detail.

The runtime plots (Fig. 6) and tables (Fig. 7) show that anisotropic Gaussian filtering can be performed faster in image space than by using the Fourier transform and the convolution theorem. This extends the results that were known for the 2-D situation to higher dimensions and another implementation. Also, the methods working in image space achieved almost constant throughput, whereas in our experiments, the runtime of the FFT varied strongly. We believe that this is because the *fftw3* library contains specially optimized code paths for certain image sizes, in particular for powers of two. Depending on the image size, different routines in the library are used, causing the varying filter speeds.

The separated implementation follows the same regular code-path for all image sizes. Its runtime is mainly limited by floating point calculations and memory latency. The last point is also our explanation for the reduced performance at certain image sizes: modern CPUs have prefetch mechanisms to load data from the main memory that is likely to be needed later in the operation. When the prefetch heuristics fail, the execution speed is significantly reduced. We believe that this is the case for the outlier image sizes. A software prefetch mechanism could be integrated to avoid this effect.

Working on 8-bit data instead of floating point values reduces the total amount of memory needed and thereby increases the filtering speed. However, this positive effect is almost completely cancelled out by the additional CPU cost for integer-to-floating point conversions that is necessary to apply the 1-D Gaussian filter.

Finally, studying the relative speed of the methods, one can see that the performance gains compared to the FFT are larger in two than in three dimensions. In 2-D, the nonorthogonal separable Gaussian clearly outperforms the convolution in the Fourier domain. In 3-D, the separated filter is still faster than the FFT-based implementation, but the gain is not as large, at least not for all possible image sizes. Our explanation for

[2]http://www.fftw.org

σ_1	σ_2	σ_3	bilinear	nearest neighbor
2.0	1.0	1.0	0.0223	0.0346
3.0	1.0	1.0	0.0170	0.0280
5.0	2.0	2.0	0.0034	0.0057
7.0	2.0	2.0	0.0028	0.0046
7.0	4.0	4.0	0.0010	0.0013
10.0	3.0	3.0	0.0017	0.0019
10.0	5.0	5.0	0.0006	0.0008
10.0	7.0	7.0	0.0004	0.0004

Fig. 3. Euclidean error between the 3-D impulse response and the true Gaussian kernel: Maximum square root of the summed square error over all rotation angles.

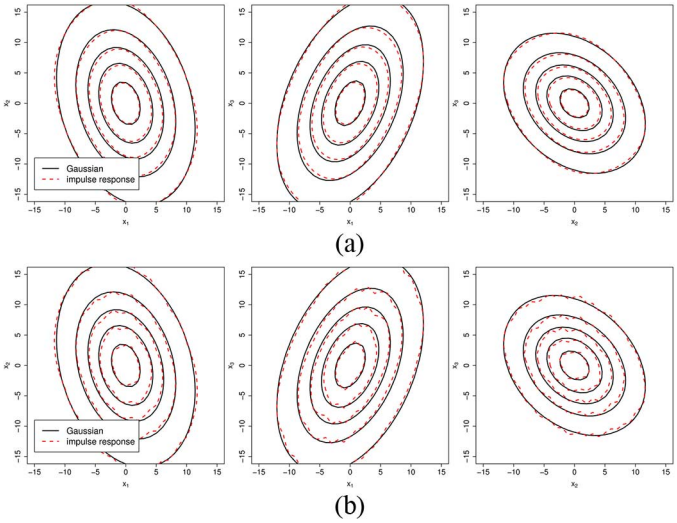

Fig. 4. Orthogonal slices through the impulse response of the 3-D implementation, $\theta = 30°$, $\varphi = 60°$, $\sigma_1 = 10$ and $\sigma_2 = \sigma_3 = 5$. For comparison, the contour lines of the true Gaussians are depicted as well. (a) Convolution using single-precision floating-point data and linear/bilinear interpolation. (b) Convolution using single-precision floating-point data and nearest neighbor interpolation.

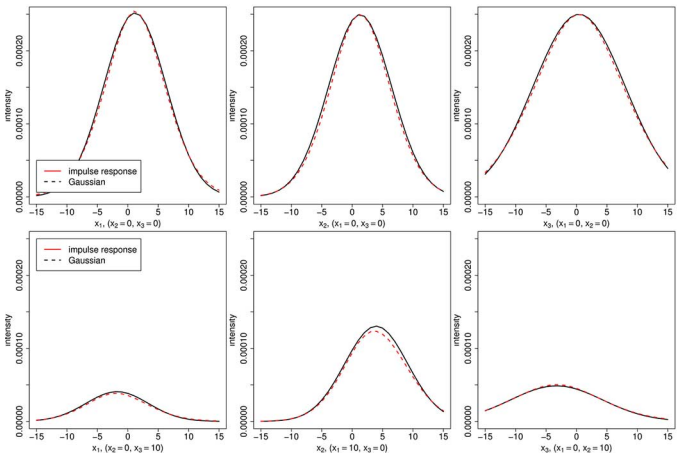

Fig. 5. 1-D profiles across the impulse responses from Fig. 4(a) against the true Gaussian function at the coordinate center (top) and shifted by 10 voxels from the center (bottom).

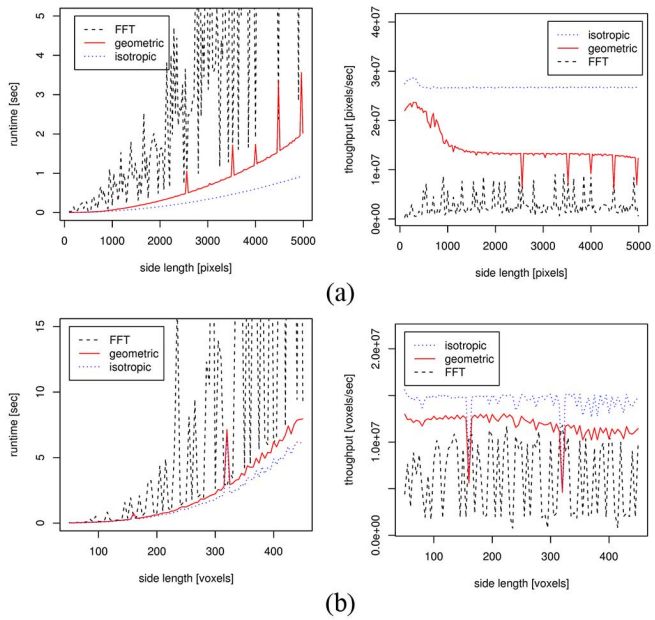

Fig. 6. Performance of different methods to calculate the anisotropic Gaussian filter: absolute speed (left) and relative throughput (right). (a) 2-D filtering performance: geometric implementation versus FFT-based convolution versus isotropic filtering (float data). (b) 3-D filtering performance: geometric implementation versus FFT-based convolution versus isotropic filtering (float data).

this is twofold. For one, the computational complexity of the FFT-method is $O(N \log N)$, if N is the size of the largest image dimension. The complexity of the separated filter is $O(N)$, which therefore has an advantage of $O(\log N)$. This factor is larger in 2-D data, where N can have values of several thousands, than for 3-D, where N is in the range of a few hundreds. The second reason is that the speeds of the different methods in 3-D are, in general, closer to each other, because the larger image size makes the problem more determined by memory access than by CPU operations. This is because not only the total size of the images influences the runtime, but also the patterns how the memory cells are accessed. In 3-D, the average distance between neighboring image elements is larger, causing more cache-misses and thus reducing the filter speed. This observation is backed up by the fact that the speed advantage of the isotropic Gaussian compared with the new methods and the FFT is larger in the 2-D than in the 3-D case.

Regarding the accuracy, the numerical and the graphical evaluations of errors (Figs. 3 and 4) show that the impulse response of the proposed filter closely matches the anisotropic Gaussian. Thus, the implementation indeed performs a Gaussian filtering with the correct rotational and variance parameters. There are only minor deviations in the peak height (Fig. 5), which we believe are caused by two factors. Firstly, the IIR filter is only an approximation of the true Gaussian. Applying it three times can cause a visible difference in the enlarged plots. Secondly, in our implementation the IIR filter acts in a wrap-around manner on the image data instead of cutting the Gaussian off at the image edges. This design was chosen to simplify the implementation.

The results also indicate that it does not play a major role which interpolation method is used to access between pixel positions. Nearest neighbor interpolation does introduce an error, but it is small and the filter still performs Gaussian smoothing

method	interpolation	rel. runtime float data	rel. runtime 8 bit data
isotropic		1.00	0.98 ± 0.07
geometric	NN	1.21 ± 0.05	1.01 ± 0.07
geometric	bilinear	1.22 ± 0.05	1.10 ± 0.05
line buffer	NN	2.11 ± 0.22	1.46 ± 0.16
line buffer	bilinear	3.73 ± 0.44	2.24 ± 0.27
FFT		4.30 ± 3.99 *(see text)*	
naive		depends on σ *(see text)*	

(a) 3D relative performance

method	interpolation	rel. runtime float data	rel. runtime 8 bit data
isotropic		1.00	1.59 ± 0.30
geometric	NN	1.98 ± 0.55	1.62 ± 0.29
geometric	linear	2.01 ± 0.55	1.70 ± 0.29
line buffer	NN	2.09 ± 0.06	2.01 ± 0.35
line buffer	linear	3.69 ± 0.15	2.89 ± 0.60
Geusebroek	linear	1.26 ± 0.07	1.42 ± 0.07
FFT		7.69 ± 6.92 *(see text)*	
naive		depends on σ *(see text)*	

(b) 2D relative performance

Fig. 7. Average performance of different implementations of the anisotropic Gaussian filter, measured in relative units. For each method, the ratio between their runtime and the runtime of the isotropic Gaussian was measured. The tables shows their mean and standard deviation over all image sizes (see text).

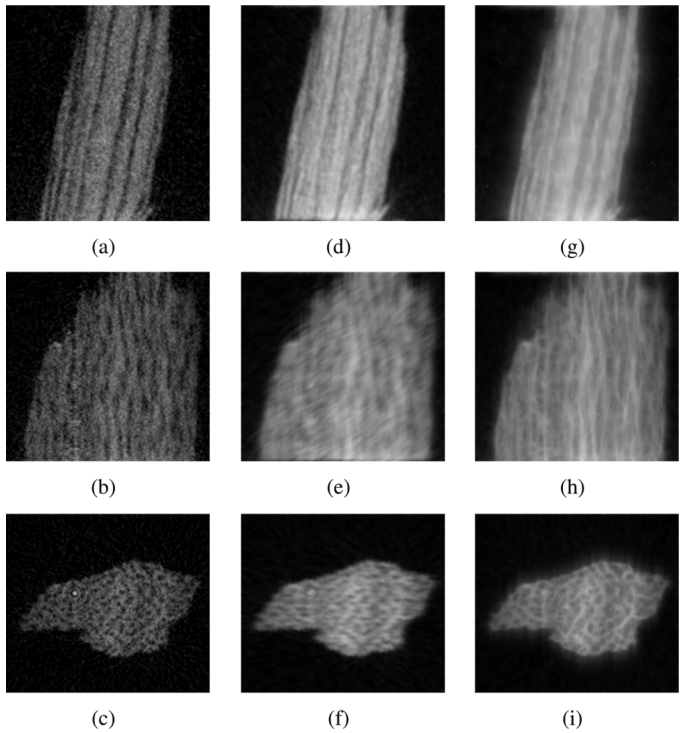

Fig. 8. (a)–(c) Orthogonal cross-sections through a μCT of a piece of wood. (d–(f) Anisotropic filtering with parameters aligned to dominant fiber direction ($\theta = 77°, \varphi = 0°, \sigma_1 = 5, \sigma_2 = \sigma_3 = 1$). (g)–(i) Orientation-space with an angular resolution of 10°, showing maximal filter response in each voxel.

with correct strength and orientation. However, linear interpolation does give better results and is fast as well, so there is no reason not to use it in practical applications.

VIII. Applications

In this section we show some application examples of anisotropic Gauss filters, both using them as directed filters and to construct orientation space images. We have applied the method to 3-D image data and to $2+1$D video.

A. Processing of Fibrous Data

Many macroscopically homogeneous materials reveal heterogeneous characteristics when examined at a microscopic scale. Quantization of such properties by means of image analysis in three dimensional data is possible using micro-tomographic (μCT) data. Typically, binarization is a prerequisite to performing such analyses, see e.g., [26]. In this setting, anisotropic filtering is a useful preprocessing tool, especially when one is dealing with images of strongly anisotropic objects such as fibers.

Fig. 8 shows orthogonal cross-sections through a 3-D image of a piece of wood obtained by μCT. Anisotropic filtering with hand-selected orientation parameters can reduce the level of noise, but Fig. 8(e) reveals that even though parameters were aligned with the dominant fiber orientation, unwanted smoothing across edges is occurring. These defects are eliminated in the orientation space representation of the data, where structures in cross-sections orthogonal to the dominant fiber direction become visible, see Fig. 8(h). In the orientation space representation of an image, one selects the parameters which maximize filter response in each voxel.

B. Spatiotemporal Smoothing of Video Sequences

For object detection in video sequences, typically low-level image features like color and texture are used, or motion between frames is analyzed. For both, de-noising usually is the first preprocessing step, which is done individually for each frame. Afterwards, motion estimation and feature extraction are performed, again frame by frame.

An alternative unified approach relies on the covariance information between subsequent frames for smoothing. Since objects do not appear or disappear spontaneously in a video sequence, they cause dependencies between the pixel values at different time steps. This can be utilized for filtering: instead of applying a 2-D-smoothing filter to each image, we make use of the correlations and apply an anisotropic 3-D-smoothing filter along space and time at the same time.

Fig. 9 illustrates this using part of the *mobcal* video sequence.[3] It shows a moving toy train in front of a differently moving background and other objects. After filtering the sequence with a prolate 3-D-Gaussian whose correlation matrix is aligned to the train motion, we can see that the image is blurred where the scene motion does not coincide with the train motion. On the front of the train, the Gaussian mainly acts a de-noising filter, without the strong blurring effect. Filtering

[3]ftp://ftp.ldv.e-technik.tu-muenchen.de/pub/test sequences/1080i/

(a) (b)

Fig. 9. Anisotropic filtering to selectively smoothen $2 + 1\mathrm{D}$ video data. A strongly prolate anisotropic 3-D-Gaussian is aligned with the motion of the front of the toy train. The resulting filter acts as a motion aware blurring filter, only weakly smoothing the first two cars of the train, but strongly blurring the background and other objects which show different movement patterns. (a) Original frame #240. (b) Anisotropically filtered.

the 474-MB clip (80 RGB frames in 1920×1080 resolution) took less than 1 min on a 2-GHz PC.

IX. Conclusion

We have presented a geometrically motivated method for separating the anisotropic Gaussian filter, i.e., for decomposing its n-dimensional convolution integral into a sequence of n 1-D ones. We were able to show that the separation based on the triangular factorization of Cholesky type is optimal. Optimality here means to require the minimal possible number of interpolation operations and memory accesses. This theoretical result generalizes and explains the mechanisms behind earlier results of the nonorthogonal separation scheme in $\mathbb{R}^2$ by Geusebroek *et al.* [15].

Since the proposed algorithm works solely in the image domain, it is very flexible in implementation. It allows to incorporate different interpolation schemes and methods to calculate the 1-D Gaussian. When using an FIR filter subroutine, our method can easily be applied locally, enabling it, for example, to cope with missing data. Any of these implementations was shown to be a good approximation of the true Gaussian filter: there was no evidence for structural errors introduced to the results by using a particular interpolation or convolution scheme. The main advantage of the separation is that it is very fast. Already our plain C implementation outperforms the usual FFT-based method, even if that relies on SIMD-optimized assembler code.

Application examples of anisotropic Gaussian filters were shown to put the proposed method into a larger context. The focus of the present work was to develop a sound mathematical basis and an optimal solution for an open problem. Additionally, we were able to demonstrate the usefulness of the filter for different tasks in image and video processing.

One problem of the nonorthogonal separation is that the calculation of Gaussian derivative filters is less straightforward. Because the directions of convolution do not coincide with the Gaussian's main axes, it is not possible to just convolve with a derived 1-D convolution kernel, as it is the case for isotropic Gaussians. The same phenomenon already exists in 2-D, where the authors of [14] suggest the use of rotated finite differences for calculating the derivatives. The necessary linear combinations of the filtered samples can be interwoven into the separated convolution steps themselves.

To improve the filtering method, it might be useful to look for even better (faster, more accurate) separations by allowing other functions than Gaussians as 1-D convolution kernels. The first candidates that come to mind would be of Gaussian form but without a normalization constraint. This corresponds to transformation matrices V that are not unit diagonal. The higher number of possible choices for the separation matrices would in some cases allow shorter direction vectors for the 1-D convolutions. This results in more accurate interpolation and better locality of memory accesses.

References

[1] G.-Z. Yang, P. Burger, D. N. Firmin, and S. R. Underwood, "Structure adaptive anisotropic image filtering," *Image Vis. Comput*, vol. 14, no. 2, pp. 135–145, 1996.

[2] M. Lynch, K. Robinson, O. Ghita, and P. Whelan, "A performance characterisation in advanced data smoothing techniques," presented at the Irish Machine Vision and Image Processing Conf. 2004.

[3] J. Wang, B. Thiesson, Y. Xu, and M. Cohen, "Image and video segmentation by anisotropic kernel mean shift," in *ECCV (2)*, ser. Lecture Notes in Computer Science, T. Pajdla and J. Matas, Eds. New York: Springer, 2004, vol. 3022, pp. 238–249.

[4] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis, "Processing and visualization of diffusion tensor MRI," *Med. Image Anal.*, vol. 6, no. 2, pp. 93–108, 2002.

[5] K. Okada, D. Comaniciu, N. Dalal, and A. Krishnan, J. M. T. Pajdla, Ed., "A robust algorithm for characterizing anisotropic local structures," in *Proc. Eur. Conf. Computer Vision*, Heidelberg, Germany, Apr. 2004, vol. 1, ECCV, pp. 549–561.

[6] K. Okada, D. Comaniciu, and A. Krishnan, "Scale selection for anisotropic scale-space: Application to volumetric tumor characterization," presented at the IEEE Conf. Computer Vision and Pattern Recognition 2004.

[7] R. Manmatha and N. Srimal, "Scale space technique for word segmentation in handwritten documents," in *Scale-Space*, ser. Lecture Notes in Computer Science, M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, Eds. New York: Springer, 1999, vol. 1682, pp. 22–33.

[8] V. Lakshmanan, "A separable filter for directional smoothing," *IEEE Trans. Geosci. Remote Sens. Lett.*, vol. 1, no. 3, pp. 192–195, Jul. 2004.

[9] F. G. A. Faas and L. J. van Vliet, "3D-orientation space; filters and sampling," in *SCIA*, ser. Lecture Notes in Computer Science, J. Bigün and T. Gustavsson, Eds. New York: Springer, 2003, vol. 2749, pp. 36–42.

[10] T. Lindeberg, *Scale-Space Theory in Computer Vision*. Norwell, MA: Kluwer, 1994.

[11] J. Loviscach, "Motion blur for textures by means of anisotropic filtering," in *Proc. Rendering Techniques (Eurographic Symp. Rendering)*, 2005, pp. 105–110.

[12] W. T. Freeman and E. H. Adelson, "The design and use of steerable filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 9, pp. 891–906, Sep. 1991.

[13] P. Perona, "Steerable-scalable kernels for edge detection and junction analysis," *Image Vis. Comput*, vol. 10, no. 10, pp. 663–672, 1992.

[14] J.-M. Geusebroek, A. W. M. Smeulders, and J. van de Weijer, "Fast anisotropic Gauss filtering," in *ECCV: Proc. 7th Eur. Conf. Computer Vision-Part I*, London, U.K., 2002, pp. 99–112.

[15] ——, "Fast anisotropic gauss filtering," *IEEE Trans. Image Process.*, vol. 12, no. 8, pp. 938–943, Aug. 2003.

[16] I. T. Young and L. J. van Vliet, "Recursive implementation of the Gaussian filter," *Signal Process.*, vol. 44, pp. 139–151, 1995.

[17] O. Wirjadi and T. M. Breuel, "Approximate separable 3D anisotropic Gauss filter," in *Proc. IEEE Int. Conf. Image Processing*, Genova, Italy, 2005, vol. II, pp. 149–152.

[18] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: Wiley-Interscience, 2000.

[19] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[20] G. Strang, *Introduction to Applied Mathematics*. Cambridge, MA: Wellesley-Cambridge, 1986.

[21] E. Weisstein, *CRC Concise Encyclopedia of Mathematics*, 2nd ed. London, U.K./Boca Raton, FL: Chapman & Hall/CRC, 2002.

[22] J. Proakis and D. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 1996.

[23] J. Lim, *Two-Dimensional Signal and Image Processing*. Upper Saddle River, NJ: Prentice-Hall, 1990.

[24] I. T. Young, L. J. van Vliet, and M. van Ginkel, "Recursive Gabor filtering," *IEEE Trans. Signal Process.*, vol. 50, no. 11, pp. 2798–2805, Nov. 2002.
[25] B. Triggs and M. Sdika, "Boundary conditions for young—Van Vliet recursive filtering," *IEEE Trans. Signal Process.*, vol. 54, no. 6, pt. 1, pp. 2365–2367, Jun. 2006.
[26] C. Lang, J. Ohser, and R. Hilfer, "On the analysis of spatial binary images," *J. Microscopy*, vol. 203, pp. 303–313, 2001.

Christoph Lampert was born in Konstanz, Germany, in 1974. He received the Ph.D. degree in mathematics from the Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany, in 2003.

His research in the area of image and video processing started in 2001, when co-founding the Open Source MPEG-4 codec XviD. Presently, he is a Senior Researcher at the German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, working on image understanding and machine learning in video processing.

Oliver Wirjadi (S'05) was born in Berlin, Germany, in 1978. He received the diploma in computer science from the Technische Universität Berlin, Berlin, Germany, in 2003. He currently holds a scholarship from Fraunhofer ITWM, Kaiserslautern, Germany, and is pursuing the Ph.D. degree at the Technische Universität Kaiserslautern.

He was a Research Assistant at the Illinois Institute of Technology, Chicago, during 2001–2002. His research interests include three-dimensional image processing and the uses of machine learning in computer vision.