HASSO-PLATTNER-INSTITUT
Fachgebiet Computergrafische Systeme

Structure Adaptive Stylization of Images and Video

Dissertation
zur Erlangung des akademischen Grades
„doctor rerum naturalium“
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Praktische Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

von

Jan Eric Kyprianidis

Potsdam,
5. Februar 2013

Abstract

In the early days of computer graphics, research was mainly driven by the goal to create realistic synthetic imagery. By contrast, non-photorealistic computer graphics, established as its own branch of computer graphics in the early 1990s, is mainly motivated by concepts and principles found in traditional art forms, such as painting, illustration, and graphic design, and it investigates concepts and techniques that abstract from reality using expressive, stylized, or illustrative rendering techniques. This thesis focuses on the artistic stylization of two-dimensional content and presents several novel automatic techniques for the creation of simplified stylistic illustrations from color images, video, and 3D renderings. Primary innovation of these novel techniques is that they utilize the smooth structure tensor as a simple and efficient way to obtain information about the local structure of an image.

More specifically, this thesis contributes to knowledge in this field in the following ways. First, a comprehensive review of the structure tensor is provided. In particular, different methods for integrating the minor eigenvector field of the smoothed structure tensor are developed, and the superiority of the smoothed structure tensor over the popular edge tangent flow is demonstrated. Second, separable implementations of the popular bilateral and difference of Gaussians filters that adapt to the local structure are presented. These filters avoid artifacts while being computationally highly efficient. Taken together, both provide an effective way to create a cartoon-style effect. Third, a generalization of the Kuwahara filter is presented that avoids artifacts by adapting the shape, scale, and orientation of the filter to the local structure. This causes directional image features to be better preserved and emphasized, resulting in overall sharper edges and a more feature-abiding painterly effect. In addition to the single-scale variant, a multi-scale variant is presented, which is capable of performing a highly aggressive abstraction. Fourth, a technique that builds upon the idea of combining flow-guided smoothing with shock filtering is presented, allowing for an aggressive exaggeration and an emphasis of directional image features.

All presented techniques are suitable for temporally coherent per-frame filtering of video or dynamic 3D renderings, without requiring expensive extra processing, such as optical flow. Moreover, they can be efficiently implemented to process content in real-time on a GPU.

Zusammenfassung

In den Anfängen der Computergrafik war die Forschung hauptsächlich von dem Anspruch getragen, realistisch aussehende synthetische Bilder zu erstellen. Im Gegensatz dazu ist die nicht-photorealistische Computergraphik, ein Untergebiet der Computergrafik, welches in den frühen 1990er Jahren gegründet wurde, vor allem motiviert durch Konzepte und Prinzipien der traditionellen Kunst wie Malerei, Illustration und Grafikdesign. Diese Arbeit beschäftigt sich mit der künstlerischen Verarbeitung von zweidimensionalen Bildinhalten und präsentiert mehrere neue automatische Verfahren für die Erstellung von vereinfachten künstlerischen Darstellungen von Farbbildern, Videos und 3D-Renderings. Wichtigste Neuerung dieser Techniken ist die Verwendung des Strukturtensors als eine einfache und effiziente Möglichkeit, Informationen über die lokale Struktur eines Bildes zu erhalten.

Konkret werden die folgenden Beiträge gemacht. Erstens wird eine umfassende Übersicht über den Strukturtensor gegeben. Insbesondere werden verschiedene Methoden für die Integration des kleineren Eigenvektorfeldes des geglätteten Strukturtensors entwickelt, und die Überlegenheit des geglätteten Strukturtensors gegenüber dem populären Edge-Tangent-Flow demonstriert. Zweitens werden separable Implementierungen des bilateralen Filters und des Difference of Gaussians Filters vorgestellt. Durch die Anpassung der Filter an die lokale Struktur des Bildes werden Bildfehler vermieden, wobei der Vorgang rechnerisch effizient bleibt. Zusammengenommen bieten beide Techniken eine effektive Möglichkeit, um einen Cartoon-ähnlichen Effekt zu erzielen. Drittens wird eine Verallgemeinerung des Kuwahara-Filters vorgestellt. Durch die Anpassung von Form, Umfang und Orientierung der Filter an die lokale Struktur werden Bildfehler verhindert. Außerdem werden direktionale Bildmerkmale besser berücksichtigt und betont, was zu schärferen Kanten und einem malerischen Effekt führt. Neben der single-scale Variante wird auch eine multi-scale Variante vorgestellt, welche im Stande ist, eine höhere Abstraktion zu erzielen. Viertens wird eine Technik vorgestellt, die auf der Kombination von flussgesteuerter Glättung und Schock-Filterung beruht, was zu einer intensiven Verstärkung und Betonung der direktionalen Bildmerkmale führt.

Alle vorgestellten Techniken erlauben die zeitlich kohärente Verarbeitung von Einzelbildern eines Videos oder einer dynamischen 3D-Szene, ohne dass andere aufwendige Verfahren wie zum Beispiel die Berechnung des optischen Flusses, benötigt werden. Darüberhinaus können die Techniken effizient implementiert werden und ermöglichen die Verarbeitung in Echtzeit auf einem Grafikprozessor (GPU).

Acknowledgments

This thesis covers the research that I carried out while working as a research scientist with the Computer Graphics Systems Group at the Hasso-Plattner-Institut in Potsdam, Germany from 2007–2012. I would like to thank Jürgen Döllner for offering me the position at HPI, introducing me to the exciting field of non-photorealistic rendering, and enabling me to pursue my research interests. I am also very grateful to Henry Kang for all the helpful discussions and feedback during the past years. Moreover, I would like to express my gratitude to John Collomosse for all the good advice, sharing his expertise on artistic rendering, and for his hospitality during my visits to the University of Surrey. I am grateful for all the support I received and would like to thank Jürgen, Henry, and John for agreeing to review this thesis and serve on my thesis committee.

I would like to thank my former colleagues for making HPI a great place to work. In particular, I would like to thank Matthias Trapp for all the discussions on graphics, GPU programming, and other related topics. I want to give special thanks to Amir Semmo and Daniel Limberger for showing so much interest in my work, and for their comments and feedback on the early drafts of this thesis.

I am grateful to several people for their encouragement and for many stimulating discussions: Mathias Eitz, Christian Richard, Sven Olsen, and Holger Winnemöller deserve special mention. Thank you also to Tinghuai Wang and Tobias Isenberg for joining the survey paper project, to Adobe Systems for enabling me to visit the CTL office in Seattle, and to Seungyong Lee for inviting me to visit his group after Pacific Graphics 2009. Some portions of this work were supported by the German Research Foundation (DFG), grant DO 697/5-1.

Finally, I would like to thank my family for their love and support.

Contents

Chapter 1

Introduction

"Style, in its finest sense, is the last acquirement of the educated mind; it is also the most useful. It pervades the whole being. The administrator with a sense for style hates waste; the engineer with a sense for style economises his material; the artisan with a sense for style prefers good work. Style is the ultimate morality of mind."

— Alfred North Whitehead

The invention of the daguerreotype process in 1837 made it possible for the first time, to capture images permanently, making photography interesting to a wider audience. Until then, painters were the only ones to receive commissioned work for portraits or to archive historical events and they tried to reproduce their subjects with the best possible accuracy in every detail. The camera's capability to capture clear and accurate images drove painters to experiment and expand their creative vision, resulting in the development of, for example, Impressionism and Expressionism. Similarly, in the early days of computer graphics, research was mainly driven by the aim to create realistic synthetic imagery. In the beginning, work focused on the development of plausible models and later on physically accurate simulations, resulting in the development of, for example, radiosity and path tracing. Just as the invention of photography stimulated artistic diversity in the late 19th century, the successes of photorealistic computer graphics in the early 1990s motivated alternative techniques for rendering in non-photorealistic styles, establishing non-photorealistic computer graphics [GG01; SS02] as its own branch of computer graphics, which, in contrast to traditional computer graphics, is mainly motivated by the concepts and principles found in traditional art forms, such as painting, illustration, and graphics design. Non-photorealistic computer graphics investigates concepts and techniques that abstract from reality using expressive, stylized, or illustrative rendering techniques. Today, the field of *non-photorealistic rendering* (NPR) has expanded into a highly active area of research, covering a wide range of expressive rendering styles, such as artistic rendering of three-dimensional objects [Mei96; LKL06; DR07], exploded diagrams [Li+08], false color [PR11; Rei+01], and artistic styles, including painterly [Bou+07; Zen+09] and constrained palette rendering [MG08; XK08]. While NPR also covers artistic rendering from the object space of a three-dimensional scene, this thesis focuses on techniques for the artistic stylization of two-dimensional content (i.e., photographs and video), which is referred to as *image-based artistic rendering* (IB-AR). For a comprehensive survey of this field see [J1]. Moreover, a book [RC12] on the subject is about to appear, including a chapter [B2] on nonlinear filtering approaches.

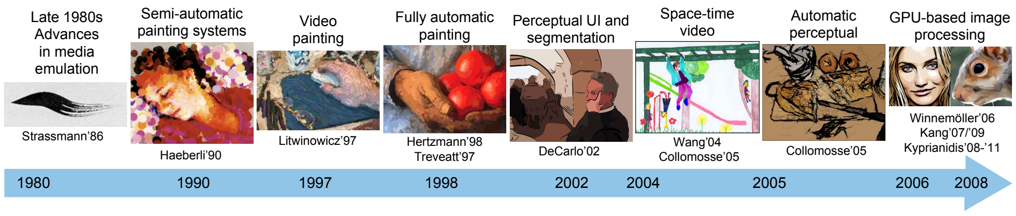
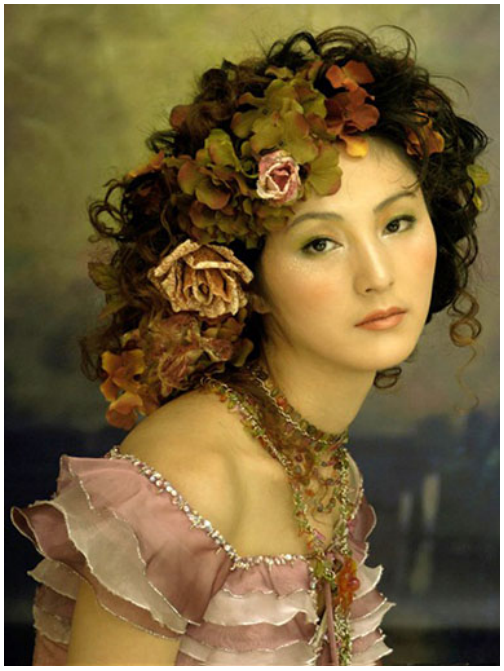


Figure 1.1: *Chronology of IB-AR development. From the semi-automated stroke-based rendering systems of the early 1990s, to increasingly automated systems based upon advanced computer vision and image processing techniques that enhanced the aesthetic gamut. Later, the availability of massive computing resources, provided by modern GPU hardware, has spawned the development of techniques operating in real-time.*

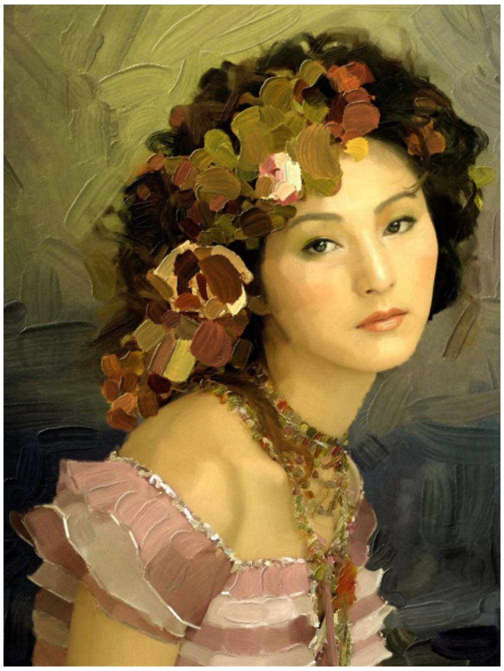
The origins of IB-AR reach back to seminal works that explore the emulation of traditional artistic media and styles [Hae90; Cur+97; Lit97; Sal+97; Her98]. While early IB-AR approaches only utilized simple low-level image processing operations, such as the image gradient, modern IB-AR methods build upon the latest methods from computer vision, perceptual modeling, human computer interaction, and computer graphics (Figure 1.1). For example, video cartooning [CRH05; Wan+04] is related to video matting and automated rotoscoping [Aga+04], and painterly rendering of video [HE04; Lit97] requires optical flow in order to obtain temporally coherent results. While advanced techniques that perform a higher-level analysis of an image or video provide a wealth of information that may be used to drive the stylization and rendering process, this information comes at a price. Advanced techniques are typically computationally demanding and techniques, such as segmentation and optical flow, do not perform equally well on all footage. Figure 1.2(b) shows an example, created by an advanced technique, utilizing computer vision and machine learning for the analysis and stoke-based rendering for synthesizing the final result.

In parallel with the trend toward more sophisticated scene analysis, techniques based on heuristics have been developed that avoid a higher-level analysis of the image. Most of these techniques are based on image processing operations, such as the cartoon pipeline by Winnemöller et al. [WOG06], which is based on the bilateral and difference of Gaussians filters. Due to the local nature of image processing decisions, parallelization and GPU implementations of image filters are straightforward in most cases and often lead to real-time performance on modern multi-core CPUs and GPUs. Consequently, they are practical for video processing and are applicable to footage, such as water, smoke, or fur, which is otherwise challenging to parse using computer vision methods such as segmentation. This simplicity, however, comes at the expense of the style diversity that is possible when a higher-level interpretation of the content is applied. Nevertheless, the lack of higher-level image understanding can, to some extent, be superseded by allowing the user to interactively experiment with a technique's parameters. Interestingly enough, a trend toward semi-automatic systems is also observable for methods based on higher-level analysis. While fully automatic techniques may be interesting from an academic point of view, tools for artists require possibilities for intervention that support the achievement of the artist's intention.

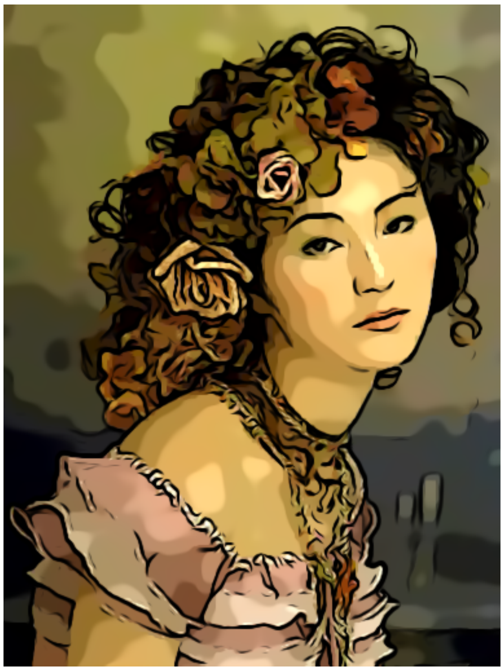
This thesis aims for a compromise between stylization methods driven by higher-level image analysis and fast and simple image processing operations. To this end, this thesis advocates for the use of the smoothed structure tensor as a simple and efficient way to obtain



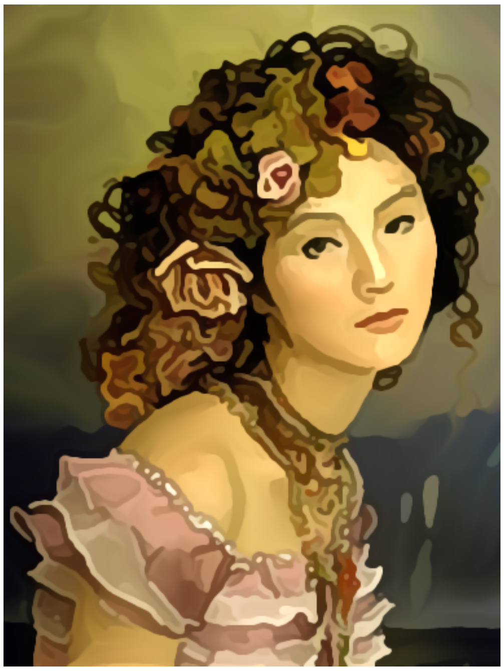
(a) *Original image*



(b) *Advanced stroke-based rendering method [Zen+09]*



(c) *Cartoon (Chapter 4)*



(d) *Coherence-enhancing filtering (Chapter 6)*

Figure 1.2: *Comparison of different IB-AR techniques. (a) Original image. (b) Result of an advanced stroke-based rendering method by Zeng et al. [Zen+09], utilizing computer vision and higher-level analysis. (c) Result obtained using the generalized cartoon rendering pipeline discussed in Section 4.6. (d) Result obtained by applying coherence-enhancing filtering discussed in Chapter 6.*

information about the local structure of an image. On this basis, two general principles to utilize the information about the local structure will be investigated. First, the information may be used to improve and generalize existing techniques (Figure 1.2(c)). Second, the information may be used to achieve artistic stylization by a reinforcement or exaggeration of directional image features and flow-like structures (Figure 1.2(d)). These are considered pleasant, harmonic, or at least interesting by most humans [Wei99]. They are also a highly sought after property in many of the traditional art forms, such as painting and illustration. Enhancing directional coherence in an image helps to clarify region boundaries and features. As exemplified by Expressionism, it also helps to evoke mood or ideas and even elicit emotional response from the viewer [Wik12]. Particular examples include van Gogh and Munch, who have emphasized these features in their paintings.

In order to demonstrate the effectiveness of the proposed approach, several classical filters will be examined and new variants developed that take the local structure intro account. More specifically, generalizations of the bilateral, difference of Gaussians, and Kuwahara filters, as well as a new filter that performs smoothing and sharpening locally in perpendicular directions will be presented.

1.1 Structure and Contributions

This thesis is organized as follows:

Chapter 2 presents a comprehensive introduction to the smoothed structure tensor and related techniques. It forms the foundation for the subsequent chapters, which utilize the structure tensor to guide local neighborhood filters. It presents several different but equivalent definitions of the structure tensor, each providing a different view and highlighting specific properties. Moreover, it discusses how to derive confidence measures and cornerness measures from the structure tensor. The chapter's main contribution is an examination of different integration methods to perform line integral convolution along the integral curves induced by the smoothed structure tensor. These techniques play a fundamental role in later chapters. Portions of this chapter are based on the publications [J3; C1; C3].

Chapter 3 presents two variants of the bilateral filter. The first variant is the orientation-aligned bilateral filter, a fast separable approximation of the bilateral filter that works by filtering in the direction of the gradient and then filtering the intermediate result in the perpendicular direction. In contrast to an axis-aligned separable implementation, this approach does not suffer from horizontal or vertical artifacts and it creates smooth output at curved boundaries when applied iteratively. Moreover, moderate stylization can be achieved when stronger smoothing is used in the tangent direction, making the filter shape elliptic. A stronger stylization effect can be achieved with the second variant presented in this chapter, the flow-based bilateral filter, which realizes the filtering in tangent direction as a line integral convolution along the integral curves induced by the smoothed structure tensor. For both approaches, special sampling schemes are presented to reduce blurring. Portions of this chapter are based on the publications [C3; B4]. The concept of the flow-based bilateral filter was developed independently from Kang et al. [KLC09], but was not formally published.

Chapter 4 presents a separable implementation of the flow-based difference of Gaussians filter, and a reparameterization of the extended difference of Gaussians thresholding scheme, which simplifies artistic control by providing uncoupled parameters. Moreover, this chapter presents a generalization of Winnemöller et al.'s cartoon pipeline [WOG06], based on the separable implementations of the flow-based bilateral and difference of Gaussians filters. This pipeline is highly efficient and its GPU implementation processes video in real-time, while achieving excellent temporal coherence with per-frame filtering. Portions of this chapter are based on the publications [C3; B4; J2]. The separable implementation of the flow-based difference of Gaussians filter has also been independently proposed in Kang et al. [KLC09].

Chapter 5 presents the anisotropic Kuwahara filter, which is a generalization of the Kuwahara filter that is adapted to the local shape of features derived from the smoothed structure tensor. Contrary to conventional edge-preserving filters, the new filter generates a painting-like flattening effect along the local feature directions while preserving shape boundaries. As opposed to conventional painting algorithms, it produces temporally coherent video abstraction without extra processing. The GPU implementation of this method processes video in real-time. The results have the clearness of cartoon illustrations but also exhibit directional information as found in oil paintings. Moreover, driven by local image flattening, it achieves a comparatively consistent level of abstraction across an image. For the smoothing process, the anisotropic Kuwahara filter uses weighting functions that use convolution in their definition. The chapter presents two methods that can be used to implement the anisotropic Kuwahara filter. The first method samples the weighting functions into a texture map. The second method models the weighting functions as polynomial functions allowing for the computation to occur directly during the filtering in real-time. Moreover, a multi-scale computation scheme is presented that simultaneously propagates local orientation estimates and filtering results up a low-pass filtered pyramid. This allows for a very strong abstraction effect and avoids artifacts in large low-contrast regions. The propagation is controlled by the local variances and anisotropies that are derived during the computation without extra overhead, resulting in a highly efficient scheme that is particularly suitable for real-time processing on a GPU. Portions of this chapter are based on the publications [J4; B3; C1; C2].

Chapter 6 presents a technique that builds upon the idea of combining diffusion with shock filtering for image abstraction. The underlying idea behind the approach is to perform smoothing in the direction where the image is changing the least and sharpening in the orthogonal direction. Instead of modeling this process by a partial differential equation (PDE) and solving it, approximations are used that operate as local filters on a neighborhood of a pixel. This has two major benefits. First, good abstraction results are already achieved after a few iterations, making it possible to perform processing at real-time rates on a GPU. Second, since only a few iterations are required, per-frame processing of videos achieves temporally coherent results. By contrast, PDE-based techniques typically require a large number of iterations and per-frame processing is unstable. Portions of this chapter are based on the publications [J3; B1].

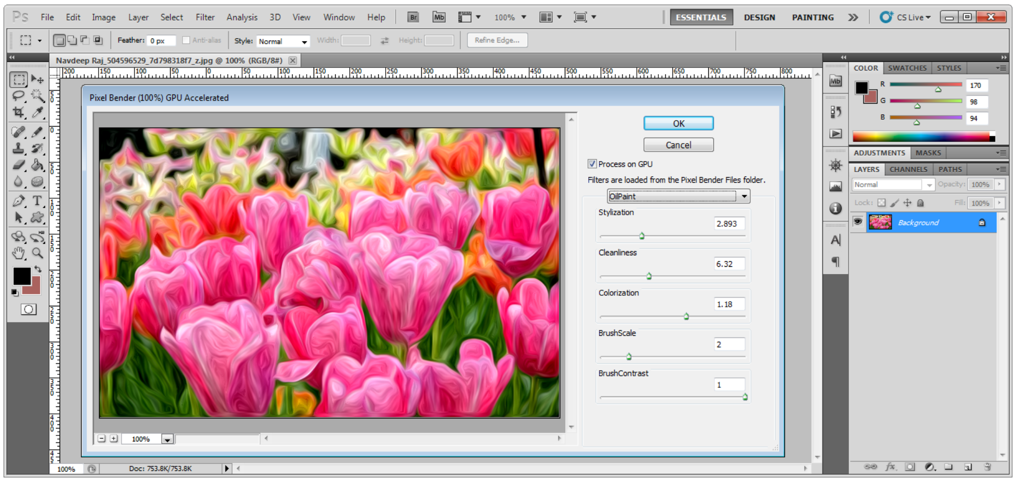
Chapter 7 draws conclusions.

1.2 Areas of Application

Digital consumer imaging devices have seen an explosive growth in recent years. Each smart-phone or tablet shipped today has a built-in camera, which, in most cases, in addition to being able to take images, is also capable of capturing video. Moreover, high-quality digital cameras and camcorders are available at reasonable prices. In addition to the highly simplified and easily accessible image acquisition process, computing resources of desktop computers and small devices have greatly increased. In particular, most modern smart phones and tablets have a GPU and support rendering with OpenGL. The availability of image content and computing capacities create opportunities for new applications that utilize these resources. In the following, three primary types of applications that come into consideration for the algorithms developed in this thesis are discussed.

First, the techniques may be incorporated into tools for skilled artists as part of a larger toolset or application. Adobe Photoshop, for example, already contains a large set of built-in image filters. The filters presented in this thesis could be made available through the Adobe Photoshop SDK or implemented using the Adobe Pixel Bender graphics language. A specific example is the oil paint filter developed by H. Winnemöller and shipped with Adobe Photoshop CS5 and CS6. That filter, utilizes the flow-guided smoothing approach presented in Chapter 2 that was first published in [C3; B4]. Processing starts with the computation of the smoothed structure tensor. Then, similar to the approach used by Inoue and Urahama [IU04], noise is added to the source image and smoothing, along the integral curves induced by the minor eigenvectors of the smoothed structure tensor, is performed. As can be seen in Figure 1.3, this creates a nice effect that is similar to oil paint.

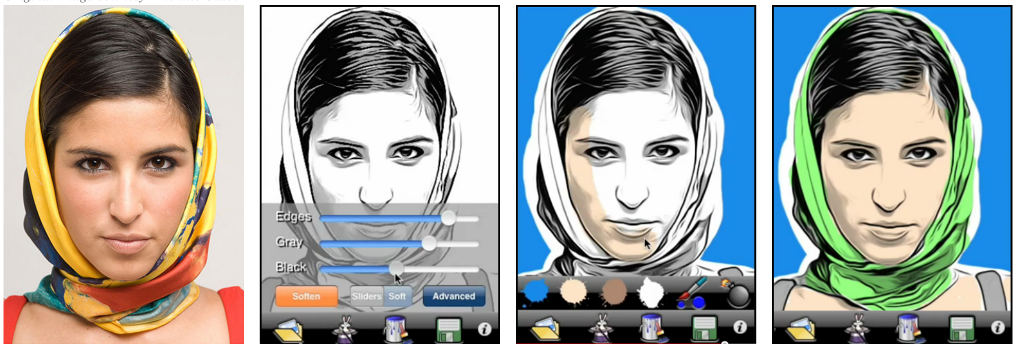
Second, the techniques may be applied directly to images and video using specialized


Ps File Edit Image Layer Select Filter Analysis 3D View Window Help
ESSENTIALS
DESIGN
PAINTING
CS Live
Navdeep Raj_504596529_7d798318f7_z.jpg @ 100% (RGB/8#)
Pixel Bender (100%) GPU Accelerated
OK
Cancel
COLOR
SWATCHES
STYLES
ADJUSTMENTS
MASKS
LAYERS
CHANNELS
PATHS
Background
Doc: 753.8K/753.8K

Process on GPU [✓]
Filters are loaded from the Pixel Bender Files folder.
OilPaint
Stylization 2.893
Cleanliness 6.32
Colorization 1.18
BrushScale 2
BrushContrast 1

Figure 1.3: *Screenshot of Adobe Photoshop CS5 executing the oil paint Pixel Bender filter.*

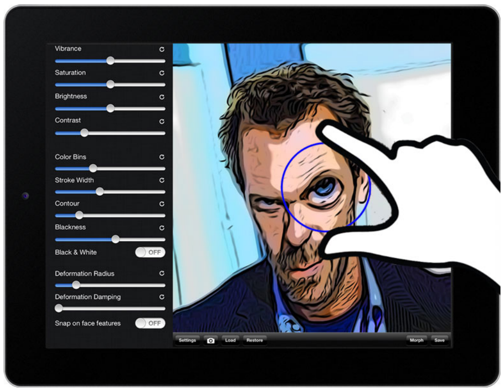
Original image courtesy Mickael Casol



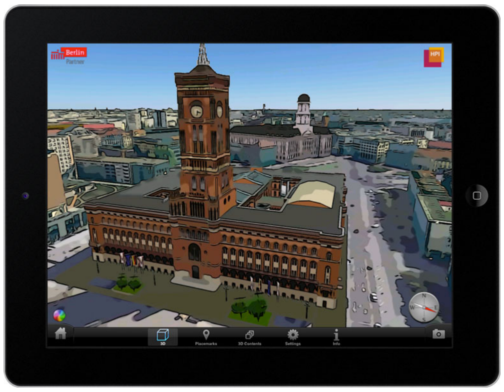
(a) (b) (c) (d)

Figure 1.4: *ToonPAINT by ToonFX, LLC. (a) Input image. (b) Selection of outlines and shading using the flow-based extended difference of Gaussians filter. The user can influence the outlines, mid-tones, and shadows. (c) Color regions are manually painted by the user. (d) Final result.*

tools and applications, thereby targeting casual users who do not have any specific artistic training. For these applications, it is of special interest that the algorithms presented in this thesis are suitable for real-time processing. This enables applications to provide immediate visual feedback when parameter changes occur, allowing for an iterative exploration of the technique's parameter space on a trial-and-error basis. To this end, it is important to abstract from the underlying technical control parameters and provide a set of meta-parameters that are intelligible to casual users. However, that topic is beyond the scope of this work. ToonPAINT by ToonFX LLC is an example of a successful productization of an application, utilizing the separable implementation of the flow-based difference of Gaussians filter discussed in Chapter 4. It is available for iOS and Android devices. Figure 1.4 shows the different steps a user takes to turn a photograph into a cartoon using ToonPAINT. First, the user selects outlines, mid-tones, and shadows. Second, the user colorizes the intermediate result using touch gestures similar to a coloring page for kids. This results in an experience that is more engaging than a fully automatic algorithm would provide. A detailed discussion of the design decisions behind ToonPAINT can be found in [Win12]. Figure 1.5 shows a screenshot of an application for the creation of caricatures from photographs. It



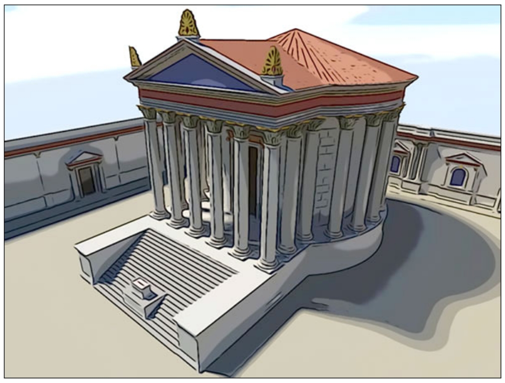
(a) *Caricature app prototype by F. Schlegel*



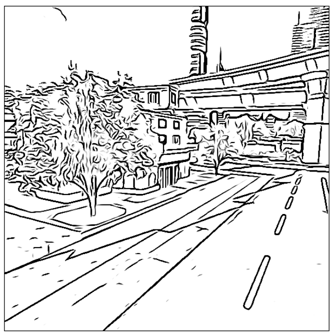
(b) *Cartoon-style 3D-rendering [DHK12]*

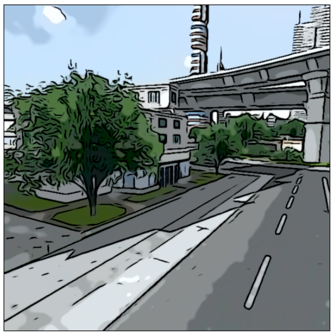
Figure 1.5: *Screenshots of application prototypes currently under development.*

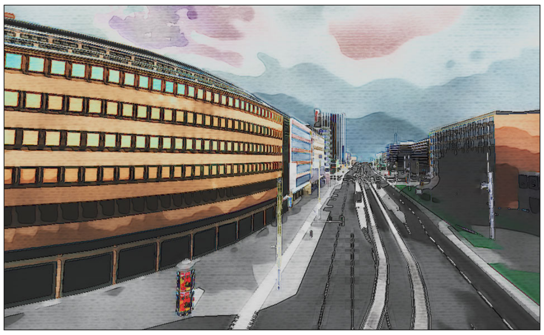
(a) *3D rendering*

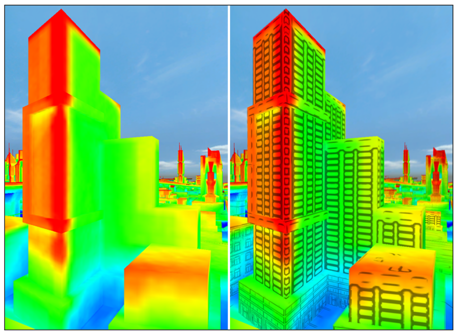

(b) *Stylized result [M7]*

(c) *3D rendering*


(d) *Edges*


(e) *Stylized result [M4]*


(f) *Watercolor rendering [Ric08]*


(g) *Thematic visualization [Eng+12]*

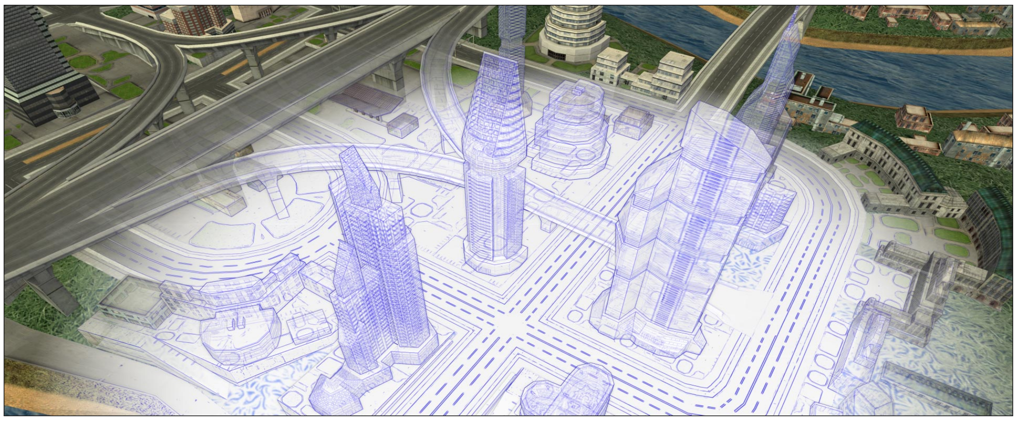

(h) *Blueprint rendering of a circular region of interest [M1]*

Figure 1.6: *The algorithms presented in this thesis may be used to post-process renderings of 3D scenes.*

utilizes the generalized cartoon pipeline (Section 4.6) for stylization, and touch gestures can be used to deform the image.

Finally, the techniques may be applied to three-dimensional renderings during the post-processing stage. In particular, the generalized cartoon pipeline (Section 4.6) has been shown to be useful for the visualization of three-dimensional city models [Eng+12; M4; Ric08; M7], due to its simplicity and efficiency. Effective communication of geospatial information has to take into account perceptional, cognitive, and graphical design issues in order to ensure a clear and efficient understanding of the information content [Job08]. These principles, which are known for two-dimensional depictions, also apply to dynamic three-dimensional presentations. Compared to well-known straightforward photorealistic depictions, these principles demand for specific perceptional, cognitive, and graphical designs. Figure 1.5(b) shows a screenshot of the client application of a service-based rendering system [DHK12]. This system performs the rendering and stylization on the server side, enabling the visualization of large models even on low-end devices. Figures 1.6(a) and 1.6(b) show a 3D model of a temple of Roman Cologne [M7], rendered in both a photorealistic and a non-photorealistic style, with the latter intending to communicate missing evidence in the reconstruction. Figures 1.6(c) to 1.6(e) show an additional example. Figure 1.6(f) shows a watercolor-style rendering of a three-dimensional city model that utilizes the orientation-aligned bilateral filter, discussed in Chapter 3, internally. Figure 1.6(g) shows a visualization of a solar potential analysis mapped to facade textures in the corresponding three-dimensional city model. On the right, the separable implementation of the flow-based difference of Gaussians filter has been used to emphasize the shape and spatial relations of the objects. Lastly Figure 1.6(h) shows an example in which the separable implementation of the flow-based difference of Gaussians filter is utilized to render a blueprint of a region of interest.

Chapter 2

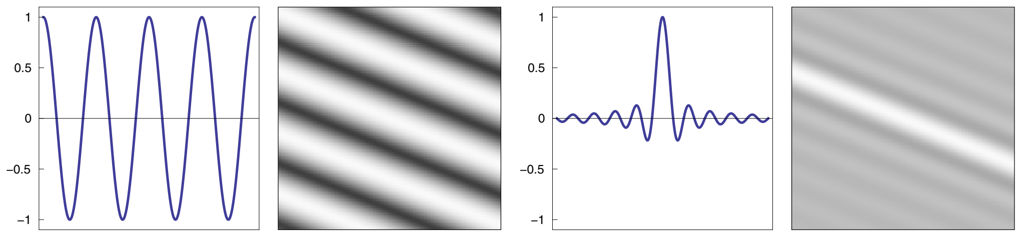
Local Structure Estimation with the Structure Tensor

Local orientation estimation refers to the task of estimating the dominant orientation in a local neighborhood of a pixel. It is a well-researched field and examples for popular approaches are methods based on quadrature filters [Knu89; GK95], steerable filters [FA91], the inertia tensor [BGW91; Big06], principle component analysis [FM02], polynomial expansion [Far02], the edge tangent flow [KLC07; KLC09] and the smoothed structure tensor [För86; Bro+06]. In this work, the smoothed structured tensor is utilized, since it provides an excellent trade-off between accuracy and computational cost, making it particularly suitable for real-time applications. This chapter provides a detailed discussion of the structure tensor and related techniques and forms the foundation for the subsequent chapters, which utilize the structure tensor to guide local neighborhood filters.

2.1 Simple Functions and Local Orientation

For arbitrary images it is not possible to define the notion of local orientation intrinsically. We therefore first consider images with the specific property of having only a single dominant orientation, as exemplified in Figure 2.1. Such images can be formally described as discretizations of simple functions and we will refer to them in the following as *simple* images. A function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is called *simple (of rank one)* if it can be represented as

$$f(x) = g\big(\langle n, x\rangle\big) , \tag{2.1}$$



(a) $g_1(t) = \cos t$ (b) $f_1(x) = g_1(\langle n, x\rangle)$ (c) $g_2(t) = \operatorname{sinc} t$ (d) $f_2(x) = g_2(\langle n, x\rangle)$

Figure 2.1: *Plots of cosine, sinus cardinalis, and the simple functions induced by them.*

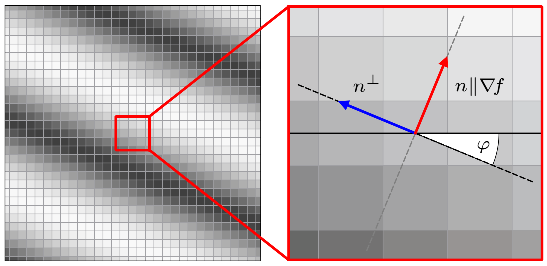


Figure 2.2: *The local orientation φ of a simple function $f = g(\langle n, x\rangle)$ is defined as the angle between the horizontal coordinate axis and the axis perpendicular to the direction of variation n. Since the gradient ∇f is parallel to the direction of variation, local orientation can be equivalently defined using the gradient.*

where $g\colon \mathbb{R} \to \mathbb{R}$ is a one-dimensional function, $n \in \mathbb{R}^2$ is a unit vector, and $\langle \cdot, \cdot \rangle$ denotes the Euclidean scalar product on $\mathbb{R}^2$. For locally non-constant functions g, the isophote curves (i.e., the level curves of constant gray value) of a simple function are parallel lines. This is easily seen by considering lines perpendicular to n, which can be described by $L(\beta) = a + \beta n^\perp$, where $a \in \mathbb{R}^2$ is a point and $n^\perp$ is the counter-clockwise rotation of n by 90 degrees. The values of a simple function f along such lines is then constant:

$$\begin{aligned} f\big(L(\alpha)\big) &= f\big(a + \beta n^\perp\big) = g\Big(\langle n, a + \beta n^\perp \rangle\Big) \\ &= g\Big(\langle n, a\rangle + \beta \underbrace{\langle n, n^\perp\rangle}_{=0}\Big) = f(a) \end{aligned} \tag{2.2}$$

Similarly, a simple function f restricted to lines $L = a + \beta n$ parallel to n, corresponds to g translated by a constant β_0:

$$f\big(L(\beta)\big) = f\big(a + \beta n\big) = g\Big(\underbrace{\langle n, a\rangle}_{=:\beta_0} + \beta \underbrace{\langle n, n\rangle}_{=1}\Big) = g(\beta_0 + \beta) \tag{2.3}$$

This shows that simple functions have, indeed, only a single dominant orientation, which motivates the definition of *local orientation* as the angle between the axis perpendicular to the dominant orientation and the horizontal axis (Figure 2.2). It is important to note that local orientation is a quantity with a periodicity of 180 degrees. This is due to the fact that $f(x)$ and $f(-x)$, which corresponds to a rotation by 180 degrees, both have the same dominant orientation and thus give rise to the same perpendicular axis. In order to better distinguish the two angular quantities, in the following *orientation* will refer to a periodicity of 180 degrees and *direction* to a periodicity of 360 degrees. Instead by an angle, local orientation will also often be described, for the sake of simplicity, by a unit vector representing the corresponding axis. In such cases, it is important to remember that the specific direction is almost always not well-defined.

Let us now consider the case that a simple function f is given and n and g are wanted. Since the gradient of a scalar field points in the direction of the greatest rate of increase, it follows that the gradient of a simple function is parallel to n. In order to find an expression for the gradient, we first calculate the partial derivatives using the chain rule:

$$\frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i} g\big(\langle n, x\rangle\big) = g'\big(\langle n, x\rangle\big) \cdot \frac{\partial}{\partial x_i} \langle n, x\rangle = g'\big(\langle n, x\rangle\big) \cdot n_i \; . \tag{2.4}$$

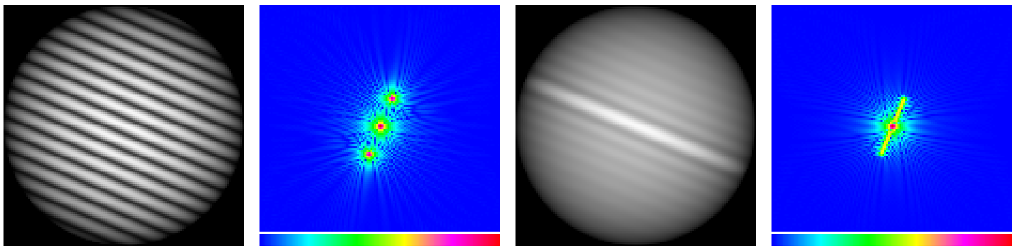

(a) $f_1 * W$ (b) $|F_1 \cdot W|$ (c) $f_2 * W$ (d) $|F_2 \cdot W|$

Figure 2.3: *Visualization of the magnitude of the discrete Fourier transform of the images shown in Figure 2.1 using a logarithmic scale. In order to avoid boundary artifacts, the functions have been multiplied with a cosine window function.*

This shows us that the gradient

$$\nabla f = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{pmatrix} = \begin{pmatrix} g'\big(\langle n, x\rangle\big) \cdot n_1 \\ g'\big(\langle n, x\rangle\big) \cdot n_2 \end{pmatrix} = g'\big(\langle n, x\rangle\big) \cdot n \tag{2.5}$$

equals the derivative of g times the vector n. Therefore, if f is not locally constant (i.e., g has non-vanishing derivative), then n can be derived from the normalized gradient $\nabla f / |\nabla f|$. At this point we have to be careful, however, since the normalized gradient only defines n up to a change of sign. After choosing a particular sign, n is well-defined and g is obtained as $g(t) = f(tn)$.

As in the spatial domain, simple functions exhibit a special structure in the frequency domain. In order to derive the Fourier transform of a simple function, let as before $f(x) = g(\langle n, x\rangle)$ and let A be the rotation matrix that rotates the first coordinate vector e_1 to n. Then

$$\begin{aligned} h(x) := f\big(A^T x\big) &= g\big(\langle n, A^T x\rangle\big) \\ &= g\big(\langle An, x\rangle\big) = g\big(\langle e_1, x\rangle\big) = g(x_1) = g(x_1) \cdot 1 \end{aligned} \tag{2.6}$$

is a simple function that varies only along the first coordinate axis. Hence, h is Cartesian separable, which means that it can be written as a product of two one-dimensional functions. The two functions are g and the constant unit function. Since the Fourier transform of Cartesian separable functions is given by the product of the Fourier transforms of the two one-dimensional functions, the Fourier transform of h is given by

$$\begin{aligned} H = \mathcal{F}(h)(\omega) &= \int_{\mathbb{R}^2} h(x)\, e^{-i\omega^T x}\, \mathrm{d}^2 x \\ &= \mathcal{F}(g)(\omega_1) \cdot \mathcal{F}(1)(\omega_2) = G(\omega_1) \cdot \delta(\omega_2)\ , \end{aligned} \tag{2.7}$$

where G denotes the Fourier transform of g. This shows that the Fourier transform is given by the product of the Fourier transform of g depending on the first coordinate multiplied with the Dirac delta function depending on the second coordinate. In other words, the whole energy is fully concentrated along the first coordinate axis. Since $f = h(Ax)$, the Fourier transform of f can be derived using the substitution theorem for multiple variables

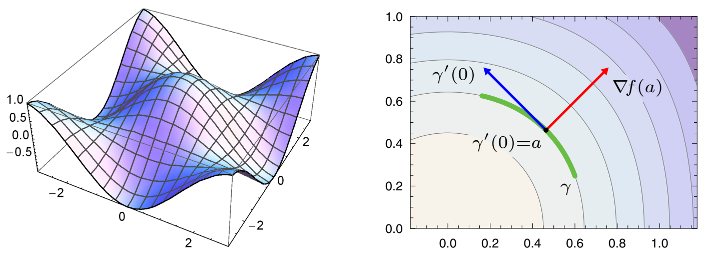


Figure 2.4: *Plots of the function $f(x) = \cos(x_1)\cos(x_2)$. On the left a 3D plot is shown. The contour plot on the right shows various level sets and illustrates that for regular level sets the gradient is orthogonal to the tangent of curves locally parameterizing the level set.*

and the fact that by construction the columns of A are given by n and $n^\perp$:

$$\begin{aligned} F(\omega) &= |\det A|^{-1} \cdot H\big((A^T)^{-1}\omega\big) \\ &= H(A\omega) \\ &= G\big(\langle n, \omega\rangle\big) \cdot \delta\big(\langle n^\perp, \omega\rangle\big) \end{aligned} \tag{2.8}$$

Consequently, all the energy of a simple function is distributed along a line passing through the origin and oriented parallel to its variation. This observation is experimentally demonstrated in Figure 2.3 and forms the foundation of the inertia tensor method discussed in Section 2.4.5.

2.2 Grayscale and Color Image Gradients

In the previous section we saw how to define local orientation for a simple function. An interesting observation was that the gradient of a simple function is orthogonal to the isophote curves. A similar result can be shown for regular level sets of smooth functions and is illustrated in Figure 2.4. Let $U \subset \mathbb{R}^2$ be an open subset, let $f\colon U \to \mathbb{R}$ be a smooth function, and let $c \in \mathbb{R}$ be any value. Then the inverse image

$$f^{-1}(c) = \{\, x \in U : f(x) = c \,\} \tag{2.9}$$

of c under f is called a *level set* or *contour line* of f. A point $x \in U$ is a *regular point* of f if $\nabla f(x) \neq 0$. If every point of a level set $f^{-1}(c)$ is regular, then c is called a *regular value*. In this case $f^{-1}(c)$ is said to be a *regular level set*.

Let us suppose that c is a regular value. From the regular value theorem [Kön04; Lee03] then follows that the level set $f^{-1}(c)$ is a one-dimensional submanifold. Hence, for every $a \in f^{-1}(c)$ there exists a diffeomorphism $\gamma\colon I \to f^{-1}(c)$ from an open interval $I \subset \mathbb{R}$ to an open neighborhood of a in $f^{-1}(c)$. In other words, there exists a smooth curve γ that is locally parameterizing the level set at a. Without loss of generality we can assume that $\gamma(0) = a$. Since $f(\gamma(t)) = c$ for all $t \in I$, it follows that the differential of $f \circ \gamma$ must vanish. Using the chain rule we can thus conclude that

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}(f \circ \gamma)(0) = \big\langle\, \nabla f(a), \gamma'(0) \,\big\rangle\,, \tag{2.10}$$

which shows that the gradient of f at a is orthogonal to the tangent vector of γ at 0.

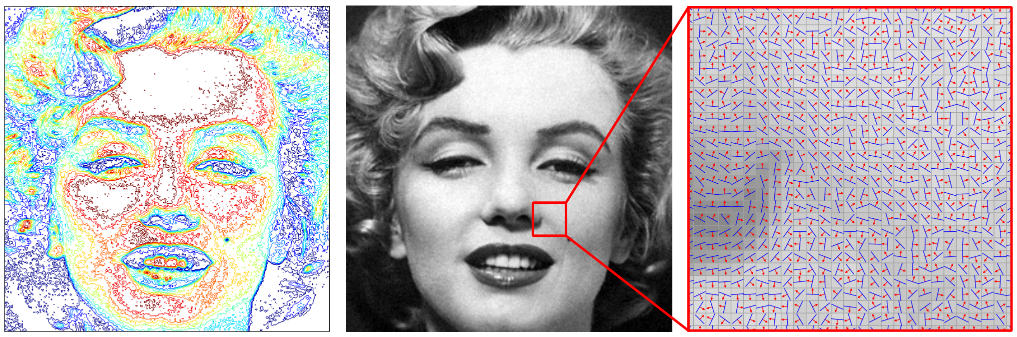


Figure 2.5: *On the left, a contour plot of a selection of level sets of the image in the middle is shown. On the right, the gradient vector field (red), which has been computed using central differences, and the orientation field (blue), which has been derived from the gradient vectors by a rotation of 90 degrees, are shown for the marked image region.*

This observation generalizes our earlier result for simple functions and motivates to derive local orientation for arbitrary images from the image gradient. Rotating the gradients by 90 degrees results in a vector field that represents the axes perpendicular to the gradients. Since these axes can also be interpreted as tangent spaces of the isophote curves, we will in the following often refer to them as *tangent vectors*. Notice that in contrast to the gradient, which has a well-defined direction, tangent vectors only represent an orientation. Therefore, their sign is not well-defined and special care is required to handle the sign ambiguity.

Unfortunately, the computation of image gradients is a difficult task. It is well-known that differentiation is an ill-posed problem [EHN96], because small perturbations in the data may result in large errors in the derivative. This can be illustrated by a simple example [BPT88; EHN96]. Let $f(t)$ be some one-dimensional function and let

$$g(t) = f(t) + \varepsilon \sin(\Omega t) \tag{2.11}$$

be a small perturbation by a sine function. By adjusting ε, we can make f and g arbitrarily close. Nevertheless, the differential

$$g'(t) = f'(t) + \varepsilon\Omega \cos(\Omega t) \tag{2.12}$$

will be very different from $f'(t)$ for large Ω, which shows that differentiation amplifies high frequency information such as noise. Due to this, differentiation is typically combined with some form of regularization [BPT88]. This is an important practical issue, since real world images often contain noise and other artifacts (e.g., introduced by image compression codecs), making the computation of differentials challenging. In Figure 2.5, an image of moderate quality, a selection of its level sets, and an excerpt of the gradient field are shown. As can be seen, the level sets are very noisy and obviously do not form smooth curves. A similar situation can be observed for the gradient field. Especially in low contrast regions, where the signal-to-noise ratio is low, the gradient field is almost random.

There are advanced differentiation techniques available in the research literature (e.g., formulated as variational problem). In general, however, these have high computational complexity. Fortunately, since our main concern is to derive a suitable orientation field, there is also the option to use a less sophisticated technique for differentiation and to perform a

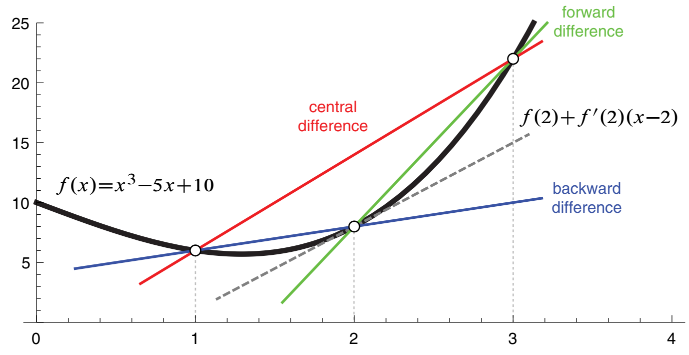


Figure 2.6: *Illustration of finite difference approximations of the derivative of a function. Shown are the forward, backward, and central finite differences for the cubic polynomial $f(x) = x^3 - 5x + 10$ at $x = 2$ with step size $h = 1$.*

regularization of the orientation field. To this end the following sections discuss popular techniques for estimating the image gradient, and their suitability for orientation estimation. Section 2.4 then discusses techniques to smooth the corresponding orientation field. As we will see later, this results in a computationally highly effective approach capable of producing orientation fields at suitable quality.

2.2.1 Finite Differences

Let us assume that $f\colon \mathbb{R}^2 \to \mathbb{R}$ is a smooth function (i.e., infinitely differentiable). Along one of the coordinate axis, we can then approximate f locally at some point $a \in \mathbb{R}^2$ by the corresponding Taylor series [Kön04]:

$$f(a+h) = f(a) + h\frac{\partial f}{\partial x_i}(a) + \frac{h^2}{2}\frac{\partial^2 f}{\partial x_i^2}(a) + \frac{h^3}{6}\frac{\partial^3 f}{\partial x_i^3}(a) + \cdots \tag{2.13}$$

$$f(a-h) = f(a) - h\frac{\partial f}{\partial x_i}(a) + \frac{h^2}{2}\frac{\partial^2 f}{\partial x_i^2}(a) - \frac{h^3}{6}\frac{\partial^3 f}{\partial x_i^3}(a) + \cdots \tag{2.14}$$

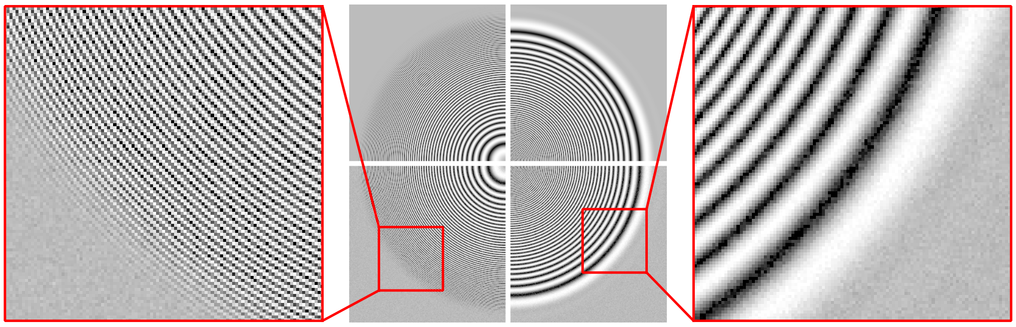
With h denoting a small step in direction of the i-th coordinate axis. Rearranging we obtain:

$$\frac{\partial f}{\partial x_i}(a) = \frac{f(a+h) - f(a)}{h} - \underbrace{\frac{h}{2}\frac{\partial^2 f}{\partial x_i^2}(a) - \frac{h^2}{6}\frac{\partial^3 f}{\partial x_i^3}(a) - \cdots}_{\text{truncation error } O(h)} \tag{2.15}$$

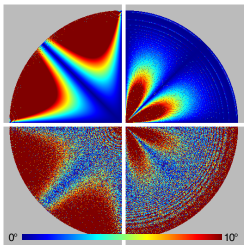
$$\frac{\partial f}{\partial x_i}(a) = \frac{f(a) - f(a-h)}{h} + \underbrace{\frac{h}{2}\frac{\partial^2 f}{\partial x_i^2}(a) - \frac{h^2}{6}\frac{\partial^3 f}{\partial x_i^3}(a) + \cdots}_{\text{truncation error } O(h)} \tag{2.16}$$

Truncating after the first term, we get for the first partial derivative two approximations that are referred to as *forward* and *backward* differences, respectively. Moreover, averaging Equations (2.15) and (2.16), we see that the terms with odd partial derivatives cancel out, resulting in a third approximation

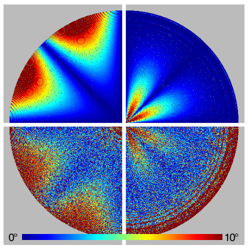
$$\frac{\partial f}{\partial x_i}(a) = \frac{f(a+h) - f(a-h)}{2h} - \underbrace{\frac{h^2}{3}\frac{\partial^3 f}{\partial x_i^3}(a) - \cdots}_{\text{truncation error } O(h^2)} , \tag{2.17}$$



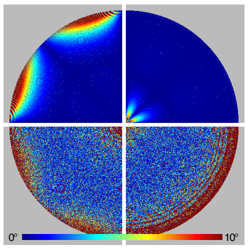
(a) *Test image*



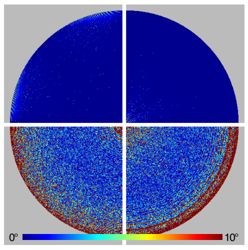
(b) *Central differences*



(c) *Sobel filter*



(d) *Optimized* 3 × 3



(e) *Optimized* 5 × 5

Figure 2.7: *Evaluation of the rotation invariance of different derivative operators. In the top row, the test pattern is shown. (Due to the high frequencies present in the pattern it may not be reproduced correctly on screen or in print.) Top-left of the test pattern is a standard zone plate pattern approaching approximately 0.9 of the Nyquist frequency, where curvature decreases with increasing frequency. Top-right is an inverted zone plate pattern, where curvature increases with increasing frequency. The bottom part of the test pattern has been corrupted with Gaussian noise of variance* $v = 0.001$. *In the bottom row, the angular errors for different derivative operators applied to the test pattern are shown. Notice the strong dependence of the error on the angle for central differences and Sobel filter. The optimized filters perform much better in this respect. However, all approaches are highly sensitive to noise.*

with quadratic truncation error that is referred to as *central* difference. Figure 2.6 illustrates the different forms of finite differences.

For a grayscale image I and step size $h = 1$, the central differences at some point (x, y) along the coordinate axes are given by:

$$\begin{aligned} \delta_x I(x,y) &= \frac{I(x+1,y) - I(x-1,y)}{2} \\ \delta_y I(x,y) &= \frac{I(x,y+1) - I(x,y-1)}{2} \end{aligned} \tag{2.18}$$

Alternatively, the central differences can also be expressed as discrete convolutions $\delta_x I = D_{2x} * I$ and $\delta_y I = D_{2y} * I$, where D_{2x} and D_{2y} are the following convolution masks:

$$D_{2x} = \frac{1}{2}\begin{pmatrix} +1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad D_{2y} = \frac{1}{2}\begin{pmatrix} +1 \\ 0 \\ -1 \end{pmatrix} \tag{2.19}$$

In Figure 2.7 an evaluation of different popular derivative operators for a test pattern with varying frequencies and curvatures is shown. As can be seen, central differences produce comparatively large errors in high frequency regions of the patterns. Also clearly noticeable is a strong dependence of the error on the angle. Moreover, it can be observed that even small amounts of noise lead to large errors in the estimated orientation.

2.2.2 Gaussian Derivatives

If nothing is known about the image and its noise characteristics, using a Gaussian filter for regularization is a good choice (see [TP84] for a more detailed discussion). Since convolution commutes with differentiation we have

$$\frac{\partial}{\partial x_i}\big(G_\sigma * I\big) = \frac{\partial G_\sigma}{\partial x_i} * I \,, \tag{2.20}$$

where I denotes the image and

$$G_\sigma(x_1, x_2) = \frac{1}{2\pi\sigma^2}\exp\Big(-\frac{x_1^2 + x_2^2}{2\sigma^2}\Big) \tag{2.21}$$

is a bivariate Gaussian with standard deviation σ. Thus, instead of using finite differences, a closed solution of the derivative of the Gaussian can be calculated

$$\frac{\partial G_\sigma}{\partial x_i} = -\frac{x_i}{2\pi\sigma^4}\exp\Big(-\frac{x_1^2 + x_2^2}{2\sigma^2}\Big) \tag{2.22}$$

and convolved with the image. This operation is known as *Gaussian derivative filter* and is also interesting from a computational point of view, since it is separable like the Gaussian filter. The standard deviation σ of the filter must be chosen carefully. If it is too large, important image features may be removed by the Gaussian smoothing. This happens even for highly anisotropic image regions. For instance, if the standard deviation is chosen, as shown in Figure 2.8, such that the frequency of the pattern is outside the passband of the Gaussian, then the smoothed image will be nearly constant and the estimated orientation highly inaccurate. Thus, the strength of regularization that can be achieved by regularizing with a Gaussian filter is limited, since strong regularization is only possible if image detail is sacrificed. Nevertheless, an interesting observation can be made. The Gaussian derivative filter with standard deviation $\sigma = 0.68$ (Figure 2.8(a)) and $\sigma = 0.867$ (Figure 2.8(b))—the σ were chosen to match a 3×3 and 5×5 filter, respectively—perform clearly better than central differences (Figure 2.7(b)). This suggests seeking for specialized derivative operators. Moreover, notice that in Figure 2.8(c) excellent results are obtained if no noise is present. Adding just a little noise, however, results in large errors, as shown in the bottom part of the test pattern.

2.2.3 Optimized Derivative Operators

In the classical image processing literature a number of different derivative operators, such as the Sobel, Roberts, or Prewitt filter, have been proposed [Pra01]. In particular, the Sobel filter, given by the convolution masks

$$S_x = \frac{1}{8}\begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad S_y = \frac{1}{8}\begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, \tag{2.23}$$

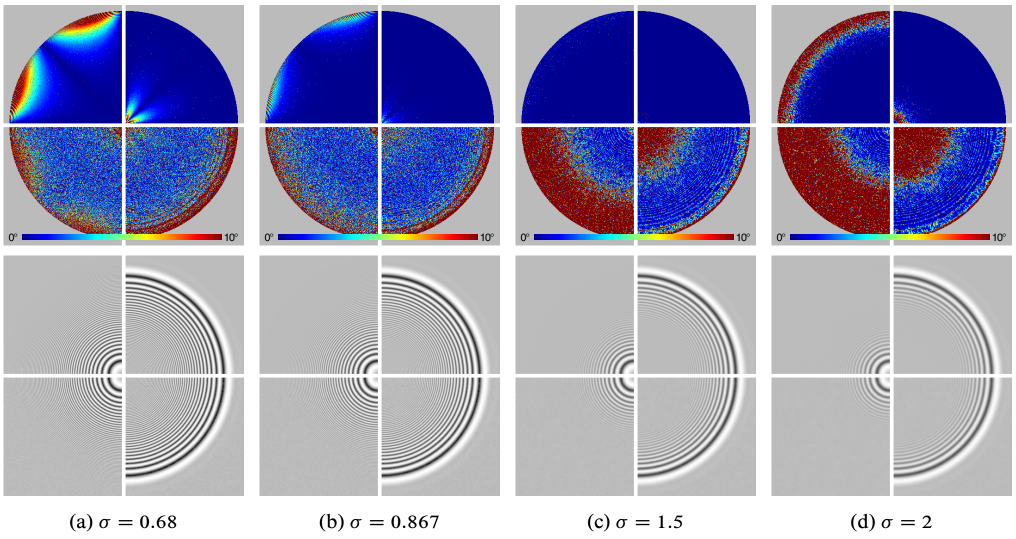


Figure 2.8: *The top row shows the angular error produced by a Gaussian derivative filter applied to the test pattern from Figure 2.7 for different standard deviations. The bottom row shows the result of smoothing the ring pattern with the corresponding Gaussian filter, illustrating that strong regularization removes high frequency structures and makes the orientation estimation impossible.*

is an often seen choice in image-based artistic rendering and other image processing applications. From the perspective of orientation estimation, however, the Sobel filter is not optimal. This can be seen in Figure 2.7(c). Although the Sobel filter performs clearly better than central differences, it still shows large errors that strongly depend on the angle.

Specialized derivative operators optimized for rotation invariance have been investigated by Simoncelli [Sim94], Farid and Simoncelli [FS97; FS04] and Scharr et. al. [SKJ97; JSK99; Sch00]. Basic idea behind these approaches is to compose a multi-dimensional derivative filter from a one-dimensional derivative filter and additional one-dimensional filters that perform smoothing perpendicular to the derivative. The parameters of the filters were found by an optimization process. The best results thereby were achieved by Scharr et. al., who directly minimized the angular error.

Since a derivative operator should not response to constant signals, the sum of the filter coefficients must be zero. In addition, a derivative operator with $2R+1$ coefficients must have odd symmetry with the center coefficient being zero [Jäh05]:

$$\begin{pmatrix} +d_R & \dots & +d_1 & 0 & -d_1 & \dots & -d_R \end{pmatrix} \tag{2.24}$$

Moreover, constraining the slope of the transfer function at the origin to the slope of the ideal transfer function, as suggested by Jähne et. al. [JSK99], results in the additional constraint $d_1 = 1/2 - \sum_{k=2}^{R} k d_k$. In particular, it follows that for $R = 1$ the only derivative operator fulfilling these constraints are the central differences. A smoothing filter should be even symmetric

$$\begin{pmatrix} b_R & \dots & b_1 & b_0 & b_1 & \dots & b_R \end{pmatrix} \tag{2.25}$$

and normalized, which is equivalent to the constraint $b_0 = 1 - \sum_{k=1}^{R} b_k$.

Based on these constraints an ansatz for a two-dimensional 3×3 derivative filter along the horizontal axis is given by the following convolution mask:

$$D_x^3 = \begin{pmatrix} +\frac{1}{2} & 0 & -\frac{1}{2} \end{pmatrix} * \begin{pmatrix} b_1 \\ b_0 \\ b_1 \end{pmatrix} \tag{2.26}$$

As usual, $*$ refers to the convolution operator and should not be confused with matrix multiplication. Similarly, the derivative in direction of the vertical axis is defined. In this model, only b_1 is a free parameter. Optimizing b_1 for rotation invariance, Jähne et. al. [JSK99] obtained:

$$b_1 = 46.84/256, \quad b_0 = 1 - 2b_1 = 162.32/256 \tag{2.27}$$

Correspondingly, an ansatz for a 5×5 derivative filter is given by:

$$D_x^5 = \begin{pmatrix} +d_2 & +d_1 & 0 & -d_1 & -d_2 \end{pmatrix} * \begin{pmatrix} b_2 \\ b_1 \\ b_0 \\ b_1 \\ b_2 \end{pmatrix} \tag{2.28}$$

In this case b_1, b_2, and d_2 are free parameters for which Jähne et al. [JSK99] obtained by optimization:

$$\begin{gathered} b_2 = 5.91/256, \quad b_1 = 61.77/256, \quad b_0 = 1 - 2(b_1 + b_2) = 120.64/256 \\ d_2 = 21.27/256, \quad d_1 = 1/2 - 2d_2 = 85.46/256 \end{gathered} \tag{2.29}$$

As can be seen in Figure 2.7, the optimized derivative filters clearly outperform central differences and Sobel filter. Thereby the optimized 5×5 filter achieves the best results, even in regions where the pattern is close to the Nyquist frequency. However, as a 5×5 filter it is also considerable more expensive from a computational point of view, and therefore for most applications the 3×3 filter is generally the preferable choice. The results of the optimized 5×5 are in line with the results we obtained earlier for the Gaussian derivative filter (Figure 2.7). It should be noted, however, that the implementation of the Gaussian derivative filter uses a larger filter size than 5×5. Also apparent from Figure 2.7 is that both optimized filters are highly sensitive to even small amounts of noise. This issue will be later addressed in Section 2.4.

2.2.4 Multi-Image Gradient

So far in our treatment of image gradients we have only considered grayscale images. For color images the situation is more complex. This can be illustrated by a simple example. Consider an RGB color image, where red, green, and blue color channels are given by linear gradients along different directions (Figure 2.9). The color image, however, has no obvious distinguished direction.

An early attempt to define a generalized image gradient for color images goes back to Di Zenzo [Di 86]. For a real-valued function, the gradient points (if nonzero) in direction of the largest change, which is the direction where the directional derivative exhibits a

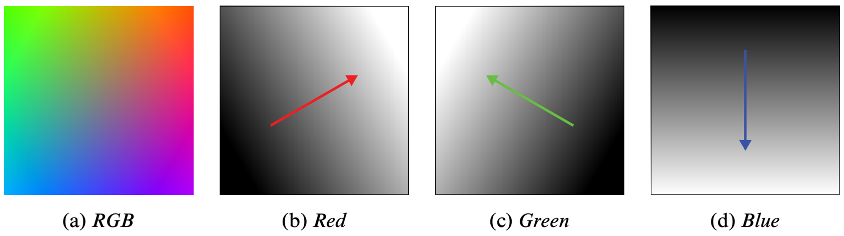


Figure 2.9: *Example for an RGB image (a), where the individual color channels (b)–(d) have distinct gradient directions.*

maximum. This property can be generalized to real-valued maps. However, since we are working with vector-valued data, we have to use a metric to compare different function values. To this end, let us suppose that a color image is given by a smooth map $f\colon \mathbb{R}^2 \to \mathbb{R}^m$ with $m \geq 2$, and let $a \in \mathbb{R}^2$ be any point. Moreover, let $h = (\cos\varphi, \sin\varphi)^T$ be a unit vector defined by the angle φ and let $\varepsilon > 0$. Then

$$F_\varepsilon(\varphi) = \left| \frac{f(a+\varepsilon h) - f(a)}{\varepsilon} \right|^2 \tag{2.30}$$

measures the squared normalized difference between the image values at a and $a + \varepsilon h$, where the squared Euclidean norm is used to avoid the square root and simplify subsequent computations. Making ε infinitesimal small by taking the limit $\varepsilon \to 0$ we get

$$F(\varphi) = \lim_{\varepsilon\to 0} F_\varepsilon(\varphi) = \lim_{\varepsilon\to 0} \left| \frac{f(a+\varepsilon h) - f(a)}{\varepsilon} \right|^2 = \left|\partial_h f(a)\right|^2 , \tag{2.31}$$

where

$$\partial_h f(a) = \lim_{\varepsilon\to 0} \frac{f(a+\varepsilon h) - f(a)}{\varepsilon} = J_f(a)\, h\, , \tag{2.32}$$

is the directional derivative of f at a in direction h, which can also be expressed as the Jacobi matrix

$$J_f(a) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \frac{\partial f_1}{\partial x_2}(a) \\ \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \frac{\partial f_m}{\partial x_2}(a) \end{pmatrix} \tag{2.33}$$

of f at a multiplied by h. Substituting $(\cos\varphi, \sin\varphi)^T$ for h and introducing the abbreviations E, F, and G we have:

$$\begin{aligned} F(\varphi) = \left|\partial_h f(a)\right|^2 = \left|J_f(a)\, h\right|^2 &= \sum_{i=1}^m \left| \langle \nabla f_i(a), h \rangle \right|^2 \\ &= \sum_{i=1}^m \left| \frac{\partial f_i}{\partial x_1}(a)\cdot\cos\varphi + \frac{\partial f_i}{\partial x_2}(a)\cdot\sin\varphi \right|^2 \\ &= \underbrace{\sum_{i=1}^m \left(\frac{\partial f_i}{\partial x_1}(a)\right)^2}_{=:E} \cos^2\varphi + 2 \underbrace{\sum_{i=1}^m \left(\frac{\partial f_i}{\partial x_1}(a)\frac{\partial f_i}{\partial x_2}(a)\right)}_{=:F} \sin\varphi\, \cos\varphi + \underbrace{\sum_{i=1}^m \left(\frac{\partial f_i}{\partial x_2}(a)\right)^2}_{=:G} \sin^2\varphi \\ &= E\cos^2\varphi + 2F\sin\varphi\,\cos\varphi + G\sin^2\varphi \end{aligned} \tag{2.34}$$

Substituting the following identities

$$\sin^2\varphi = \frac{1-\cos 2\varphi}{2}\,, \qquad \cos^2\varphi = \frac{1+\cos 2\varphi}{2}\,, \qquad \sin\varphi\,\cos\varphi = \frac{\sin 2\varphi}{2}\,, \tag{2.35}$$

yields the following simplification

$$\begin{aligned} F(\varphi) &= \frac{E\,\big(1+\cos 2\varphi\big) + 2F\,\sin 2\varphi + G\,\big(1-\cos 2\varphi\big)}{2} \\ &= \frac{E+G}{2} + \frac{E-G}{2}\cos 2\varphi + F\,\sin 2\varphi\ . \end{aligned} \tag{2.36}$$

In order to find the direction of largest change, we have to find φ that maximizes $F(\varphi)$. As can be easily seen, $F(\varphi + k\pi) = F(\varphi)$ for $k \in \mathbb{Z}$. Without loss of generality, we can therefore assume that φ lies in the interval $(-\pi/2, \pi/2]$. If $F(\varphi)$ is constant, then $E = G$, $F = 0$, and there is no direction φ that maximizes $F(\varphi)$. So let us assume F is non-constant. A necessary condition for an extremum is that the derivative

$$\frac{\mathrm{d}F}{\mathrm{d}\varphi} = -(E-G)\sin 2\varphi + 2F\cos 2\varphi \tag{2.37}$$

vanishes. Setting $\mathrm{d}F/\mathrm{d}\varphi$ to zero and solving for φ we get:

$$\tan 2\varphi = \frac{2F}{E-G} \tag{2.38}$$

This equation has two solutions. If $\varphi_0 \in [-\pi/2, -\pi/2)$ is a solution, then also $\varphi \pm \pi$ is a solution. F attains a maximum for one of these and a minimum for the other. In Section 2.3.3 this will be discussed further and it will be shown that the maximum is always attained at

$$\varphi_{\max} = \frac{1}{2}\,\mathrm{atan2}\big(2F, E-G\big)\,, \tag{2.39}$$

where atan2 denotes the inverse tangent with values in $[-\pi, +\pi)$. By combining the squared maximum rate of change $F(\varphi)$ and orientation $\varphi_{\max}$, we get Di Zenzo's multi-image gradient:

$$\nabla_{\text{Di Zenzo}} f(a) = \sqrt{F(\varphi_{\max})}\begin{pmatrix}\cos\varphi_{\max}\\ \sin\varphi_{\max}\end{pmatrix}. \tag{2.40}$$

From the geometric perspective, as pointed out by Cumani [Cum91], a better approach is to consider the image embedded as a graph in $\mathbb{R}^{2+m}$. We will review this in more detail in Section 2.4.2.

2.3 Symmetric Positive Semidefinite 2 × 2 Matrices

The structure tensor introduced in the next section is a field of positive semidefinite 2×2 matrices. In this section some properties of such matrices will be reviewed. Although most results are valid in a more general setting [Jän08; Fis08; Lan87], most of the discussion is restricted to 2×2 matrices for the sake of simplicity. An excellent treatment of advanced linear algebra topics, such as the notion of definiteness, is given in the book by Horn and Johnson [HJ85]. A detailed discussion of 2×2 matrices can be found in [Bli96] and complements the topics presented here.

2.3.1 Eigenanalysis of 2×2 Matrices

Let $A \in \mathbb{R}^{n \times n}$ be a matrix. A scalar $\lambda \in \mathbb{R}$ is called an *eigenvalue* of A if there exists a non-zero vector $v \in \mathbb{R}^n$ satisfying $Av = \lambda v$. Every vectors satisfying this equation is called an *eigenvector* of A associated to the eigenvalue λ. The linear subspace

$$\begin{aligned} \operatorname{Eig}(A, \lambda) &= \left\{ v \in \mathbb{R}^n : Av = \lambda v \right\} \\ &= \left\{ v \in \mathbb{R}^n : (A - \lambda I)v = 0 \right\} = \operatorname{Ker}(A - \lambda I) \end{aligned} \tag{2.41}$$

defined by all vectors satisfying $Ax = \lambda x$ is called the *eigenspace* of A corresponding to the scalar λ. If λ is an eigenvalue of A then $\operatorname{Eig}(A, \lambda)$ is the union of all the eigenvectors and the null vector. Hence, λ is an eigenvalue of A if and only if $\operatorname{Eig}(A, \lambda) \neq \{0\}$, that is, $A - \lambda I$ has a non-trivial kernel, which is equivalent to $A - \lambda I$ being singular. Since a matrix is singular exactly if its determinant is zero, it follows that λ is an eigenvalue of A if and only if $\det(A - \lambda I) = 0$. The determinant $\det(A - \lambda I)$ is an n-dimensional polynomial with variable λ and is called the *characteristic polynomial* of A. The eigenvalues of A can thus be found by finding the roots of the characteristic polynomial. This result is mainly of theoretical importance. For example, it shows that every $n \times n$ matrix has exactly n (possibly complex and not necessarily distinct) eigenvalues. Moreover, using the properties of the determinant it can be shown that similar matrices have the same characteristic polynomial. Hence, A and $S^{-1}AS$ have the same eigenvalues for non-singular matrices $S \in \operatorname{GL}(\mathbb{R}, n)$. In practice the characteristic polynomial is rarely used for computing the eigenvalues, because even if the eigenvalue problem itself is well-conditioned finding the roots of the characteristic polynomial may be ill-conditioned.

For 2×2 matrices, fortunately, finding the roots of the characteristic polynomial leads to a closed form solution that can be implemented in a, for our purposes, sufficiently numerically stable way, which will be discussed in detail in Section 2.3.4. To this end, let

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{2.42}$$

be a two-dimensional square matrix. Then the characteristic polynomial of A is given by:

$$\begin{aligned} \det(\lambda I - A) &= \begin{vmatrix} a - \lambda & b \\ c & d - \lambda \end{vmatrix} \\ &= (a - \lambda)(d - \lambda) - bc = \lambda^2 - (a + d)\lambda + ad - bc \\ &= \lambda^2 - \operatorname{tr}(A)\lambda + \det(A) \end{aligned} \tag{2.43}$$

Setting the characteristic polynomial equal to zero and solving for λ using the monic form of the quadratic formula then yields the following eigenvalues:

$$\begin{aligned} \lambda_{1,2} &= \frac{a + d \pm \sqrt{(a + d)^2 - 4(ad - bc)}}{2} \\ &= \frac{\operatorname{tr}(A) \pm \sqrt{\operatorname{tr}(A)^2 - 4\det(A)}}{2} = \frac{\operatorname{tr}(A) \pm \sqrt{\Delta(A)}}{2} . \end{aligned} \tag{2.44}$$

Depending on the value of the discriminant $\Delta(A) = \operatorname{tr}^2(A) - 4\det(A)$, we have to distinguish three different cases. If $\Delta(A) > 0$, then there are two distinct real eigenvalues. If $\Delta(A) = 0$, there is a single real eigenvalue with multiplicity two. And finally, if $\Delta(A) < 0$, there are two complex conjugate eigenvalues. Since we are only interested in real eigenvalues, we will assume $\Delta(A) \geq 0$ in the following.

From the last equality of Equation (2.44), we can explicitly derive the relation between eigenvalues, trace, and determinant of a square matrix:

$$\lambda_1 + \lambda_2 = \operatorname{tr}(A) = a + d \tag{2.45}$$

$$\lambda_1 \cdot \lambda_2 = \det(A) = ad - bc \tag{2.46}$$

Since the eigenvalues are similarity invariants, it follows that trace and determinant are also similarity invariants.

In order to find the eigenvectors of A, we have to compute the eigenspaces $\operatorname{Eig}(A, \lambda_1)$ and $\operatorname{Eig}(A, \lambda_2)$ by solving the following system of linear equations for $\lambda \in \{\lambda_1 \lambda_2\}$:

$$(A - \lambda I)v = \begin{pmatrix} a-\lambda & b \\ c & d-\lambda \end{pmatrix}\begin{pmatrix} v_x \\ v_y \end{pmatrix} = 0 \quad \Longleftrightarrow \quad \begin{aligned} (a-\lambda)v_x + bv_y &= 0 \\ cv_x + (d-\lambda)v_y &= 0 \end{aligned} \tag{2.47}$$

By construction $A - \lambda I$ is singular and therefore the two equations are linearly dependent. Hence, a solution of one equation will also be a solution of the other. Apparently,

$$v_1(\lambda) = \begin{pmatrix} b \\ \lambda - a \end{pmatrix} \quad \text{and} \quad v_2(\lambda) = \begin{pmatrix} \lambda - d \\ c \end{pmatrix} \tag{2.48}$$

are solutions for the first and second equation. For $b, c \neq 0$, both vectors are non-zero and indeed differ only by a constant scalar multiple. This can be seen by observing that the ratios of their coordinates are equal, which directly follows from the assumption that λ is an eigenvalue of A:

$$\frac{b}{\lambda - d} = \frac{\lambda - a}{c} \quad \Longleftrightarrow \quad (\lambda - a)(\lambda - d) - bc = \det(\lambda I - A) = 0 \tag{2.49}$$

If $b = 0$ or $c = 0$ then

$$\lambda_{1,2} = \frac{a + d \pm \sqrt{(a-d)^2}}{2} = \frac{a + d \pm |a - d|^2}{2}\,, \tag{2.50}$$

and thus $\lambda_1 = \max\{a, d\}$ and $\lambda_2 = \min\{a, d\}$. Hence, v_1 or v_2 might be zero and therefore no valid eigenvector. For example, if $b = 0$ and $\lambda = a$ then $v_1(\lambda) = 0$. Assuming $\lambda_1 \neq \lambda_2$, a simple strategy to pick valid eigenvectors is to choose v_1 for $b \neq 0$ and v_2 for $c \neq 0$. If both b and c happen to be zero, then A is a diagonal matrix and the eigenvectors are given by the corresponding canonical basis vectors. Finally, if $\lambda_1 = \lambda_2$, then any two vectors spanning $\mathbb{R}^2$ can be chosen.

2.3.2 Symmetric 2 × 2 Matrices

Let $A \in \mathbb{C}^{n \times n}$ be a matrix and let A^* denote its complex conjugate transpose. A matrix that commutes with its conjugate transpose, that is $A^*A = AA^*$, is said to be *normal*. If $AA^* = I$, then A is said to be *unitary*. Moreover, if $A = A^*$, then A is said to be *hermitian*. A unitary matrix in which all entries are real is called *orthogonal* and an hermitian matrix in which all entries are real is called *symmetric*. Or more specifically, a real matrix is orthogonal if $AA^T = I$ and it is symmetric if $A = A^T$.

Since $AA^T = A^2 = A^TA$ holds for a symmetric matrix, it follows that a symmetric matrix is especially a normal matrix. This means the following interesting properties of normal matrices also apply directly to symmetric matrices. Firstly, eigenvectors associated to distinct eigenvalues are orthogonal. Secondly, normal matrices are unitarily diagonalizable; that is, there exists a unitary matrix U and a diagonal matrix Λ such that $A = U\Lambda U^*$. This means, in particular, that the columns of U form an orthonormal basis.

A property that is specific to hermitian matrices, and which characterizes them among the normal matrices, is that all the eigenvalues of a hermitian matrix are real. To this end, suppose that A is hermitian and that v is an eigenvector with $v^*v = 1$ associated to the eigenvalue λ. From

$$\begin{aligned}\lambda = \lambda v^*v = v^*\lambda v = v^*Av \\ = v^*A^*v = (Av)^*v = (\lambda v)^*v = \overline{\lambda}v^*v = \overline{\lambda}\end{aligned} \tag{2.51}$$

then follows that λ is equal to its complex conjugate and thus must be real. The reverse direction, which shows that a normal matrix with real eigenvalues is hermitian, can be found in [HJ85].

Subsequently, we will often use the letters E, F, and G to denote the entries of a symmetric 2×2 matrix:

$$A = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \tag{2.52}$$

This notation is inspired by Gauss [Gau02], who used E, F, and G to denote the first fundamental form in his seminal work on curved surfaces. Using this notation, the eigenvalue computation given by Equation (2.44) simplifies to:

$$\begin{aligned}\lambda_{1,2} &= \frac{E + G \pm \sqrt{(E+G)^2 - 4(EG - F^2)}}{2} \\ &= \frac{E + G \pm \sqrt{(E-G)^2 + 4F^2}}{2}\end{aligned} \tag{2.53}$$

Hence, the discriminant $\Delta = (E - G)^2 + F^2$ is always greater or equal to zero and the eigenvalues are always real, as expected. The computation of the eigenvectors will be discussed in more detail in Section 2.3.4. For now, we just note that the eigenvectors v_1 and v_2, as given by Equation (2.48), are indeed orthogonal ($i, j \in \{1, 2\}, i \neq j$):

$$\begin{aligned}\langle (v_1(\lambda_i), v_2(\lambda_j) \rangle &= \left\langle \begin{pmatrix} F \\ \lambda_i - E \end{pmatrix}, \begin{pmatrix} \lambda_j - G \\ F \end{pmatrix} \right\rangle \\ &= F(\lambda_j - G) + (\lambda_i - E)F = F\big(\overbrace{\lambda_i + \lambda_j}^{=\mathrm{tr}(A)} - (E + G)\big) = 0\end{aligned} \tag{2.54}$$

As already mentioned, symmetric matrices are orthogonally diagonalizable. Hence, if A is a symmetric matrix, then there exists an orthogonal matrix $Q \in \mathrm{O}(2)$ and a diagonal matrix Λ, such that $A = Q\Lambda Q^T$, which is called the *spectral decomposition* of A. Two-dimensional orthogonal matrices are either rotations or reflections. Without loss of generality we can assume that Q is a rotation, since negating one column of a reflection will turn it into a rotation. In order to derive an explicit expression for the decomposition, let us assume that R_φ is a rotation by $\varphi \in (-\pi/2, \pi/2]$ and that Λ is the diagonal matrix of the eigenvalues of A. The decomposition $\Lambda = R_\varphi^T A R_\varphi$ is then given by:

$$\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} E & F \\ F & G \end{pmatrix} \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \tag{2.55}$$

Hence, we get the following system of three equations:

$$0 = F\left(\cos^2\varphi - \sin^2\varphi\right) - (E - G)\sin\varphi\cos\varphi \tag{2.56}$$
$$\lambda_1 = E\cos^2\varphi + 2F\sin\varphi\cos\varphi + G\sin^2\varphi \tag{2.57}$$
$$\lambda_2 = E\sin^2\varphi - 2F\sin\varphi\cos\varphi + G\cos^2\varphi \tag{2.58}$$

By substituting the identities

$$\sin^2\varphi = \frac{1-\cos 2\varphi}{2}, \quad \cos^2\varphi = \frac{1+\cos 2\varphi}{2}, \quad \sin\varphi\cos\varphi = \frac{\sin 2\varphi}{2}, \tag{2.59}$$

we obtain the following simplifications:

$$0 = F\cos 2\varphi - \frac{E-G}{2}\sin 2\varphi \tag{2.60}$$
$$\lambda_1 = \frac{E+G}{2} + \frac{E-G}{2}\cos 2\varphi + F\sin 2\varphi \tag{2.61}$$
$$\lambda_2 = \frac{E+G}{2} - \frac{E-G}{2}\cos 2\varphi - F\sin 2\varphi \tag{2.62}$$

Equation (2.60) does not depend on the eigenvalues and a simple rearrangement yields:

$$\tan 2\varphi = \frac{\sin 2\varphi}{\cos 2\varphi} = \frac{2F}{E-G} \tag{2.63}$$

This equation has multiple solutions. If 2φ is a solution, then $2\varphi + k\pi$, $k \in \mathbb{Z}$, are also solutions. Since Equations (2.61) and (2.62) depend on 2φ with periodicity 2π, it is sufficient to consider only the solutions 2φ and $2\varphi + \pi$. Both are valid and affect the order in which the eigenvalues are placed on the diagonal of Λ. To ensure that the eigenvalues are sorted in ascending order $\lambda_1 \geq \lambda_2$, the angle φ must be chosen such that

$$\frac{E-G}{2}\cos 2\varphi + F\sin 2\varphi \geq -\frac{E-G}{2}\cos 2\varphi - F\sin 2\varphi\ . \tag{2.64}$$

Apparently, this is the case if the signs of $E - G$ and $\cos 2\varphi$ and the signs of F and $\sin 2\varphi$ match. This is exactly the purpose of the `atan2` function, which is available in most programming languages:

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \\ \textit{undefined} & x = 0, y = 0 \end{cases} \tag{2.65}$$

Note that atan2 makes an additional differentiation to ensure that the resulting angle is in the range $(-\pi, \pi]$. Hence, the solution we are looking for is:

$$\varphi = \frac{1}{2}\,\mathrm{atan2}\big(2F, E - G\big) \tag{2.66}$$

In the following we will often require $\varphi^\perp = \varphi - \pi/2$, which can also be obtained in the range $(-\pi/2, \pi/2]$ by rotating the vector consisting of $E - G$ and $2F$ by 180 degrees:

$$\varphi^\perp = \frac{1}{2}\,\mathrm{atan2}\big(-2F, G - E\big) \tag{2.67}$$

In addition, the spectral decomposition $A = R_\varphi \Lambda R_\varphi^T$ is useful for constructing symmetric matrices with prescribed orientation and eigenvalues. For the sake of completeness, the explicit expression is noted here:

$$\begin{aligned}\begin{pmatrix} E & F \\ E & G \end{pmatrix} &= \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \\ &= \begin{pmatrix} \lambda_1\cos^2\varphi + \lambda_2\sin^2\varphi & (\lambda_1 - \lambda_2)\,\sin\varphi\,\cos\varphi \\ (\lambda_1 - \lambda_2)\,\sin\varphi\,\cos\varphi & \lambda_1\sin^2\varphi + \lambda_2\cos^2\varphi \end{pmatrix}\end{aligned} \tag{2.68}$$

Since a symmetric matrix has real eigenvalues, the notion of largest or smallest eigenvalue is meaningful. In fact, the largest and smallest eigenvalues of a symmetric matrix can be characterized as solution of a variational problem [HJ85]. To this end, let $A = Q\Lambda Q^T$ be a spectral decomposition of a symmetric matrix $A \in \mathbb{R}^{n\times n}$. Then we get for the induced quadratic form:

$$x^T A x = x^T Q\Lambda Q^T x = \big(Q^T x\big)^T \Lambda \big(Q^T x\big) = \sum_{i=1}^{n} \lambda_i \big|(Q^T x)_i\big|^2 \tag{2.69}$$

Since Q is orthogonal, we have $\sum_{i=1}^{n} \big|(Q^T x)_i\big|^2 = \|x\|^2$ and it follows that

$$\lambda_{\min} \cdot \|x\|^2 \le x^T A x \le \lambda_{\max} \cdot \|x\|^2 \,. \tag{2.70}$$

For the eigenvectors associated to the largest and smallest eigenvalues the inequalities are sharp. Therefore, the largest and smallest eigenvalues are characterized by:

$$\lambda_{\max} = \max_{0 \neq x \in \mathbb{R}^2} \frac{x^T A x}{\|x\|^2} = \max_{\|x\|=1} x^T A x \tag{2.71}$$

$$\lambda_{\min} = \min_{0 \neq x \in \mathbb{R}^2} \frac{x^T A x}{\|x\|^2} = \min_{\|x\|=1} x^T A x \tag{2.72}$$

In the following, we will refer to the largest and smallest eigenvalues as *major* and *minor eigenvalues*, respectively. Correspondingly, the eigenvectors associated to these will be referred to as *major* and *minor eigenvectors*.

2.3.3 Positive Semidefinite 2 × 2 Matrices

A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is said to be *positive definite* if the induced quadratic form $q(x) = x^T A x$ is positive definite; that is, if $x^T A x > 0$ for all $x \in \mathbb{R}^n$. If the strict inequality

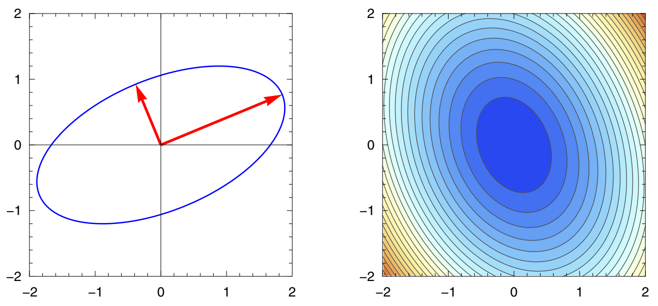


Figure 2.10: *Geometric illustration of the positive semidefinite matrix* $A = R_{\pi/8}\Lambda(2,1)R^T_{\pi/8}$. *(a) Plot showing how the elements of the unit circle are mapped by A. (b) Contour plot showing the level sets of the induced quadratic function* $q(x) = x^T Ax$.

is weakened to $x^T Ax \geq 0$, then a symmetric matrix is said to be *positive semidefinite*. Positive definite and semidefinite matrices can be characterized by their eigenvalues, which are positive and non-negative real numbers, respectively, as follows from the following computation

$$v^T Av = v^T \lambda v = \lambda v^T v = \lambda \cdot \|v\|^2 \,, \tag{2.73}$$

where v was assumed to be an eigenvector associated to the eigenvalue λ of A. Conversely, a symmetric matrix with positive or non-negative eigenvalues is positive definite or positive semidefinite, respectively. This immediately follows from Equation (2.69). A direct consequence of this observation is that trace and determinant are also positive (or non-negative) real numbers. Positive definite matrices have an interesting geometric interpretation that is illustrated in Figure 2.10.

Let A and B be positive semidefinite. Then for $\alpha, \beta \geq 0$ the linear combination $\alpha A + \beta B$ is also positive semidefinite, because

$$x^T(\alpha A + \beta B)x = \alpha\big(x^T Ax\big) + \beta\big(x^T Bx\big) \geq 0 \tag{2.74}$$

for all $x \in \mathbb{R}^2$. Similarly, it follows that a linear combination with positive coefficients of positive definite matrices is positive definite.

In the next section, of special interest will be matrices that arise from scalar products of a set of vectors. Let $v_1, \ldots, v_m \in \mathbb{R}^n$ be a set of vectors. Then the matrix $G = (g_{ij}) \in \mathbb{R}^{m\times m}$ defined by $g_{ij} = \langle v_i, v_j\rangle$ is called the *Gram matrix* of the vectors. A Gram matrix is positive semidefinite, as the following computation shows:

$$\begin{aligned} x^T Gx &= \sum_{i,j=1}^{m} \langle v_i, v_j\rangle\, x_i x_j = \sum_{i,j=1}^{m} \langle x_i v_i, x_j v_j\rangle \\ &= \left\langle \sum_{i=1}^{m} x_i v_i, \sum_{j=1}^{m} x_j v_j \right\rangle = \left\| \sum_{i=1}^{m} x_i v_i \right\|^2 \geq 0 \end{aligned} \tag{2.75}$$

Finally, let us consider the special case of positive semidefinite 2×2 matrices. To this end, let

$$A = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \tag{2.76}$$

be positive semidefinite. The induced quadratic form is then given by:

$$q(x) = x^T A x = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} E & F \\ F & G \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = E x_1^2 + 2F x_1 x_2 + G x_2^2 \tag{2.77}$$

By restricting the vector x along the coordinate axis, it immediately follows that $E, G \geq 0$. Moreover, we can deduce from $\det(A) = EG - F^2 \geq 0$ that $EG \geq F^2$.

If $x \in \mathbb{R}^2 \setminus \{0\}$, then the outer product xx^T is positive semidefinite, since it is the Gram matrix of $\{x_1, x_2\}$. The eigenvalues are $\|x\|^2$ and 0. Thus, it follows that xx^T is of rank 1. Let $A = R_\varphi \Lambda R_\varphi^T$ be a spectral decomposition with $R_\varphi = (\xi, \eta)$, then A can be written as

$$A = \lambda_1 \, \xi \xi^T + \lambda_2 \, \eta \eta^T \, . \tag{2.78}$$

2.3.4 Numerical Implementation

In this section it will be explained how the eigenvalues and eigenvectors of a symmetric positive semidefinite 2×2 matrix

$$A = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \tag{2.79}$$

can be computed in a numerically stable way. Though a straightforward implementation based on the discussion in the previous sections leads to reasonable results, an implementation, such as the one shown in Listing 2.1, is able to achieve much better accuracy by taking care of the numerical subtleties of floating point calculations [Gol91; Hig96].

We already saw that A being positive semidefinite implies $E, G \geq 0$ and $EG \geq F^2$. Hence, from $E + G = 0$ it follows that $F = 0$, which means that the matrix is zero. Let us now assume $E + G > 0$. Computation of the eigenvalues requires solving a quadratic equation, which is known [Bli05; Pre+07; Gol91] to have numerical issues if implemented as in Equation (2.53). Problematic are the two subtractions, which may result in loss of accuracy due to cancellation. In case of the major eigenvalue, only the subtraction under the square root is an issue, as $E + G > 0$. The subtraction is best implemented as $(E - G)^2$, since then catastrophic cancellation is avoided and replaced with benign cancellation [Gol91]:

$$\lambda_1 = \frac{E + G}{2} + R \, , \qquad R = \sqrt{\left(\frac{E - G}{2}\right)^2 + F^2} \tag{2.80}$$

For the computation of the minor eigenvalue, we have to take care of the subtraction in front of the square root. A common approach [Bli05; Pre+07; Gol91] is to use the fact that the product of the eigenvalues equals its determinant:

$$\lambda_2 = \frac{\det(A)}{\lambda_1} = \frac{EG - F^2}{\lambda_1} \tag{2.81}$$

Notice that $\lambda_1 > 0$, since $E + G > 0$. Because of rounding errors, $EG - F^2$ may become negative and therefore it is advisable to compute $\max(0, EG - F^2)$.

Recall that a symmetric matrix has orthogonal eigenvectors. Therefore, it is sufficient to compute only one of the eigenvectors. The other can be found applying a rotation by 90 degrees. Let us first assume that the eigenvalues are distinct, which can be easily verified by

```
inline __host__ __device__
void solve_eig_psd( float E, float F, float G, float& lambda1,
                    float& lambda2, float2& ev )
{
    float B = (E + G) / 2;
    if (B > 0) {
        float D = (E - G) / 2;
        float FF = F*F;
        float R = sqrtf(D*D + FF);
        lambda1 = B + R;
        lambda2 = fmaxf(0, E*G - FF) / lambda1;

        if (R > 0) {
            if (D >= 0) {
                float nx = D + R;
                ev = make_float2(nx, F) * rsqrtf(nx*nx + FF);
            } else {
                float ny = -D + R;
                ev = make_float2(F, ny) * rsqrtf(FF + ny*ny);
            }
        } else {
            ev = make_float2(1, 0);
        }
    } else {
        lambda1 = lambda2 = 0;
        ev = make_float2(1, 0);
    }
}
```

Listing 2.1: *Eigenanalysis of a symmetric positive semidefinite* 2×2 *matrix given.*

checking that the square root R is nonzero. Then we have a well-defined major eigenvalue and, as discussed in Section 2.3.1, there are two possible candidates for the computation of the eigenvector. Substituting the expression for the major eigenvalue λ_1 into Equation (2.48), we get:

$$\begin{aligned} v_1(\lambda_1) &= \begin{pmatrix} F \\ \lambda_1 - E \end{pmatrix} = \begin{pmatrix} F \\ -\frac{E-G}{2} + R \end{pmatrix} \\ v_2(\lambda_1) &= \begin{pmatrix} \lambda_1 - G \\ F \end{pmatrix} = \begin{pmatrix} \frac{E-G}{2} + R \\ F \end{pmatrix} \end{aligned} \tag{2.82}$$

Hence, by choosing v_1 if $E - G < 0$ and v_2 if $E - G \geq 0$, we can kill two birds with one stone. Firstly, subtractive cancellation is avoided, since the first term in the sum is positive. Secondly, if $F = 0$ and correspondingly $R = |E - G|/2$, then the computed vector is guaranteed to be nonzero. Finally, if the square root R is zero, we have a single eigenvalue with multiplicity two and the eigenspace is two-dimensional.

A limitation of the discussed implementation is that computations having the form $\sqrt{a^2 + b^2}$ may underflow or overflow if not computed at higher precision. A common approach to avoid such issues, which is for example used in LAPACK's SLEV2 function [And+99], is to exchange a and b if necessary, such that $|a| > |b|$, and compute $|a|\sqrt{1 + (b/a)^2}$.

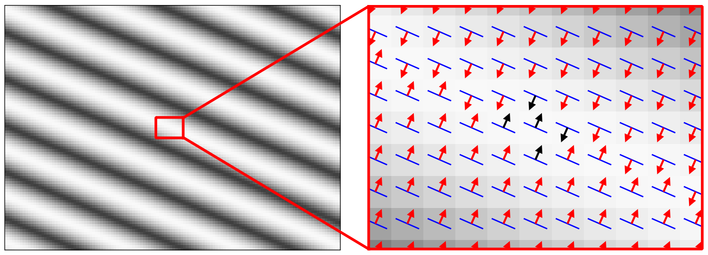


Figure 2.11: *In the neighborhood of local extrema, gradients have opposite signs and cancel out when averaged, resulting in a loss of accuracy.*

2.4 Definition of the Structure Tensor

The gradient vectors gained with a small sized derivative filter are, in general, too noisy for our purposes. To obtain a smooth vector field, further smoothing is required. However, smoothing the vectors directly generally fails to produce reasonable results. For instance, gradients in the neighborhood of a minimum or maximum have opposite signs (Figure 2.11) and cancel out when averaged. As the right tool to obtain a smooth vector field, the structure tensor will be introduced in this section. It is a well-known and popular tool in computer vision and image processing and was introduced by Förster [För86]. The major eigenvector of the structure tensor corresponds to Di Zenzo's multi-image gradient [Di 86; GW06], and also appeared, although not explicitly, in Harris and Stephens's corner detector [HS88] and the classical work on texture synthesis by Kass and Witkin [KW87] and texture analysis by Rao and Schunck [RS89; Rao90].

Let $f : \mathbb{Z}^2 \to \mathbb{R}$ denote a grayscale image. Then the *structure tensor* at a point x is defined as the outer product of the gradient at x:

$$J(x) = \nabla f(x) \nabla f(x)^T = \begin{pmatrix} \left(\frac{\partial f}{\partial x_1}(x)\right)^2 & \frac{\partial f}{\partial x_1}(x) \cdot \frac{\partial f}{\partial x_2}(x) \\ \frac{\partial f}{\partial x_1}(x) \cdot \frac{\partial f}{\partial x_2}(x) & \left(\frac{\partial f}{\partial x_2}(x)\right)^2 \end{pmatrix} \tag{2.83}$$

The *smoothed structure tensor*

$$J_\rho(x) = \frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, \nabla f(y) \nabla f(y)^T , \tag{2.84}$$

is obtained by convolving the structure tensor with a two-dimensional Gaussian function

$$G_\rho(z) = \frac{1}{2\pi\rho^2} \exp\left(-\frac{\|z\|}{2\rho^2}\right) \tag{2.85}$$

with standard deviation ρ. The set

$$\mathcal{N}(x) = \left\{ y \in \mathbb{Z}^2 \ : \ \|x-y\| < r \right\} \tag{2.86}$$

thereby refers to a local neighborhood of x with reasonable cut-off (e.g., with radius $r = 3\rho$), and

$$|G_\rho| = \sum_{y \in \mathcal{N}(0)} G_\rho(y) \tag{2.87}$$

denotes the corresponding normalization term. Notice that both the structure tensor and the smoothed structure tensor are symmetric and positive semidefinite, since they are an outer product and a positively weighted sum of outer products (Section 2.3.3). In the remainder of this section, different ways to define or interpret the structure tensor will be discussed. Each definition or interpretation will provide a different perspective on the structure tensor.

2.4.1 Mean Axis and Double Angle

Before approaching the computation of averages of local orientations, we first review how to compute an average of a set of angles. For a set of data points $x_i \in \mathbb{R}^n$, it is well-known that the *arithmetic mean* has the property to minimize the sum of the squared distances

$$\bar{x} = \operatorname*{arg\,min}_{x \in \mathbb{R}^n} \sum_i (x_i - x)^2 \,. \tag{2.88}$$

Means characterized by this property are called *Fréchet means*. Applying this definition to a set of angles $\phi_i \in [0, 2\pi)$ results in the minimization problem

$$\bar{\phi} = \operatorname*{arg\,min}_{\phi \in [0,2\pi)} \sum_i d^2(\phi_i, \phi) \,, \tag{2.89}$$

with d being suitable metric. Since an angle in the range $[0, 2\pi)$ can be identified with a point of the unit circle $\mathbb{S}^1$, a possible choice for the metric is the geodesic distance in $\mathbb{S}^1$. This leads to the *intrinsic mean* [Pen04] that, however, is difficult to compute. An alternative that leads to reasonable results is to consider the embedding of $\mathbb{S}^1$ into $\mathbb{R}^2$ and use the induced Euclidean metric (Figure 2.12), leading to the *extrinsic mean*:

$$\bar{\phi} = \operatorname*{arg\,min}_{\phi \in [0,2\pi)} \sum_i \big(1 - \cos(\phi_i - \phi)\big) \tag{2.90}$$

Setting the derivative with respect to ϕ equal to zero yields:

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}\phi} \sum_i \big(1 - \cos(\phi_i - \phi)\big) = \sum_i \sin(\phi_i - \phi) \\ &= \sum_i \big(\cos\phi \sin\phi_i - \cos\phi_i \sin\phi\big) = \cos\phi \sum_i \sin\phi_i - \sin\phi \sum_i \cos\phi_i \end{aligned} \tag{2.91}$$

Thus, solving for ϕ gives

$$\bar{\phi} = \operatorname{atan2}\left(\sum_i \sin\phi_i, \sum_i \cos\phi_i \right), \tag{2.92}$$

where atan2 denotes the inverse tangent with values in the range $-\pi$ to π. The angle $\bar{\phi}$ is called the *directional mean*, and alternatively could have been derived as maximum-likelihood estimator of the *von Mises distribution* [Bis07; MJ99].

Since local orientation is represented by an angle in the range $(-\pi/2, \pi/2]$, or optionally by points on the unit circle with opposite points identified, the directional mean is not directly applicable. We can, however, consider doubling the angle

$$\cos\phi + \mathrm{i}\sin\phi = e^{\mathrm{i}\phi} \mapsto e^{\mathrm{i}2\phi} = \cos 2\phi + \mathrm{i}\sin 2\phi \,, \tag{2.93}$$

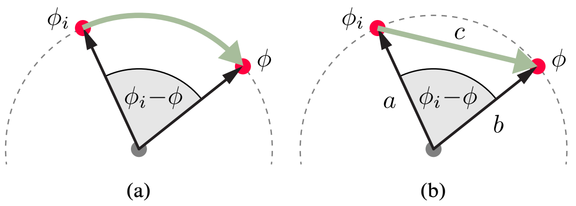


Figure 2.12: *(a) Intrinsic (geodesic) distance on the unit circle. (b) Extrinsic distance. From the law of cosines it follows that the squared extrinsic distance between two angles ϕ_i and ϕ is given by $c^2 = a^2 + b^2 - 2ab\cos(\phi_i - \phi) = 2 - 2\cos(\phi_i - \phi)$.*

as proposed by Granlund [Gra78], which transforms the periodicity from π to 2π. Applying the directional mean after the transformation and transforming back then yields what is called the *mean axis* [MJ99]:

$$\bar{\phi} = \frac{1}{2}\operatorname{atan2}\left(\sum_i \sin 2\phi_i\,,\ \sum_i \cos 2\phi_i\right). \tag{2.94}$$

The mean axis already creates reasonable smooth vector fields, but we can do better. Firstly, a spatial weighting term that gives more weight to gradient vectors at spatially closer positions can be incorporated. Secondly, gradient vectors with low magnitude belong to almost flat image regions. In these regions, the signal-to-noise ratio is low and, due to this, the accuracy of the gradient vectors also low. By including a weighting term based on the magnitude of the gradient vectors, the influence of gradient vectors with large magnitude, which correspond to strong edges, can be increased. More precisely, let $r_x = \|\nabla f(x)\|$ denote the gradient magnitudes and let $\phi_x = \arg \nabla f(x)$ be the arguments of the gradient vectors. The *weighted mean axis* at a point x is then defined as:

$$\begin{aligned}\tilde{\phi}(x) = \frac{1}{2}\operatorname{atan2}\Bigg(&\frac{1}{|G_\rho|}\sum_{y\in\mathcal{N}(x)} G_\rho(y-x)\, r_y^2 \sin 2\phi_y\,, \\ &\frac{1}{|G_\rho|}\sum_{y\in\mathcal{N}(x)} G_\rho(y-x)\, r_y^2 \cos 2\phi_y \Bigg)\end{aligned} \tag{2.95}$$

The influence of the magnitudes r_x has been chosen to be quadratic. Although other choices would be possible, this gives vectors with large magnitude strong influence, helps to create more coherent results, and matches well many historical painting styles [HE04]. In addition, this choice matches the definition of the smoothed structure tensor, which can be easily verified by computing the outer product of the gradient in polar coordinates:

$$\begin{aligned}\nabla f(x)\nabla f(x)^T &= \begin{pmatrix} r_x\cos\phi_x \\ r_x \sin\phi_x \end{pmatrix} \begin{pmatrix} r_x\cos\phi_x, & r_x\sin\phi_x \end{pmatrix}^T \\ &= \begin{pmatrix} r_x^2\cos^2\phi_x & r_x^2 \sin\phi_x\,\cos\phi_x \\ r_x^2\sin\phi_x\,\cos\phi_x & r_x^2\sin^2\phi_x \end{pmatrix} =: \begin{pmatrix} E & F \\ F & G \end{pmatrix}\end{aligned} \tag{2.96}$$

Using the identities from Equation (2.59), it then immediately follows that

$$\begin{aligned} 2F &= 2r_x^2\,\sin\phi_x\,\cos\phi_x = r_x^2\,\sin 2\phi_x \\ E - G &= r_x^2(\cos^2\phi_x - \sin^2\phi_x) = r_x^2\,\cos 2\phi_x\,. \end{aligned} \tag{2.97}$$

2.4.2 Metric Tensor of a Riemannian Manifold

Let $U \subset \mathbb{R}^2$ and let us assume for a moment that the input image is given by a differential function $f\colon U \to \mathbb{R}^n$. Here, the case $n = 1$ corresponds to gray level images and the case $n = 3$ corresponds to color images. Given this function, we can now consider its graph as a two dimensional submanifold of $\mathbb{R}^{2+n}$. The map

$$F\colon \begin{cases} U & \longrightarrow & \mathbb{R}^{2+n} \\ (u,v) & \mapsto & \big(u, v, f^1(u,v), \dots, f^n(u,v)\big) \end{cases} \tag{2.98}$$

then defines a smooth global parameterization of the embedding and the Euclidean metric $\bar{g}$ of $\mathbb{R}^{2+n}$ induces a Riemannian metric on the graph that is given in chart coordinates by:

$$\begin{aligned} g = F^*\bar{g} &= F^*\Big(\big(\mathrm{d}x^1\big)^2 + \dots + \big(\mathrm{d}x^{2+n}\big)^2\Big) \\ &= \mathrm{d}\big(x^1 \circ F\big)^2 + \dots + \mathrm{d}\big(x^{2+n} \circ F\big)^2 \\ &= \mathrm{d}u^2 + \mathrm{d}v^2 + \textstyle\sum_{i=1}^n \big(\mathrm{d}f^i\big)^2 \end{aligned} \tag{2.99}$$

Here, ω^2 denotes the common abbreviation for the symmetric product of a tensor ω with itself. Now, let us consider a point $p \in U$ of the image domain and a tangent vector $X \in T_pU \simeq \mathbb{R}^2$. The Euclidean length of X in local coordinates corresponds to the length in the image domain and the length given by the induced Riemannian metric

$$\sqrt{\langle X, X\rangle_p} = \sqrt{g_p(X,X)} \tag{2.100}$$

corresponds to measuring length on the image embedded as graph in $\mathbb{R}^{2+n}$. Hence, for tangent vectors of fixed Euclidean length, e.g. $\|X\| = 1$, the Riemannian metric g can be interpreted as measuring the squared local rate of change in direction X.

We are interested in finding the local orientation at p; i.e., the directions where the rate of change of the image regarded as graph is either minimum or maximum. For unit length tangent vectors, the term $\mathrm{d}u^2 + \mathrm{d}v^2$ of g is constant and equal to one. To find the minimum and maximum local rate of change, it is therefore sufficient to consider only the term $\sum_{i=1}^n (\mathrm{d}f^i)^2$. The differential of the i-th component of f is given by

$$\mathrm{d}f^i = \frac{\partial f^i}{\partial u}\mathrm{d}u + \frac{\partial f^i}{\partial v}\mathrm{d}v \tag{2.101}$$

and its symmetric product with itself is given by

$$\begin{aligned} \big(\mathrm{d}f^i\big)^2 &= \mathrm{d}f^i \otimes \mathrm{d}f^i \\ &= \left(\frac{\partial f^i}{\partial u}\right)^2 \mathrm{d}v^2 + 2\frac{\partial f^i}{\partial u}\frac{\partial f^i}{\partial v}\mathrm{d}u\mathrm{d}v + \left(\frac{\partial f^i}{\partial v}\right)^2 \mathrm{d}v^2 \,. \end{aligned} \tag{2.102}$$

Thus we get

$$\textstyle\sum_{i=1}^n \big(\mathrm{d}f^i\big)^2 = E\,\mathrm{d}u + 2F\,\mathrm{d}u\mathrm{d}v + G\,\mathrm{d}v^2 \,, \tag{2.103}$$

where

$$J := \begin{pmatrix} E & F \\ F & G \end{pmatrix} := \begin{pmatrix} \sum_{i=1}^n \left(\frac{\partial f^i}{\partial u}\right)^2 & \sum_{i=1}^n \frac{\partial f^i}{\partial u}\frac{\partial f^i}{\partial v} \\ \sum_{i=1}^n \frac{\partial f^i}{\partial u}\frac{\partial f^i}{\partial v} & \sum_{i=1}^n \left(\frac{\partial f^i}{\partial v}\right)^2 \end{pmatrix} \tag{2.104}$$

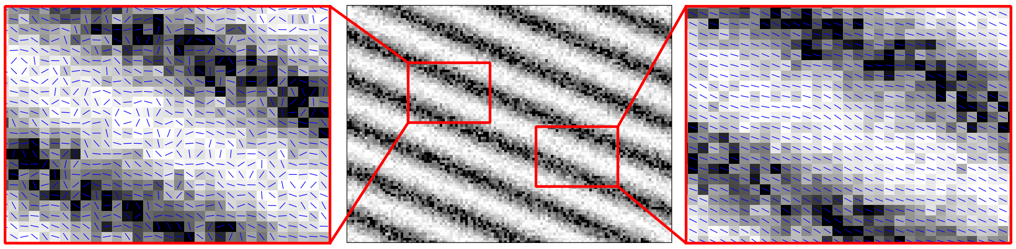


Figure 2.13: *Visualization of the minor eigenvector of the structure tensor for a pattern with distinguished orientation, corrupted by Gaussian noise. On the left the minor eigenvector has been computed from the structure tensor without smoothing. On the right the minor eigenvector of the smoothed structure tensor is shown.*

is the so called *metric tensor*. In case of gray level images, the metric tensor is simply given by the outer product of the gradient

$$J = \nabla f \nabla f^T \tag{2.105}$$

and in case of a color image it is given by the sum of the outer products of the gradients of the different color channels. For example, in case of an RGB color image we have:

$$J = \nabla R \nabla R^T + \nabla G \nabla G^T + \nabla B \nabla B^T \tag{2.106}$$

As discussed in Section 2.3.2, the extremal values of the quadratic form on the unit circle, as defined by Equation (2.103), correspond to the eigenvalues of the metric tensor J.

2.4.3 Least Squares Optimization

In the previous section we made the assumption that the image is given by a smooth function. This is obviously wrong for typical natural images. As discussed earlier in Section 2.2.2, the common practice to smooth the image prior to derivative computation (e.g., with a Gaussian filter), presents a problem. In this section it will be shown that a better approach is to smooth the structure tensor (Figure 2.13), which can be best understood from an optimization point of view [WB05; Bro+06].

Let $g(y)$ denote the gradients of a grayscale image, and let x be any point of the image. Now, suppose a unit vector v is given. Then the deviation of a single gradient $g(y)$ from v can be defined as (see also Figure 2.14):

$$e(y, v) = \left\| g(y) - \langle g(y), v \rangle v \right\| \tag{2.107}$$

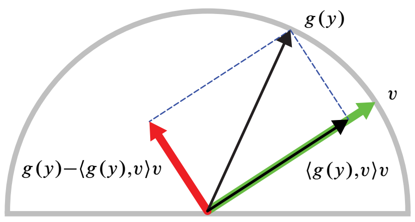


Figure 2.14: *Deviation of a gradient $g(y) := \nabla(y)$ from a given direction $v(x)$. The term $\langle g(y), v(x)\rangle v(x)$ is the projection of $g(y)$ onto $v(x)$.*

The total squared error at x can then be defined by the convolution of e^2 with a two-dimensional Gaussian function G_ρ:

$$E(x,v) = \frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, e^2(y,v) \tag{2.108}$$

Using the linearity of the scalar product, $e^2(y)$ can be simplified to:

$$\begin{aligned} e^2 &= \langle e,e \rangle = \big\langle e, g - \langle g,v \rangle v \big\rangle = \big\langle e,g \big\rangle - \underbrace{\big\langle e, \langle g,v \rangle v \big\rangle}_{=0} \\ &= \langle g,g \rangle - \langle g,v \rangle^2 \\ &= g^T g - v^T\big(gg^T\big)v \end{aligned} \tag{2.109}$$

By substituting this into equation (2.108) we get:

$$\begin{aligned} E(x,v) = \frac{1}{|G_\rho|}\Bigg(&\sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, g(y)^T g(y) \\ &- \sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, v^T\big(g(y)g(y)^T\big)v \Bigg) \end{aligned} \tag{2.110}$$

The first term of E is constant. Therefore, minimizing E is equivalent to maximizing the second term. Hence, the vector we are looking for is given by:

$$\mathfrak{v}(x) = \operatorname*{arg\,min}_{\|v\|=1} E(x,v) \tag{2.111}$$

$$= \operatorname*{arg\,max}_{\|v\|=1} \frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, v^T\big(g(y)g(y)^T\big)v \tag{2.112}$$

Moreover, since v does not depend on y, it follows by linearity that

$$\mathfrak{v}(x) = \operatorname*{arg\,max}_{\|v\|=1} v^T J_\rho(x)\, v\ , \tag{2.113}$$

with

$$J_\rho(x) = \frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho(x-y)\, g(y)g(y)^T \tag{2.114}$$

being the smoothed structure tensor at x. From Equation (2.71) we know that maximizing $v^T J_\rho v$ with the constrained $\|v\| = 1$ is equivalent to an eigenanalysis of J_ρ and that the vector that minimizes $E(x,v)$ is given by the major eigenvector. Hence, we see that smoothing the structure tensor corresponds to solving a weighted least squares problem.

Up to now, only the case of grayscale images has been considered. The most straightforward way to extend the previous discussion to color images is by minimizing Equation (2.111) jointly for all color channels

$$\mathfrak{v}(x) = \operatorname*{arg\,min}_{\|v\|=1} \sum_{i=1}^{n} E^i(x,v)\ , \tag{2.115}$$

resulting in the following generalization of Equation (2.84)

$$J_\rho(x) = \frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho(y-x) \sum_{i=1}^{n} g^i(y)\, g^i(y)^T\ , \tag{2.116}$$

where n is the number of color channels (i.e., $n = 3$ for RGB images) and $g^i(y)$ denotes the gradient of the i-th color channel. Another convenient way to express the sum of outer products is as the scalar product of the partial derivatives:

$$\begin{aligned}\sum_{i=1}^{n} g^i(y)\, g^i(y)^T &= \begin{pmatrix} \sum_{i=1}^{n} g_1^i(y)^2 & \sum_{i=1}^{n} g_1^i(y) g_2^i(y) \\ \sum_{i=1}^{n} g_1^i(y) g_2^i(y) & \sum_{i=1}^{n} g_2^i(y)^2 \end{pmatrix} \\ &= \begin{pmatrix} \langle g_1(y), g_1(y)\rangle & \langle g_1(y), g_2(y)\rangle \\ \langle g_1(y), g_2(y)\rangle & \langle g_2(y), g_2(y)\rangle \end{pmatrix}\end{aligned} \tag{2.117}$$

Instead of minimizing the total error $E(x, v)$, an alternative but equivalent approach [Rao90; RS91] is to maximize the squared scalar product $\langle g(y), v\rangle$:

$$\underset{\|v\|=1}{\arg\max}\ \frac{1}{|G_\rho|} \sum_{y\in\mathcal{N}(x)} G_\rho(x-y)\,\langle g(y), v\rangle^2 \tag{2.118}$$

2.4.4 Singular Value Decomposition

Let $A \in \mathbb{R}^{m\times n}$ be a matrix. Then A may be written in the form

$$A = U\Sigma V^T \tag{2.119}$$

with $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ being orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ being a diagonal matrix. The decomposition in Equation (2.119) is called the *singular value decomposition* (SVD) and can be regarded as a generalization of spectral decomposition of symmetric matrices. The diagonal elements of Σ are referred to as the *singular values*. They are non-negative, the square roots of the eigenvalues of AA^T, and usually arranged in order of decreasing magnitude. The columns of U are eigenvectors of AA^T, and the columns of V are eigenvectors of A^TA, arranged in both cases in the order of the corresponding squared eigenvalues on the diagonal of Σ, and called the *left-singular* and *right-singular vectors*, respectively. The singular value decomposition is one of the most important tools in numerical linear algebra. A detailed coverage can be found in the books by Horn and Johnson [HJ85] and Golub and Van Loan [GVL96].

To understand how the singular value decomposition relates to the structure tensor, let x_0 be any point, let

$$\{x_0, \ldots, x_{N-1}\} = \{x \in \mathbb{R}^2 \ :\ \|x - x_0\| < r\} \tag{2.120}$$

be the set of sequentially enumerated points from the local neighborhood of x_0 with distance smaller than some radius r, and let $g(x_i)$ denote the gradient vectors at x_i. Moreover, let

$$G = \begin{pmatrix} g(x_0)^T \\ g(x_1)^T \\ \vdots \\ g(x_{N-1})^T \end{pmatrix} \tag{2.121}$$

be the $N \times 2$ matrix having the gradient vectors as columns. Forming the product $G^T G$, we see that

$$\begin{aligned} G^T G &= \begin{pmatrix} g_1(x_0) & \dots & g_1(x_{N-1}) \\ g_2(x_0) & \dots & g_2(x_{N-1}) \end{pmatrix} \begin{pmatrix} g_1(x_0) & g_2(x_0) \\ \vdots & \vdots \\ g_1(x_{N-1}) & g_2(x_{N-1}) \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=0}^{N-1} g_1(x_i)^2 & \sum_{i=0}^{N-1} g_1(x_i) g_2(x_i) \\ \sum_{i=0}^{N-1} g_1(x_i) g_2(x_i) & \sum_{i=0}^{N-1} g_2(x_i)^2 \end{pmatrix} = \sum_{i=0}^{N-1} g(x_i) g(x_i)^T \end{aligned} \tag{2.122}$$

is equivalent to the structure tensor. Moreover, by defining a $N \times N$ diagonal matrix W with elements

$$w_{ii} = \sqrt{\frac{G_\rho(x_0 - x_i)}{\sum_{i=0}^{N-1} G_\rho(x_0 - x_i)}} = \frac{1}{\sqrt{|G_\rho|}} \sqrt{G_\rho(x_0 - x_i)} \tag{2.123}$$

we have

$$WG = \Big(w_{00}\, g(x_0),\ w_{11}\, g(x_1),\ \dots,\ w_{N-1,N-1}\, g(x_{N-1}) \Big) \tag{2.124}$$

and we see that

$$\begin{aligned} (WG)^T WG &= G^T W^T W G = G^T W^2 G \\ &= \sum_{i=0}^{N-1} w_{ii}^2\, g(x_i) g(x_i)^T \\ &= \frac{1}{|G_\rho|} \sum_{i=0}^{N-1} G_\rho(x_0 - x_i)\, g(x_i) g(x_i)^T = J_\rho(x_0) \end{aligned} \tag{2.125}$$

is just the smoothed structure tensor. The eigenvalues and eigenvectors of the smoothed structure tensor may thus be obtained from the singular value decomposition of WG, where the eigenvalues are given by the squared singular values and the eigenvectors given by the right-singular vectors.

Feng and Milanfar [FM02; Fen03] used the singular value decomposition as part of their multi-scale local orientation estimation approach. A key advantage of the singular value decomposition is that it is numerically highly stable and "cannot fail" [Pre+07, p. 795]. For this reason, the singular value decomposition is often recommended as a default choice for solving least squares problems. The numerical stability, however, comes at a price. Compared to using the normal equations $G^T G$, the singular value decomposition requires more storage and is computationally significantly more expensive, making it in our case impractical for GPU implementation. Moreover, using the singular value decomposition in experiments did not provide observable improvements.

2.4.5 Inertia Tensor

In Section 2.1 we saw that patterns with distinguished orientation cluster along a line in the power spectrum. This observation is the foundation for the approach by Bigün et al. [BGW91; Big06], who proposed to derive the average local orientation as the axis k of minimum inertia of the power spectrum:

$$\underset{\|k\|=1}{\arg\min} \int d^2(\omega, k) |F(\omega)|^2 \, \mathrm{d}\omega \tag{2.126}$$

The distance of ω to the axis k is thereby given as illustrated in Figure 2.14:

$$\begin{aligned} d^2(\omega,k) &= \left\| \omega - \langle w,k\rangle k \right\| \\ &= \omega^T\omega - k^T(\omega\omega^T)\,k = k^T\left(\omega^T\omega I - \omega\omega^T\right)k \end{aligned} \tag{2.127}$$

Hence, we get

$$\int d^2(\omega,k)|F(\omega)|^2\,\mathrm{d}\omega = k^T\mathcal{J}k\,, \tag{2.128}$$

with

$$\mathcal{J} = \begin{pmatrix} \displaystyle\int \omega_2^2\,|F(\omega)|^2\,\mathrm{d}\omega & \displaystyle -\int \omega_1\omega_2\,|F(\omega)|^2\,\mathrm{d}\omega \\ \displaystyle -\int \omega_1\omega_2\,|F(\omega)|^2\,\mathrm{d}\omega & \displaystyle\int \omega_1^2\,|F(\omega)|^2\,\mathrm{d}\omega \end{pmatrix}. \tag{2.129}$$

Performing the convolutions in the frequency domain is not practical. Fortunately, by Parseval's theorem (Rudin 1987, p. 187) we have:

$$\begin{aligned} \int \omega_p\omega_q\,|F(\omega)|^2\,\mathrm{d}\omega = \int i\omega_p\,\overline{i\omega_q}\,F(\omega)\overline{F(\omega)}\,\mathrm{d}\omega &= \int i\omega_p\,F(\omega)\,\overline{i\omega_q\,F(\omega)}\,\mathrm{d}\omega \\ &= \int \mathcal{F}^{-1}\big(i\omega_p\,F(\omega)\big)\,\overline{\mathcal{F}^{-1}\big(i\omega_q\,F(\omega)\big)}\,\mathrm{d}x \\ &= \int \frac{\partial f}{\partial x_p}\frac{\partial f}{\partial x_q}\,\mathrm{d}x \end{aligned} \tag{2.130}$$

Hence, we see that

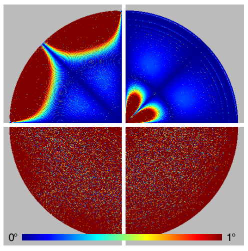
$$\mathcal{J} = \mathrm{tr}(J)I - J\,, \tag{2.131}$$

where J is the structure tensor averaged (i.e., box filtered) over the whole image. Thus, in physical terms $\mathcal{J}$ is the inertia tensor, while the structure tensor corresponds to the covariance of the pointwise density. Introducing appropriate windowing and reconstruction functions yields the smoothed structure tensor.

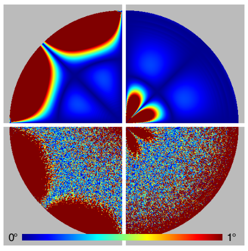
Kass and Witkin [KW87] also start the derivation of their approach in frequency space. However, instead of computing the axis of minimum inertia, they used a specifically designed directional filter. In the spatial domain this filter turned out to be the derivative of a difference of Gaussians. Another approach for deriving the structure tensor in frequency space based on quadrature filters has been developed by Knutsson and Granlund [Knu89; GK95; KN09]. In contrast to Gaussian derivative filters, quadrature filters are phase invariant, a highly desired property for local orientation analysis [Jäh05].

2.4.6 Implementation

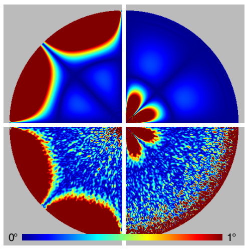
In the previous sections, several equivalent definitions of the structure tensor were given, providing motivation and explaining its usefulness. Interestingly enough, besides its theoretical benefits, the structure tensor is also straightforward and simple to use in practice, and it may be computed as follows: First, for each pixel, the gradient is obtained by computing the horizontal and vertical partial derivatives, using either the 3×3 or 5×5 optimized derivative filter (Section 2.2.3). For best performance, advantage of the separability of the 5×5 derivative filter should be taken. Alternatively, Gaussian derivatives may be used. However, these require a considerable larger filter kernel, and are therefore not recommended. In the



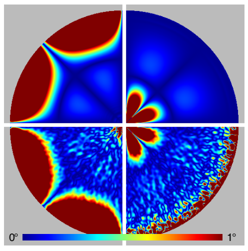
(a) 3×3: $\rho = 0$



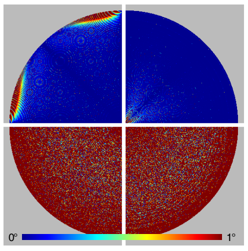
(b) 3×3: $\rho = 1$



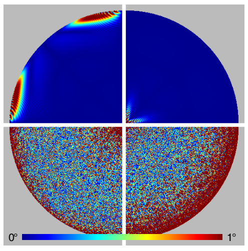
(c) 3×3: $\rho = 2$



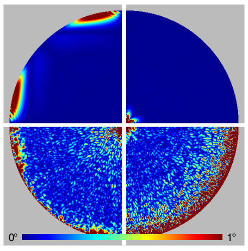
(d) 3×3: $\rho = 3$



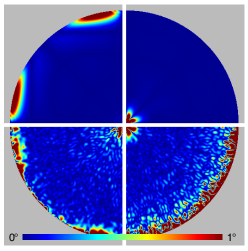
(e) 5×5: $\rho = 0$



(f) 5×5: $\rho = 1$



(g) 5×5: $\rho = 2$



(h) 5×5: $\rho = 3$

Figure 2.15: *Angular errors of the orientation computed from the smoothed structure for different ρ are shown. The test pattern of Figure 2.7 is used.*

```
__global__ void st_scharr_3x3( gpu_plm2<float4> dst ) {
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    const float b1 = 46.84f / 256;
    const float b0 = 1 - 2 * b1;

    float3 g1 = 0.5f * (
          -b1 * make_float3(tex2D(texSRC, ix-1, iy-1)) +
          -b0 * make_float3(tex2D(texSRC, ix-1, iy  )) +
          -b1 * make_float3(tex2D(texSRC, ix-1, iy+1)) +
          +b1 * make_float3(tex2D(texSRC, ix+1, iy-1)) +
          +b0 * make_float3(tex2D(texSRC, ix+1, iy  )) +
          +b1 * make_float3(tex2D(texSRC, ix+1, iy+1)));

    float3 g2 = 0.5f * (
          -b1 * make_float3(tex2D(texSRC, ix-1, iy-1)) +
          -b0 * make_float3(tex2D(texSRC, ix,   iy-1)) +
          -b1 * make_float3(tex2D(texSRC, ix+1, iy-1)) +
          +b1 * make_float3(tex2D(texSRC, ix-1, iy+1)) +
          +b0 * make_float3(tex2D(texSRC, ix,   iy+1)) +
          +b1 * make_float3(tex2D(texSRC, ix+1, iy+1)));

    dst(ix, iy) = make_float4( dot(g1, g1), dot(g1, g2), dot(g2, g2), 1 );
}
```

Listing 2.2: *Computation of the structure tensor using the optimized derivative operator. The result is stored in the first three components of a `float4`.*

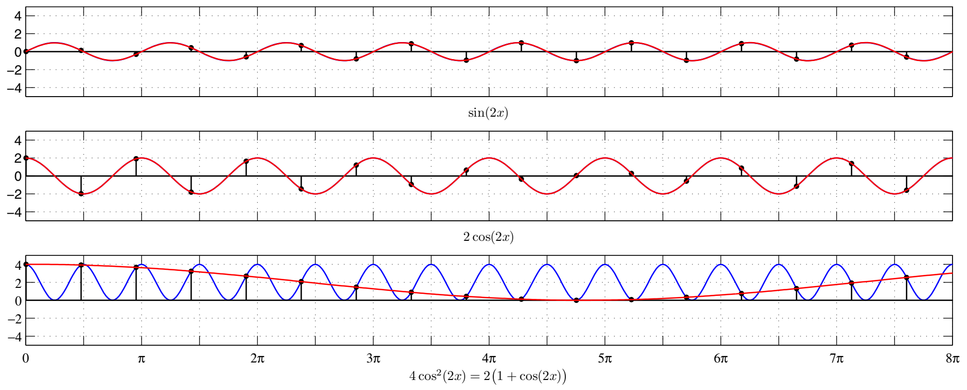


Figure 2.16: *Correct sampling of the structure requires doubling the sampling rate. In the top and middle, an oscillating signal and its derivative, respectively, are shown. Both signals are sampled at 0.95 of the Nyquist rate $\frac{\pi}{2}$, which allows for a perfect reconstruction of the signal. On the bottom, the squared derivative is shown in blue. Due to its doubled signal frequency, the reconstruction shown in red fails.*

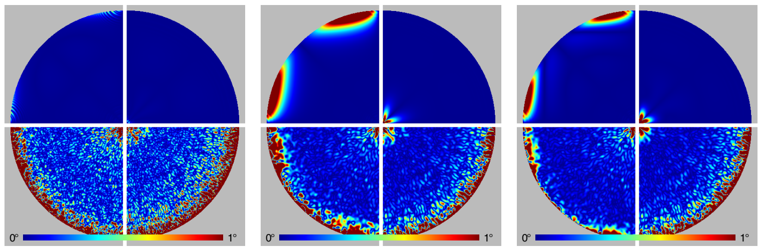
next step, the structure tensor is computed, which is most conveniently done by calculating the scalar products of the partial derivatives (Equation (2.117)), and storing the result in a floating point texture map. The smoothed structure tensor is obtained by a subsequent smoothing pass, performing a convolution with the Gaussian function G_ρ. As can be seen in Figure 2.15, smoothing the structure tensor indeed improves the quality of the orientation estimate for noisy images. For the sake of completeness, the computation of the structure tensor using the optimized 3×3 derivative filter is shown in Listing 2.2.

A limitation of this approach is that it violates Shannon's sampling theorem [Köt03b; Köt03a; Köt08]. To understand why, let us consider the case of grayscale images. Let f be an image that is properly sampled at the Nyquist rate. Then f is band-limited and has a cut-off frequency of π, if unit pixel size is assumed. Since the optimized derivative filters are finite impulse response filters, they are not band-limited, which is also the case for the Gaussian derivatives. Hence, the partial derivatives obtained by convolution with some derivative operators $f_1 = D_x * f$ and $f_2 = D_y * f$ will be band-limited with cut-off frequency π as well. To compute the structure tensor, the partial derivatives have to be multiplied. By the convolution theorem, this corresponds to a convolution $\mathcal{F}(f_i) * \mathcal{F}(f_j)$ in the frequency domain, with the result having doubled support. Thus, to correctly sample the structure tensor, the sampling rate has to be doubled as well. While this may seem counter-intuitive at the first glance, it can be illustrated by a simple example that is shown in Figure 2.16. Let $f(t) = \sin(\omega t + \phi)$ be an oscillating signal with frequency ω and phase ϕ. Then its derivative is given by $f'(t) = \omega \cos(\omega t + \phi)$. Both $f(t)$ and $f'(t)$ can be sampled properly at sampling rates $\lambda < \frac{\pi}{\omega}$. However, the squared derivative

$$\big(f'(t)\big)^2 = \omega^2 \cos^2(\omega t + \phi) = \frac{\omega^2}{2}\big(1 + \cos(2\omega t + 2\phi)\big) \tag{2.132}$$

has doubled frequency and thus requires a sampling rate $\lambda < \frac{\pi}{2\omega}$.

There are two approaches to ensure proper sampling. First of all, the high-frequencies in the image could be removed. This could be achieved, for instance, by increasing the standard deviation of the Gaussian derivatives. However, to achieve the necessary damping

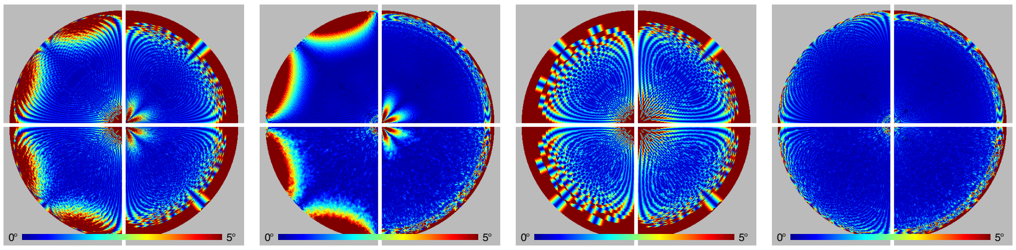


(a) *Oversampled Gaussian derivative:* $\sigma = 1$ (b) *Gaussian derivative:* $\sigma = 1$ (c) 5×5

Figure 2.17: *Angular errors of the orientation obtained from smoothed structure tensors computed with and without oversampling. (a) Oversampled Gaussian derivative computed for double image size. Smoothing and downsampling of the structure tensor is performed in a single step using resampling convolution with* G_ρ. *(b)–(c) The structure tensor is computed without oversampling and then smoothed with the* G_ρ. *In all cases, a standard deviation of* $\rho = 2$ *is used for the Gaussian function.*

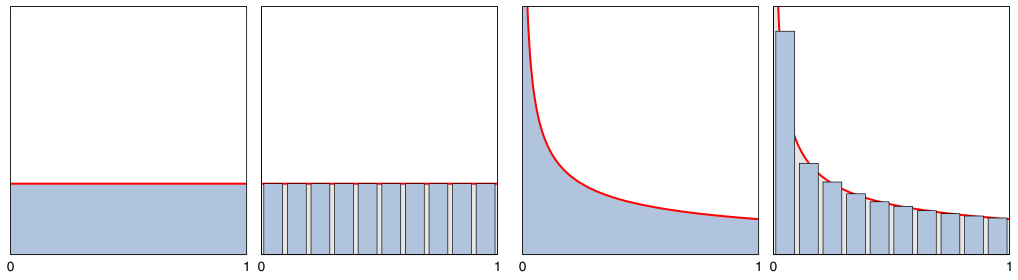
in the frequency domain, a standard deviation of $\sigma \approx 2$ would be required, resulting in large errors, as we saw in Figure 2.8(d). The second approach is to oversample the image by computing the Gaussian derivatives using resampling convolution [Sch92]. The smoothed structure tensor can then be obtained by downsampling using resampling convolution with a Gaussian function. As can be seen in Figure 2.17, oversampling indeed leads to improved results if no noise is present. If noise is present, however, the results gained by oversampling are similar to those obtained without over-sampling. Consequently, for the computation of the smoothed structure tensor, the optimized 5×5 derivative filter is the best choice in terms of accuracy.

The discussion so far assumed that the computed structure tensor is stored in a floating point texture. However, with a few exceptions of modern devices, such as the iPhone 4S and iPad 2, neither single nor half precision floating point texture are supported on most small devices. Straightforward quantization and storage in 8-bit textures produces significant errors, as demonstrated in Figure 2.18(a) and Figure 2.18(b). This is again due to the computation of the products of the partial derivatives. If we assume that all gradient values are equally likely to occur, that is, the distribution of the gradients is uniform, then the product of the gradient values has a non-uniform distribution where smaller values are more likely to occur (Figure 2.19). This is problematic from the point of quantization, since the data is not distributed equally into the different quantization bins. The reason for the



(a) 3×3 (b) 3×3 *(normalized)* (c) 5×5 (d) 5×5 *(normalized)*

Figure 2.18: *Comparison of the angular errors produced by storing the structure tensor in an 8-bit RGB texture with and without normalization of the gradient magnitude.*



(a) X uniform$(0,1)$ (b) X^2

Figure 2.19: *The square of the uniform distribution is non-uniform and given by* $f_{X^2}(x) = \frac{1}{2\sqrt{x}}$, *resulting in a non-uniform quantization.*

quadratic nature of the structure becomes apparent when looking at Equation (2.96): The structure tensor can be regarded as the outer product of the normalized gradients multiplied with the squared gradient magnitude:

$$J(x) = \nabla f(x) \nabla f(x)^T = \|\nabla f(x)\|^2 \left(\frac{\nabla f(x)}{\|\nabla f(x)\|} \right) \left(\frac{\nabla f(x)}{\|\nabla f(x)\|} \right)^T \tag{2.133}$$

A simple and straightforward workaround is to normalize the structure tensor by dividing by the gradient magnitude. As can be seen in Figure 2.18(c) and Figure 2.18(d) this indeed provides a significant improvement. This approach can also easily be generalized to color images by computing for each color channel the normalized structure tensor independently, and then taking their sum. Notice that smoothing the normalized structure tensor produces a different result, since the normalization changes the definition of the residual Equation (2.108).

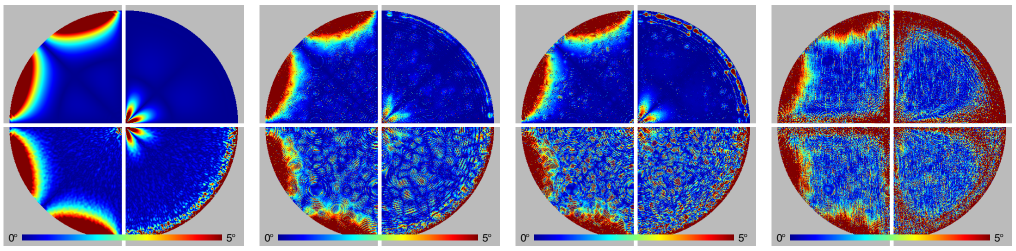
2.5 Comparison with Edge Tangent Flow

The *edge tangent flow* filter (ETF) was first proposed by Kang et. al. [KLC07; KLC09]. Technically, it can be understood as a multilateral filter that has been specifically designed to operate on orientation vectors. To achieve reasonable results, multiple iterations of the filter are applied. The processing starts with an initial tangent field t^0 obtained by rotating the image gradients 90 degrees. A single iteration of the edge tangent flow filter is then defined by

$$t^{n+1}(x) = \sum_{y \in \mathcal{N}(x)} w_s(x,y)\, w_m(x,y)\, w_d(x,y)\, \frac{t^n(y)}{\|t^n(y)\|} \,, \tag{2.134}$$

where $\mathcal{N}(x)$ denotes the local neighborhood of x and w_s, w_m and w_d are functions controlling the weight given to a summand.

The term $w_s(x,y)$ denotes the *spatial weight function*. It controls the weight depending on the spatial relation of x and y. Kang et al. use a radially-symmetric box filter with radius r that is one for $\|x - y\| < r$ and zero otherwise. Another reasonable choice would be to use, for example, a Gaussian function. This would give pixels that are farther away from the filter origin less influence.



(a) *SST* (b) *Box ETF* (c) *Gaussian ETF* (d) *xy-separable ETF*

Figure 2.20: *Comparison of the angular errors produced by the smoothed structure tensor and different variants of the edge tangent flow. The gradients used in this experiment were computed using the optimized* 3×3 *derivative filter. The standard deviation of the Gaussian filter was set to* $\rho = 2$*, and for the cut-off of the Gaussian and Box filters,* $r = 3\rho$ *was used.*

The *magnitude weight function* is defined by

$$w_m = \frac{1}{2}\left(\frac{\|t^0(y)\| - \|t^0(x)\|}{t^0_{\max}} + 1\right), \tag{2.135}$$

where $t^0_{\max} = \max_x \|t^0(x)\|$ is the maximum length of the initial tangents. It controls the weight depending on the relation of the lengths $\|t^0(x)\|$ and $\|t^0(y)\|$. If $\|t^0(y)\|$ has greater magnitude than $\|t^0(x)\|$, the weight will be in the range $(0.5, 1]$, and vice-versa in the case of lower magnitude. Thus, more weight is given to tangents with greater magnitude.

Lastly, the *directional weight function* is set to the scalar product

$$w_d = \left\langle \frac{t^n(y)}{\|t^n(y)\|}, \frac{t^n(x)}{\|t^n(x)\|} \right\rangle \tag{2.136}$$

of the normalized tangents. Since scalar product is equivalent to the cosine of the angle between both tangents, more weight is given to tangents that have similar orientation. If the angle is greater than 90 degrees, the weight is negative. Thus, $w_d(x, y)\, t^n(y)$ always lies in the half space defined by $t^n(x)$.

The edge tangent flow creates smooth tangent fields suitable for IB-AR, but it is computationally expensive due to its quadratic per pixel complexity. Kang et. al. [KLC09] therefore proposed to implement the filter in a separable way, similar to the xy-separable bilateral filter that will be discussed later. Figure 2.20 shows an evaluation of the smoothed structure tensor and the edge tangent flow for the test pattern of Figure 2.7. As can be seen the structure tensor is not only computationally more efficient, it also clearly outperforms the edge tangent flow in terms of accuracy of the local orientation estimation. Another disadvantage of the ETF is that it does not provide a confidence measure for the averaged orientation. The next section discusses how such a measure can be defined for the structure tensor.

2.6 Anisotropy and Cornerness Measures

Besides the superior robustness, the smoothed structure tensor has a second advantage over orientation averaging techniques like for example the mean of angles or the edge tangent flow. Its eigenvalues allow for defining a measure for the anisotropy of a pixel's neighborhood. Moreover, several corner detectors utilize the eigenvalues of the structure tensor in their definition.

2.6.1 Anisotropy Measures

The minor eigenvalue λ_2 of the structure tensor measures how much the gradients deviate from the axis defined by the major eigenvector. It provides a way to measure the quality of the orientation estimation. If $\lambda_1 \gg \lambda_2$ there is a clear dominant orientation in the considered neighborhood. If on the other hand $\lambda_1 \approx \lambda_2$ there is no particular distinguished axis among the gradients.

The normalized difference of the eigenvalues thus provides a *measure of anisotropy*

$$\mathcal{A} = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} = \frac{\sqrt{(E-G)^2 + 4F^2}}{E+G} = \frac{\sqrt{\operatorname{tr}^2(J) - 4\det(J)}}{\operatorname{tr}(J)} \,, \tag{2.137}$$

which has been adopted by many authors [Yan+96; BVV99; Pha06]. The anisotropy $\mathcal{A}$ ranges from 0 to 1, where 0 corresponds to isotropic, and 1 corresponds to entirely anisotropic regions.

Since the eigenvalues λ_1 and λ_2 correspond to the squared rate of change and the squared error of the estimate, respectively, another reasonable definition of anisotropy is given by [FM02; Fen03]:

$$\mathcal{A}_1 = \frac{\sqrt{\lambda_1} - \sqrt{\lambda_2}}{\sqrt{\lambda_1} + \sqrt{\lambda_2}} \tag{2.138}$$

Probably to avoid the evaluation of the square root in Equation (2.137), Förstner and Gülch [För86; FG87] as well as Weickert [Wei99] used the squared anisotropy in their works:

$$\mathcal{A}_2 = \mathcal{A}^2 = \left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}\right)^2 = \frac{(E-G)^2 + 4F^2}{(E+G)^2} = 1 - \frac{4\det(J)}{\operatorname{tr}^2(J)} \tag{2.139}$$

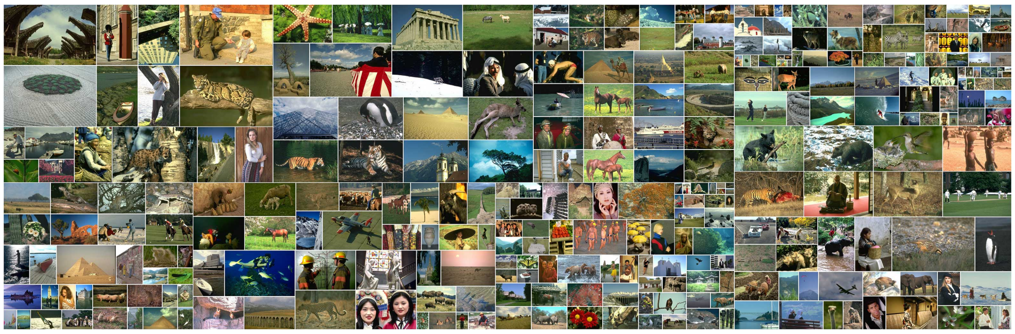
In order to decide which of the above anisotropy measures to use in this work, an experiment was conducted, where the anisotropy measures for the 300 natural images in the Berkeley segmentation dataset (BSDS300) were computed. The resulting histograms are shown in Figure 2.21. Since the histogram of the anisotropy measure defined by Equation (2.137) is more uniformly distributed (Figure 2.21(b)) than the others (Figure 2.21(c)–(d)), it is adopted here for the purpose of image stylization.

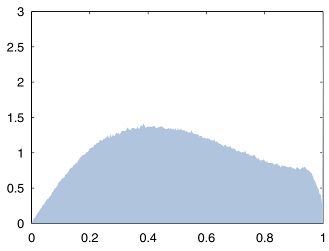
2.6.2 Relation of Anisotropy and Condition Number

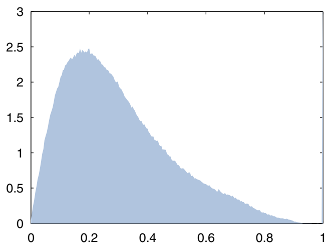
Let $A \in \mathbb{R}^{m \times n}$ be a matrix with $\operatorname{rank}(A) = \min(m,n)$, then the *condition number* of A is defined as the ratio between the largest and smallest eigenvalue of A:

$$\kappa = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \tag{2.140}$$

The condition number plays a central role in the numerical analysis of matrix computations and provides an upper bound on the amplification of relative errors. The definitions of


(a) *The Berkeley segmentation dataset (BSDS300) [Mar+01]*


(b) $\mathfrak{A}$


(c) $\mathfrak{A}_1$

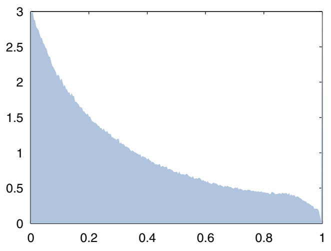

(d) $\mathfrak{A}_2$

Figure 2.21: *Histograms of the different anisotropy variants for the Berkeley segmentation dataset.*

anisotropy and condition number are both based on the eigenvalues and are related by:

$$\mathfrak{A} = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} = \frac{\frac{\lambda_1}{\lambda_2} - 1}{\frac{\lambda_1}{\lambda_2} + 1} = \frac{\kappa - 1}{\kappa + 1} \tag{2.141}$$

$$\kappa = \frac{\lambda_1}{\lambda_2} = \frac{1 + \mathfrak{A}}{1 - \mathfrak{A}} \tag{2.142}$$

2.6.3 Anisotropy of Random Gradients

A real *Wishart matrix* is a symmetric $m \times m$ matrix of the form $W = A^T A$, where $A \in \mathbb{R}^{m \times n}$ has normally, independent, and identically distributed elements [ER05]. If an image has normally, independent, and identically distributed random pixels, then also its gradients are normally, independent, and identically distributed. Thus, by Equation (2.122), the structure tensor $J = G^T G$ computed from such random gradients is a Wishart matrix. This is interesting, since a closed expression of the joint eigenvalue distribution of Wishart matrices is known [Ede88; Ede89]. In this section it will be shown how this joint eigenvalue distribution can be used to derive the probability distribution of the anisotropy for structure tensors that are Wishart matrices. The distribution can then be used to check if it is likely that a structure tensor has been obtained from a white noise image neighborhood.

The joint eigenvalue distribution of a real Wishart matrix is given by

$$K_{m,n} \cdot \exp\left(-\frac{1}{2}\sum_{i=1}^{n} \lambda_i\right) \cdot \prod_{i=1}^{n} \lambda_i^{\frac{m-n-1}{2}} \cdot \prod_{i<j} (\lambda_i - \lambda_i) \,, \tag{2.143}$$

with

$$K_{m,n}^{-1} = \left(\frac{2^m}{\pi}\right)^{n/2} \prod_{i=1}^{n} \Gamma\left(\tfrac{m-i+1}{2}\right)\Gamma\left(\tfrac{n-i+1}{2}\right) . \tag{2.144}$$

For $n = 2$ this simplifies to:

$$K_{m,2}\, e^{-\frac{1}{2}\left(\lambda_1+\lambda_2\right)} \left(\lambda_1\lambda_2\right)^{\frac{m-3}{2}} \left(\lambda_1 - \lambda_2\right) \tag{2.145}$$

Using the recurrence relation $\Gamma(z+1) = z\Gamma(z)$ and the duplication formula $\Gamma(z)\Gamma\big(z + \frac{1}{2}\big) = 2^{1-2z}\sqrt{\pi}\Gamma(2z)$ of the gamma function yields the following simplification of the normalization term for $n = 2$:

$$\begin{aligned} K_{m,2}^{-1} &= \frac{2^m}{\pi} \prod_{i=1}^{2} \Gamma\left(\tfrac{m-i+1}{2}\right)\Gamma\left(\tfrac{2-i+1}{2}\right) = \frac{2^m}{\pi}\Gamma\left(\tfrac{m}{2}\right)\underbrace{\Gamma(1)}_{=1}\Gamma\left(\tfrac{m-1}{2}\right)\underbrace{\Gamma\left(\tfrac{1}{2}\right)}_{=\sqrt{\pi}} \\ &= \frac{2^m}{\sqrt{\pi}}\Gamma\left(\tfrac{m}{2}\right)\Gamma\left(\tfrac{m-1}{2}\right) = \frac{2^m}{\sqrt{\pi}} 2^{2-m}\sqrt{\pi}\,\Gamma(m-1) = \frac{4\Gamma(m)}{m-1} \end{aligned} \tag{2.146}$$

In order to obtain the probability density function of the condition number κ, the quotient distribution is computed. Following the common approach an auxiliary variable t is introduced:

$$\kappa = \frac{\lambda_1}{\lambda_2} \qquad t = \lambda_2 \tag{2.147}$$

The inverse and Jacobi determinant of the transformation are then given by:

$$\begin{aligned} \lambda_1 &= \kappa t \\ \lambda_2 &= t \end{aligned} \qquad \det\begin{pmatrix} \frac{\partial\lambda_1}{\partial\kappa} & \frac{\partial\lambda_1}{\partial t} \\ \frac{\partial\lambda_2}{\partial\kappa} & \frac{\partial\lambda_2}{\partial t} \end{pmatrix} = \begin{pmatrix} t & \kappa \\ 0 & 1 \end{pmatrix} = t \tag{2.148}$$

Applying the transformation to Equation (2.145) we obtain

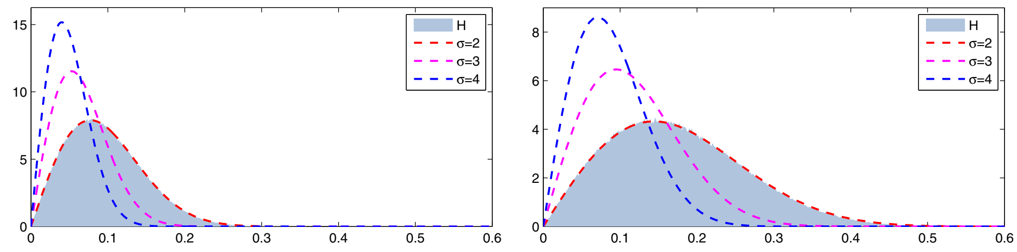
$$K_{m,2}\, e^{-\frac{1}{2}(\kappa+1)t} \left(\kappa t^2\right)^{\frac{m-3}{2}} (\kappa - 1)\, t^2 , \tag{2.149}$$

and marginalizing out t now yields the probability density function of the condition number κ:

$$\begin{aligned} \int_0^\infty & K_{2,n}\, e^{-\frac{1}{2}(\kappa+1)t} \left(\kappa t^2\right)^{\frac{m-3}{2}} (\kappa-1)\, t^2 \,\mathrm{d}t \\ &= K_{m,2}\, \kappa^{\frac{m-3}{2}} (\kappa-1) \int_0^\infty e^{-\frac{1}{2}(\kappa+1)t}\, t^{m-1} \,\mathrm{d}t \\ &= K_{m,2}\, \kappa^{\frac{m-3}{2}} (\kappa-1) \frac{2^m}{(\kappa+1)^m} \int_0^\infty e^{-u}\, u^{m-1} \,\mathrm{d}u \\ &= K_{m,2}\, \kappa^{\frac{m-3}{2}} (\kappa-1) \frac{2^m}{(\kappa+1)^m} \Gamma(m) \\ &= (m-1)\, 2^{m-2}\, \frac{\kappa-1}{(\kappa+1)^m}\, \kappa^{\frac{m-3}{2}} \end{aligned} \tag{2.150}$$

The probability density function of the anisotropy $\mathfrak{A}$ can now easily be derived through another change of variables $\mathfrak{A} = (\kappa-1)/(\kappa+1)$. The derivative of the inverse (Equation (2.142)) is given by

$$\frac{\mathrm{d}\kappa}{\mathrm{d}\mathfrak{A}} = \frac{\mathrm{d}}{\mathrm{d}\mathfrak{A}}\left(\frac{1+\mathfrak{A}}{1-\mathfrak{A}}\right) = \frac{2}{(1-\mathfrak{A})^2} \tag{2.151}$$



(a) $m = (2r+1)^2$, $r = 3\sigma$ *(Box)* (b) $m = 4\pi\sigma^2$ *(Gaussian)*

Figure 2.22: *Probability distribution of the anisotropy of random structure tensors. (a) Box filtered structure tensor. (b) Smoothed structure tensor obtained with Gaussian filter and cut-off radius* 3σ. *In both cases, plots of the distribution for different neighborhood sizes are shown. Moreover, the histograms of an experiment are shown, where the anisotropies of* 10^6 *randomly generated matrices were computed.*

and applying the transform to Equation (2.150) we get:

$$\begin{aligned} p(\mathcal{A}) &= (m-1)\,2^{m-2}\left(\frac{1+\mathcal{A}}{1-\mathcal{A}}-1\right)\left(\frac{1+\mathcal{A}}{1-\mathcal{A}}+1\right)^{-m}\left(\frac{1+\mathcal{A}}{1-\mathcal{A}}\right)^{\frac{m-3}{2}}\frac{2}{(1-\mathcal{A})^2} \\ &= (m-1)\mathcal{A}\left(1-\mathcal{A}^2\right)^{\frac{m-3}{2}} \end{aligned} \tag{2.152}$$

A similar result for the anisotropy measure defined in Equation (2.138) was obtained by Feng and Milanfar [FM02; Fen03]. Figure 2.22 shows an experimental validation of this result, as well as plots for different n. The cumulative distribution function of the anisotropy is given by:

$$\begin{aligned} \Pr(\mathcal{A} \le a) &= \int_0^a p(\mathcal{A})\,\mathrm{d}\mathcal{A} = \int_0^a (m-1)\mathcal{A}\left(1-\mathcal{A}^2\right)^{\frac{m-3}{2}}\,\mathrm{d}\mathcal{A} \\ &= \left[-\left(1-\mathcal{A}^2\right)^{\frac{m-1}{2}}\right]_0^a = 1-\left(1-a^2\right)^{\frac{m-1}{2}} \end{aligned} \tag{2.153}$$

The cumulative distribution function can now be used to perform a significance test [Dra67; YG05]. To this end, given some significance level $\alpha \in [0,1]$, we have to solve $\Pr(\mathcal{A} \le a) = 1-\alpha$ and thus obtain as a threshold:

$$\mathcal{A}^* = \sqrt{1-\alpha^{2/(m-1)}} \tag{2.154}$$

For example, if a structure tensor computed on a 13×13 neighborhood has anisotropy $\mathcal{A} > \sqrt{1-0.05^{2/(13^2-1)}} = 0.1872$ then the hypothesis that the structure tensor is a random Gaussian matrix can be rejected with a 95% confidence.

The discussion so far only applies to box-filtered structure tensors G^TG. Apparently, if Gaussian weighting is included, then $(WG)^TWG$ is not a Wishart matrix. It is therefore not surprising that the corresponding anisotropy distribution, shown in Figure 2.22(b), is different. However, as can be seen, the overall shape of the distribution is very similar, leading to the idea to search for a matching m by fitting Equation (2.152) to the anisotropy distribution obtained by simulation. To this end, an experiment was conducted in MATLAB. For different σ and cut-off radii $p\sigma$, 10^6 random Gaussian $\times$ matrices were generated, and their structure tensors and anisotropies computed. The histogram of the anisotropies was

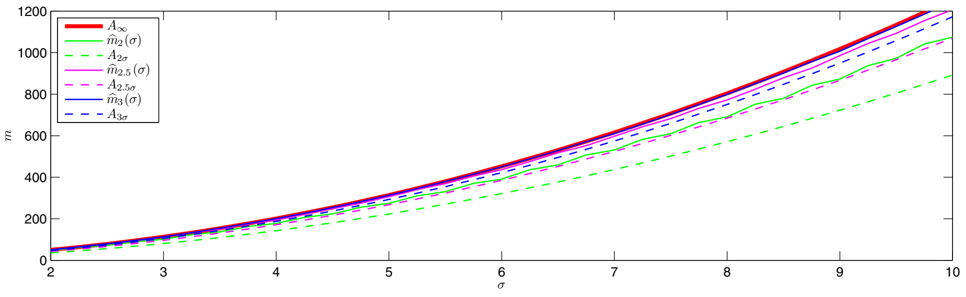


Figure 2.23: *Results of the least squares fit of Equation* (2.152) *for the cut-off radii* $p\sigma = 2\sigma, 2.5\sigma, 3\sigma$. *Moreover, the area* A_∞ *under the Gaussian function, and the areas* $A_{p\sigma}$ *under the Gaussian filter kernel are shown.*

then used to perform a least squares fit of Equation (2.152), which resulted in an L2-error of 0.1% or less, showing that histogram and distribution indeed match. The resulting $\hat{m}_{p\sigma}(\sigma)$ for different cut-off radii are shown in Figure 2.23. As can be seen, the results depend on the cut-off radius. However, it can be observed that the result $\hat{m}_{p\sigma}(\sigma)$ of the least squares fit is bounded by

$$A_{p\sigma} < \hat{m}_{p\sigma}(\sigma) < A_\infty\,, \tag{2.155}$$

where

$$A_{p\sigma} = \int_{-p\sigma}^{+p\sigma}\int_{-p\sigma}^{+p\sigma} \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)\,\mathrm{d}x = 4\pi\sigma^2\,\mathrm{erf}^2\left(\frac{p}{2}\right) \tag{2.156}$$

is the area of the exponential functional under the filter neighborhood, and

$$A_\infty = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)\,\mathrm{d}x = 4\pi\sigma^2 \tag{2.157}$$

is the area of the exponential functional for all of $\mathbb{R}^2$. Consequently, for cut-off radii $p\sigma$ with $p \geq 2.5$ a reasonable approximation is given by:

$$m(\sigma) = 4\pi\sigma^2 \tag{2.158}$$

2.6.4 Cornerness Measures

Similar to anisotropy measures, which measure if a neighborhood has a distinguished orientation, it is also possible to define measures that identify corner regions. One of the best known corner detectors is probably the *Harris corner detector* [HS88]. Interestingly, this detector is based on the smoothed structure tensor, and its corner response function given by:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2) = \det(J_\rho) - k\,\mathrm{tr}(J_\rho)^2 \tag{2.159}$$

The response function R is thresholded and non-maximum suppression performed by checking an 8-neighborhood. The control parameter k is an empirically determined constant and typically chosen to lie in the range $[0.04, 0.06]$. Underlying idea of the corner response function R is that if the major eigenvector is large and the minor eigenvector small, then R is negative, indicating an edge region. If both eigenvalues are small, then $|R|$ is small,

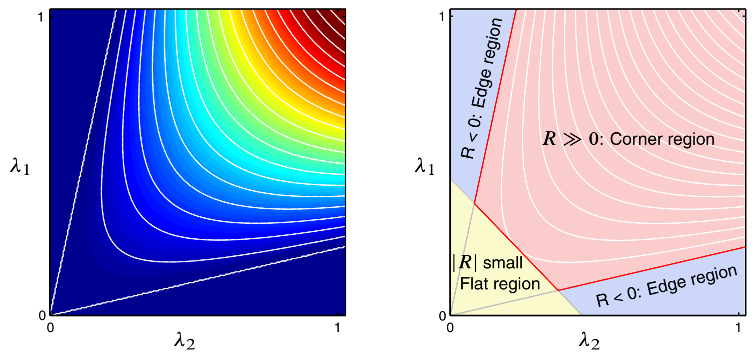


Figure 2.24: *Illustration of the Harris-Stephens corner response function $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$. On the left, a plot of R is shown. On the right, the different regions corresponding to edge regions, flat regions, and corner regions are shown.*

indicating a flat region. If both eigenvalues are comparatively larger, R is positive, and the larger the eigenvalues are, the larger R is. Figure 2.24 illustrates the relationship between R and the corresponding region classification.

The *Förstner interest operator* [För86; FG87], whose inverse is also known as the *Plessey detector* [Nob88], is another approach that is based on the structure tensor:

$$\lambda_1 + \lambda_2 = \operatorname{tr}(J_\rho) > \Theta \quad \text{and} \quad \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2} = \frac{\det(J_\rho)}{\operatorname{tr}(J_\rho)} \equiv \text{local maximum} \tag{2.160}$$

The Harris corner detector as well as the Förstner interest operator, both avoid using the eigenvalues of the structure tensor directly and instead use the determinant and trace. Nevertheless, a large minor eigenvalue is good a indicator for a corner region and proposed by Tomasi and Kanade [TK91]:

$$\lambda_2 > \Theta \quad \text{and} \quad \lambda_2 \equiv \text{local maximum} \tag{2.161}$$

As a further approach Rohr [Roh87] has proposed to use the local maxima of the determinant $\det(J_\rho(x))$ for corner detection. He also provided an analysis of localization properties of different corner detectors [Roh94].

2.7 Sampling and Interpolation

In this section, sampling of the structure tensor at arbitrary locations, bilinear sampling, and the replacement of unreliable structure tensors will be discussed.

2.7.1 Sampling at Arbitrary Locations

In Section 2.4, the smoothed structure tensor was defined as the discrete convolution of the structure tensor with a Gaussian function. However, instead of regarding the convolution as smoothing operation, we may alternatively interpret it as reconstruction, yielding a continuous structure tensor from a set of samples. Or in other words, although the image gradients are only defined at integer points, the Gaussian function may be evaluated at arbitrary positions. However, if non-integer values are allowed for x in Equation (2.84),

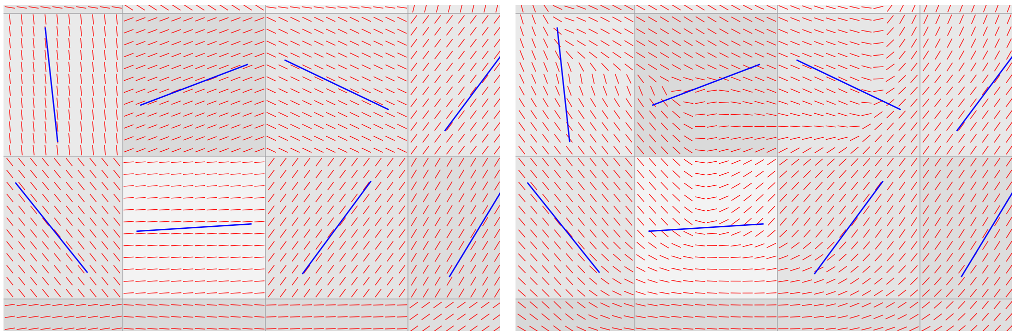

(a) *Nearest-neighbor* (b) *Bilinear*

Figure 2.25: *Illustration of the minor eigenvector of the smoothed structure tensor (blue) for nearest-neighbor and bilinear sampling (red).*

we have to account for that graphics hardware defines the center of a pixel at half-integer positions. Hence, computation of the smoothed structure tensor at non-integer positions is given by:

$$\frac{1}{|G_\rho|} \sum_{y \in \mathcal{N}(x)} G_\rho\Big(x - \big(y + \big(\begin{smallmatrix}0.5\\0.5\end{smallmatrix}\big)\big)\Big)\, g(y)g(y)^T \tag{2.162}$$

with $|G_\rho|$ denoting the corresponding normalization term.

2.7.2 Bilinear Interpolation

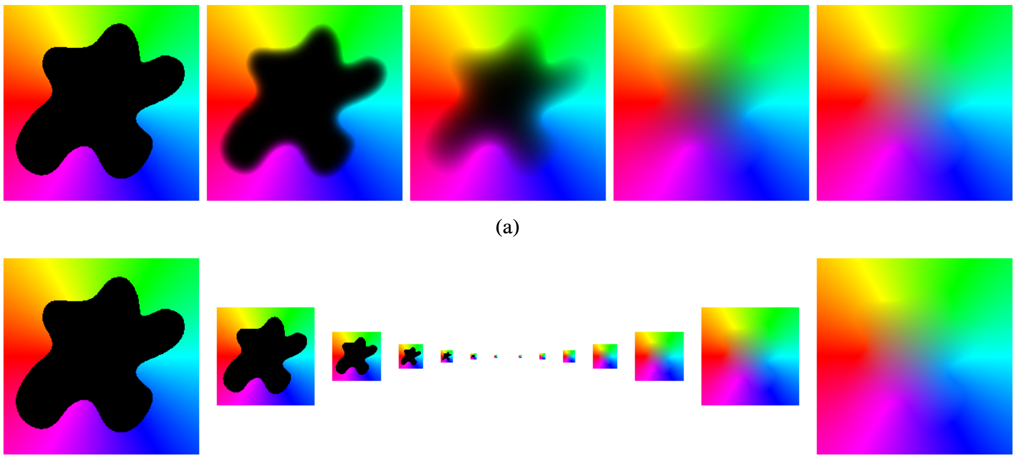
The reconstruction at arbitrary positions discussed in the previous section is computationally expensive when performed for a large amount of samples. An interesting alternative is to approximate Equation (2.162) by bilinearly sampling the smoothed structure tensor. Let $x = \sum_{i=1}^{4} \alpha_i x_i$, where $x_1, \ldots, x_4$ are four neighboring pixels and $\sum_{i=1}^{4} \alpha_i = 1$ with $a_i \geq 0$. Moreover, let $J_1, \ldots, J_4$ denote the smoothed structure tensors at the corresponding pixels. As already noted in Section 2.3.3, the weighted sum $\sum_{i=1}^{4} \alpha_i J_i$ is symmetric and positive semidefinite. Performing the convolution of the structure tensor with the Gaussian function for the sake of simplicity over $\mathbb{Z}^2$, we have:

$$\begin{aligned} \sum_{i=1}^{4} \alpha_i J_i &= \sum_{i=1}^{4} \alpha_i \left(\frac{1}{|G_\rho|} \sum_{y \in \mathbb{Z}^2} G_\rho(x_i - y)\, g(y)g(y)^T \right) \\ &= \frac{1}{|G_\rho|} \sum_{y \in \mathbb{Z}^2} \left(\sum_{i=1}^{4} \alpha_i\, G_\rho(x_i - y) \right) g(y)g(y)^T \end{aligned} \tag{2.163}$$

Hence, we see that bilinear interpolation of the smoothed structure tensor corresponds to a reconstruction of the structure, where instead of a Gaussian function, the bilinear interpolation of four Gaussian functions is used.

2.7.3 Inpainting of Low-contrast Regions

In low-contrast regions, the signal-to-noise ratio is low, making the gradient information unreliable. Accordingly, the estimated orientation is almost random and of little value. In


(a)

(b)

Figure 2.26: *Solution of the Laplace equation. (a) Iterative solution using 100, 1,000, 5,000, and 50,000 Jacobi iterations. (b) Multi-scale approach.*

this section an approach for replacing unreliable structure tensors will be presented. As explained earlier, the square root of the major eigenvalue is a generalization of the gradient magnitude, and can thus be used to identify points with reliable structure tensors

$$\partial\Omega = \left\{ x \in \Omega \mid \sqrt{\lambda_1} > \tau_r \right\}, \tag{2.164}$$

where τ_r is a control parameter. The idea is now to look for a smooth function that interpolates the structure tensors S defined on $\partial\Omega$. Such a function is given by the membrane that minimizes

$$\arg\min_s \int_\Omega |\nabla s|^2 \, \mathrm{d}x \quad \text{with} \quad s|_{\partial\Omega} = S|_{\partial\Omega} \,. \tag{2.165}$$

This problem is known to be equivalent to solving Laplace's equation with a corresponding Dirichlet boundary condition

$$\Delta s = 0 \quad \text{with} \quad s|_{\partial\Omega} = S|_{\partial\Omega} \,, \tag{2.166}$$

where

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} \tag{2.167}$$

denotes the Laplace operator, which is given by the sum of the second order partial derivatives. These can be discretized using finite differences

$$\frac{\partial^2 s}{\partial x_1^2} \approx \frac{s_{i+1,j} - 2s_{i,j} + s_{i-1,j}}{h^2} \quad \text{and} \quad \frac{\partial^2 s}{\partial x_2^2} \approx \frac{s_{i,j+1} - 2s_{i,j} + s_{i,j-1}}{h^2} \,, \tag{2.168}$$

leading to the following discretization of the Laplace operator:

$$\Delta s|_{i,j} \approx \frac{s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} - 4s_{i,j}}{h^2} \tag{2.169}$$

The equations $\Delta s|_{i,j}$ form a large sparse linear system of equations, which is best visualized by concatenating all rows of the image into a large vector. Neglecting boundary conditions

```
__global__ void jacobi_step( gpu_plm2<float4> dst ) {
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float4 st = tex2D(texST, ix, iy);
    if (st.w < 1) {
        st = make_float4((
                make_float3(tex2D(texST, ix+1, iy  )) +
                make_float3(tex2D(texST, ix-1, iy  )) +
                make_float3(tex2D(texST, ix,   iy+1)) +
                make_float3(tex2D(texST, ix,   iy-1))) / 4,
                0);
    }

    dst(ix, iy) = st;
}
```

Listing 2.3: *Implementation of a Jacobi relaxation step.*

for a moment, this results in the following system of equations

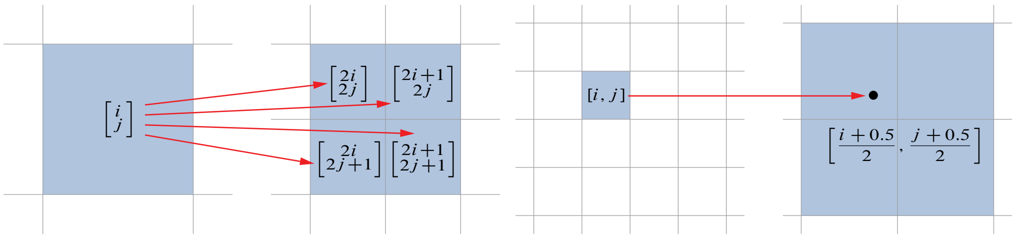
$$\begin{pmatrix}
-4 & 1 & \cdots & 1 & \cdots & & & \cdots & 0 \\
1 & -4 & 1 & \cdots & 1 & \cdots & & & \vdots \\
 & \ddots & \ddots & \ddots & & \ddots & & & \\
\ddots & & \ddots & \ddots & \ddots & & \ddots & & \\
\cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots \\
 & & \ddots & & \ddots & \ddots & \ddots & & \ddots \\
 & & & \ddots & & \ddots & \ddots & \ddots & \\
\vdots & & & & 1 & \cdots & 1 & -4 & 1 \\
0 & \cdots & & & & 1 & \cdots & 1 & -4
\end{pmatrix} \cdot \begin{pmatrix} s_{0,0} \\ s_{0,1} \\ s_{0,2} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ s_{M-1,N-3} \\ s_{M-1,N-2} \\ s_{M-1,N-1} \end{pmatrix} = 0 \qquad (2.170)$$

with minus four on the diagonal, ones on four secondary diagonals, and zeros elsewhere. In order to incorporate boundary conditions, rows corresponding to boundary pixels must be replaced with a one on the diagonal, and zeros elsewhere.

In principle, any technique for solving linear systems can be used. One of the simplest approaches is to assume that in the i-th equation only the i-th parameter is unknown, leaving the other parameters fixed. Solving each of these equations independently and iterating the process converges to the solution, and is known as the *Jacobi* method. More specifically, let $s^k_{i,j}$ denote the k-th step's structure tensor at pixel (i, j); then a Jacobi relaxation step is given by

$$s^{k+1}_{i,j} = \begin{cases} s^k_{i,j} & \text{if } (i,j) \in \partial\Omega \\ \dfrac{s^k_{i+1,j} + s^k_{i-1,j} + s^k_{i,j+1} + s^k_{i,j-1}}{4} & \text{otherwise} \end{cases} \qquad (2.171)$$

Since the computation involves a convex combination, the result is again a positive semidefinite matrix and, thus, is well-defined. Unfortunately, a very large number of iterations is


$\begin{bmatrix} i \\ j \end{bmatrix}$
$\begin{bmatrix} 2i \\ 2j \end{bmatrix}$
$\begin{bmatrix} 2i+1 \\ 2j \end{bmatrix}$
$\begin{bmatrix} 2i \\ 2j+1 \end{bmatrix}$
$\begin{bmatrix} 2i+1 \\ 2j+1 \end{bmatrix}$
$[i, j]$
$\left[\frac{i+0.5}{2}, \frac{j+0.5}{2}\right]$

(a) *Restriction* (b) *Interpolation*

Figure 2.27: *Illustration of the restriction and interpolation operations. (a) Restriction collapses four pixels into a single pixel by averaging those that are boundary pixels. (b) Interpolation is performed using bilinear interpolation.*

typically required, which is demonstrated in Figure 2.26(a). Obtaining a sufficient approximation of the solution, takes approximately 50,000 Jacobi iterations. Even on modern high-end GPUs, this takes several seconds to compute.

Jacobi iterations are effective for removing high-frequency oscillations in the residual, but perform rather poorly when the residual becomes smooth. Multigrid methods [BHM00] address this issue by solving for the residual on a coarser level. A similar approach, which can be regarded as a simplified variant of a multigrid solver, where computation of the residual is avoided, is adopted here. As a first step, which structure tensors should be kept unmodified is determined by using Equation 2.164. To this end, the fourth color channel is used, with one indicating a boundary pixel and zero used otherwise. The reason why convergence of the Jacobi method is slow is illustrated in Figure 2.26(a). One Jacobi iteration computes the average of the neighboring pixels; consequently, it takes a large number of iterations until values on the boundary diffuse into the inner parts. Apparently, a simple way to speed up the diffusion is to compute it on a coarser grid. Since the transfer to a coarser grid can be repeated recursively, this yields a pyramid of images. Moving from a finer to a coarser level is referred to as restriction. The pixels on a coarser pyramid level are defined

```
__global__ void restrict( const gpu_plm2<float4> st, gpu_plm2<float4> dst ) {
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float4 sum = make_float4(0);
    float4 tmp;
    tmp = st(2*ix,   2*iy  ); if (tmp.w > 0) { sum += tmp; }
    tmp = st(2*ix+1, 2*iy  ); if (tmp.w > 0) { sum += tmp; }
    tmp = st(2*ix,   2*iy+1); if (tmp.w > 0) { sum += tmp; }
    tmp = st(2*ix+1, 2*iy+1); if (tmp.w > 0) { sum += tmp; }

    if (sum.w > 0)
        sum /= sum.w;

    dst(ix, iy) = sum;
}
```

Listing 2.4: *Implementation of the restriction operation.*

```
__global__ void interpolate( const gpu_plm2<float4> st_fine, gpu_plm2<float4> dst ) {
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float4 st = st_fine(ix, iy);
    if (st.w < 1) {
        st = make_float4(make_float3(
                tex2D(texST, 0.5f * (ix + 0.5f),
                             0.5f * (iy + 0.5f) )), 0);
    }

    dst(ix, iy) = st;
}
```

Listing 2.5: *Implementation of the interpolation operation.*

as the average of four pixels on the finer pyramid level, with non-boundary pixels being excluded (Figure 2.27(a)). The left of Figure 2.26(b) exemplifies the pyramid construction. Once the finest pyramid level is reached, the pyramid is processed in a coarse-to-fine manner. On each pyramid level, 1–3 Jacobi iterations are performed. Non-boundary pixels on the next-finer pyramid level are then replaced by sampling the result using bilinear interpolation. These operations are repeated until the finest pyramid level has been reached, as shown on the right of Figure 2.26(b). The implementations of the restriction and interpolation operations are shown in Listing 2.4 and 2.5.

Nevertheless, performing the relaxation for every computation of the structure tensor is expensive. Therefore, the relaxation is only performed for the first computation of the structure tensor. All subsequent computations use the structure tensor of the previous computation for points not in $\partial\Omega$.

2.8 Multi-scale Estimation

The multi-scale local structure estimation is inspired by the multi-scale orientation estimation methods by Wilson et al. [WCB90] and Feng and Milanfar [FM02]. Similar to these approaches, a weighted linear combination is used to propagate the estimates from coarser to finer pyramid levels (Figure 2.28). However, in contrast to the other techniques that propagate gradient estimates, in this work the structure tensor is propagated.

The processing starts at the coarsest level of the pyramid, where the smoothed structure

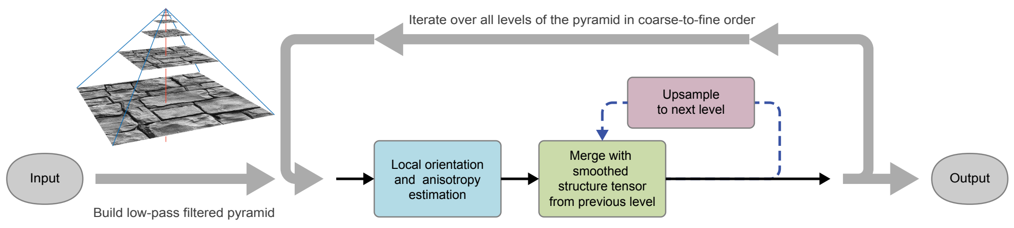

Iterate over all levels of the pyramid in coarse-to-fine order
Upsample to next level
Input
Build low-pass filtered pyramid
Local orientation and anisotropy estimation
Merge with smoothed structure tensor from previous level
Output

Figure 2.28: *Schematic overview of the multi-scale structure tensor estimation approach.*

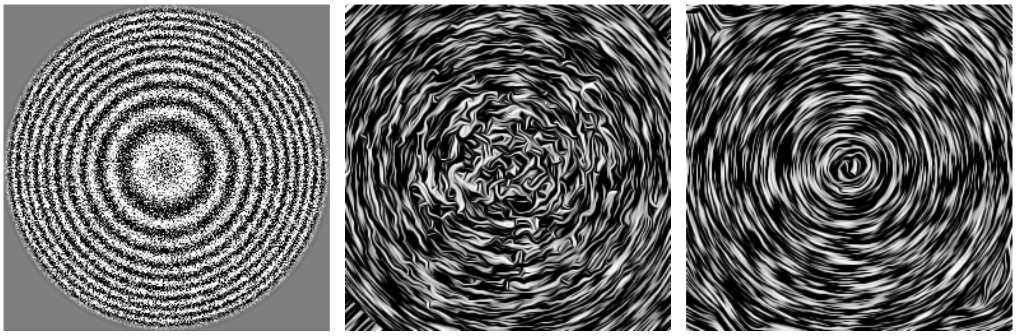

(a) *Test image* (b) *Single-scale* (c) *Multi-scale*

Figure 2.29: *Visualization of the minor eigenvector field of the smoothed structure tensor ($\rho = 2$) for a test ring pattern corrupted by Gaussian noise of variance $\nu = 0.25$.*

tensor is calculated as usual. For the other pyramid levels, the smoothed structure tensor is calculated and then combined with the upsampled structure tensor from the previous level using linear interpolation:

$$\tilde{J}_\rho^k = \alpha^k J_\rho^k + \left(1 - \alpha^k\right) \bar{J}_\rho^{k+1} \tag{2.172}$$

Here, $\bar{J}_\rho^{k+1}$ denotes the upsampled structure tensor from the previous level, that is, $\tilde{J}_\rho^{k+1}$ upsampled to the next level. J_ρ^k is the smoothed structure tensor computed from the merged image data of the current pyramid level. The linear weighting factor is defined per pixel and based on the anisotropy measure of the current level $\mathcal{A}$ and the upsampled anisotropy measure of the previous level $\bar{\mathcal{A}}^{k+1}$:

$$\alpha^k = \frac{\mathcal{A}}{\mathcal{A} + \bar{\mathcal{A}}^{k+1}} \tag{2.173}$$

Hence, more weight is given to the structure tensor that is more anisotropic. This leads to a more robust estimation as shown in Figure 2.29.

2.9 Integral Curves and Line Integral Convolution

Let $v\colon \mathbb{R}^2 \to \mathbb{R}^2$ be a vector field and let (a, b) be an open interval. A curve $\gamma\colon (a, b) \to \mathbb{R}^2$ satisfying $\gamma'(t) = v(\gamma(t))$ for all $t \in (a, b)$ is called an *integral curve* or *stream line* of the vector field v. Taken together, the minor eigenvectors of the smoothed structure tensors at each pixel define a vector field, which is smooth up to a change of sign and closely aligned to image features. The general idea behind flow-guided filtering methods is to perform the filtering operation by following the minor eigenvectors through tracing the corresponding stream line. In contrast to the isophote curves of the image, the stream lines defined by the smoothed structure tensor are much smoother, and smoothing along them results in a regularization of the geometry of the isophote curves. The next sections discuss different approaches for the computation of the stream lines. Moreover, filtering along them will be examined.

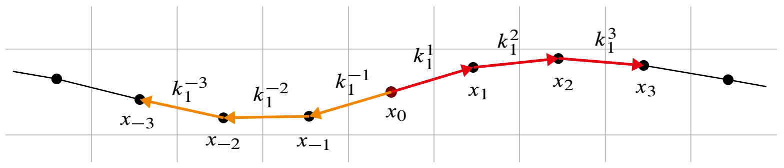


Figure 2.30: *Integration of a stream line passing through a point x_0 is performed iteratively in forward and backward directions.*

2.9.1 Euler Integration

Formally, finding the stream line $\gamma\colon (a,b) \to \mathbb{R}$ passing through a point x_0 of a vector field v can be described as solving the system of ordinary differential equations $\gamma'(t) = v(\gamma(t))$ with initial condition $\gamma(t_0) = x_0$. Approximating $\gamma'(t_0)$ by a forward finite difference we obtain:

$$v\big(\gamma(t_0)\big) = \gamma'(t_0) = \frac{\gamma(t_0+h) - \gamma(t_0)}{h} + O(h) \tag{2.174}$$

Rearranging terms yields an approximation for $\gamma(t_0+h)$, depending on $\gamma(t_0)$ and $v\big(\gamma(t_0)\big)$ only. Taking the initial condition as a starting point, the underlying idea of *Euler's method* is to apply this process iteratively. Setting $t^{i+1} = t^i + h$, we get the following approximation for the stream line γ:

$$\gamma(t^{i+1}) = \gamma(t^i) + hv\big(\gamma(t^i)\big) \tag{2.175}$$

If the considered vector field is given by the minor eigenvectors, special attention must be paid to their sign, since the structure tensor defines only orientation, but no particular direction. This is due to the quadratic nature of the structure tensor. A straightforward way to define the sign of the minor eigenvectors is to choose the sign that minimizes the curvature of the stream line. This can be achieved by ensuring that the scalar product between the minor eigenvectors of the current and the previous step is positive. Since for filtering operations a stream line passing through the filter origin is required, the integration is carried out in forward and backward directions, as shown in Figure 2.30. To this end, let x^0 be the starting point, and let $i \pm 1$ denote the next step in the corresponding direction (i.e., either forward or backward), then the next minor eigenvector is given by

$$k_1^{i\pm1} = \begin{cases} \pm\xi(x^0) & \text{if } i = 0 \\ \operatorname{sign}\big\langle k_1^i, \xi(x^i)\big\rangle \cdot \xi(x^i) & \text{otherwise} \end{cases} \tag{2.176}$$

where $\xi(x)$ denotes the minor eigenvector sampled at a point x and computed using Listing 2.1. Correspondingly, a step with step size h of Euler's method is given by

$$x^{i\pm1} = x^i + h\,k_1^{i\pm1}\,. \tag{2.177}$$

Integral curves also play a central role in stroke-based rendering.Early approaches used the image gradient [Hae90; Lit97] or image moments [TC97] to place short dabs of paint, typically, in form of opaque rounded or textured [HE04] rectangular strokes.However, such constant-sized rectangular strokes generate an artificial regularity that may degrade the resulting aesthetic. Hertzmann [Her98] was the first to address this issue by proposing curved brush strokes of multiple sizes. In order to derive the path of a brush stroke, he

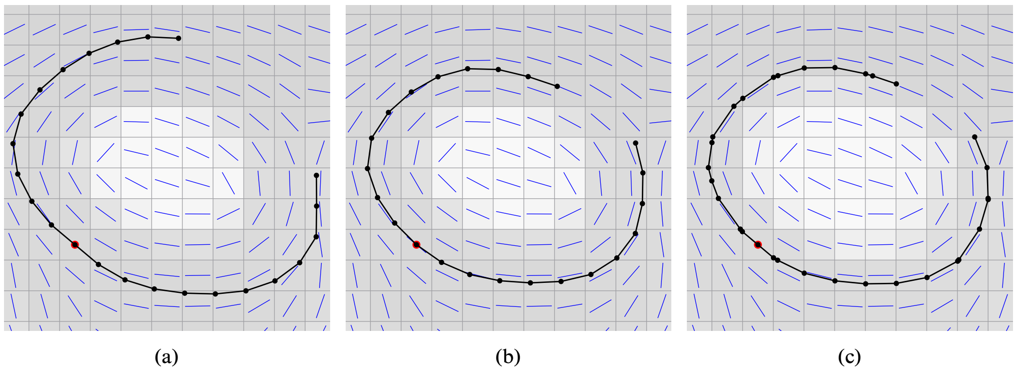


Figure 2.31: *Illustrations of different stream line integration methods. (a) First-order Euler integration. (b) Second-order Runge-Kutta. (c) Cabral-Leedom method.*

computed the luminance gradients of the blurred source image.At each seed point of a stroke, the stroke's path was then traced using Euler integration following the gradient field in normal direction. Thereby, the direction minimizing the curvature, as explained above,was chosen. Tracing stopped when either the maximum curve length was reached or the color under the path's current point deviated too much from that of the seed point. To account for noisy gradients, the traced points where smoothed by applying an infinite impulse response filter. While this approach would lead to severe artifacts in image filtering, this is typically not an issue for stroke-based rendering, since later painting passes can compensate for image errors introduced in previous passes.For this reason the less precise Euler integration method is usually sufficientfor stroke-based rendering approaches.Because of its simplicity and effectiveness, Hertzmann's curve tracing algorithm was adopted by later work [CH05; Zen+09] and became a common approach for stroke placement.

2.9.2 Second-order Integration

At least for long stream lines, which are typically required by flow-guided smoothing approaches, the Euler method is comparatively inaccurate. This is illustrated in Figure 2.31(a), where the stream line generated by the Euler integration method crosses adjacent isophote curves of the image. When smoothing is performed along the stream line, this corresponds to a smoothing operation in the direction of the major eigenvector, and a loss of information (Figure 6.7(a)). Therefore, instead of using the Euler's method, which is first-order, a more precise second-order method should be used, which traces stream lines at a higher accuracy (Figure 2.31(b)) and thereby reduces blurring across edges (Figure 6.7(c)).

Similar to Euler's method, a second-order method may be derived by approximating a derivative of γ by a finite difference. Instead of the forward difference, however, in this case $\gamma'(t_0 + h/2)$ is approximated by a central difference:

$$v\Big(\gamma\big(t_0 + \tfrac{h}{2}\big)\Big) = \gamma'\big(t_0 + \tfrac{h}{2}\big) = \frac{\gamma(t_0 + h) - \gamma(t_0)}{h} + O(h^2) \tag{2.178}$$

Rearranging terms we get the following second-order approximation of $\gamma(t_0 + h)$:

$$\gamma(t_0 + h) \approx \gamma(t_0) + h\, v\Big(\gamma\big(t_0 + \tfrac{h}{2}\big)\Big) \tag{2.179}$$

In this approximation $\gamma(t_0 + h/2)$ is unknown. However, since the last term of the above equation contains a multiplication with h, it is sufficient to approximate $\gamma(t_0 + h/2)$ with first-order accuracy, for which Euler's method can be used:

$$\gamma\big(t_0 + \tfrac{h}{2}\big) \approx \gamma(t_0) + \tfrac{h}{2}\, v\big(\gamma(t_0)\big) \tag{2.180}$$

Substituting Equation (2.180) into Equation (2.179), we obtain the recursive scheme

$$\gamma(t^{i+1}) = \gamma(t^i) + h\, v\Big(\gamma(t^i) + \tfrac{h}{2} v\big(\gamma(t^i)\big)\Big)\,, \tag{2.181}$$

which is known as *second-order midpoint* method, a specific example of a second-order Runge-Kutta method [DB08; Pre+07]. In case of the structure tensor, again, the sign of the minor eigenvector must be handled properly. A step with step size h is given by:

$$\begin{aligned} k_1^{i\pm1} &= \begin{cases} \pm\xi(x^0) & \text{if } i = 0 \\ \operatorname{sign}\langle k_2^i, \xi(x^i)\rangle \cdot \xi(x^i) & \text{otherwise} \end{cases} \\ k_2^{i\pm1} &= \operatorname{sign}\Big\langle k_2^i, \xi\big(x^i + \tfrac{h}{2} k_1^{i\pm1}\big)\Big\rangle \cdot \xi\big(x^i + \tfrac{h}{2} k_1^{i\pm1}\big) \\ x^{i\pm1} &= x^i + h\, k_2^{i\pm1}\,. \end{aligned} \tag{2.182}$$

The second-order Runge-Kutta method requires values of the minor eigenvector for arbitrary positions. One option is to calculate these in one pass, and then use nearest-neighbor sampling while tracing the stream lines. Bilinear interpolation of the eigenvectors is complicated, since opposite vectors may cancel each other out. A better approach is to sample the structure tensor directly using bilinear interpolation as discussed in Section 2.7.2. This is more expensive, since the minor eigenvector has to be computed for every sample, but also provides superior results. An implementation of the first- and second-order methods is shown in Listing 2.6.

2.9.3 Line Integral Convolution

Let $\gamma\colon (a,b) \to \mathbb{R}^2$ be a smooth curve, and let $f\colon \mathbb{R}^2 \to \mathbb{R}$ be a scalar field. Then the *line integral* of f along γ is defined by

$$\int_\gamma f \,\mathrm{d}s = \int_a^b f(\gamma(t))\, \|\gamma'(t)\|\, \mathrm{d}t\,. \tag{2.183}$$

The factor $\|\gamma'(t)\|$ adjusts for the velocity of the curve's parameter and assures that the line integral is invariant under orientation-preserving reparameterizations. Based on this definition, the convolution of a scalar field with a one-dimensional function $g\colon \mathbb{R} \to \mathbb{R}$ along a curve can be defined as:

$$\big(g *_\gamma f\big)(u) = \int_a^b g(u-t)\, f\big(\gamma(t)\big)\, \|\gamma'(t)\|\, \mathrm{d}t \tag{2.184}$$

If g is normalized, that is $\int_a^b g(t)\,\mathrm{d}t = 1$, then the convolution above defines a weighted average of the values of f along the curve.

```
template <class ST, class F, int order, bool adaptive>
inline __host__ __device__ void st_int( float2 p0, const ST& st, F& f,
                                        unsigned w, unsigned h, float step_size )
{
    f(0, p0);
    float2 v0 = st2tangent(st(p0));
    float sign = -1;
    float dr = f.radius() / CUDART_PI_F;
    do {
        float2 v = v0 * sign;
        float2 p = p0;
        float u = 0;

        for (;;) {
            float2 t = st2tangent(st(p));
            if (order == 2) {
                if (dot(v, t) < 0) t = -t;
                t = st2tangent(st(p + 0.5f * step_size * t));
            }
            float vt = dot(v, t);
            if (vt < 0) {
                t = -t;
                vt = -vt;
            }

            v = t;
            p += step_size * t;
            u += step_size;

            if ((u >= f.radius()) ||
                (p.x < 0) || (p.x >= w) || (p.y < 0) || (p.y >= h)) break;

            f(copysignf(u, sign), p);
        }
        sign *= -1;
    } while (sign > 0);
}
```

Listing 2.6: *Generic implementation of first- and second-order stream line integration.*

Now, let $v\colon \mathbb{R}^2 \to \mathbb{R}^2$ be a vector field consisting of normalized vectors. Then, for the vector field's stream lines, we have $\|\gamma'(t)\| = \|v(\gamma(t))\| = 1$, which is equivalent to an arc length parameterization. Overlaying the vector field with an image, the convolution along the stream line passing through the pixel can be computed for each pixel. This operation is known as *line integral convolution*, and increases the correlation of the image's pixel values along the stream lines. If the vector field is closely aligned with the image features, such as the minor eigenvector field of the smoothed structure tensor, convolution along stream lines effectively enhances the coherence of image features, while simplifying the image at the same time.

In order to implement line integral convolution, Equation (2.184) must be discretized. Let us suppose that g is given by a one-dimensional Gaussian function

$$G_\sigma(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t^2}{2\sigma^2} \right) \tag{2.185}$$

with standard deviation σ. Since the stream lines become less accurate the longer they are, a comparatively short cut-off for the Gaussian function is used. It is truncated after two times the standard deviation rounded down to the next integer, $L = \lfloor 2\sigma \rfloor$, corresponding to approximately 95% of the weights. The integral is approximated by a sum of rectangle functions using the midpoint rule. Sampling of the image is thereby performed using bilinear filtering. Thus, if $x^{-L}, \dots, x^0, \dots, x^L$ denote the stream line points obtained with step size h, as described in the previous section, then the result of the line integral convolution at x^0 is given by

$$\frac{1}{|G_\sigma|} \sum_{i=-L}^{L} G_\sigma(kh)\, f(x^i)\,, \tag{2.186}$$

where

$$|G_\sigma| = \sum_{i=-L}^{L} G_\sigma(ih) \tag{2.187}$$

denotes the corresponding normalization term.

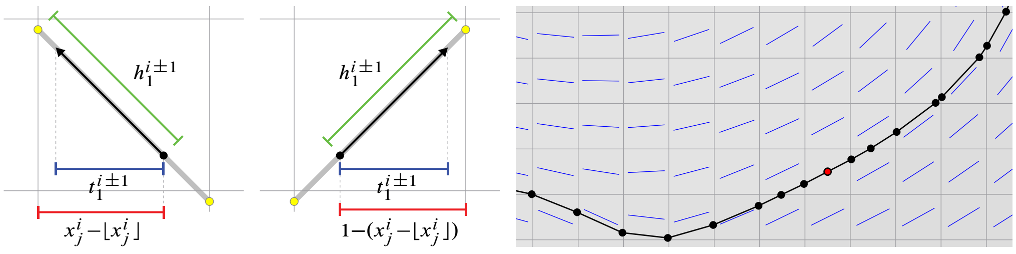
Line integral convolution was first proposed by Cabral and Leedom [CL93] in the context of flow-visualization. When performed over white noise, line integral convolution strongly increases the coherence of pixels along the vector field's stream lines, making the stream lines easily recognizable and thereby leading to a highly effective visualization of the vector field. In addition, performing line integral convolution over an image allows for the creation of a painting-like effect without actually placing brush strokes. Line integral convolution is a computationally expensive process, since a stream line must be traced for every pixel. An alternative method based on texture warping and blending, which can be efficiently implemented using fixed-function graphics hardware, was proposed by Wijk [Wij02]. This approach was adopted as rendering technique by Zhang et al. [ZHT07] to create painterly renderings guided by interactively designed tensor fields and further refined by Kagaya et al. [Kag+11], who proposed a new blending method that prevents mixing of different strokes. A fast implementation, which avoids tracing of every stream line, was presented by Stalling and Hege [SH95]. Their approach enables an efficient sequential implementation of line integral convolution, but whether it can be adapted for parallel processing is unclear. By contrast, tracing every stream line naturally generalizes to a parallel implementation and is therefore adopted in this work.

2.9.4 Cabral-Leedom Integration

The Euler and Runge-Kutta methods are classical numerical solvers for ordinary differential equations. A different approach has been presented by Cabral and Leedom [CL93]. Similar to Euler's method the integral curve is linearly approximated, but instead of a fixed step size, the linear curve segment is elongated until intersecting the current pixel's boundary. Although originally developed for the visualization of vector fields, their approach can also be applied to the minor eigenvector field of the structure tensor.

As in case of the Euler and Runge-Kutta methods, the sign of the minor eigenvector has to be treated with special care:

$$t^{i\pm1} = \begin{cases} \pm\xi(x^0) & \text{if } i = 0 \\ \operatorname{sign}\left\langle t^i, \xi(x^i)\right\rangle \cdot \xi(x^i) & \text{otherwise} \end{cases} \tag{2.188}$$


$h_1^{i\pm 1}$
$t_1^{i\pm 1}$
$x_j^i - \lfloor x_j^i \rfloor$
$h_1^{i\pm 1}$
$t_1^{i\pm 1}$
$1-(x_j^i - \lfloor x_j^i \rfloor)$

(a) *Left* (b) *Right* (c) *Resulting path*

Figure 2.32: *The Cabral-Leedom integration approach. (a)–(b) Illustrate the computation of the intersection with the pixel's horizontal limits. (c) Example of a stream line generated by the approach.*

The computation of the intersections with the pixel's horizontal and vertical limits is illustrated in Figure 2.32. The parameter values of the intersections are thus given by

$$h_j^{i\pm 1} = \begin{cases} \frac{\lfloor x_j^i \rfloor - x_j^i}{t_j^{1\pm 1}} & \text{if } t_j^{i\pm 1} < 0 \text{ and } \lfloor x_j^i \rfloor \neq x_j^i \\ \frac{1+\lfloor x_j^i \rfloor - x_j^i}{t_j^{i\pm 1}} & \text{if } t_j^{i\pm 1} > 0 \\ \infty & \text{otherwise} \end{cases} \qquad (j = 1, 2)\,, \tag{2.189}$$

with the smaller of both corresponding to the intersection with the pixel's boundary:

$$h^{i\pm 1} = \min\{\, h_1^{i\pm 1}, h_2^{i\pm 1} \,\} \tag{2.190}$$

Since the intersection computation may be affected by rounding errors, a small correction term ε is added to the parameter value to ensure that new points lie strictly inside a pixel:

$$x^{i\pm 1} = x^i + (h^{i\pm 1} + \varepsilon)\, t^{i\pm 1} \tag{2.191}$$

An example for a stream line obtained using this method is shown in Figure 2.32(c). As can be seen, the curves points are not equidistantly distributed. Moreover, the curve's points are placed very close to the pixels boundary. Hence, for computing the line integral convolution with a function g, Equation (2.186) does not apply. Since the stream line is a polygonal line, it may be expressed as sum $\gamma = \sum_i \gamma_i$ of line segments $\gamma_i \colon [u_i, u_{i+1}] \to \mathbb{R}^2$, and the convolution of a function g with an image f along γ given by $g *_\gamma f = \sum_i g *_{\gamma_i} f$. When excluding the respective end points, the line segments of the curve lie within a pixel. The image is thus constant on the line segments, if nearest neighbor sampling is used.

$$(g *_{\gamma_i} f)(0) = \int_{u_i}^{u_{i+1}} g(-t) f(\gamma_i(t))\, \mathrm{d}t = f(\gamma(u_i)) \int_{u_i}^{u_{i+1}} g(-t)\, \mathrm{d}t \tag{2.192}$$

This integral may be computed analytically. For instance, if g is a one-dimensional Gaussian function G_σ with standard deviation σ we get

$$\begin{aligned} (G_\sigma *_{\gamma_i} f)(0) &= f(\gamma(u_i)) \int_{u_i}^{u_{i+1}} G_\sigma(-t)\, \mathrm{d}t \\ &= \frac{f(\gamma(u_i))}{2} \left[\operatorname{erf}\!\left(\frac{u_{i+1}}{\sigma\sqrt{2}}\right) - \operatorname{erf}\!\left(\frac{u_i}{\sigma\sqrt{2}}\right) \right], \end{aligned} \tag{2.193}$$

where

$$\operatorname{erf}(x) = \frac{2}{\pi} \int_0^x \exp\bigl(-t^2\bigr)\,\mathrm{d}t \tag{2.194}$$

denotes the error function. In case of a general function g, the integral can be approximated using, for instance, the midpoint rule:

$$\int_{u_i}^{u_{i+1}} g(-t)\,\mathrm{d}t \approx (u_{i+1} - u_i) \cdot g\left(\frac{u_{i+1} + u_i}{2}\right) \tag{2.195}$$

Chapter 3

Bilateral Filter

The bilateral filter is arguably the simplest and most straightforward edge-preserving filtering method available today. Despite its simplicity, it is highly effective (Figure 3.1) and has originated a large number of extensions and applications in computer graphics and computer vision that utilize it. Particular examples include texture editing and relighting [Oh+01], tone management [BPD06; Ela05; ED04; DD02], denoising [Liu+06; BM05; Ale06], upsampling [Kop+07], and optical-flow estimation [ST08; Xia+06]. An excellent review of the bilateral filter can be found in the survey by Paris et al. [Par+09], which also discusses the various applications in detail. The ability of the bilateral filter to smooth image regions while preserving discontinuities makes it also an obvious candidate for carrying out image abstraction. Classical examples that employ the bilateral filter as part of their

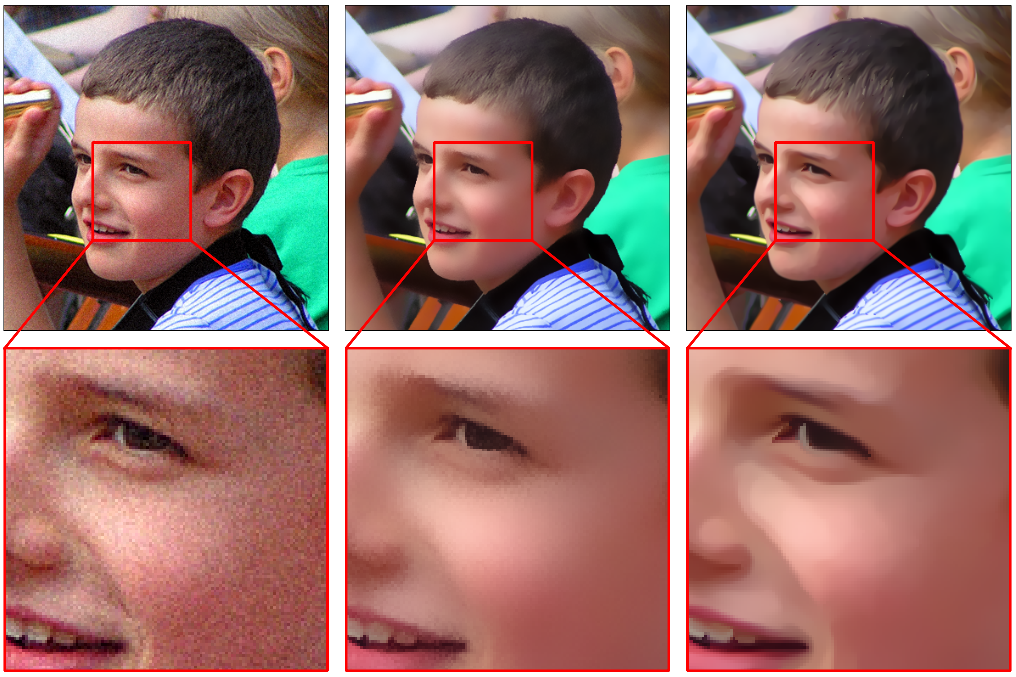
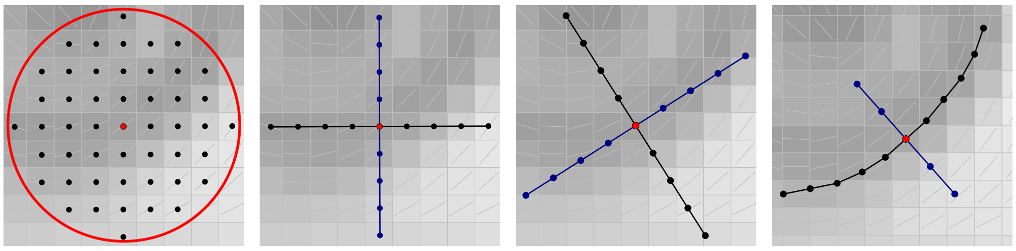

(a) *Original image* (b) *Bilateral filter* (c) *Flow-based bilateral filter*

Figure 3.1: *The bilateral filter smoothes an image while preserving its discontinuities. Taking the local structure into account allows for enhancement and exaggeration of image features, a useful property for carrying out image stylization.*



(a) *2D kernel* (b) *xy-separable* (c) *Orientation-aligned* (d) *Flow-based*

Figure 3.2: *Different variants of the bilateral filter discussed in this chapter. (a) A classical isotropic single pass bilateral filter performs a weighted average of a two-dimensional neighborhood. (b) Separable implementation with the first pass along the x-axis and second pass along the y-axis. (c) Orientation-aligned separable implementation with the first pass perpendicular and the second pass parallel to a vector field derived form the local structure, such as edge tangent flow or smoothed structure tensor. (d) Separable implementation of the flow-based bilateral filter with the first pass perpendicular to the integral curve and the second pass along the integral curve.*

methods are the works by Fischer et al. [FBSe05] and Winnemöller et al. [WOG06].

In this chapter two extensions of the traditional bilateral filter will be presented. Both take the local structure into account, which is derived from the smoothed structure tensor, making them specifically feasible for image abstraction purposes. The first is the orientation-aligned bilateral filter (Figure 3.2(c)), which provides a fast separable approximation of the traditional bilateral filter, including the ability to enhance anisotropic image features. The second is the flow-based bilateral filter, which enables an even more aggressive enhancement of anisotropic structures by closely following the image features (Figure 3.2(d)). The remainder of this chapter is organized as follows. First, anisotropic diffusion, which is closely related to the bilateral filter, is briefly reviewed. Then, the traditional bilateral is introduced and its properties reviewed, followed by a detailed discussion of the orientation-aligned and flow-based bilateral filters and their implementations. The chapter closes with a short discussion of limitations of the bilateral filter.

3.1 Anisotropic Diffusion

Let I be a grayscale image, then the solution of the heat equation

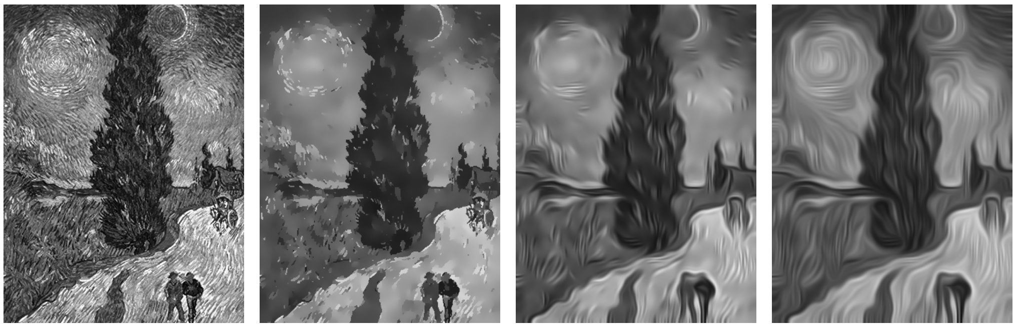
$$\frac{\partial u}{\partial t} = \Delta u = \operatorname{div}(\nabla u) \tag{3.1}$$

at a particular time t with initial condition $u(x,0) = I(x)$ is given by the convolution with a two-dimensional Gaussian having standard deviation $\sqrt{2t}$ [Wei98]. To overcome the limitations of linear shift-invariant smoothing, Perona and Malik [PM90] added the regularization terms

$$g(s^2) = \exp\Big(-\frac{s^2}{K^2}\Big) \qquad \text{and} \qquad g(s^2) = \frac{1}{1+\frac{s^2}{\lambda^2}} \qquad (K, \lambda > 0) \tag{3.2}$$

to the heat equation to stop the diffusion at the edges:

$$\frac{\partial u}{\partial t} = \operatorname{div}\Big(g\big(|\nabla u|^2\big)\nabla u\Big) \tag{3.3}$$



(a) *Original image* (b) *AD* (c) *EED* (d) *CED*

Figure 3.3: *Examples of different anisotropic diffusion techniques. (a) van Gogh—Road with Cypress and Star. (b) Anisotropic diffusion [PM90]. (c) Edge-enhancing diffusion [Wei98]. (c) Coherence-enhancing diffusion [Wei98].*

This is known as *anisotropic diffusion* and adding such penalization terms a standard technique often found in PDE-based approaches (Figure 3.3). For instance, the edge-enhancing and coherence-enhancing diffusion techniques developed by Weickert [Wei99] guide the diffusion using a tensor derived from the smoothed structure tensor. The Perona-Malik and other popular diffusion equations can be written as decomposition in local gauge coordinates [KDA97]

$$\frac{\partial u}{\partial t} = c_\xi \, u_{\xi\xi} + c_\eta \, u_{\eta\eta} \ ,$$

where $u_{\xi\xi}$ and $u_{\eta\eta}$ denote the second derivatives in the directions of the normalized gradient $\eta = \nabla u / \|\nabla u\|$ and tangent $\xi = \eta^\perp$. This can be interpreted as two simultaneous 1D heat flows, where the amount of smoothing is controlled by the functions c_ξ and c_η that typically depend on the gradient ∇u. This decomposition allows to interpret different approaches. For example, mean curvature flow (Section 6.1) corresponds to $c_\xi = 1$ and $c_\eta = 0$. Instead of obtaining the local gauge coordinates from the gradient, the eigenvectors of the smoothed structure tensor may be used.

Weickert [Wei98] replaced the scalar diffusivity by a diffusion tensor that depends on the structure tensor. This approach can be directly extended to color images using Di Zenzo's [Di 86] multi-channel image gradient (Section 2.2.4). In particular, Weickert [Wei99] showed that by adapting the diffusion tensor, coherent image enhancement can be performed (Figure 3.3). The results created by this technique look very similar to those generated by line integral convolution (Section 2.9.3). Tschumperlé [Tsc06] later clarified the connection between line integral convolution and PDEs and proposed tensor-driven PDEs that take the integral curves into account. An approximate solution to these curvature preserving PDEs can be found with an algorithm based on line integral convolution. More details about anisotropic diffusion and other PDE-based image processing techniques may be found in the books by Weickert [Wei98] and Aubert and Kornprobst [AK06].

The bilateral filter uses a similar principle as anisotropic diffusion to avoid smoothing across images. However, instead of a diffusion process modeled by a PDE, it is explicitly designed as a local neighborhood filter. Crucial to the success is thereby the the use of local gauge coordinates to perform the image filtering. A detailed discussion of the relationship between anisotropic diffusion and the bilateral filter is beyond the scope of this work, but

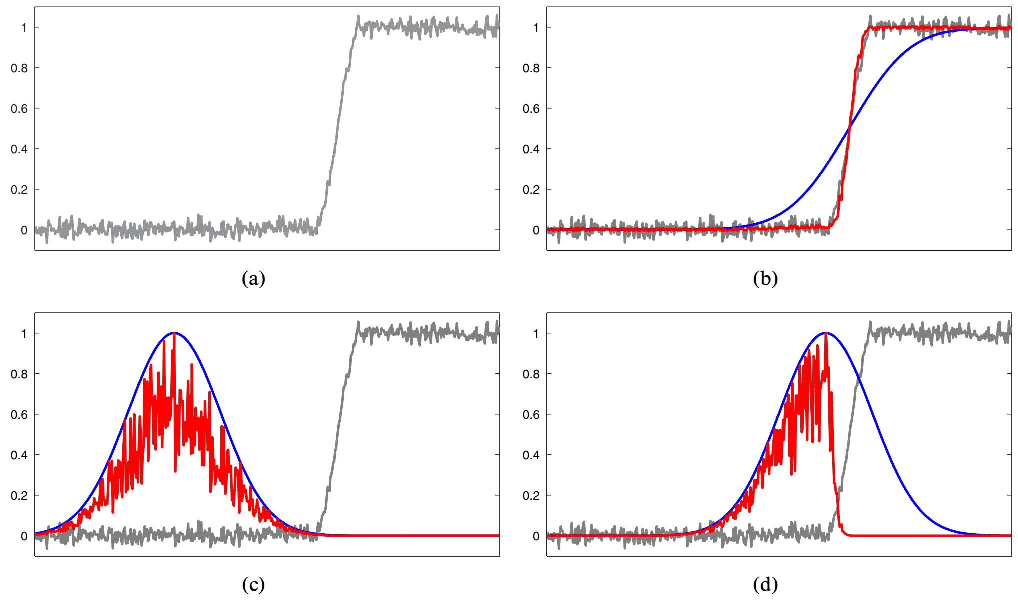


Figure 3.4: *Illustration of the working principle of the bilateral filter. (a) Noisy input signal (gray). (b) Result of the convolution with Gaussian (blue) and bilateral (red) filter. Notice how the Gaussian kernel blurs the signal, while the bilateral filter keeps the sharp transition. In (c) and (d) the local filter kernel profiles of the Gaussian filter (blue) and bilateral filter (red) are shown at two different positions. The local filter kernel of the Gaussian filter does not depend on the signal (shift invariance) and is the same in both case. The bilateral filter adapts its local filter kernel to the signal and thereby limits smoothing across the transition.*

can be found in [BC04], where also the connection to mean-shift and adaptive smoothing is covered. The bilateral filter is also related to W-estimators, local M-smoothers, and regularization approaches. A detailed discussion of these topics may be found in [MWB06].

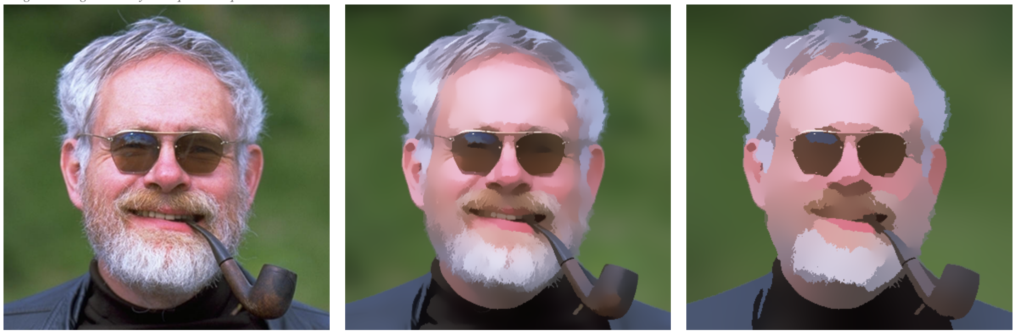
3.2 Isotropic Bilateral Filter

The *bilateral filter* was first introduced by Aurich and Weule [AW95] as a nonlinear extension of the Gaussian filter. Independently, the bilateral filter also appeared as part of the SUSAN corner detector by Smith and Brady [SB97]. However, it got popularized by Tomasi and Manduchi [TM98], who provided the general definition that is used today and also coined the name. For a given image f and position x the bilateral filter is defined by

$$\frac{\sum_{y\in\mathcal{N}(x)} f(y)\, \overbrace{k_d\big(\|y-x\|\big)}^{\text{domain weight}}\, \overbrace{k_r\big(\|f(y)-f(x)\|\big)}^{\text{range weight}}}{\sum_{y\in\mathcal{N}(y)} k_d\big(\|y-x\|\big)\, k_r\big(\|f(y)-f(x)\|\big)}\,, \tag{3.4}$$

where $\mathcal{N}(x)$ denotes a sufficiently large neighborhood of x, and k_d and k_r are two weighting functions. The *domain weight*, given by k_d, is based on the spatial distance from the filter origin x, whereas the *range weight*, given by k_r, is based on the distance between the image's values at the corresponding positions, acting as a stopping function similar to the stopping

Original image courtesy Philip Greenspun



(a) *Original image* (b) *4 iterations* (c) *25 iterations*

Figure 3.5: *An iterative application of the bilateral filter smoothes the image while preserving edges, achieving a strong simplification effect.*

function in anisotropic diffusion. Typically, following the proposal by Aurich and Weule [AW95], a one-dimensional Gaussian

$$G_\sigma(r) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma} r^2\right) \tag{3.5}$$

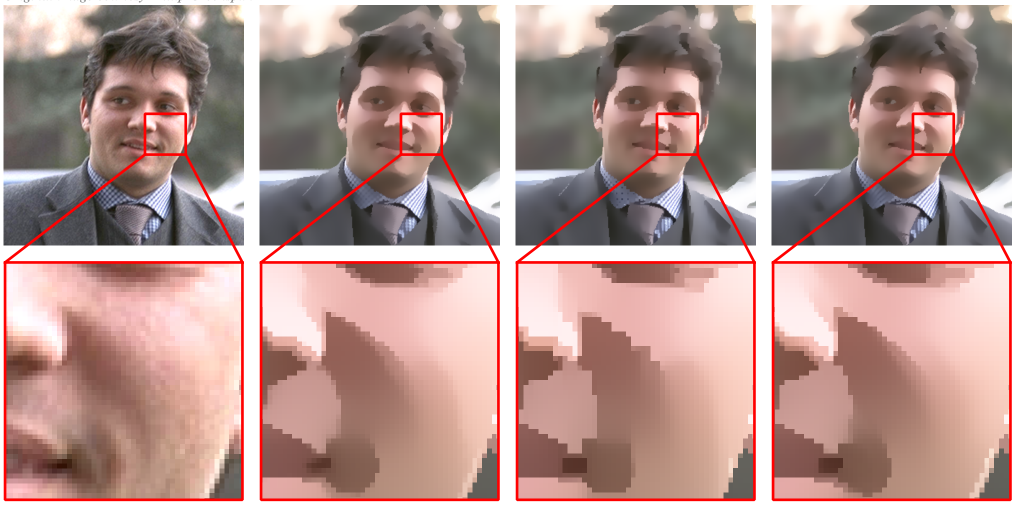
is chosen for both weighting functions, but other choices are possible [DD02; Par+09]. If k_d is chosen as Gaussian and $k_r \equiv 1$, then the bilateral filter simplifies to the Gaussian filter. If, vice versa, $k_d \equiv 1$, and k_r is chosen as Gaussian, then one gets a filter originally proposed by Yaroslavsky [Yar85]. The bilateral filter smoothes regions of similar color, while regions with detail are preserved. For instance, if the local neighborhood of a pixel contains an edge, then pixels on the opposite side of the edge receive a low and all others a high weight by the range function, resulting in the preservation of the edge. Figure 3.4 illustrates the principle using a one-dimensional signal.

By using a suitable metric for the computation of the range weight, the bilateral filter extends naturally to color images. For instance, a possible choice is to use the Euclidean metric in RGB color space. Another choice, proposed by Tomasi and Manduchi [TM98], is using the Euclidean metric in CIELAB color space [WS82], which is known to correlate with human perception for short distances. This approach is adopted in the original WOG-pipeline [WOG06] and subsequent work.

If domain and range weight are chosen to be Gaussians, increasing the standard deviation of the domain weight generally does not lead to a stronger abstraction effect. Moreover, increasing the range weight results, in most cases, in blurred edges. Instead, to achieve a cartoon-like effect, it is better to apply multiple iterations of the bilateral filter (Figure 3.5). This was already noted by Tomasi and Manduchi [TM98], and can be explained theoretically by the connection of bilateral filtering to anisotropic diffusion.

A limitation of the bilateral filter for practical applications, especially in the case of real-time processing, is that the direct evaluation of Equation (3.4) is computationally expensive. For a local neighborhood with radius r the complexity is $O(r^2)$ per pixel, which means that linear growth of the neighborhood leads to quadratic growth in computational costs. Because of the importance of the bilateral filter for practical applications, several approaches have been developed to speed up the bilateral filter. Examples are fast implementations

Original image courtesy Philip Greenspun



(a) *Original image* (b) *2D-kernel* (c) *xy-separable* (d) *Orientation-aligned*

Figure 3.6: *Comparison of the full-kernel, xy-separable, and orientation-aligned (with manual sampling) variants of the bilateral filter (5 iterations, $\sigma_d = 3$, $\sigma_r = 5\%$).*

based on sliding windows [Wei06; Bro11] and approaches that avoid redundant operations by using histograms [Por08; YTA09; CSU11; Gun11], channel smoothing [FFS06], or subsampling [Ban+12]. Another class of approaches builds upon the idea to represent the image data as a two-dimensional manifold in a higher dimensional space [Bar02], making the bilateral filter become a linear filtering operation. While the bilateral grid [CPD07] and its predecessors [PD06; DD02] use a voxel to represent the image data, alternative approaches have been proposed that efficiently store the image data in a sparse data structure, such as the Gaussian KD-tree [Ada+09] or the permutohedral lattice [ABD10].

3.3 xy-Separable Bilateral Filter

An interesting property of the Gaussian filter is its separability. This means that a convolution with a two-dimensional Gaussian filter may be decomposed into two convolutions with a one-dimensional Gaussian filter. From a computational perspective this is highly desirable, since it reduces the computational complexity from a quadratic dependence (Figure 3.2(a)) on the filter radius to a linear dependence (Figure 3.2(b)). Unfortunately, in contrast to the Gaussian filter, the bilateral filter is not separable, since it depends on the filter origin's image value. Nevertheless, in the context of video compression, [PV05] were able to show that for small filter sizes a separable implementation of the bilateral filter can provide reasonable results. Their approach was adopted in the original cartoon pipeline by [WOG06] and was a crucial factor for achieving real-time performance on consumer GPUs at that time. However, the xy-separable bilateral filter [PV05] suffers from horizontal and vertical artifacts. These artifacts appear particularly when the bilateral filter is applied iteratively (Figure 3.6), which is the typical way of applying it for the purpose of image stylization and abstraction.

To achieve a closer match to the bilateral filter, Pham [Pha06] proposed to align the separable bilateral filter with the local image structure. To this end, he derived local

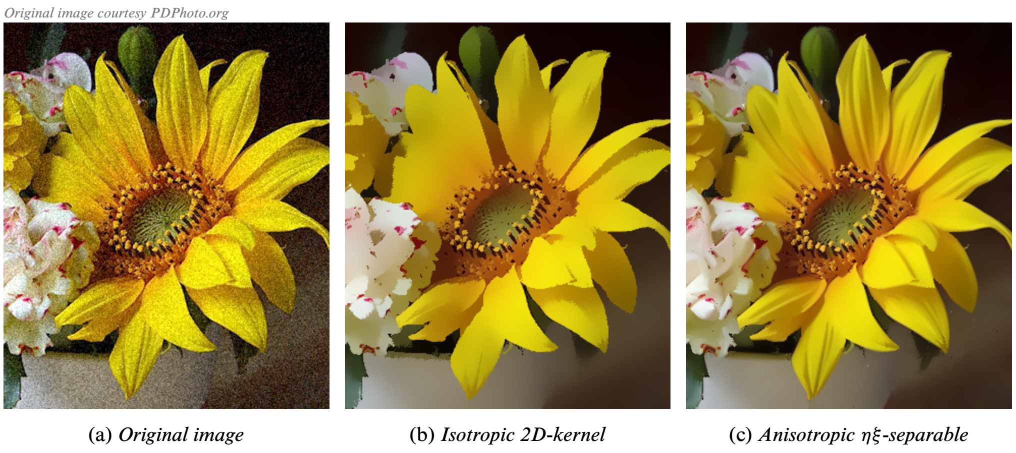

Original image courtesy PDPhoto.org
(a) *Original image*
(b) *Isotropic 2D-kernel*
(c) *Anisotropic $\eta\xi$-separable*

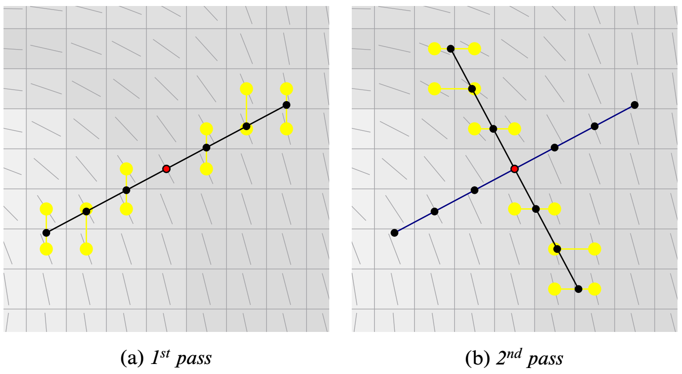
Figure 3.7: *Adapting the separable implementation of the bilateral filter to the local structure enables the enhancement of image structures. (a) Original low quality image. (b) Bilateral filter: $\sigma_d = 6$, $\sigma_r = 8\%$. (c) Orientation-aligned bilateral filter with linear sampling: $\rho = 4$, $\sigma_{d,g} = 3$, $\sigma_{r,g} = 2\%$, $\sigma_{d,t} = 8$, $\sigma_{r,t} = 10\%$. In both cases 5 iterations were performed.*

orientation and curvature information using a method proposed by Ginkel et al. [Gin+99]. Separable filtering was carried out by first filtering perpendicular to the local orientation and then along an arc curved profile matching the curvature. The next sections discuss two approaches that are similar in spirit.

3.4 Orientation-aligned Bilateral Filter

As we saw in the previous section, implementing the bilateral filter in a separable way results in horizontal and vertical artifacts. A simple way to reduce the artifacts is to perform the filtering along the local gauge coordinates of the image, instead of along the coordinate axes. In case of an edge, filtering along the gradient direction results in strengthening of the edge, while filtering along the tangent direction results in a smoothing operation along the edge, avoiding artifacts and producing more coherent region boundaries, as demonstrated in Figure 3.6(d). For flat and homogeneous image regions, smoothing is performed in both directions. Another advantage of this approach is that the filter shape can be adapted to the local image structure, since the parameters for each pass can be controlled individually on a per-pixel basis. For instance, by decreasing the size in the direction of the gradient and increasing it in the direction of the tangent, the overall filter shape becomes elliptic, leading to an enhancement of anisotropic structures (Figure 3.7). For instance, if the standard deviations are chosen in a similar way to Weickert's coherence-enhancing diffusion technique [Wei99], one gets a macroscopic variant of it on basis of the bilateral filter [Bar02].

For an implementation of the orientation-aligned bilateral filter, as a first step the local orientation has to be estimated, which is derived from the smoothed structure tensor, whose major and minor eigenvectors provide the local gauge coordinates to filter along. In general, it is sufficient to estimate the local orientation once for both passes and all iterations performed. Not recomputing the orientation also helps to ensure the filtered output does not deviate too much from the original input. The next step is to carry out the actual



(a) *1st pass* (b) *2nd pass*

Figure 3.8: *Orientation-aligned bilateral filter. Yellow points indicate sampling positions that are linearly interpolated. (a) First pass filters along the direction of the gradient. (b) Second pass filters along the direction of the tangent.*

filtering. As pointed out earlier, the first pass should filter along the gradient (i.e., major eigenvector) direction, since then the second pass can smooth rough regions boundaries introduced by the first pass by filtering along the tangent (i.e., minor eigenvector) direction. Figure 3.8 illustrates the two passes that have to be performed. As can be seen, except for obvious cases, pixel values have to be evaluated at arbitrary positions. A possible approach would be to perform nearest-neighbor or bilinear sampling. However, the former may introduce artifacts, while the latter may result in blurred results. A much better approach is to use a unit step size in either horizontal or vertical direction. The step size along the axis can be derived from the intercept theorem as illustrated in Figure 3.9 and is given by

$$\Delta s = \begin{cases} \dfrac{1}{t_1} t & \text{if } |t_1| > |t_2| \\ \dfrac{1}{t_2} t & \text{otherwise} \end{cases} \tag{3.6}$$

where t denotes the filtering direction. Notice that the sign of t does not matter, since the filtering operates in both directions and is symmetric. Hence, we obtain the following sampling positions

$$x^k = x^0 + k\,\Delta s\,, \qquad k \in \mathbb{Z}, \tag{3.7}$$

where x^0 denotes the filter origin. Sampling can now be performed in three different ways, with nearest-neighbor and bilinear sampling being the obvious ones. For these two cases,

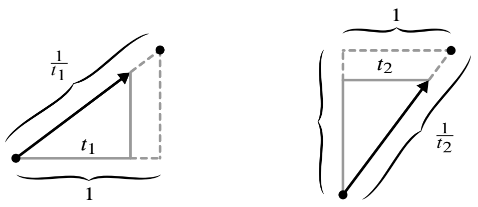


Figure 3.9: *Calculation of the step size for the orientation-aligned bilateral filter using the intercept theorem.*

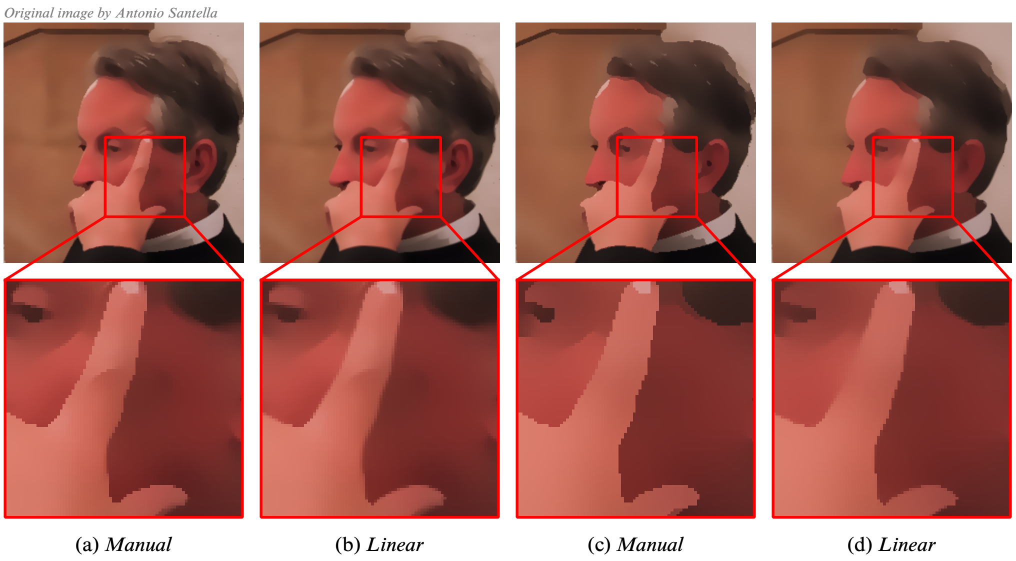


Figure 3.10: *Comparison of manual versus linear sampling for different numbers of iterations. Left: 4 iterations. Right: 12 iterations.*

Equation (3.4) becomes

$$\frac{\sum_{i=-L}^{L} f(x^i)\, G_{\sigma_d}\left(\|x^i - x^0\|\right) G_{\sigma_r}\left(\|f(x^i) - f(x^0)\|\right)}{\sum_{i=-L}^{L} G_{\sigma_d}\left(\|x^i - x^0\|\right) G_{\sigma_r}\left(\|f(x^i) - f(x^0)\|\right)}\,, \tag{3.8}$$

where L denotes the cut-off radius, which is typically chosen based on σ_d, such as, for instance, $L = \lceil 3\sigma_d / \|t\| \rceil$. Because of the unit step size in either horizontal or vertical direction, bilinear sampling in fact performs a linear interpolation between two pixels, resulting in less blurring introduced by the sampling. The other option is to manually perform linear sampling and incorporate weighting with the range function into the linear interpolation. To this end, the two neighboring pixels are sampled, and for both the range weights are computed. The pixel values are multiplied by the range weights and the results linearly weighted depending on the sampling position. In a nutshell, let $\lceil x^i \rceil$ and $\lfloor x^i \rfloor$ denote the two sampling positions, then the summands in Equation (3.8) have to be updated to:

$$\begin{aligned} G_{\sigma_d}\left(\|x^i - x^0\|\right)\Big[&\left\|\lfloor x^i \rfloor - x^0\right\| f\left(\lceil x^i \rceil\right) G_{\sigma_r}\left(\left\| f\left(\lceil x^i \rceil\right) - f\left(x^0\right)\right\|\right) + \\ &\left\|\lceil x^i \rceil - x^0\right\| f\left(\lfloor x^i \rfloor\right) G_{\sigma_r}\left(\left\| f\left(\lfloor x^i \rfloor\right) - f\left(x^0\right)\right\|\right)\Big] \end{aligned} \tag{3.9}$$

In Figure 3.10, a comparison between manual and bilinear sampling is shown. As can be seen, manual sampling creates sharp region boundaries similar to the 2D-kernel bilateral filter, yielding a stronger separation of image regions. Since linear sampling operates on average values of the image, it creates smoother transitions between image regions, achieving an effect similar to anti-aliasing (Figure 3.10(b)). While this may be inappropriate for other applications, for image stylization and abstraction it is a useful feature. Another advantage of linear sampling is that it may be realized by utilizing the bilinear texture sampling support available in GPUs. Linear sampling typically works well for a small number of iterations.

```
template <bool tangential, bool src_linear>
__global__ void oabf_filter( gpu_plm2<float3> dst, const gpu_plm2<float2> tf,
                             float sigma_d, float sigma_r, float precision)
{
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float2 t = tf(ix, iy);
    if (!tangential) t = make_float2(t.y, -t.x);

    float twoSigmaD2 = 2 * sigma_d * sigma_d;
    float twoSigmaR2 = 2 * sigma_r * sigma_r;
    int halfWidth = int(ceilf(precision * sigma_d));
    float2 tabs = fabs(t);
    float ds = (tabs.x > tabs.y)? 1.0f / tabs.x : 1.0f / tabs.y;

    float2 uv = make_float2(0.5f + ix, 0.5f + iy);
    float3 c0 = make_float3(tex2D(texSRC, uv.x, uv.y));
    float3 sum = c0;
    float norm = 1;
    float sign = -1;
    do {
        for (float d = ds; d <= halfWidth; d += ds) {
            float2 p = uv + d * t * sign;

            float kd = __expf( -dot(d,d) / twoSigmaD2 );
            if (src_linear) {
                float3 c = make_float3(tex2D(texSRC, p.x, p.y));
                float kr = __expf( -squared(c - c0) / twoSigmaR2 );
                sum += kd * kr * c;
                norm += kd * kr;
            } else {
                p -= make_float2(0.5f, 0.5f);

                float3 c1, c2;
                float f;
                if (tabs.x < tabs.y) {
                    float2 q = make_float2(floorf(p.x), p.y);
                    c1 = make_float3(tex2D(texSRC, q.x, q.y));
                    c2 = make_float3(tex2D(texSRC, q.x + 1, q.y));
                    f = p.x - q.x;
                } else {
                    float2 q = make_float2(p.x, floorf(p.y));
                    c1 = make_float3(tex2D(texSRC, q.x, q.y));
                    c2 = make_float3(tex2D(texSRC, q.x, q.y + 1));
                    f = p.y - q.y;
                }

                float kr1 = (1 -  f) * __expf( -squared(c1 - c0) / twoSigmaR2 );
                float kr2 = f * __expf( -squared(c2 - c0) / twoSigmaR2 );
                sum += kd * (kr1 * c1 + kr2 * c2);
                norm += kd * (kr1 + kr2);
            }
        }
        sign *= -1;
    } while (sign > 0);
    dst.write(ix, iy, sum / norm);
}
```

Listing 3.1: *Implementation of the orientation-aligned bilateral filter.*

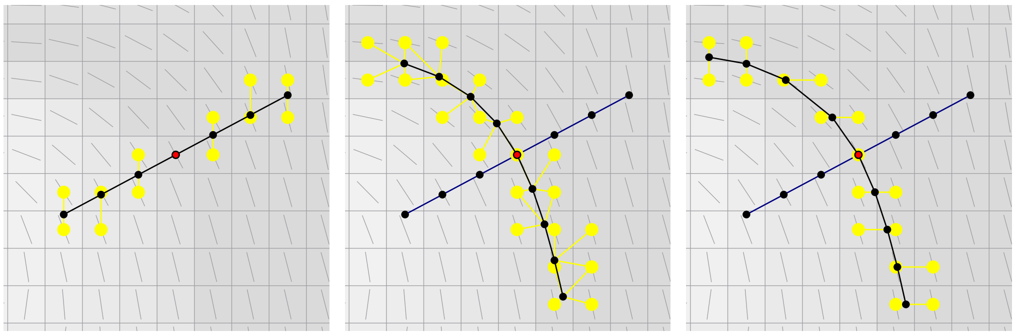

(a) *1st pass* (b) *2nd pass (bilinear sampling)* (c) *2nd pass (linear sampling)*

Figure 3.11: *Flow-based bilateral filter. (a) First pass filters in direction of the gradient. (b) Second pass using unit step size and bilinear sampling. (c) Second pass using unit step size in either horizontal or vertical direction and linear sampling. Yellow points indicate sampling positions.*

For a large number of iterations, region boundaries may be smoothed out, as demonstrated in Figure 3.10(d). An implementation of both sampling approaches is shown in Listing 3.1. Notice that in the case of linear sampling, the texture unit must be configured to perform bilinear texture sampling, while for manual sampling the texture unit is expected to be configured to perform nearest-neighbor sampling.

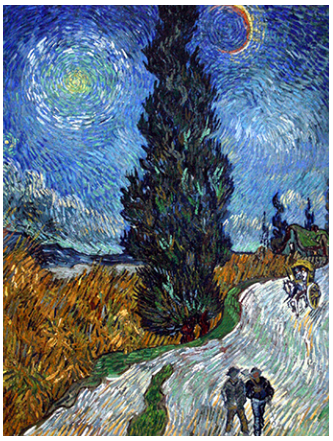
3.5 Flow-based Bilateral Filter

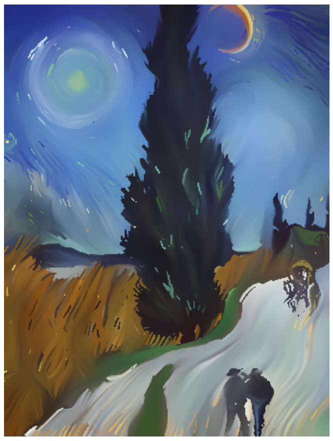
The orientation-aligned bilateral filter provides good results for circular filter shapes, where it features a good approximation of the traditional bilateral filter, and for elliptic filter shapes with moderate eccentricity, where it achieves reasonable enhancement for approximately linear image features. For curved and irregular image features, however, sampling along the tangent direction provides a poor approximation to the image if the filter radii are comparatively large, resulting in image features being smoothed out. Hence, to allow for a strong enhancement of anisotropic image features without blurring them, the local approximation must be improved. This can be achieved by modifying the second pass filter to filter along the integral curves derived from the minor eigenvectors of the smoothed structure tensor. More precisely, if x^0 denotes the filter origin and γ is the integral curve passing through x^0, then computing a one-dimensional bilateral filter along γ corresponds to computing the following line integral:

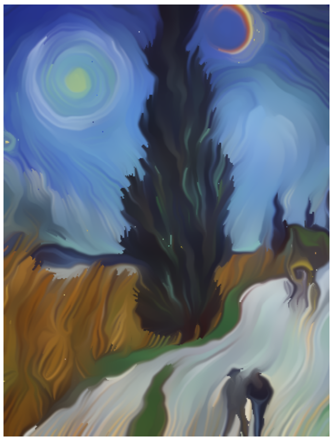
$$\int_\gamma f(x)\, G_{\sigma_d}\left(x - x^0\right) G_{\sigma_r}\left(\left\| f(x) - f(x^0)\right\|\right) \mathrm{d}x \tag{3.10}$$

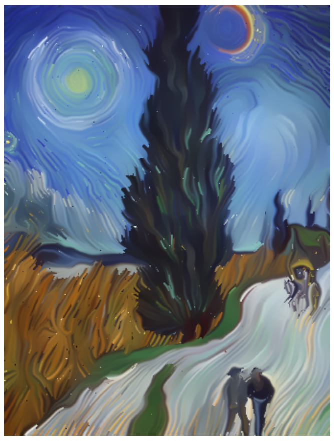
When nearest-neighbor or bilinear sampling is used, this line integral can be easily computed using either the Euler or second-order integration method from Section 2.9. Figure 3.11(b) exemplifies the corresponding integral curve. Notice the equidistant sampling positions that require sampling at arbitrary positions.

As can be seen in Figure 3.12(c), the flow-guided filtering indeed preserves directional image features better, but the introduced blurring is still quite strong, stemming from the use of bilinear sampling. In order to reduce the amount of blurring, a similar approach as for


(a) *Original image*


(b) *Orientation-aligned (linear)*


(c) *Flow-based (bilinear)*


(d) *Flow-based (linear)*

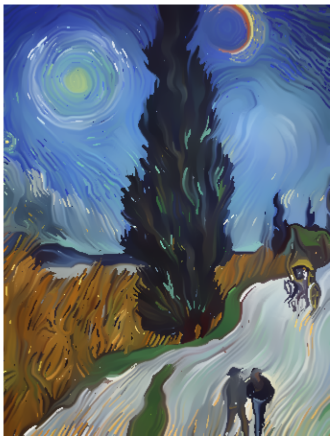

(e) *Flow-based (manual)*

Figure 3.12: *Comparison of the orientation-aligned bilateral filter with different variants of the flow-based bilateral filter: 8 iterations, $\rho = 3$, $\sigma_{d,g} = 3$, $\sigma_{r,g} = 2\%$, $\sigma_{d,t} = 8$, $\sigma_{r,t} = 14\%$.*

the orientation-aligned bilateral filter can be taken. The underlying idea is again to step with unit size in either horizontal or vertical direction. To this end, the integral curve is computed as usual, but sampling points are only emitted where the curve intersects the horizontal or vertical center of a pixel. For computing the intersection, there are two possible ways. The first is to seek from the previous point in forward direction, while the second seeks from the current point in backward direction. The first possibility is numerically unstable. For instance, consider the center of a pixel and a step vector that is almost horizontal. In this case, the distance to the intersection is greater than one. However, the sum of the position and the step vector may, due to rounding errors, lie beyond the intersection point. By computing the intersection in backward direction this issue is avoided.

The distance to the horizontal or vertical intersection can be computed using the intercept theorem, in a similar way as already explained for the Cabral-Leedom integration method (Section 2.9.4). However, since the intersection with horizontal or vertical pixel centers is intended, a correction factor of ½ must be considered. Moreover, since the intersection is computed in backward direction, a sign change occurs. Let $p^i = p^{i-1} + h\,t^i$

be the current position with t^i denoting the current tangent vector and h being the step size. The distances to the horizontal and vertical intersections are then given by

$$\Delta u^i_j = \begin{cases} \frac{p^i_j + \frac{1}{2} - \lfloor p^i_j + \frac{1}{2} \rfloor}{t^i_j} & \text{if } t^i_j > 0 \\ \frac{p^i_j - \frac{1}{2} - \lfloor p^i_j + \frac{1}{2} \rfloor}{t^i_j} & \text{if } t^i_j < 0 \\ \infty & \text{otherwise} \end{cases} \qquad (j = 1, 2)\,, \tag{3.11}$$

and the distance to the closest intersection is given by $\Delta u^i = \min\{\Delta u^i_1, \Delta u^i_2\}$. Sampling must be performed if the distance is smaller than the step size. The corresponding intersection point is given by $p^i - \Delta u^i t^i$, and the parameter value is given by $u^i - \Delta u^i$. In Listing 3.2 the implementation is shown, which is identical to Listing 2.6, with the exception of lines 29–54.

In Figure 3.12 the resulting sampling points are shown. As can be seen, the rectangle method cannot be used for computing Equation (3.10), since the sampling points are not equidistantly distributed along the integral curve. Instead, the trapezoidal rule

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)\,\frac{f(a)+f(b)}{2} \tag{3.12}$$

may be used. An excerpt of the implementation is shown in Listing 3.3, which handles, similar to the implementation of the orientation-aligned bilateral filter, linear as well as manual sampling.

3.6 Discussion

The bilateral filter smoothes low-contrast regions while preserving high-contrast edges, and may, therefore, fail for high-contrast images, where either no abstraction is performed or relevant information is removed because of the parameters chosen. In addition, the bilateral filter also often fails for low-contrast images, where typically too much information is removed. Moreover, iterative application of the bilateral filter may blur edges, resulting in a washed-out look. To some extent, these limitations can be alleviated by overlaying the output of the bilateral filter with outlines, such as the output of the difference of Gaussians filter, which will be discussed in the next chapter. Accordingly, the bilateral filter is rarely applied independently. Although the difference of Gaussians filter can be used independently, preprocessing with the bilateral filter can often reduce artifacts caused by noise in the image, making it a useful tool for preprocessing.

```
template <class ST, class F, int order>
inline __host__ __device__ void st_int_ustep( float2 p0, const ST& st, F& f,
                                     unsigned w, unsigned h, float step_size )
{
    float2 v0 = st2tangent(st(p0));
    float sign = -1;
    float dr = f.radius() / CUDART_PI_F;
    do {
        float2 v = v0 * sign;
        float2 p = p0;
        float u = 0;
        f(0, p0);
        for (;;) {
            float2 t = st2tangent(st(p));
            if (order == 2) {
                if (dot(v, t) < 0) t = -t;
                t = st2tangent(st(p + 0.5f * step_size * t));
            }
            float vt = dot(v, t);
            if (vt < 0) {
                t = -t;
                vt = -vt;
            } else if (vt == 0) break;

            v = t;
            p += step_size * t;
            u += step_size;

            float2 fp = make_float2(fract(p.x + 0.5f), fract(p.y + 0.5f));
            float du;
            if ((fp.x == 0) || (fp.y == 0)) du = 0;
            else {
                du = CUDART_NORM_HUGE_F;

                if (t.x > 0)
                    du = fp.x / t.x;
                else if (t.x < 0)
                    du = (fp.x - 1) / t.x;

                if (t.y > 0)
                    du = fminf(du, fp.y / t.y);
                else if (t.y < 0)
                    du = fminf(du, (fp.y - 1) / t.y);
            }

            if (du < step_size) {
                float2 q = p - t * du;
                float qu = u - du;

                if ((qu >= f.radius()) ||
                    (q.x < 0) || (q.x >= w) || (q.y < 0) || (q.y >= h)) break;

                f(copysignf(qu, sign), q);
            }
        }
        sign *= -1;
    } while (sign > 0);
}
```

Listing 3.2: *Generic implementation of the stream line integration approach that samples at either horizontal or vertical pixel centers.*

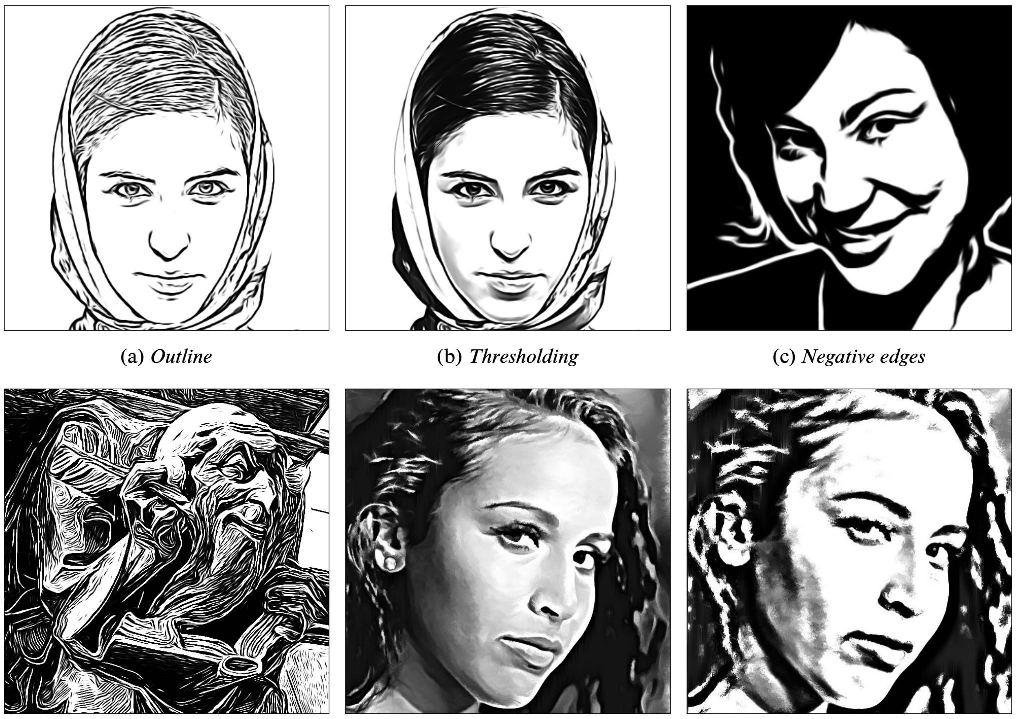
```
__host__ __device__ void operator()( float u, float2 p ) {
    if (u == 0) {
        u_ = 0;
        cp_ = c0_;
        wp_ = 1;
    } else {
        float du = fabsf(u - u_);
        u_ = u;

        c_ += cp_ * du / 2;
        w_ += wp_ * du / 2;

        float kd = expf(-u * u / twoSigmaD2_);
        if (src_linear) {
            T c1 = src_(p.x, p.y);
            float kr = expf(-squared(c1 - c0_) / twoSigmaR2_);
            cp_ = kd * kr * c1;
            wp_ = kd * kr;
        } else {
            p -= make_float2(0.5f, 0.5f);
            float2 ip = floor(p);
            float2 fp = p - ip;

            T c1, c2;
            float f;
            if (fp.x > 1e-4f) {
                float2 q = make_float2(ip.x, p.y);
                c1 = src_(q.x, q.y);
                c2 = src_(q.x + 1, q.y);
                f = fp.x;
            } else if (fp.y > 1e-4f) {
                float2 q = make_float2(p.x, ip.y);
                c1 = src_(q.x, q.y);
                c2 = src_(q.x, q.y + 1);
                f = fp.y;
            } else {
                c1 = c2 = src_(p);
                f = 0;
            }

            float kr1 = (1 -  f) * __expf( -squared(c1 - c0_) / twoSigmaR2_ );
            float kr2 = f * __expf( -squared(c2 - c0_) / twoSigmaR2_ );

            cp_ = kd * (kr1 * c1 + kr2 * c2);
            wp_ = kd * (kr1 + kr2);

        }

        c_ += cp_ * du / 2;
        w_ += wp_ * du / 2;
    }
}
```

Listing 3.3: *Excerpt of the implementation of the functor performing the integration using the trapezoidal rule for linear and manual sampling.*

Chapter 4

Difference of Gaussians

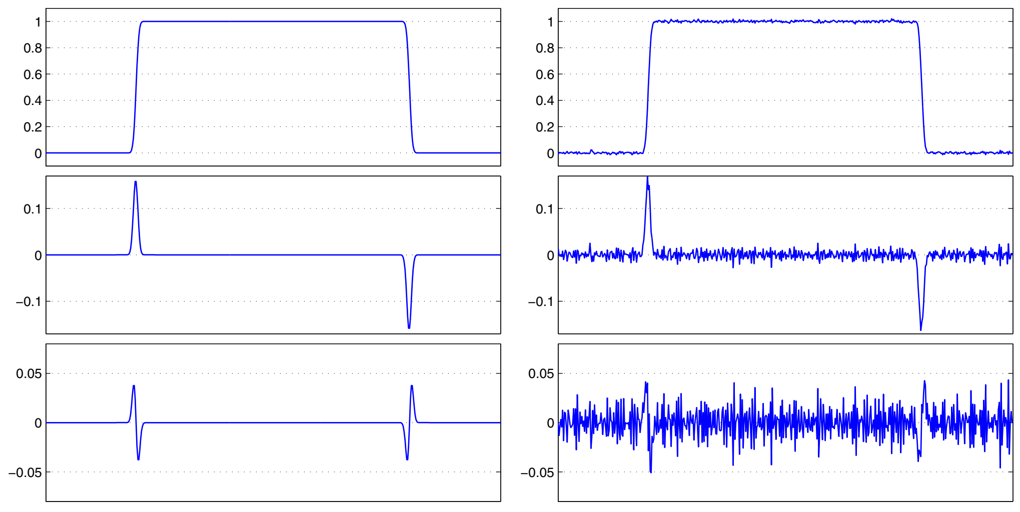
The difference of Gaussians filter has its origin in early work on edge detection by Marr and Hildreth [MH80]. More recently, it gained popularity in IB-AR because of its capacity to create aesthetically pleasing edge lines without post-processing, particularly when synthesizing line drawings and cartoons (Figure 4.1). For instance, Sýkora et al. [SBv05] used the thresholded output of the Laplacian of Gaussian, which is approximated by the difference of Gaussians filter, to create outlines for colorizing hand-drawn black-and-white cartoons, and Gooch et al. [GRG04] used the difference of Gaussians filter in combination with a model of brightness perception to create human facial illustrations. Moreover, it was used in the cartoon pipeline proposed by Winnemöller et al. [WOG06].



(a) *Outline* (b) *Thresholding* (c) *Negative edges*

(d) *Woodcut* (e) *Pastel* (f) *Charcoal*

Figure 4.1: *The extended flow-based difference of Gaussians filter allows for the generation of a large variety of styles. See [J2] for a detailed discussion of effects and styles. (c)–(f) Courtesy of H. Winnemöller.*



(a) *Without noise* (b) *With noise*

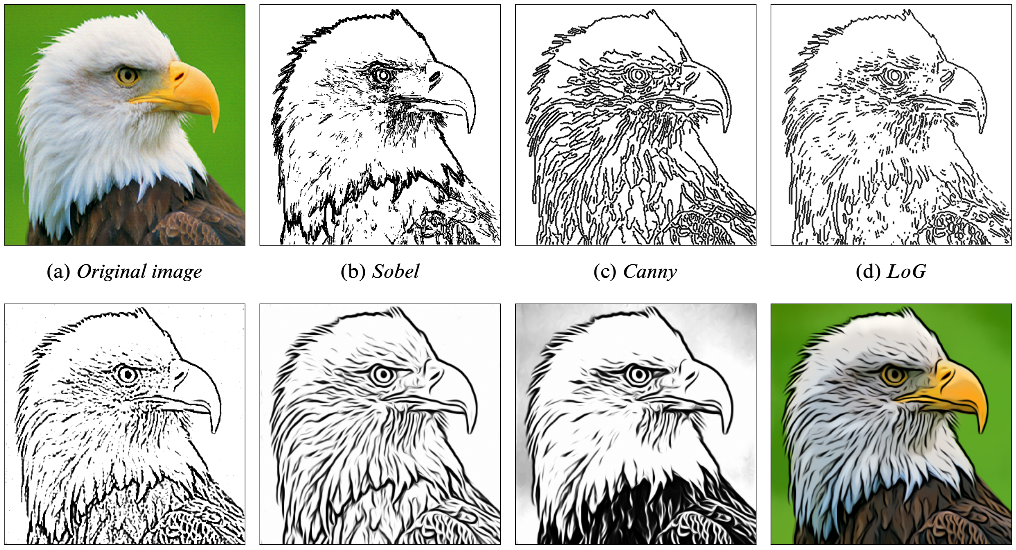
Figure 4.2: *Differentiation is an ill-posed problem and very sensitive to noise. In (a) a smooth step function is shown at the top. In this case, the edges can be easily localized by finding the extrema of the first derivative (middle) or the zero-crossings of the second derivative (bottom). In (b) it is demonstrated what happens when just a little noise is added. The first derivative has now a large number of extrema, and the second derivative almost randomly oscillates, making the identification of the zero-crossings corresponding to the edges impossible.*

In this chapter the flow-based difference of Gaussians filter will be presented, which achieves significant quality improvements in comparison to the traditional isotropic variant. In addition, related concepts are reviewed. The chapter is organized as follows. It begins with remarks on early edge detection and then discusses the Laplacian of Gaussian, which is closely related to difference of Gaussians. After discussing the classical difference of Gaussians filter and how it can be derived as approximation to the Laplacian of Gaussian, the extended difference of Gaussians thresholding scheme (XDoG) is reviewed and a reparameterization of it is presented. Then, the flow-based difference of Gaussians filter and its separable implementation are discussed. The chapter closes with a presentation of a modern form of the cartoon pipeline using the flow-based bilateral and difference of Gaussians filters.

4.1 Gradient-based Edge Detection

The first approaches to edge detection focused on identifying pixels associated with high gradient magnitude. To this end, simple approximations of the image gradient were computed by convolving an image with a small filter kernel, such as the Prewitt or Sobel filter [Pra01]. Then the gradient magnitude was thresholded. A small kernel size allows for an efficient computation, but also makes these filters highly sensitive to noise (Figure 4.2). In addition, they fail to reliably detect edges at large scales without further processing.

The Canny edge detector [Can86] provides several improvements over simple thresholding of the gradient magnitude. Smoothing and differentiation are combined into a single operator (i.e., Gaussian derivatives) and a non-maximum suppression scheme is applied to


(a) *Original image* (b) *Sobel* (c) *Canny* (d) *LoG*

(e) *Isotropic DoG* (f) *Flow-based DoG* (g) *Flow-based XDoG* (h) *Cartoon*

Figure 4.3: *Example showing the output of different edge detection and stylization methods. (a) The original image. (b) The globally thresholded gradient magnitude of the Sobel filter. (c) The output of the Canny edge detector [Can86]. (d) Zero-crossing of the Laplacian of Gaussian [MH80]. (e) The thresholded DoG as proposed in Winnemöller et al. [WOG06]. (f) The thresholded output of the separable implementation of the flow-based DoG. (g) The flow-based DoG with XDoG thresholding. The image is pre-processed with a bilateral filter to suppress noise. (h) Cartoon-style abstraction generated with bilateral and flow-based DoG filters using the generalized cartoon pipeline.*

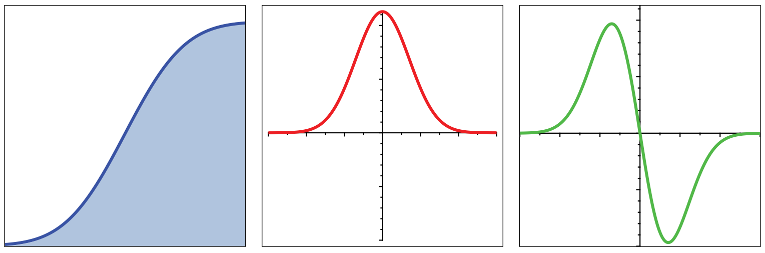
detect local maxima and thereby localize edges. Finally, hysteresis thresholding enhances the coherence of detected edges and reduces false positives. These attributes of the Canny detector, along with its widespread availability, have made it one of the most popular edge detectors, particularly for computer vision. However, as demonstrated in Figure 4.3(c), the single pixel-wide edges it creates are typically not attractive from an artistic point of view, since edges representing traces or outlines are commonly expected to exhibit a certain amount of width and width-variability. Therefore, techniques employing it for artistic purpose typically perform additional processing, such as scale-space analysis [Orz+07] or curve fitting [DS02]. By contrast, the difference of Gaussians operator (Figure 4.3(e)–(g)) offers a good compromise between computational efficiency and stylistic versatility.

4.2 Isotropic Difference of Gaussians

This section reviews the development of Laplacian-based edge detection and how this led to the difference of Gaussians filter.

4.2.1 Laplacian and Marr-Hildreth Theory

Even before Canny suggested using non-maxima suppression to localize edges, Marr and Hildreth [MH80] investigated the problem from a computational point of view and proposed an approach to edge detection based on the second derivatives of an image (Figure 4.4). For



(a) *Smooth step edge* (b) *First derivative* (c) *Second derivative*

Figure 4.4: *Edge detection based on the second derivative. In the one-dimensional case, searching for a maximum in the first derivative is equivalent to finding a zero-crossing in the second derivative.*

one-dimensional functions, a maximum of the gradient magnitude is equivalent to a zero-crossing in the second derivative. This also generalizes to two dimensions, where the second derivative in perpendicular direction to the zero-crossing has to be considered. However, this presents a chicken-and-egg problem, as the second derivative must be computed in a direction that is yet to be determined by the result of the computation (i.e., the edge). Marr and Hildreth suggested circumventing this problem by using the *Laplacian*

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \,, \tag{4.1}$$

which is rotationally invariant. While the Laplacian was known at that time to be useful for sharpening images [GW06], it had not been used for edge detection due to its high sensitivity to noise. Marr and Hildreth's key insight was to smooth the image before applying the Laplacian. This has two important effects. First, noise is reduced and the differentiation regularized. Second, the bandwidth is restricted, which means that the range of possible scales at which edges can occur is reduced. For the smoothing filter, a two-dimensional Gaussian function

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(- \frac{\|x\|^2}{2\sigma^2} \right) \tag{4.2}$$

with standard deviation σ was chosen, since the Gaussian function is known to minimize uncertainty, which simultaneously measures the spread of a function in the spatial and frequency domains. Since the Laplacian commutes with convolution, for an image I it follows that

$$\nabla^2 \big(G_\sigma \star I \big) = \big(\nabla^2 G_\sigma \big) * I \,, \tag{4.3}$$

where $*$ denotes the convolution operator. Thus, instead of applying smoothing and differentiation in sequence, both operations can be combined into a single operator $\nabla^2 G_\sigma$, which is known as the *Laplacian of Gaussian* (LoG) and which can be symbolically computed.

To extract edges from a Laplacian of Gaussian filtered image, the local neighborhood of a pixel is typically examined to detect the zero-crossings. This, however, results again in artistically questionable 1-2 pixel-wide edges (Figure 4.3(d)) similar to those produced by the Canny edge detector. To achieve an artistically interesting effect, it turns out that simple thresholding works surprisingly well, and will be discussed in detail in Section 4.3.

4.2.2 Difference-of-Gaussians (DoG)

When Marr and Hildreth developed their theory of edge detection based on the Laplacian, they were faced with the fact that the Laplacian is not separable, and computationally efficient ways to obtain it were not known at that time. While today a number of methods are available to compute the Laplacian efficiently [Kin82; CHM87; JG97; PH02], Marr and Hildreth [MH80, App. B] proposed to approximate the Laplacian of Gaussian by the differences of two Gaussian functions, which are themselves separable. This also matched results in biological vision, which showed that the ganglion cell receptive fields of cats can be modeled in this way [You87], and provided motivation for their approach and helped to popularize the technique. This approximation may be verified by looking at the difference of Gaussians with infinitesimally small change in the standard deviation σ, yielding the differential quotient

$$\lim_{k\to 1} \frac{G_{k\sigma}(x) - G_\sigma(x)}{k\sigma - \sigma} = \frac{\partial G_\sigma}{\partial \sigma} = \sigma\, \nabla^2 G\,. \tag{4.4}$$

Thus, it can be seen that the *difference of Gaussians* (DoG) filter

$$D_{\sigma,k}(x) = G_\sigma(x) - G_{k\sigma}(x) \approx -(k-1)\,\sigma^2\, \nabla^2 G \tag{4.5}$$

approximates the negated scale-normalized Laplacian (as defined by Lindeberg [Lin94a]) up to a constant positive factor. The scale-normalization is a useful property of the DoG, since it ensures that the response does not change when modifying the scale σ. Thresholding values, for instance, can therefore be defined independently of the scale. In this work, the commonly cited suggestion by Marr and Hildreth [MH80, App. B] to use $k = 1.6$ as a good engineering trade-off between small bandwidth and adequate sensitivity will be followed. However, other choices are possible, for example, when deriving DoG for several levels of an image pyramid [Low04].

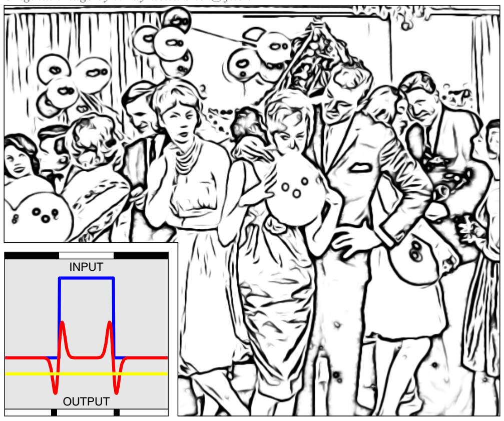
There is also an interesting interpretation of the DoG in terms of signal processing [OS75]. A Gaussian filter is a low-pass filter, that is, it allows low spatial frequencies to pass, while attenuating or eliminating high spatial frequencies. Accordingly, the subtraction of two Gaussians creates a band-pass filter that attenuates all frequencies between the cut-off frequencies of the two Gaussians. A DoG filter can therefore detect edges whose spatial extent falls within this characteristic frequency band [OS75; Lim90].

4.3 Extended Difference of Gaussians (XDoG)

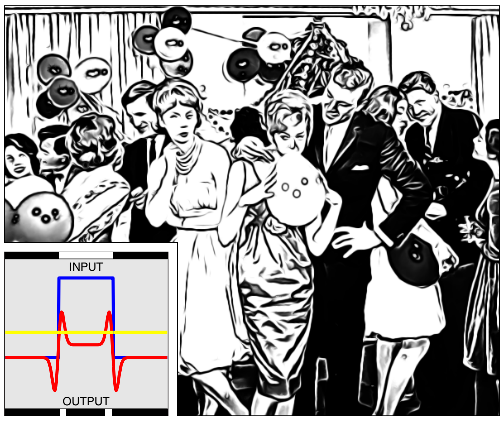
Comparing the two rows of images in Figure 4.3, it becomes evident that edge "detection", useful in computer vision, is qualitatively quite different from edge "enhancement" for stylistic and artistic applications. While the former is primarily concerned with the exact localization and extent of an edge, the latter is more appropriately focused on the weight (i.e., thickness) and structure (i.e., shape) of an edge.

If we wish to generate a two-tone edge image, we essentially have two choices. Either we start with a white image and make certain image regions darker (i.e., set them to black) or we start with a black image and perform highlighting (i.e., set those regions to white). Because DoG is a band-pass filter, the sign of its response describes whether capturing

Original image by x-ray delta one @ flickr



(a) *DoG*



(b) *XDoG*

Figure 4.5: *Illustration of the XDoG thresholding scheme for a step edge (blue). The output of the DoG and XDoG operator before thresholding is shown in red; the threshold ε is indicated by the yellow line. (a) DoG has no tone mapping effect. Light regions are outlined. (b) XDoG allows for a tone mapping effect. Light regions get a black outline, dark regions receive a white outline. In both cases the flow-based variant was used.*

the shape and structure of any nearby edges requires making each pixel darker or brighter than most of its neighbors. This is exactly the information we need to generate an "edge enhancement image". Such an image may be formally defined as a thresholding of the DoG response, $T_\varepsilon(D_{\sigma,k} * I)$, where,

$$T_\varepsilon(u) = \begin{cases} 1 & u \geq \varepsilon \\ 0 & \text{otherwise} \end{cases} . \tag{4.6}$$

The parameter ε is used to control the sensitivity to noise. Figure 4.3(e) demonstrates the effectiveness of the approach. Despite being comparatively simple, the result captures many important image features and is aesthetically pleasing.

In the context of computer vision, the word "edges" is used to refer to the thin lines formed by locally maximal gradient points, such as the DoG zero-crossings shown in Figure 4.3(d). However, in the context of image stylization, it is more natural to use the word "edges" to refer to an image like the thresholded DoG shown in Figure 4.3(e). As this work focuses on stylization applications, the term "edge image" will be used when referring to results like those in Figure 4.3(e)–(g), though it should be noted that in the computer vision community, such images would not typically be said to contain "edges".

The edge images generated by simple thresholding of the DoG are closely related to the biological models proposed by Young and others [You87]. Inspired by those models, Winnemöller et al. [WOG06] generated edge images using a DoG variant in which the strength of the inhibitory effect of the larger Gaussian is allowed to vary, resulting in the following equation:

$$D_{\sigma,k,\tau}(x) = G_\sigma(x) - \tau \cdot G_{k\sigma}(x) \tag{4.7}$$

That modification makes it possible to achieve a much wider range of stylistic effects, particularly after replacing the binary thresholding function T_ε with a continuous ramp:

$$T_{\varepsilon,\varphi}(u) = \begin{cases} 1 & u \geq \varepsilon \\ 1 + \tanh\big(\varphi \cdot (u - \varepsilon)\big) & \text{otherwise.} \end{cases} \tag{4.8}$$

Taken together, $T_{\varepsilon,\varphi}(D_{\sigma,k,\tau} * I)$ is referred to as the *XDoG* filter for a given image I. Figure 4.5(b) demonstrates how the base thresholded DoG is extended as a result of the soft thresholding and variable inhibition strength.

4.4 Reparameterization of the XDoG

The XDoG thresholding scheme is difficult to control. Increasing the sensitivity of the filter to edges typically requires adjusting τ, φ, and ε in concert. We can see the reason for this by decomposing $D_{\sigma,k,\tau}(x)$ as follows:

$$\begin{aligned} D_{\sigma,k,\tau}(x) &= G_\sigma(x) - \tau \cdot G_{k\sigma}(x) \\ &= (1-\tau) \cdot G_\sigma(x) + \tau \cdot D_{\sigma,k}(x) \end{aligned} \tag{4.9}$$

Hence, we see that Equation (4.7) is equivalent to a weighted average of the blurred image and the standard DoG. Notice that the average response of the standard DoG is zero, while the blurred image will have the same average brightness as the input image. Thus, the average brightness of $D_{\sigma,k,\tau} * I$ will decrease as τ increases. However, increasing τ is the only way to increase the weight of the edge emphasis lines. Thus, in order to create XDoG outputs having different edge emphasis strengths but the same average brightness, any adjustment to τ must be coupled with compensating changes to the soft thresholding parameters φ and ε.

In order to simplify artistic control of the XDoG filter, a reparameterization having the following properties would be desirable:

1. Removal of the tight parameter-interdependency of the previous parameterization;
2. More intuitive parameters by mapping to known image processing operations, such as blurring or sharpening;
3. Invertibility, that is, it should be possible to convert back-and-forth between the old and new parameter spaces.

Fortunately, a parameterization that fulfills these requirements can be found by simply dividing Equation (4.9) by $\tau - 1$, resulting in a representation of the XDoG filter as an adjusted image sharpening operator:

$$\begin{aligned} S_{\sigma,k,p}(x) = \frac{D_{\sigma,k,p}(x)}{\tau - 1} &= G_\sigma(x) + p \cdot D_{\sigma,k}(x) \\ &= (1+p) \cdot G_\sigma(x) - p \cdot G_{k\sigma}(x) \end{aligned} \tag{4.10}$$

Obviously, the range of images that can be generated using $T_{\varepsilon,\varphi}(S_{\sigma,k,p} * I)$ is identical to the range of images that can be generated using the original formulation. However, replacing τ with p makes it possible to control the strength of the edge sharpening effect without influencing any other aspects of the filter. In addition, ε is now measured proportionally to image intensity, leading to an effective decoupling of the parameters.

The sharpened image generated by $S_{\sigma,k,p}$ can be understood as a digital approximation of the classical darkroom technique of *unsharp masking*. To perform an unsharp mask, a photographer uses a negative duplication technique to create a blurred version of the original negative. Using the blurred negative as a mask when creating a print has the effect of sharpening the original edges [Lan74]. The same effect is present in Equation (4.10), which can be understood as an unsharp mask of the blurred image $G_\sigma * I$, in which the brightness has been increased in order to compensate for any darkening due to the mask.

A wide range of different stylistic effects can be achieved by applying the soft thresholding function $T_{\varepsilon,\varphi}$ to the sharpened image $S_{\sigma,k,p} * I$. Larger or smaller φ control the sharpness of the black/white transitions in the image, while ε controls the level above which the adjusted luminance values will become white. However, $T_{\varepsilon,\varphi}$ is only one of many luminance adjustments that can be applied to the sharpened image. While the images in this work are created using the soft thresholding function, additional effects may be achieved by replacing $T_{\varepsilon,\varphi}$ with a more general luminance adjustment function (e.g., see [J2]).

The XDoG filter is still rather sensitive to noise. To some extent ε can be used to reduce sensitivity, but a simple and highly effective approach is to apply 1–2 iterations of the bilateral filter (Chapter 3) before applying the XDoG filter. The next section discusses another approach to make the DoG less sensitive to noise.

4.5 Flow-based Difference of Gaussians (FDoG)

An explanation for the high sensitivity to noise of the DoG can be given by looking at the decomposition of the Laplacian of Gaussian operator in the directions of the local gradient and tangent. The second derivative in direction of the gradient contributes to the edge localization, while the one in the direction of the tangent merely increases the sensitivity to noise. This observation motivates to consider detecting zero-crossings in the second derivative in direction of the gradient. Such an edge detector was first proposed by Haralick [Har84], and also the maximum suppression of the Canny edge detector [Can86] is essentially equivalent to looking for a zero-crossing in the second directional derivative. A detailed discussion of the relationship between the Laplacian and directional derivatives can be found in the work by Torre and Poggio [TP86].

The success of second derivative methods for edge detection suggests changing the XDoG filter from an isotropic to a directional operator. However, simply replacing the two-dimensional XDoG with its one-dimensional equivalent in the direction of the gradient does not lead to better results. To the contrary, the results are typically even worse, due to a missing regularization along the edge. The reason for this is twofold. First, a one-dimensional XDoG is very sensitive to an accurate estimation of the gradient direction, which is typically performed using first order Gaussian derivative operators along the coordinate axes. The scale of these derivatives must be similar to the scale of the XDoG. For instance, if their scale is too large, the estimated gradient direction will generally not match the underlying image structure, which limits opportunities for noise suppression. Second, the missing regularization in the tangent direction further increases the sensitivity to noise.

The first work that addressed these issues and provided significantly improved quality over the isotropic DoG is the *flow-based difference of Gaussians* (FDoG) filter by Kang et al.

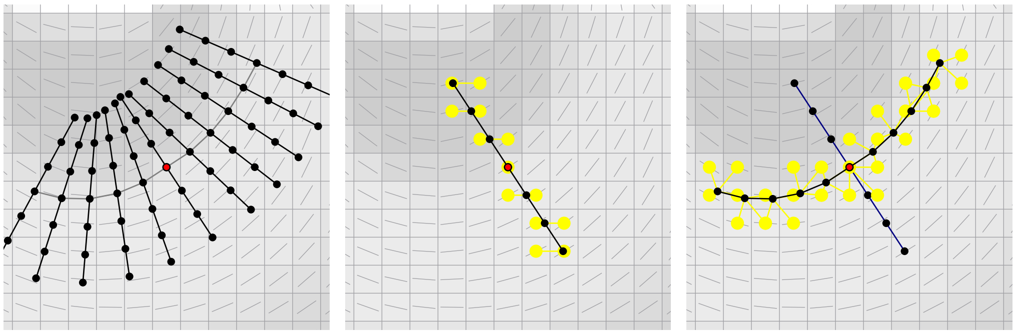

(a) *2D-kernel FDoG [KLC07]* (b) *Separable FDoG (1st pass)* (c) *Separable FDoG (2nd pass)*

Figure 4.6: *Different variants of the flow-based difference of Gaussians filter. (a) The originally proposed 2D-kernel is computationally expensive. (b)–(c) The separable implementation first computes a one-dimensional DoG in perpendicular to the tangent vectors and then smoothes along the integral curves.*

[KLC07]. In order to derive a smooth orientation field from the image, they introduced the edge tangent flow (Section 2.5). It provided Kang et al. with a vector field that is closely aligned to the underlying image structure and allowed them to derive an average gradient orientation that is less affected by noise. The originally proposed FDoG performs steps along the integral curves of the edge tangent flow by using an Euler integration scheme. At each step, a one-dimensional DoG filter in the direction perpendicular to the integral curve is applied and all these filter responses are accumulated by weighting them, using a one-dimensional Gaussian filter (Figure 4.6(a)). This accumulation performs a regularization in the tangent direction and shares similarity with the hysteresis thresholding of the Canny edge detector. This results in a better noise suppression and a significantly increased coherence of image features (i.e., longer, connected edges).

The flow-based difference of Gaussians filter as it was proposed by Kang et al. in [KLC07] has two fundamental limitations. First, since a one-dimensional DoG has to be computed for each sampling position on the integral curve, it is from perspective of computationally complexity equivalent to a 2D-kernel and, thus, comparatively expensive. Fortunately, there is a simple alternative[1]. The tangents of an integral curve match, by definition, the corresponding vectors of the vector field the integral curve was derived from. Hence, instead of computing a one-dimensional DoG filter for each sampling point on the integral curve, these may be computed for each pixel and stored as an intermediate result (Figure 4.6(b)). In a second pass, the responses of the one-dimensional DoG filters are then accumulated along the integral curve, by line integral convolution with a Gaussian kernel (Figure 4.6(c)). Assuming nearest-neighbor sampling along the integral curve, this approach is equivalent to Kang et al.'s method. In case of subsampling (e.g., bilinear) the proposed approach is an approximation, which, however, produces results that are indistinguishable from the original method. The second limitation is that the edge tangent flow is used to obtain a smooth vector field that is aligned to the image features. As pointed

[1] The separable FDoG implementation was first proposed in Kyprianidis and Döllner [C3] and independently later in Kang et al. [KLC09] as well.

```
__global__ void fdog( gpu_plm2<float> dst, const gpu_plm2<float2> tf,
                      float sigma_e, float sigma_r, float tau, float precision)
{
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float2 t = tf(ix, iy);
    float2 n = make_float2(t.y, -t.x);
    float2 nabs = fabs(n);
    float ds = 1.0f / ((nabs.x > nabs.y)? nabs.x : nabs.y);

    float twoSigmaE2 = 2 * sigma_e * sigma_e;
    float twoSigmaR2 = 2 * sigma_r * sigma_r;
    float halfWidth = precision * sigma_r;

    float sumE = tex2D(texSRC, ix + 0.5f, iy + 0.5f);
    float sumR = sumE;
    float2 norm = make_float2(1, 1);

    for( float d = ds; d <= halfWidth; d += ds ) {
        float kE = __expf( -d * d / twoSigmaE2 );
        float kR = __expf( -d * d / twoSigmaR2 );

        float2 o = d*n;
        float c = tex2D(texSRC, 0.5f + ix - o.x, 0.5f + iy - o.y) +
                  tex2D(texSRC, 0.5f + ix + o.x, 0.5f + iy + o.y);
        sumE += kE * c;
        sumR += kR * c;

        norm += 2 * make_float2(kE, kR);
    }
    sumE /= norm.x;
    sumR /= norm.y;

    dst.write(ix, iy, sumE - tau * sumR);
}
```

Listing 4.1: *Implementation of the first pass of the separable FDoG.*

out in Section 2.5, the edge tangent flow is comparable to a specialized bilateral filter. Since multiple iterations of the edge tangent flow have to be performed, this has a significant impact on the overall performance. As made evident in Section 2.5, the smoothed structure tensor is a better alternative.

The XDoG thresholding scheme and the flow-alignment (FDoG) are mutually independent extensions of the DoG operator and may therefore be combined, as desired. For the images in this work, Equation (4.8) is employed to produce stylistic variations, while on the separable FDoG implementation and the smoothed structure tensor is relied for noise suppression and increased coherence. The implementation of the separable FDoG follows the principles developed for the orientation-aligned bilateral filter (Section 3.4). The first pass uses unit step size in either horizontal or vertical direction. The second pass performs line integral convolution as discussed in Section 2.9. In contrast to the bilateral filter, blurring is not an issue, and therefore bilinear sampling is used for both passes. For the sake of completeness, the implementation of the first pass is shown in Listing 4.1.

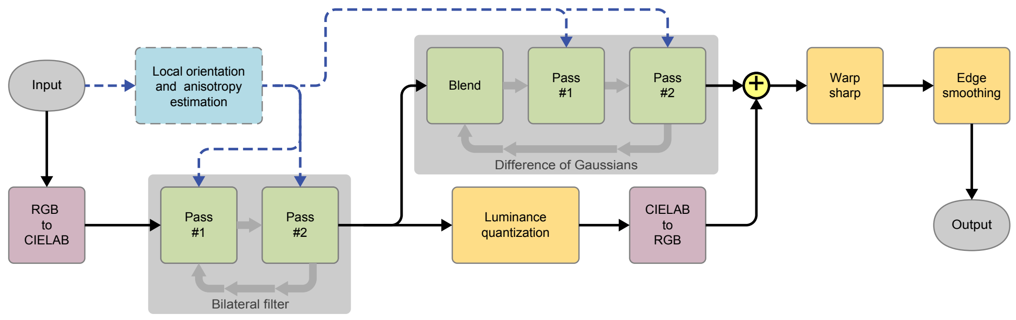


Figure 4.7: *The pipeline for the creation of a cartoon-like effect in modern generalized form. Processing starts with the conversion of the input to CIELAB color space. Then, the input is iteratively abstracted by using a variant of the bilateral filter. After one or two iterations of the bilateral filter to suppress noise, outlines are extracted from the intermediate result using a variant of the DoG filter. Then more iterations of the bilateral filter are performed, typically up to four, with luminance quantization applied afterwards. The DoG edges and the output of the luminance quantization are then composited, followed by an optional sharpening by warping and smoothing of the edges.*

4.6 Application: Enhanced Cartoon Filter

Taken together, the bilateral and DoG filters provide an effective way to create a cartoon-like effect. In Section 3.2 we saw that multiple iterations of the bilateral filter lead to a cartoon-like effect. Motivated by this, Fischer et al. [FBSe05] applied the bilateral filter in the context of augmented reality to make virtual objects less distinct from the camera stream by applying stylization to the virtual and camera inputs. However, at that time evaluating the bilateral filter at full resolution was computationally too expensive. Due to this, Fischer et al. applied the bilateral filter at reduced resolution followed by upsampling, resulting in an inferior result. Winnemöller et al. [WOG06] were faced with the same problem, but applied iteratively the xy-separable implementation of the bilateral filter discussed in Section 3.3. Although this brute force separation is prone to horizontal and vertical artifacts, it provides a reasonable trade-off in terms of quality and speed, and enabled real-time processing on consumer GPUs of that time. In addition to the bilateral filter, Winnemöller et al. added another processing step performing smooth luminance quantization. The quantization is applied in CIELAB space, with only the luminance channel being modified, creating a strong cartoon-like effect. The quantization is performed using a smooth step function, whose steepness is chosen depending on the luminance gradient. This makes the output of the quantization less sensible to small changes in the input, and increases temporal coherence when processing video frame by frame. Outlines at edges are obtained with the DoG filter.

A schematic overview of a modern generalized version of the cartoon pipeline is shown in Figure 4.7. Input is typically an image, a frame of a video, or the output of a three-dimensional rendering. In the original pipeline by Winnemöller et al. [WOG06], the local orientation estimation step was not present; this step was added later in [C3] to adapt the bilateral and DoG filters to the local image structure as discussed in this thesis. Also not present were the iterative application of the DoG filter, which was first proposed in [KLC07], and the final smoothing pass to further reduce aliasing of edges. The introduction of the flow-based DoG filter significantly increased the quality of the produced outlines, and made the warp-based sharpening step of the original pipeline less important. Therefore, this step is typically not present in later work. Figure 4.8 shows a few examples created with the generalized cartoon pipeline.

Original images courtesy Philip Greenspun and Antonio Santella

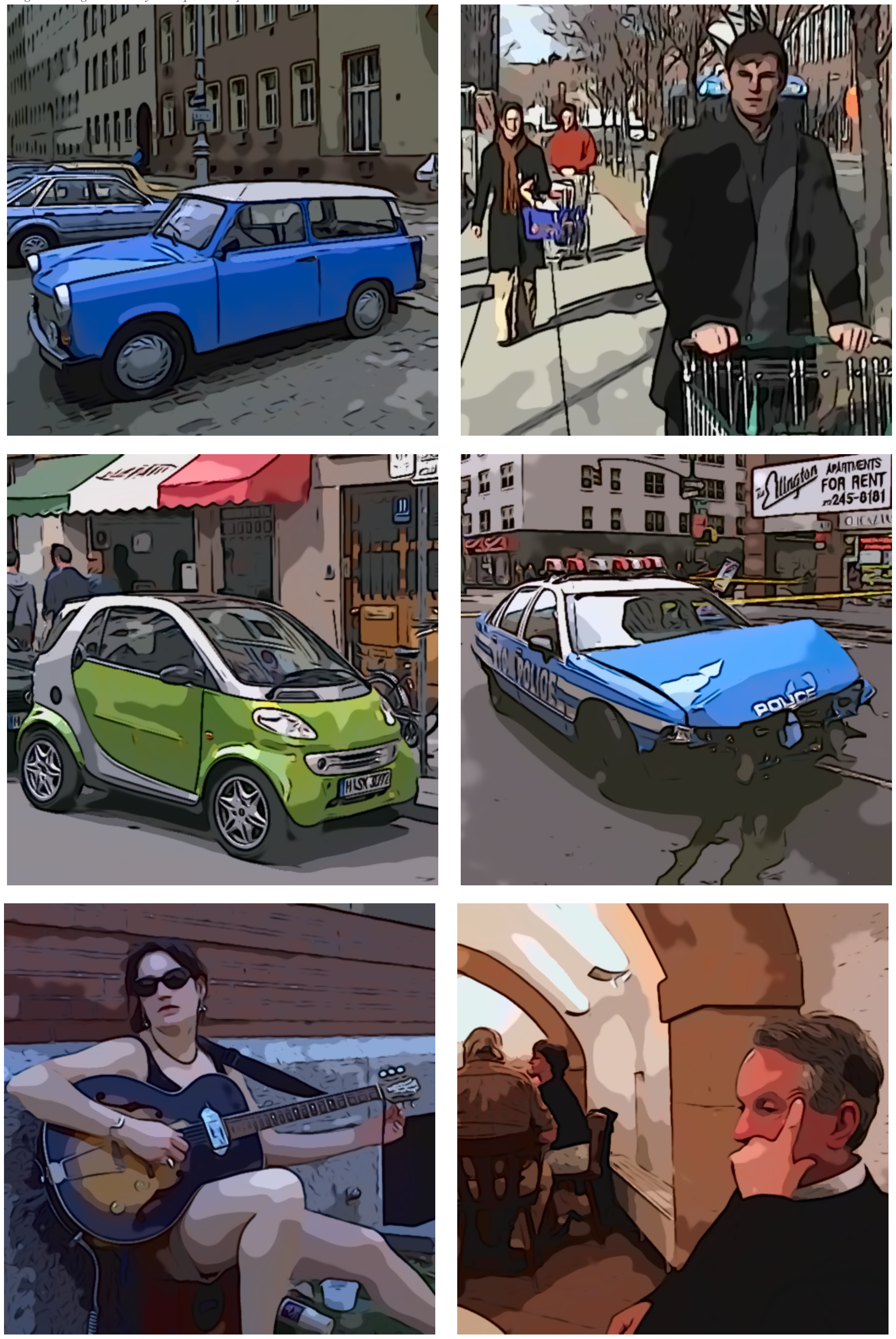


Figure 4.8: *Examples generated with the generalized cartoon pipeline. Separable orientation-aligned bilateral filter:* $n_a = 4$, $\sigma_d = 3.0$, $\sigma_r = 4.25\%$. *Separable flow-based DoG:* $n_e = 1$, $\tau = 0.99$, $\sigma_e = 1.0$, $\sigma_m = 3.0$, $\varphi = 2.0$. *Color quantization:* $q = 8$, $\varphi_q = 3.4$.

Chapter 5

Anisotropic Kuwahara Filtering

An interesting class of edge-preserving filters performing comparatively well on high-contrast images are variants of the *Kuwahara filter* [Kuw+76; TT77; NM79; Lee80; Lee81]. Based on local area flattening, these filters properly remove detail in high-contrast regions and protect shape boundaries in low-contrast regions, resulting in a roughly uniform level of abstraction across the image. The original Kuwahara filter was initially proposed in the mid-seventies as a noise reduction approach in the context of biological image processing, and belongs to the broader class of value-and-criterion filters [SP94], with value corresponding to the mean and the criterion being least variance. However, it is unstable in the presence of noise and suffers from block artifacts. Therefore, several extensions and modifications have been proposed to improve the original filter. Most notably, the work by Papari et al. [PPC07] introduced new weighting windows and replaced the selection criterion by a new combination rule. Even though this improved the output quality significantly, clustering artifacts proportional to the used filter radius remain noticeable.

In this chapter, the *anisotropic Kuwahara filter* is presented (Figure 5.1), a further generalization that removes clustering artifacts by adapting shape, scale, and orientation of the filter to the local structure of the input. In addition, directional image features are better preserved and emphasized, resulting in overall sharper edges and the enhancement

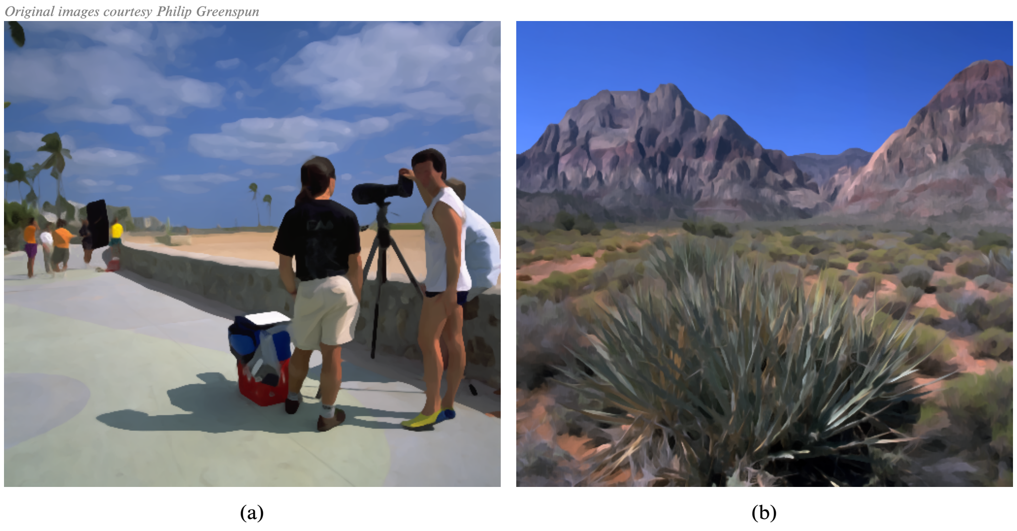


Figure 5.1: *Examples created by the anisotropic Kuwahara filter.*

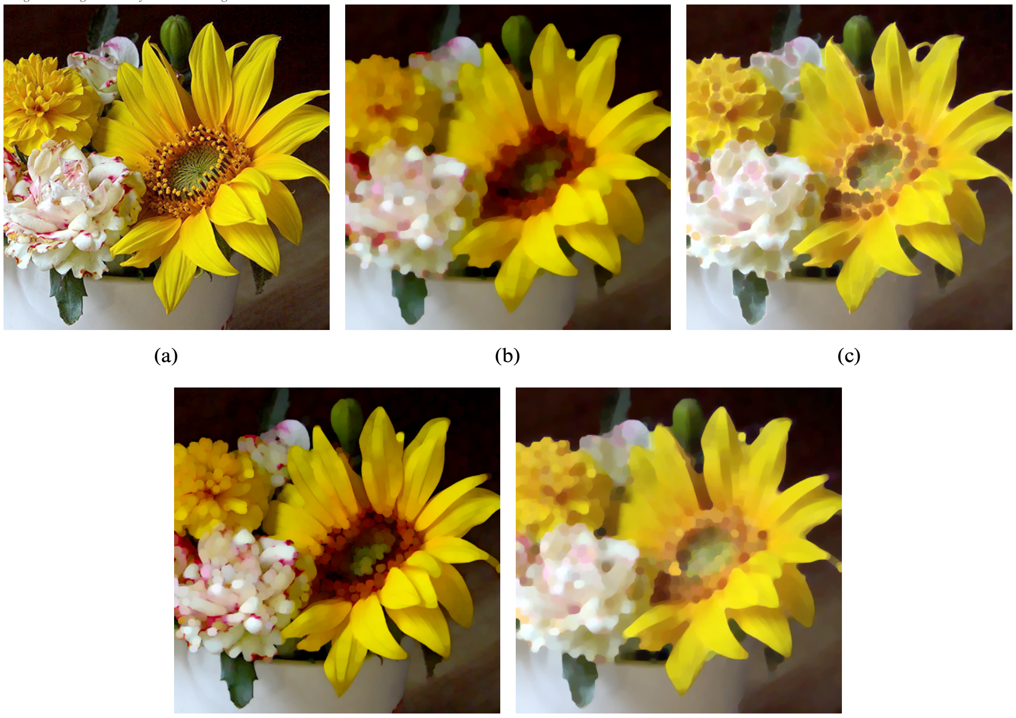
of anisotropic image features. Before introducing the anisotropic Kuwahara filter, a brief review of morphological filtering approaches in IB-AR is provided. Then, the classical Kuwahara filter and the generalized Kuwahara filter, as proposed by Papari et al. [PPC07], are discussed in detail. In particular, the implementation of the generalized Kuwahara filter in the frequency and spatial domains will be explained. For the anisotropic Kuwahara filter two alternative implementations are presented: Texture-based weighting functions and an approximation of the weighting functions by polynomials that can be evaluated on the fly at computation time. Furthermore, a multi-scale processing scheme for the anisotropic Kuwahara filter will be discussed. The chapter closes with a discussion of results and comparisons.

5.1 Morphological Filtering

Mathematical morphology provides a set-theoretic approach to image analysis and processing. Besides being useful the for extraction of object boundaries, skeletons, and convex hulls, it also has been applied successfully to many pre- and post-processing tasks. A good introduction to the subject, covering aspects of image processing and computer vision, is the tutorial by Haralick et al. [HSZ87]. Fundamental operations in mathematical morphology are *dilation* and *erosion*. From these, a large number of other operators can be derived, most notably *opening*, defined as erosion followed by dilation, and *closing*, defined as dilation followed by erosion. For grayscale images, dilation is equivalent to a maximum filter and erosion corresponds to a minimum filter. Therefore, opening removes light image features by removing peaks, while closing removes dark features by filling holes. Applying opening and closing in sequence results in a smoothing operation that is often referred to as *morphological smoothing*, which, similar to a median filter, quite effectively suppresses salt-and-pepper noise, while being computationally less expensive. In fact, openings and closings are closely related to order-statistics filters. A further in-depth discussion of morphological filters and their relations to other image processing operators can be found in Maragos and Schafer [MS87a; MS87b].

Morphological smoothing is applied in Bousseau et. al.'s [Bou+06; Bou+07] work on watercolor rendering and in Bangham et al. [BGH03] oil paintings to simplify input images and videos before rendering. In the case of video, Bousseau et al. [Bou+07] use a spatio-temporal kernel aligned to the motion trajectory derived from optical flow. Applying opening and then closing generally results in a darkened result. Since watercolor paintings typically have light colors Bousseau et al. [Bou+07] proposed to swap the order of the morphological operators and apply closing followed by opening. Because opening and closing are dual, this is equivalent to inverting the output of morphological smoothing applied to the inverted image (Figure 5.2). Papari and Petkov [PP09] described another technique that applied morphological filtering in the context of IB-AR. Motivated by *glass patterns* and similar to line integral convolution (Section 2.9.3), they performed a one-dimensional dilation in form of a maximum filter over noise along the integral curves defined by a vector field. In contrast to line integral convolution, this technique is more capable of producing thick piecewise constant coherent lines with sharp edges, resulting in a stronger brush-like effect. Moreover, it can also be applied to color images by using the

Original image courtesy PDPhoto.org



(a) (b) (c)

(d) (e)

Figure 5.2: *Mathematical morphology operators. (a) Original image. (b) Opening. (c) Closing. (d) Opening followed by closing. (e) Closing followed by opening: The morphological operator chosen by Bousseau et al. [Bou+06; Bou+07].*

location of the first maximum noise value along the integral curve as a look-up position.

Morphological operators can be efficiently implemented by using distance transforms [Fab+08]. Criminisi et al. [Cri+10] recently demonstrated that edge-sensitive smoothing based on the *generalized geodesic distance transform* can be used for the creation of cartoon-style abstractions. The image is first clustered into a fixed number of colors. Then, for every pixel, the probability of the pixel's value belonging to a certain cluster is defined. These probabilities form a soft mask to which the generalized geodesic distance transform is applied. The output is then defined as the weighted sum of the cluster's mean values where the weights are defined based on the corresponding distances.

As pointed out by Schulze and Pearce [SP93; SP94],the value-and-criterion structure allows the use of different linear and nonlinear elements in a single filter but also provides the shape control of morphological filters, which is of central importance for the anisotropic Kuwahara filter.

5.2 Kuwahara Filter

In this section the classical Kuwahara filter will be reviewed. Two perspectives on the subject are provided. First, the Kuwahara filter will be defined, in the traditional way, based on subregions. Second, the definition will be reformulated as convolution with appropriate

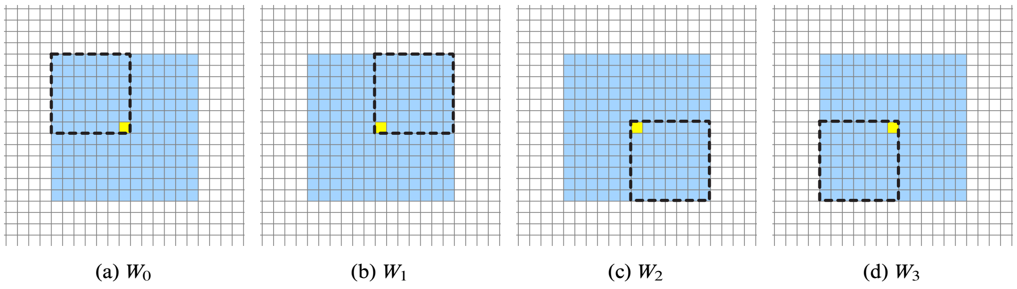


Figure 5.3: *Subregions used by the classical Kuwahara filter for $r = 7$.*

weighting functions, which serves as the foundation for further generalizations discussed later in the other sections of this chapter.

5.2.1 Classical Approach based on Regions

The general idea behind the classical Kuwahara filter [Kuw+76] is to divide the local filter neighborhood into four rectangular subregions that all contain the filter origin and overlap by one pixel (Figure 5.3). For all subregions the variance is computed, and the response of the filter defined as the mean of a subregion with minimum variance. More precisely, let $f : \mathbb{Z}^2 \to \mathbb{R}$ denote an grayscale image, let $r > 0$ be the radius of the filter, and let $x \in \mathbb{Z}^2$ be any point. The rectangular subregions are then given by:

$$\begin{aligned} W_0 &= [x_1 - r, x_1] \times [x_2, x_2 + r] \\ W_1 &= [x_1, x_1 + r] \times [x_2, x_2 + r] \\ W_2 &= [x_1, x_1 + r] \times [x_2 - r, x_2] \\ W_3 &= [x_1 - r, x_1] \times [x_2 - r, x_2] \end{aligned} \tag{5.1}$$

Let $|W_k| = (r+1)^2$ be the number of pixels in each subregion. The mean (average) of a subregion W_i is then given by:

$$m_k(x) = \frac{1}{|W_k|} \sum_{y \in W_k} f(y) \tag{5.2}$$

The variance is defined as the average of the square of the distance of each pixel to the mean, which is equivalent to subtracting the squared mean from the averaged squared image values:

$$\begin{aligned} s_k^2(x) &= \frac{1}{|W_k|} \sum_{y \in W_k} \big(f(y) - m_k(x)\big)^2 \\ &= \frac{1}{|W_k|} \sum_{y \in W_k} f^2(y) - m_k^2(x) \end{aligned} \tag{5.3}$$

Now, the output of the Kuwahara filter is defined as the mean of a subregion with minimum variance

$$F(x) = m_{\hat{k}}(x) \quad \text{with} \quad \hat{k} = \operatorname*{arg\,min}_{k \in \{0,\ldots,3\}} s_k(x) \,. \tag{5.4}$$

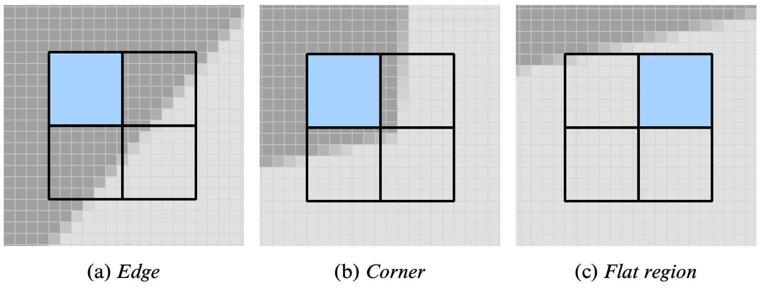

(a) *Edge* (b) *Corner* (c) *Flat region*

Figure 5.4: *The Kuwahara filter defines an edge-preserving smoothing approach. Subregions containing pixels from both sides of an edge have high variance. By contrast, subregions completely lying inside a color region have low variance.*

For color images, the mean and variance are computed for each color channel independently. Assuming that the variances of the color channels do not correlate, the variance of the subregion W_k is defined as the sum of the squared variances of the color channels:

$$s_k^2(x) = s_{k,r}^2(x) + s_{k,g}^2(x) + s_{k,b}^2(x) \tag{5.5}$$

As can be seen in Figure 5.4, this avoids averaging between differently colored regions for corners and edges. However, for flat or homogeneous regions the variances of the different subregions are similar or even the same. Therefore, a subregion with minimum variance is generally not well-defined, and the selection highly unstable, especially in the presence of noise. For small filter sizes, the Kuwahara filter produces reasonable results. However, for image-based artistic rendering, comparatively large filter sizes are necessary to achieve an interesting abstraction or stylization effect, resulting in clearly noticeable artifacts, as demonstrated in Figure 5.5. These are due to the unstable subregion selection process and the use of rectangular subregions.

5.2.2 Variant based on Weighting Functions

In order to avoid the Gibbs ringing artifacts resulting from box filtering, alternative shapes for the subregions have been proposed. For instance, Nagao and Matsuyama [NM79] used pentagons and hexagons as subregions and Bakker et al. [BVV99] considered circular

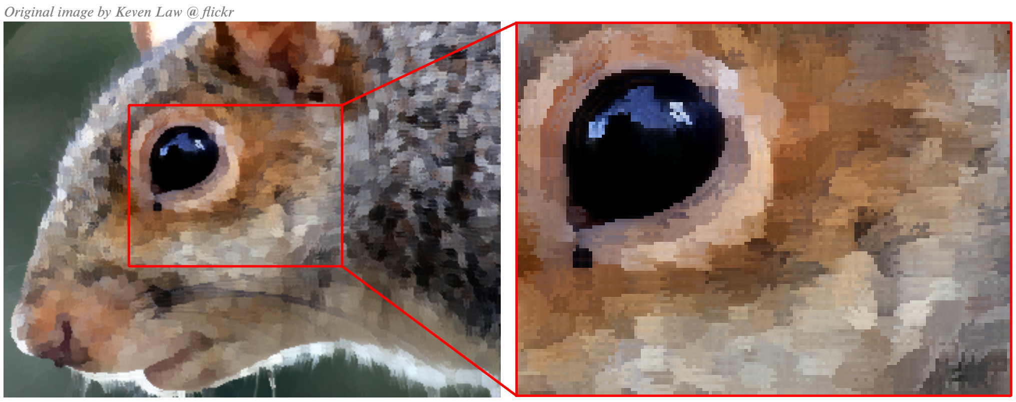
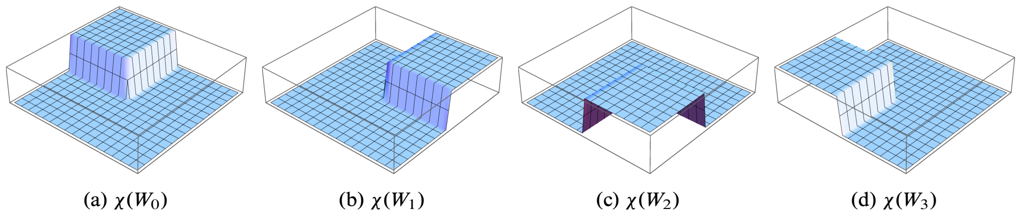


Figure 5.5: *Output of the classical Kuwahara filter.*



(a) $\chi(W_0)$ (b) $\chi(W_1)$ (c) $\chi(W_2)$ (d) $\chi(W_3)$

Figure 5.6: *Characteristic functions of the subregions used by the Kuwahara filter.*

subregions. A more general approach due to Boomgaard [Boo02] is to compute the means and variances by convolving the image with a set of suitable weighting functions. More specifically, given a weighting function $w_k : \mathbb{R}^2 \to \mathbb{R}$, the *weighted mean* of a function $f : \mathbb{R}^2 \to \mathbb{R}$ with respect to w_k is given by

$$\begin{aligned} m_k(x) &= \frac{1}{|w_k|}\,(f * w_k)(x) \\ &= \frac{1}{|w_k|}\int f(y)\, w_k(x-y)\,\mathrm{d}y \end{aligned} \tag{5.6}$$

and *weighted variance* with respect to w_k is given by

$$\begin{aligned} s_k^2(x) &= \frac{1}{|w_k|}\big((f-m_k)^2 * w_k\big)(x) \\ &= \frac{1}{|w_k|}\int f^2(y)\, w_k(x-y)\,\mathrm{d}y - m_k^2(x)\;, \end{aligned} \tag{5.7}$$

with

$$|w_k| = \int w_k(y)\,\mathrm{d}y \tag{5.8}$$

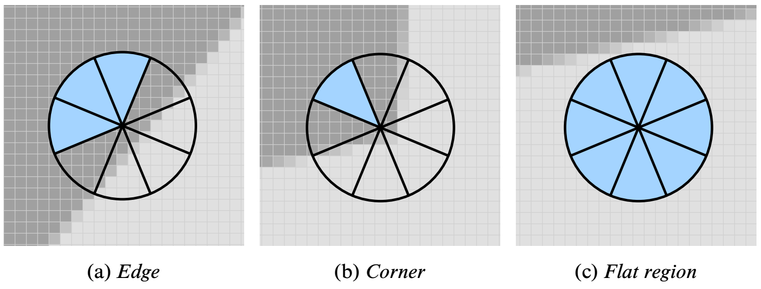
denoting the corresponding normalization term. Choosing the characteristic functions

$$\chi(W_k)(x) = \begin{cases} 1 & \text{if } x \in W_k \\ 0 & \text{otherwise} \end{cases}, \quad k = 0,\dots,3, \tag{5.9}$$

of the subregions W_k (Figure 5.6) as weighting functions, yields the classical Kuwahara filter. Using overlapping two-dimensional Gaussian functions for the weighting functions is known as the Gaussian Kuwahara filter, which enabled Boomgaard [Boo02] to link the Kuwahara filter to modern PDE-based edge-preserving and edge-enhancing smoothing techniques. He showed that the Gaussian Kuwahara filter can be interpreted as a PDE with linear diffusion and shock filter terms. Since shock filters are related to a morphological sharpening operator originally proposed by Kramer and Bruckner [KB75], this also provides a link to morphological filtering. Shock filter will be discussed in more detail later in Section 6.4.1.

5.3 Generalized Kuwahara Filter

In this section, the generalized Kuwahara filter [PPC07] is examined. It replaces the rectangular subregions by smooth weighting functions over sectors of a disc (Figure 5.7).



(a) *Edge* (b) *Corner* (c) *Flat region*

Figure 5.7: *The generalized Kuwahara filter uses weighting functions defined over sectors of a disc. The filter response is defined as a weighted sum of the local averages, where more weight is given to those averages with low variance.*

Moreover, to overcome the limitations of the unstable subregion selection process, a new criterion is defined. Instead of selecting a single subregion, the result is defined as the weighted sum of the means of the subregions. The weights are defined based on the variances of the subregions. This results in smoother region boundaries and fewer artifacts. As can be seen in Figure 5.8, this significantly improves the quality of the output. The next section reviews the construction of the weighting functions and the new combination rule. Then, the implementation of the generalized Kuwahara filter in the frequency and spatial domains are discussed.

5.3.1 Definition of the Generalized Kuwahara Filter

To construct a set of smooth weighting functions over the sectors of a disc, the plane is divided into N equal sectors by defining characteristic functions which are 1 over the sector and 0 otherwise

$$\chi_k(x) = \begin{cases} 1 & \text{if } \frac{(2k-1)\pi}{N} < \arg(x) \leq \frac{(2k+1)\pi}{N} \\ 0 & \text{otherwise} \end{cases} \qquad i = 0, \ldots, N-1\,, \tag{5.10}$$

where the arg function is assumed to return values in the range $[0, 2\pi)$. The characteristic functions of the different sectors χ_i are first convolved and then multiplied with a two-

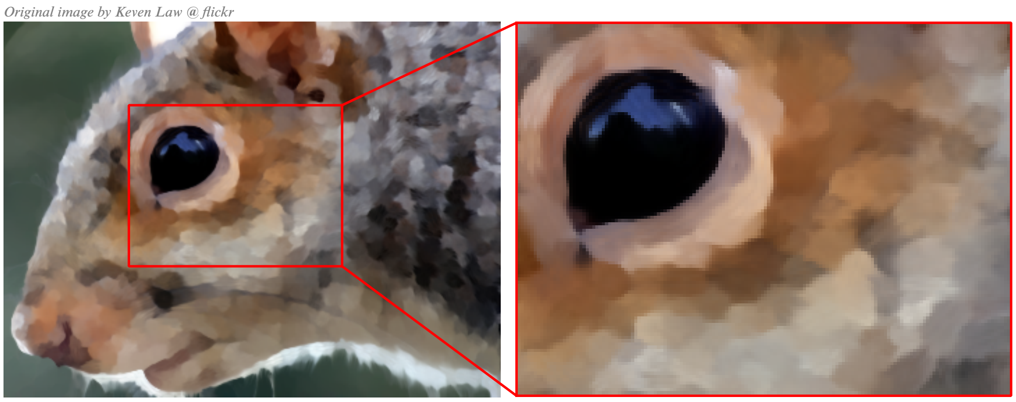


Figure 5.8: *Output of the generalized Kuwahara filter.*

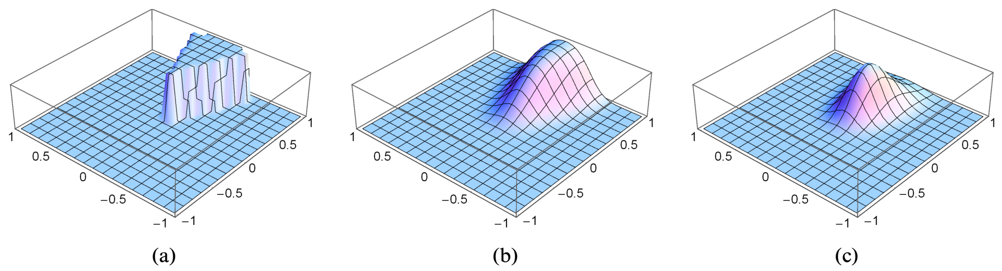


Figure 5.9: *Construction of the weighting function h_0 for $N = 8$: (a) Characteristic function χ_0. (b) Characteristic function χ_0 convolved with Gaussian function G_{σ_s}. (c) Finally, multiplication with Gaussian function G_{σ_r}.*

dimensional Gaussian:

$$h_k = \left(\chi_i * G_{\sigma_s} \right) \cdot G_{\sigma_r} \tag{5.11}$$

The convolution smoothes the characteristic functions such that they slightly overlap, and the multiplication achieves a decay with increasing radius. The construction of h_0 is illustrated in Figure 5.9. In Figure 5.10, plots of the weighting functions for eight sectors are shown. Notice that the sum of the weighting functions is equivalent to a Gaussian filter, since $\sum_k h_k(x) = G_{\sigma_r}(x)$, for $x \in \mathbb{Z}^2$. Let f denote the input image, then the weighted mean at any point $x \in \mathbb{Z}^2$ is given by

$$m_k(x) = \frac{1}{|h_k|} \sum_{y \in \mathbb{Z}^2} f(y)\, h_k(x-y)\,, \tag{5.12}$$

and the weighted variance is given by

$$s_k^2(x) = \frac{1}{|h_k|} \sum_{y \in \mathbb{Z}^2} f^2(y)\, h_k(x-y) - m_k^2\,, \tag{5.13}$$

where $|h_k| = \sum_{y \in \mathbb{Z}^2} h_k(y)$ denotes the corresponding normalization term.

Instead of selecting the weighted mean of a single subregion, the result of the filter is defined as the weighted sum of the weighted means, where the weights are based on the

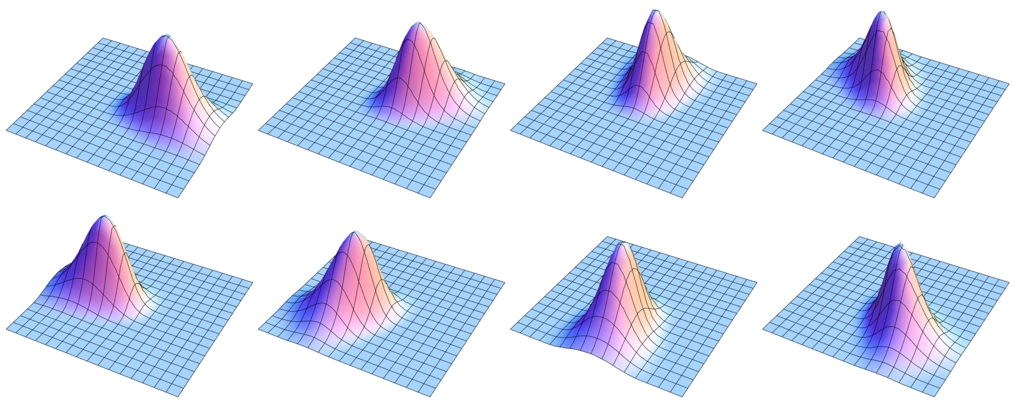


Figure 5.10: *Plots of the weighting functions $h_k = h_0 \circ R_{-2\pi k/N}$ for $N = 8$ and $k = 0, \ldots, N-1$.*

weighted variances, with sectors of low variance receiving a high weight and sectors of high variance receiving a low weight:

$$F(x) = \sum_{k=0}^{N-1} \alpha_k \, m_k(x) \Bigg/ \sum_{k=0}^{N-1} \alpha_k \tag{5.14}$$

This is achieved by taking the inverted weighted variance to the power of a user-provided parameter q. In flat and smooth regions, the variances are very small and sensitive to noise, resulting in a poorly approximated Gaussian. To avoid this, a simple solution is to threshold the variances using a control parameter τ_s, which also avoids the indetermination when some of the variances s_k^2 are zero. More precisely, the weights α_k are defined by:

$$\alpha_k = \max\big(s_k(x), \tau_s\big)^{-q} \tag{5.15}$$

The parameter q controls the sharpness of the output. A good choice is $q = 8$ and used for all examples. In Figure 5.7, the behavior of the generalized Kuwahara filter for different local neighborhoods is illustrated. As can be seen, for corners and edges the filter adapts itself to the local neighborhood by averaging one or more sectors, thus avoiding blurring across region boundaries. In case of homogeneous or flat image regions, the filter acts as a smoothing filter. From a symmetry point of view, obvious choices for the number of sectors are $N = 4$ and $N = 8$. In this chapter, we will mainly focus on $N = 8$, since it provides the best results.

5.3.2 FFT-based Implementation

The implementation of Equations (5.12) and (5.13) requires the computation of a large number of convolutions. For an RGB color image and $N = 8$ sectors, $6N = 48$ convolutions have to be computed. A standard way to speed up the convolution operations is to use the convolution theorem, which states that convolution in the spatial domain is equivalent to a multiplication in the frequency domain:

$$m_k = f * h_k = \mathcal{F}^{-1}\big(\mathcal{F}(f) \cdot \mathcal{F}(h_k)\big) \tag{5.16}$$

$$s_k^2 = f^2 * h_k = \mathcal{F}^{-1}\big(\mathcal{F}(f^2) \cdot \mathcal{F}(h_k)\big) \tag{5.17}$$

This approach was used for the MATLAB reference implementation by Papari et al. [PPC07]. Since the Fourier transforms of the filter kernels $\mathcal{F}(h_k)$ may be computed in a preprocessing stage, a total of 26 Fourier transforms have to be computed. In order to avoid artifacts at the boundary, it is important to add proper padding (e.g., $2.5\sigma_r$) to the border for a clamp-to-edge behavior [Pod07]. On the GPU, Fourier transforms can be efficiently implemented using NVidia's CUFFT library, which ships as part of the CUDA toolkit. Figure 5.11 compares the running times of an unoptimized FFT-based implementation against an implementation in the spatial domain that has been optimized as discussed later in Section 5.4.2. As expected, the FFT-based implementation outperforms the implementation in the spatial domain for large filter radii. Unfortunately, the FFT approach only works for linear shift-invariant filters. Adaptively per-pixel controlled filtering operations, therefore, have to be implemented in the spatial domain, which is discussed in the next section.

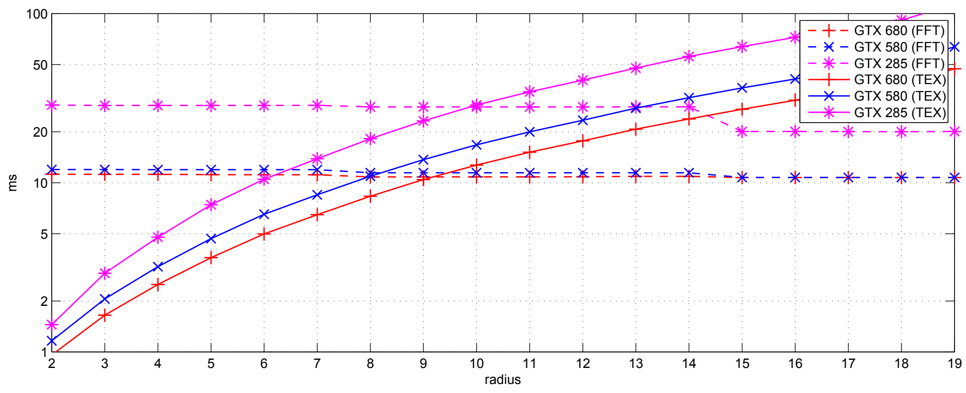


Figure 5.11: *Benchmark comparing a CUFFT-based implementation of the generalized Kuwahara filter against a texture-based implementation in the spatial domain.*

5.3.3 Implementation in the Spatial Domain

The weighting functions h_k, as defined in Equation (5.11), are difficult to compute on the fly, since their computation requires convolution, and a closed form solution is currently not known. Therefore, the most straightforward way to implement the generalized Kuwahara filter in the spatial domain is by defining a cut-off and sampling each of the weighting functions h_k into a texture map. Nevertheless, a slightly different approach that directly generalizes to anisotropic filtering will be presented in this section.

Let

$$R_\varphi = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \tag{5.18}$$

be the rotation matrix that performs a rotation by the angle φ in counter-clockwise order. Since $\chi_i = \chi_0 \circ R_{-2\pi i/N}$, and because Gaussian functions are rotationally symmetric, we have

$$\begin{aligned} h_k &= (\chi_k * G_{\sigma_s}) \cdot G_{\sigma_r} = \big((\chi_0 * G_{\sigma_s}) \cdot G_{\sigma_r}\big) \circ R_{-2\pi k/N} \\ &= h_0 \circ R_{-2\pi k/N} \end{aligned}, \tag{5.19}$$

where $\circ$ denotes composition of functions (i.e., $(f \circ g)(x) = f(g(x))$). Thus, all weighting functions h_k can be derived from h_0 by an appropriate rotation. The weighting function h_0 is sampled into a texture map H_0 of fixed size $H_{\text{size}} \times H_{\text{size}} = 32 \times 32$ and accessed using bilinear interpolation, corresponding to a linear reconstruction of the sampled function. Assuming $H_{\text{size}} = 2R + 1$ for the diameter and $R = 2.5\sigma_r$ for the cut-off radius, yields for the standard deviation of the Gaussian function G_{σ_r}:

$$\sigma_r = \frac{1}{2} \cdot \frac{H_{\text{size}} - 1}{2.5} = \frac{31}{5} = 6.2 \tag{5.20}$$

The standard deviation of the Gaussian G_{σ_s} responsible for the inter-sector smoothing is set to $\sigma_s = \sigma_r/3$, and the origin is moved to the center of the texture map as shown in Figure 5.12. Now suppose that $r > 0$ denotes the desired filter radius, then the weighting functions can be approximated by:

$$h_k(x) \approx H_0\left(\frac{1}{2r} R_{-2\pi k/N}(x)\right) \tag{5.21}$$

```
static texture<float4, 2> texSRC; // filter=Point,  address=Clamp, normalized=false
static texture<float, 2> texH0;   // filter=Linear, address=Wrap,  normalized=true

template <int N>
__global__ void gkf_filter( gpu_plm2<float3> dst, float radius,
                            float q, float threshold )
{
    int ix = blockDim.x * blockIdx.x + threadIdx.x;
    int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float3 m[N];
    float3 s[N];
    float w[N];
    for (int k = 0; k < N; ++k) {
        m[k] = s[k] = make_float3(0);
        w[k] = 0;
    }

    float piN = -2 * CUDART_PI_F / N;
    float4 RpiN = make_float4( cosf(piN), sinf(piN), -sinf(piN), cosf(piN) );

    int r = (int)ceilf(radius);
    for (int j = -r; j <= r; ++j) {
        for (int i = -r; i <= r; ++i) {
            float2 v = make_float2( 0.5f * i / radius,
                                    0.5f * j / radius);

            if (dot(v,v) <= 0.25f) {
                float3 c = make_float3(tex2D(texSRC, ix + i, iy + j));
                float3 cc = c * c;

                for (int k = 0; k < N; ++k) {
                    float wx = tex2D(texH0, v.x, v.y);

                    m[k] += c * wx;
                    s[k] += cc * wx;
                    w[k] += wx;

                    v = make_float2( RpiN.x * v.x + RpiN.z * v.y,
                                     RpiN.y * v.x + RpiN.w * v.y );
                }
            }
        }
    }

    float3 o = make_float3(0);
    float ow = 0;
    for (int k = 0; k < N; ++k) {
        m[k] /= w[k];
        s[k] = fabs(s[k] / w[k] - m[k] * m[k]);
        float sigma2 = fmaxf(threshold, sqrtf(s[k].x + s[k].y + s[k].z));
        float alpha_k = __powf(sigma2, -q);
        o += m[k] * alpha_k;
        ow += alpha_k;
    }

    dst.write(ix, iy, o / ow);
}
```

Listing 5.1: *Implementation of the generalized Kuwahara filter.*

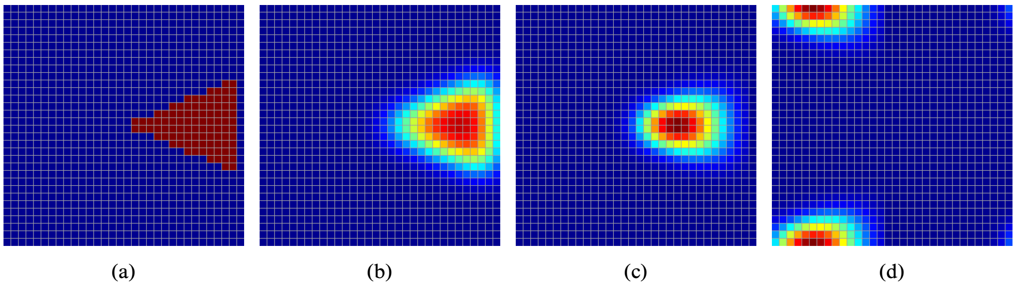


Figure 5.12: *Construction of the texture map H_0 with size 32×32. (a) Characteristic function. (b) Smoothed characteristic function. (c) Smoothed characteristic function multiplied with Gaussian function. To avoid artifacts during bilinear sampling, the last column is set to zero. (d) The last step shifts the center to the origin.*

An unoptimized implementation of the discussed approach is shown in Listing 5.1. The outer loop of the implementation iterates over a rectangular neighborhood of size $2r + 1$. To avoid a bias in direction of the coordinate axes, pixels farther away than the filter radius r are ignored. Inside the inner loop, the implementation iterates over all sectors and computes the corresponding weighting function h_k by sampling the texture map H_0. This implementation serves as the foundation for further generalization and optimizations, which will be discussed in the next section.

5.4 Anisotropic Kuwahara Filter

A limitation of the generalized Kuwahara filter is that it fails to capture directional features, resulting in clustering artifacts proportional to the filter radius (Figure 5.8). This issue is addressed by the *anisotropic Kuwahara filter*, which adapts the filter to the local structure of the input. The underlying idea is that in homogeneous regions the shape of the filter should be a circle while in anisotropic regions the filter should become an ellipse whose major axis is aligned with the principal direction of image features (Figure 5.13). As demonstrated in Figure 5.14, this avoids clustering and moreover creates a painterly look for directional image features.

The next section examines the definition of the elliptic filter shape. Then, two alternative

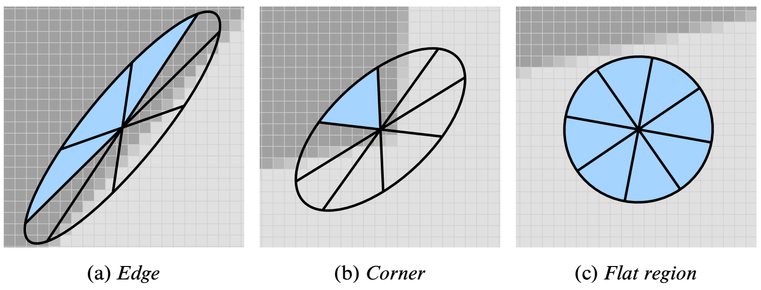


Figure 5.13: *The anisotropic Kuwahara filter uses weighting functions defined over an ellipse, whose shape is based on the local orientation and anisotropy. The filter response is defined as a weighted sum of the local averages, where more weight is given to those averages with low standard deviation.*

Original image by Keven Law @ flickr

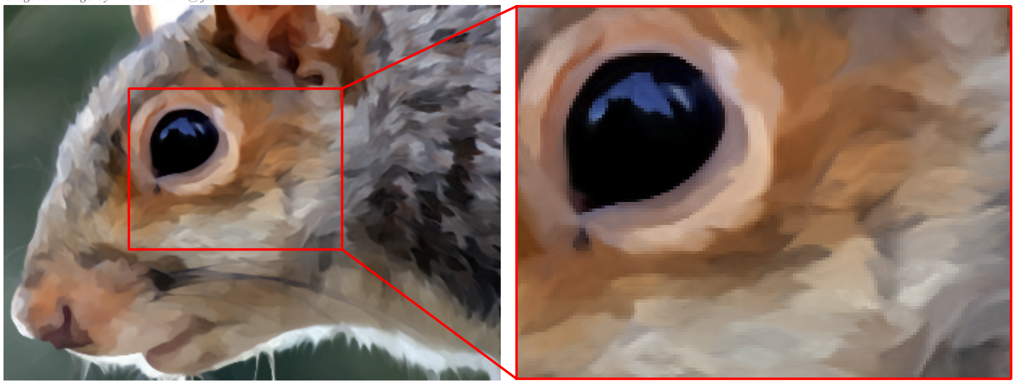


Figure 5.14: *Output of the anisotropic Kuwahara filter.*

approaches for defining weighting functions over the elliptic filter shape will be discussed. The first approach is a generalization of the texture-based weighting function. The second approach uses a polynomial approximation that can be evaluated on the fly.

5.4.1 Elliptical Filter Shape

Let $r > 0$ be the desired filter radius, let $t = (\cos\varphi, \sin\varphi)$ be the normalized minor eigenvector of the smoothed structure tensor computed using Listing 2.1, and let A be the anisotropy as defined in Section 2.6.1. To define the elliptical filter shape the approach in [PVS06; Pha06] is adopted. The eccentricity of the ellipse is defined by specifying the lengths of its major and minor axes depending on the amount of anisotropy:

$$a = \frac{\alpha + A}{\alpha} r \quad \text{and} \quad b = \frac{\alpha}{\alpha + A} r \tag{5.22}$$

The parameter $\alpha > 0$ is a tuning parameter. For $\alpha \to \infty$ the major axis a and the minor axis b converge to 1. In all examples, $\alpha = 1$ is used, which results in a maximum eccentricity of 4. The ellipse defined by a, b and φ has its major axis aligned to the local image orientation. It has high eccentricity in anisotropic regions and becomes a circle in isotropic regions.

For the implementation it will be important to know which pixels overlap the ellipse. In the remainder of this section it is therefore explained how to compute the ellipse's bounding box. An axis-aligned ellipse with major axis a and minor axis b may be defined as an implicit function by:

$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} = 1 \tag{5.23}$$

Rotating x by the angle φ, yields the equation for a rotated ellipse:

$$\frac{(x_1 \cos\varphi - x_2 \sin\varphi)^2}{a^2} + \frac{(x_1 \sin\varphi + x_2 \cos\varphi)^2}{b^2} = 1 \tag{5.24}$$

This is a quadratic polynomial in two variables, and by expanding and collecting terms it can be rewritten in normalized form as

$$P(x) = Ax_1^2 + Bx_2^2 + Cx_1 + Dx_2 + Ex_1x_2 + F = 0 \tag{5.25}$$

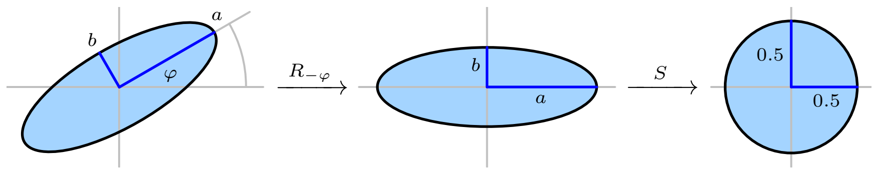


Figure 5.15: *The mapping $SR_{-\varphi}$ defines a linear coordinate transform that maps an ellipse defined by major axis a, minor axis b and angle φ to a disc with radius* 0.5.

with

$$\begin{aligned} A &= a^2 \sin^2 \varphi + b^2 \cos^2 \varphi \ , \\ B &= a^2 \cos^2 \varphi + b^2 \sin^2 \varphi \ , \\ C &= 0 \ , \\ D &= 0 \ , \\ E &= 2(a^2 - b^2) \sin \varphi \cos \varphi \ , \\ F &= -a^2 b^2 \ . \end{aligned} \tag{5.26}$$

The horizontal extrema are located where the partial derivative in x_2-direction vanishes:

$$\frac{\partial P}{\partial x_2} = 2Bx_2 + Ex_1 = 0 \quad \Leftrightarrow \quad x_2 = \frac{-Ex_1}{2B} \tag{5.27}$$

Substituting x_2 into Equation (5.25) we obtain:

$$\left(A - \frac{E^2}{4B} \right) x_1^2 + F = 0 \tag{5.28}$$

The horizontal extrema of the ellipse are therefore given by:

$$x_1 = \pm \sqrt{\frac{F}{\frac{E^2}{4B} - A}} = \pm \sqrt{a^2 \cos^2 \varphi + b^2 \sin^2 \varphi} = \pm \sqrt{a^2 t_1^2 + b^2 t_2^2} \tag{5.29}$$

A similar calculation gives the vertical extrema:

$$x_2 = \pm \sqrt{a^2 \sin^2 \varphi + b^2 \cos^2 \varphi} = \pm \sqrt{a^2 t_2^2 + b^2 t_1^2} \tag{5.30}$$

5.4.2 Convolution-based Weighting Functions

After defining the elliptical filter shape, the next step is to define weighting functions over the ellipse. As in the previous section, let $t = (\cos \varphi, \sin \varphi)$ be the normalized minor eigenvector of the smoothed structure tensor, and let a and b the major and minor axes of the ellipse. Moreover, let

$$S = \begin{pmatrix} \frac{1}{2a} & 0 \\ 0 & \frac{1}{2b} \end{pmatrix}, \tag{5.31}$$

then the linear mapping

$$SR_{-\varphi} = \begin{pmatrix} \frac{1}{2a} & 0 \\ 0 & \frac{1}{2b} \end{pmatrix} \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \frac{\cos \varphi}{a} & \frac{\sin \varphi}{a} \\ -\frac{\sin \varphi}{b} & \frac{\cos \varphi}{b} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \frac{t_1}{a} & \frac{t_2}{a} \\ -\frac{t_2}{b} & \frac{t_1}{b} \end{pmatrix} \tag{5.32}$$

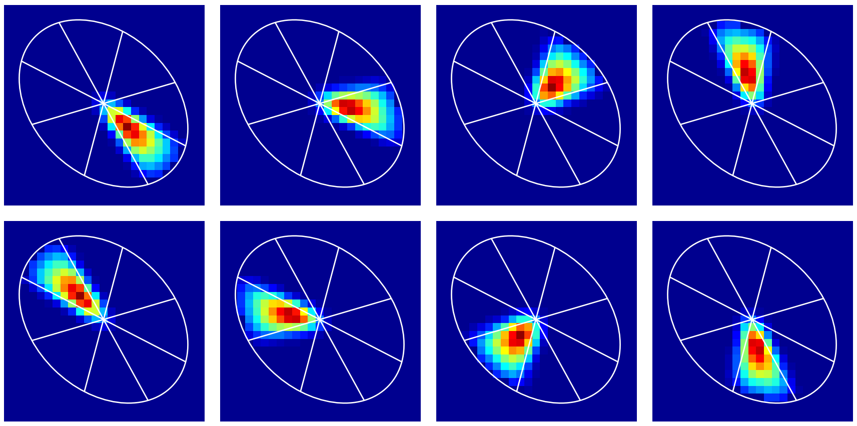


Figure 5.16: *Filter kernel used by the anisotropic Kuwahara filter ($N = 8$). The shape and orientation of the filter kernel is adjusted to the local structure of the input.*

maps points from the ellipse to a disc of radius 0.5, as illustrated in Figure 5.15. This mapping allows us to define a set of weighting function

$$\begin{aligned} w_k(x) &= h_k\big(SR_{-\varphi}(x)\big) \\ &\approx H_0\big(R_{-2\pi k/N}SR_{-\varphi}(x)\big) \end{aligned} \tag{5.33}$$

over the ellipse by pulling back the weighting functions h_k defined over the circle. In Figure 5.16, an ellipse and the corresponding weighting functions w_k are shown. As in the case of the generalized Kuwahara filter, the weighting functions are used to compute the weighted means and weighted variances of the sectors, and the filter response of the anisotropic Kuwahara filter is defined by Equation (5.14).

An implementation of the anisotropic Kuwahara filter may be obtained by a straightforward modification of the implementation of the generalized Kuwahara filter shown in Listing 5.1. All that must be done is replacing Equation (5.21) by Equation (5.33), or more specifically, the computation of the variable v has to be updated. Moreover, the outer loop must be changed to iterate over all pixels contained in the bounding box of the ellipse. Figure 5.17 shows the results of a benchmark comparing different implementations of the anisotropic Kuwahara filter. As can be seen, the straightforward implementation has comparatively poor performance for $N = 4$ (`tex4`) as well as $N = 8$ (`tex8`) sectors. In fact, this is not surprising, since for every processed pixels in the filter neighborhood, N texture fetches have to be performed. The number of texture fetches can be reduced by packing four consecutive weighting functions $h_0, \dots, h_3$ into an RGBA texture map. In this case, for $N = 4$ only a single texture fetch and for $N = 8$ only two texture fetches are necessary. This texture map, H_{0123}, is constructed by sampling the weighting functions h_0,...,h_3, as explained in Section 5.3.3, and storing the result in the corresponding color channel. Let x be the current relative pixel position, and let $v = SR_{-\varphi}(x)$, then Equation (5.33) generalizes to

$$\big(w_0(x), \dots, w_3(x)\big) \approx H_{0123}\big(v\big) \tag{5.34}$$

and

$$\big(w_4(x), \dots, w_7(x)\big) \approx H_{0123}\big(R_{-\pi}(v)\big) = H_{0123}\big(-v\big)\,, \tag{5.35}$$

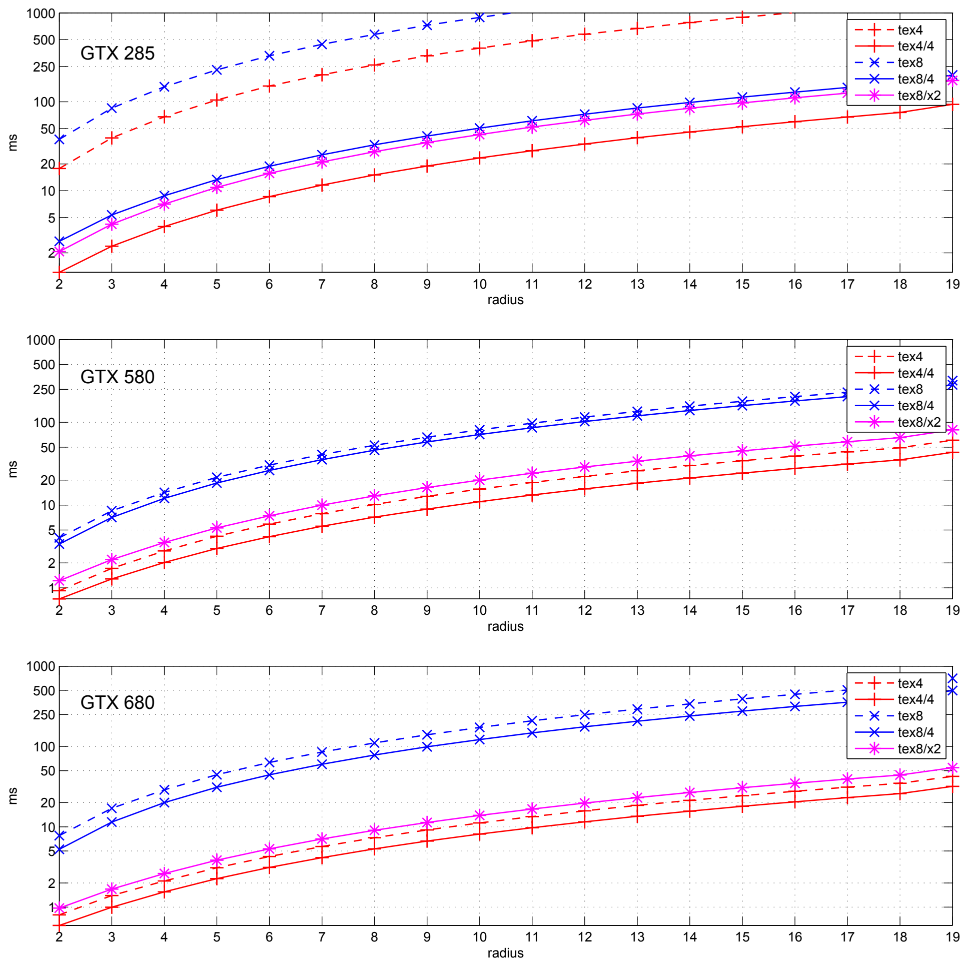


Figure 5.17: *Benchmark comparing different implementations of the anisotropic Kuwahara filter for different generations of NVidia GPUs.* `tex4`, `tex8`*: Unoptimized texture-based implementation using a grayscale image as texture.* `tex4/4`, `tex8/4`*: Texture-based implementation using an RGBA image as texture.* `tex8/x2` *Optimized texture-based implementation using two passes over the neighborhood.*

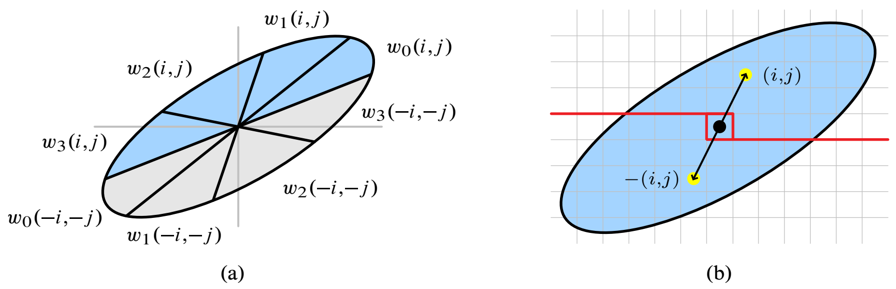


Figure 5.18: *(a) For $N = 8$ the values of the weighting functions can be fetched using two RGBA texture lookups. (b) Symmetry about the origin of an ellipse.*

```
for (int j = 0; j <= max_y; ++j) {
    for (int i = -max_x; i <= max_x; ++i) {
        if ((j !=0) || (i > 0)) {
            float2 v = make_float2( SR.x * i + SR.y * j, SR.z * i + SR.w * j );

            float dot_v = dot(v,v);
            if (dot_v <= 0.25f) {
                float3 c0 = make_float3(tex2D(texSRC, ix + i, iy + j));
                float3 c1 = make_float3(tex2D(texSRC, ix - i, iy - j));

                float3 cc0 = c0 * c0;
                float3 cc1 = c1 * c1;

                float4 tmp0 = tex2D(texH0123, v.x, v.y);
                float4 tmp1 = tex2D(texH0123, -v.x, -v.y);
                float const wx[8] = { tmp0.x, tmp0.y, tmp0.z, tmp0.w,
                                      tmp1.x, tmp1.y, tmp1.z, tmp1.w };

                for (int k = 0; k < 8; ++k) {
                    m[k] += wx[k] * c0  + wx[(k+4)&7] * c1;
                    s[k] += wx[k] * cc0 + wx[(k+4)&7] * cc1;
                    w[k] += wx[k]       + wx[(k+4)&7];
                }
            }
        }
    }
}
```

Listing 5.2: *Excerpt of the optimized implementation of the anisotropic Kuwahara filter for $N = 8$, using an RGBA texture map and exploiting the point symmetry of the elliptic filter shape.*

since a rotation of v by -180 degrees is equivalent to negating v (Figure 5.18(a)). The number of texture fetches can be further reduced by exploiting the point symmetry of the elliptic filter shape as illustrated in Figure 5.18(b). This allows for processing two pixels for each fetch of weights. Notice that the filter origin must be handled explicitly. Listing 5.2 shows the implementation of the optimizations discussed so far. As can be seen in Figure 5.17, the optimizations, denoted as `tex4/4` ($N = 4$) and `tex8/4` ($N = 8$), result in a significant performance improvement in case of the GTX 285.

In case of the GTX 580 and GTX 680, performance improvements are achieved as well, but for these two cards based on the Fermi and Kepler GPU architectures, respectively, the performance for $N = 8$ sectors is still comparatively poor. The reason for the poor

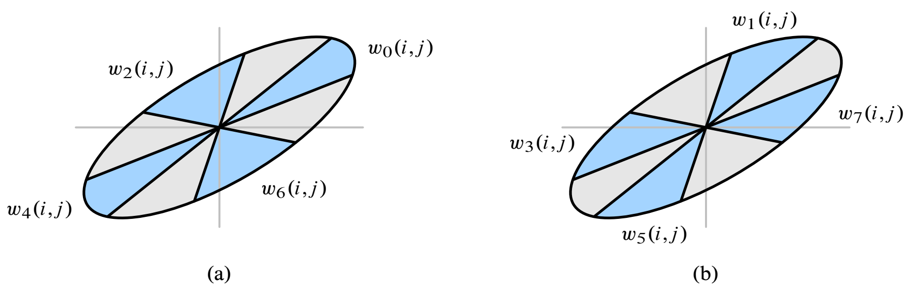


Figure 5.19: *(a) Even sectors. (b) Odd sectors.*

```
float4 SR = 0.5f * make_float4(t.x/a, t.y/a, -t.y/b, t.x/b);
float3 o = make_float3(0);
float ow = 0;
for (int rr = 0; rr < 2; ++rr) {
    float3 m[4];
    float3 s[4];
    float w[4];
    {
        float3 c = make_float3(tex2D(texSRC, ix, iy));
        float wx = tex2D(texH0246, 0, 0).x;
        for (int k = 0; k < 4; ++k) {
            m[k] =  c * wx;
            s[k] = c * c * wx;
            w[k] = wx;
        }
    }
    for (int j = 0; j <= max_y; ++j) {
        for (int i = -max_x; i <= max_x; ++i) {
            if ((j !=0) || (i > 0)) {
                float2 v = make_float2( SR.x * i + SR.y * j, SR.z * i + SR.w * j );

                const float dot_v = dot(v,v);
                if (dot_v <= 0.25f) {
                    const float4 tmp = tex2D(texH0246, v.x, v.y);
                    float const wx[4] = { tmp.x, tmp.y, tmp.z, tmp.w };

                    float3 c = make_float3(tex2D(texSRC, ix + i, iy + j));
                    float3 cc = c * c;
                    for (int k = 0; k < 4; ++k) {
                        const float wk = wx[k];
                        m[k] += wk * c;
                        s[k] += wk * cc;
                        w[k] += wk;
                    }

                    c = make_float3(tex2D(texSRC, ix - i, iy - j));
                    cc = c * c;
                    for (int k = 0; k < 4; ++k) {
                        const float wk = wx[(k + 2) & 3];
                        m[k] += wk * c;
                        s[k] += wk * cc;
                        w[k] += wk;
                    }
                }
            }
        }
    }
    for (int k = 0; k < 4; ++k ) {
        m[k] /= w[k];
        s[k] = fabs(s[k] / w[k] - m[k] * m[k]);
        float sigma2 = fmaxf(threshold, sqrtf(sum(s[k])));
        float alpha_k = __powf(sigma2, -q);
        o += m[k] * alpha_k;
        ow += alpha_k;
    }
    SR = CUDART_SQRT_HALF_F * make_float4( SR.x - SR.z, SR.y - SR.w,
                                           SR.x + SR.z, SR.y + SR.w );
}
dst.write(ix, iy, o / ow);
```

Listing 5.3: *Excerpt of the optimized implementation of the anisotropic Kuwahara filter for* $N = 8$.

performance becomes apparent when inspecting the verbose output of CUDA's NVCC compiler, which reports a large amount of registers spilled to local memory. The Fermi and Kepler GPU architectures restrict the maximum number of registers available to a thread to 63. While the kernel for $N = 4$ sectors meets these restrictions, the kernel for $N = 8$ does not, resulting in registers temporarily spilled to local memory. The reason for the large number of used registers can be observed in Listing 5.1 line 12–14. The storage for accumulating the means, variances, and weights of the different sectors requires $8 \times (3 + 3 + 1) = 56$ floats. Fortunately, the filter result defined by Equation (5.14) may be computed incrementally, since the sector weights α_k can be computed independently for each sector. This strategy is used in the implementation shown in Listing 5.3, where two passes are made over the local neighborhood. In order to exploit the point symmetry property of the ellipse, the sectors are not processed consecutively. Instead, the first pass processes the even sectors and the second pass processes the odd sectors (Figure 5.19). To this end, the even weighting functions h_0, h_2, h_4, and h_6 are sampled into an RGBA texture map. The even weighting functions are obtained through a rotation by -45 degrees. As demonstrated in Figure 5.17, this implementation, denoted by `tex8/x2`, achieves not only significant performance improvements for the Fermi and Kepler architectures, but also performs best for the GTX 285.

5.4.3 Polynomial Weighting Functions

Since the weighting functions w_k are adapted to the local image structure, they are generally different per pixel. Therefore, the calculation has to be carried out in the spatial domain. Hence, an efficient implementation depends on the possibility to quickly evaluate K_i. The direct computation of K_i would be absurd, since the computation requires convolution. In the previous section h_0, h_1, h_2 and h_3 were sampled into an RGBA texture map.

In this section, instead of discretizing or approximating the h_k, we aim for replacing them with other functions that are simpler to compute. By construction, the sum $\sum_k h_k$ is equal to the Gaussian function G_{σ_r}. Hence, the h_k define a smooth partition of the Gaussian function G_{σ_r}. The new weighting functions also have this property and their construction is illustrated in Figure 5.20. Basis for the construction is the polynomial $(x_1 + \zeta) - \eta x_2^2$ that is shown in Figure 5.20(a). The red parabola shows the zero-crossing of this polynomial. By clamping negative values to zero, one gets a function that is non-zero inside and zero outside the parabola. By taking the square, it is ensured that the transition at the zero-crossing is smooth (Figure 5.20(c)):

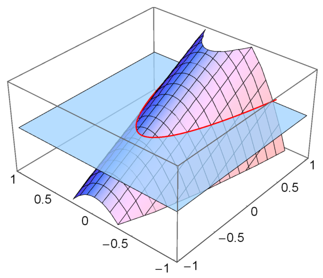
$$p_0(x) = \begin{cases} \left[(x_1 + \zeta) - \eta x_2^2\right]^2 & \text{if } x_1 \geq \eta x_2^2 - \zeta \\ 0 & \text{otherwise} \end{cases} \tag{5.36}$$

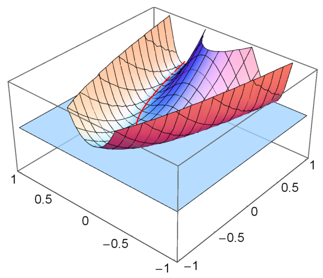
The functions for the other sectors are defined as corresponding rotations of p_0:

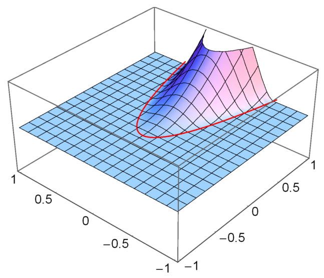
$$p_k = p_0 \circ R_{-2\pi k/N} \tag{5.37}$$

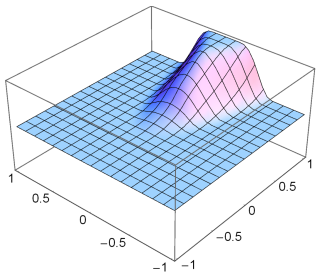
By normalizing p_k and multiplying with G_{σ_r}, we get weighting functions defined over the unit disc:

$$\tilde{h}_k(x) = \frac{p_k(x)}{\sum_{i=0}^{N-1} p_i(x)} \cdot G_{\sigma_r}(x) \tag{5.38}$$


(a)


(b)


(c)


(d)

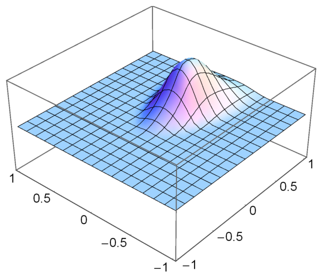
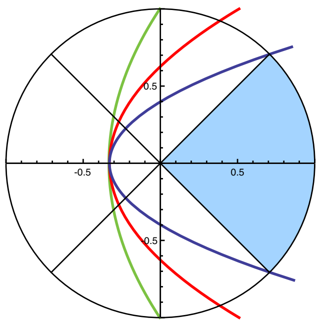

(e)

Figure 5.20: *Construction of the proposed weighting functions: (a) Polynomial* $(x_1 + \zeta) - \eta x_2^2$*. (b) Squared polynomial* $[(x_1 + \zeta) - \eta x_2^2]^2$*. (c) Clamped polynomial* p_0*. (d) Normalized* p_0*. (e) New weighting function* $\widetilde{h}_0$*.*

As desired, the sum $\sum_k \widetilde{h}_k$ is equal to G_{σ_r} by construction, and the new weighting functions $\widetilde{w}_k$ are now defined as in the previous section by:

$$\widetilde{w}_k = \widetilde{h}_k \circ SR_{-\varphi} \tag{5.39}$$

The definition of $\tilde{h}_0$ includes two parameters ζ and η. The parameter ζ controls how much the different weighting functions overlap at the filter origin. For the anisotropic Kuwahara filter to work as expected, it is required that all weighting functions overlap at their boundaries, especially at the filter origin. Since the anisotropy A lies in the interval $[0, 1]$, it follows from Equation (5.22) that, for $\alpha = 1$, the minor radius of the ellipse will not be


(a) $N = 4$

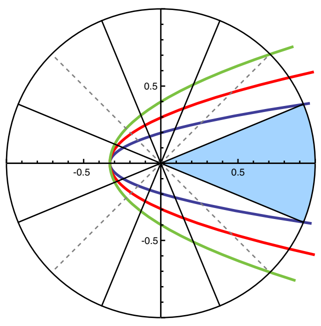

(b) $N = 8$

Figure 5.21: *Zero-crossings of the polynomial* $(x_1 + \zeta) - \eta x_2^2$ *for* $\zeta = \frac{1}{3}$ *and* $\eta = \eta(\zeta, \frac{\pi}{N})$ *(green),* $\eta = \eta(\zeta, \frac{3\pi}{4N})$ *(red), and* $\eta = \eta(\zeta, \frac{2\pi}{N})$ *(blue).*

```
for (int j = 0; j <= max_y; ++j) {
    for (int i = -max_x; i <= max_x; ++i) {
        if ((j !=0) || (i > 0)) {
            float2 v = make_float2(SR.x * i + SR.z * j, SR.y * i + SR.w * j);

            float dot_v = dot(v,v);
            if (dot_v <= 1.0f) {
                float3 c0 = src_(ix + i, iy + j);
                float3 c1 = src_(ix - i, iy - j);

                float3 cc0 = c0 * c0;
                float3 cc1 = c1 * c1;

                float sum = 0;
                float wx[8];
                float z, xx, yy;

                xx = zeta - eta * v.x * v.x;
                yy = zeta - eta * v.y * v.y;
                z = fmaxf(0,  v.y + xx);  sum += wx[0] = z * z;
                z = fmaxf(0, -v.x + yy);  sum += wx[2] = z * z;
                z = fmaxf(0, -v.y + xx);  sum += wx[4] = z * z;
                z = fmaxf(0,  v.x + yy);  sum += wx[6] = z * z;

                v = HALF_SQRT2 * make_float2( v.x - v.y, v.x + v.y );

                xx = zeta - eta * v.x * v.x;
                yy = zeta - eta * v.y * v.y;
                z = fmaxf(0,  v.y + xx);  sum += wx[1] = z * z;
                z = fmaxf(0, -v.x + yy);  sum += wx[3] = z * z;
                z = fmaxf(0, -v.y + xx);  sum += wx[5] = z * z;
                z = fmaxf(0,  v.x + yy);  sum += wx[7] = z * z;

                const float g = __expf(-3.125f * dot_v) / sum;

                for (int k0 = 0; k0 < 8; ++k0) {
                    const float wk = wx[k0] * g;
                    const int k1 = (k0 + 4) & 7;

                    m[k0] += c0 * wk;
                    s[k0] += cc0 * wk;
                    w[k0] += wk;

                    m[k1] += c1 * wk;
                    s[k1] += cc1 * wk;
                    w[k1] += wk;
                }
            }
        }
    }
}
```

Listing 5.4: *Excerpt of the implementation of the anisotropic Kuwahara filter based on polynomial weighting functions for $N = 8$ sectors.*

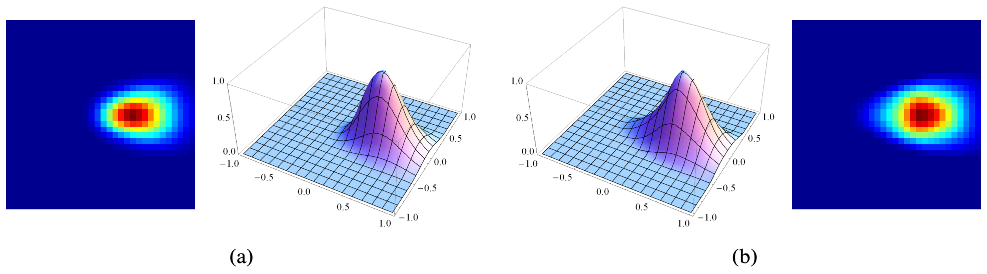


Figure 5.22: *(a) Weighting function* $(\chi_0 * G_\rho) \cdot G_\sigma$ *based on convolution. (b) New weighting function based on the polynomial* $[(x+\zeta) - \eta y^2]^2$.

smaller than half of the radius. Hence

$$\zeta = \frac{2}{r} \tag{5.40}$$

is a reasonable choice that ensures that all sectors overlap at the filter origin. The parameter η controls how much the sectors overlap at their sides. Different choices for η are shown in Figure 5.21. A convenient way to parameterize η is by the intersection of the unit circle with the zero-crossing of the polynomial $(x_1 + \zeta) - \eta x_2^2$, where the intersection point is represented in polar coordinates $(\cos\gamma, \sin\gamma)$. Solving

$$(x_1 + \zeta) - \eta x_2^2 = (\cos\gamma + \zeta) - \eta \sin^2\gamma^2 = 0 \tag{5.41}$$

for η then yields:

$$\eta(\zeta, \gamma) = \frac{\zeta + \cos\gamma}{\sin^2\gamma} \tag{5.42}$$

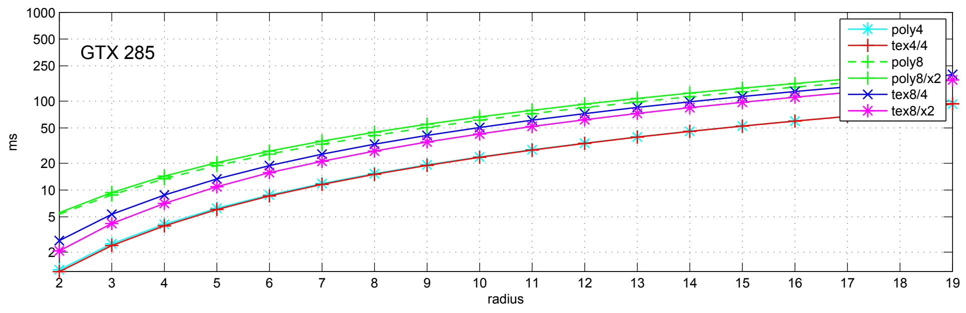
In order for the $\widetilde{h}_k$ to be well-defined on the unit disc, it is necessary that the sum $\sum_k \widetilde{h}_k$ is non-zero for every point of the unit disc. This can be achieved by requiring $\gamma \geq \frac{\pi}{N}$ (blue plot of Figure 5.21). Conversely, it is undesirable that more than two sectors overlap on one side. An approximate bound for this is $\gamma \leq \frac{2\pi}{N}$ (green plot of Figure 5.21). Hence, reasonable choices for η are in the range:

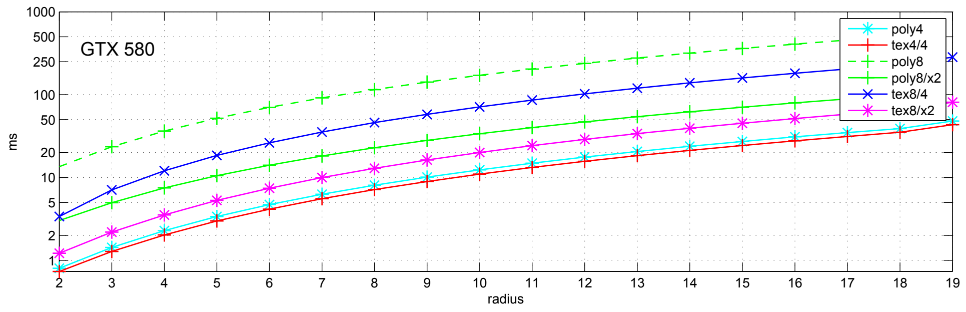
$$\eta_{\min}(\zeta) = \eta\left(\zeta, \tfrac{2\pi}{N}\right) \;\leq\; \eta \;\leq\; \eta\left(\zeta, \tfrac{\pi}{N}\right) = \eta_{\max}(\zeta) \tag{5.43}$$

In this work, all examples use $\zeta = \frac{2}{r}$ and

$$\eta\left(\tfrac{1}{3}, \tfrac{3\pi}{2N}\right) \approx \begin{cases} 0.84 & \text{if } N = 4 \\ 3.77 & \text{if } N = 8 \end{cases}. \tag{5.44}$$

Figure 5.22 demonstrates that the polynomial weighting functions closely approximate the convolution-based weighting functions. Listing 5.4 shows how the polynomial weighting functions may be implemented efficiently for $N = 8$. Here, it is important to use the `max` function. On GPUs this function is generally available as intrinsic function and much faster, since no branching is performed. For $N = 8$ the vector `v` has to be rotated by $\pi/4$. Since a rotation by multiple of $\pi/2$ can be performed by swapping and negating coordinates, the computation is performed in two stages. First, the weights are calculated for 0, $\pi/2$, π and $3/2\pi$. Second, a rotation by $\pi/4$ is performed and the weights calculated





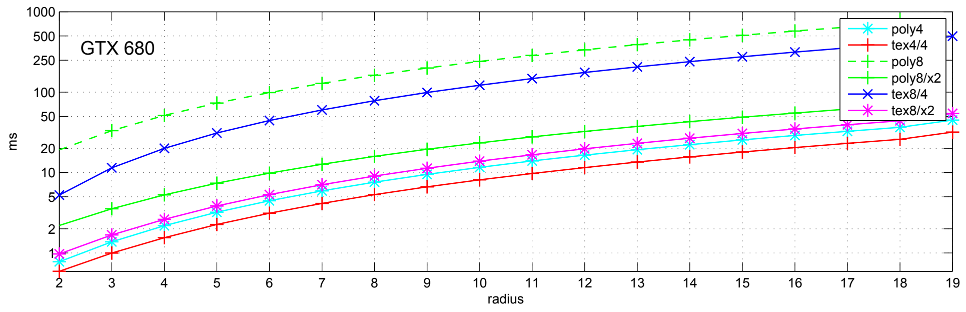
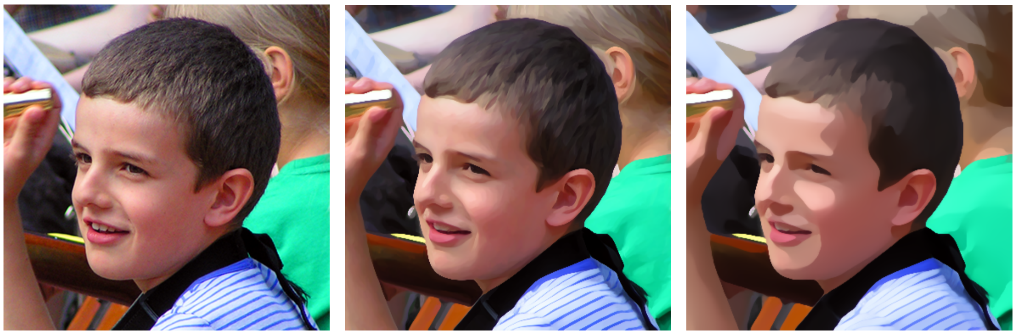


Figure 5.23: *Benchmark comparing different implementations of the anisotropic Kuwahara filter for different generations of NVidia GPUs.* `poly4`, `poly8`: *Polynomial weighting functions with* 4 *or* 8 *sectors implemented as in Listing 5.4.* `poly8/x2`: *Optimized polynomial weighting function implementation, similar to Listing 5.3.* `tex4/4`, `tex8/4`, `tex8/x2` *as in Figure 5.17.*

for $\pi/4$, $3/4\pi$, $5/4\pi$ and $7/4\pi$. For the sake of simplicity, the implementation in Listing 5.4 performs a single pass over the filter neighborhood, resulting in registers being spilled to local memory as in case of the texture-based implementation. Again, better results are achieved by an implementation similar to those in Listing 5.3. Figure 5.23 compares the execution times of different implementations of the new approach with the texture-based implementation. For all GPU architectures, the optimized texture-based implementation outperforms the optimized implementation based on the polynomial weighting functions. Nonetheless, the polynomial weighting functions remain interesting for platforms where no high-performance texture hardware is available (e.g., texture support is optional in OpenCL), or where auxiliary textures are difficult or inconvenient to use, such as the Adobe Pixel Bender language.



(a) *Original image* (b) *Anisotropic Kuwahara filter* (c) *Proposed method*

Figure 5.24: *Example comparing the multi-scale approach with the single-scale approach.*

5.5 Multi-scale Processing

The level of abstraction achievable with the generalized and the anisotropic Kuwahara filter is limited by the filter radius. Simply increasing the filter radius is typically not a solution, as it often results in artifacts. Another possibility would be to control the radius adaptively per pixel depending on the local neighborhood, but the computational cost would be very high, as the filter depends quadratically on the radius. In this section, a multi-scale approach, where the anisotropic Kuwahara filter is applied at multiple scales, will be presented. The computations are carried out on an image pyramid, and processing is performed in a coarse-to-fine manner, with intermediate results being propagated up the pyramid. Figure 5.24 shows an example image processed with single-scale and multi-scale approaches.

A schematic overview of the multi-scale technique is shown in Figure 5.25. Processing starts with building an image pyramid of the input image. Next, the pyramid is processed from the coarsest to the finest level. On the coarsest level, the smoothed structure tensor and anisotropic Kuwahara filter are computed, and their results then upsampled to the next finer level. Based on an approximation of the local variances, the upsampled result of the anisotropic Kuwahara filter is then merged with the image data of the current pyramid level. From this merged result, the smoothed structure tensor is calculated, and based on their anisotropy measure, the smoothed structure tensor and the upsampled structure tensor from the previous level are merged. Using the merged structure tensor, the anisotropic Kuwahara filter is applied to the merged image data. Until the finest level of the pyramid is reached, the result of the anisotropic Kuwahara filter and the merged structure tensor are upsampled and the process is repeated for the next finer level. The next sections discuss the different parts of the algorithm in detail.

5.5.1 Pyramid Construction

The pyramid construction is performed using resampling convolution [Sch92]. A downsampling factor of 2 has been used for all experiments. Motivated by scale-space theory [Lin94b] and the classical approach by Burt and Adelson [BA83] for building a pyramid, initially a windowed Gaussian was used as resampling filter. But the Gaussian filter tends to remove

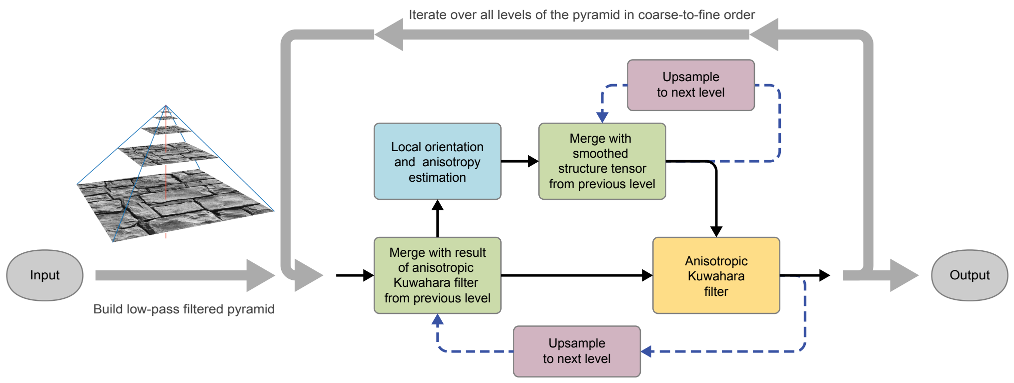

Iterate over all levels of the pyramid in coarse-to-fine order
Upsample to next level
Local orientation and anisotropy estimation
Merge with smoothed structure tensor from previous level
Input
Build low-pass filtered pyramid
Merge with result of anisotropic Kuwahara filter from previous level
Anisotropic Kuwahara filter
Output
Upsample to next level

Figure 5.25: *Schematic overview of the multi-scale anisotropic Kuwahara filter.*

too much low frequencies, creating a less sharp looking result. Therefore, several popular resampling filters, including bilinear, cubic, Catmull-Rom, Mitchell-Netravali and Lanczos, were considered. Among these, the Lanczos3 filter [Bli89] was found to give the best result for most examples. As pointed out by Blinn, the Lanczos3 filter keeps low frequencies and rejects high-frequencies better than most other well-known filter. This makes it well-suited for the multi-scale approach presented here. However, notice that using the Lanczos3 filter might create or enhance local extrema and violates the causality axiom of scale-space theory [Lin94b]. Nonetheless, it works very well in this particular case.

5.5.2 Filtering and Propagation

The multi-scale filtering is performed similar to the multi-scale local structure estimation (2.8). The pyramid is processed in coarse-to-fine order. For the coarsest level, the anisotropic Kuwahara filter is computed as usual. For the other levels, the upsampled filtering result from the previous level and the image data of the current level are merged using a linear combination:

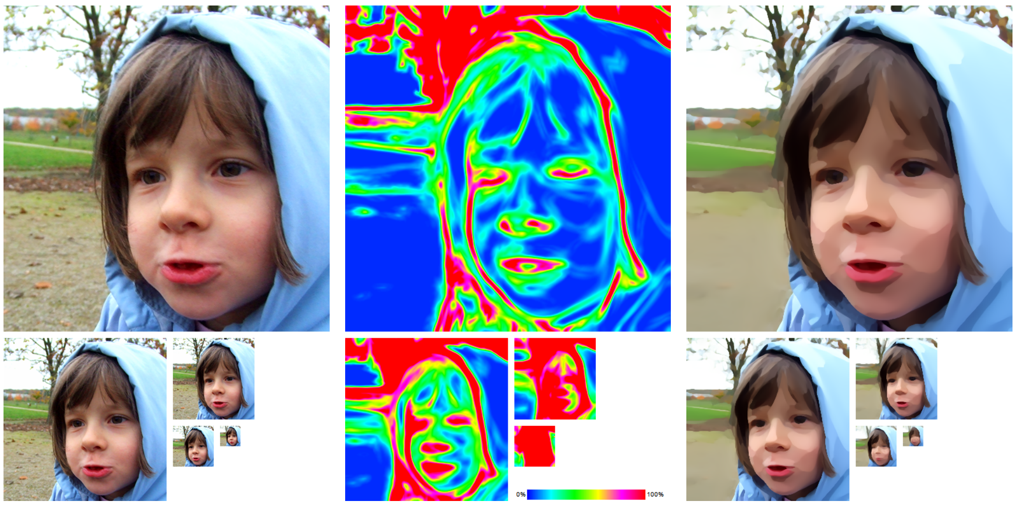
$$\tilde{f}^k = \beta^k f^k + (1 - \beta^k)\, \bar{f}^{k+1}$$

Here, $\tilde{f}^k$ denotes the merged result, f^k is the original image data of the current level and $\bar{f}^{k+1}$ is the upsampled filtering result from the previous level. The merged result is now used to calculate the structure tensor of the current level, which is merged with the upsampled structure tensor from the previous level. Finally, the merged result is processed using the single-scale anisotropic Kuwahara filter. This process is then repeated until the finest level of the pyramid is reached. The weighting factor β^k is defined based on the standard deviation s^k by:

$$\beta^k = \operatorname{clamp}\left(s^k \cdot p_s\,(p_d)^k - \tau_v,\ 0, 1\right)$$

Additional user control is provided through the parameters p_s and p_d. The parameter p_s applies to all level in a uniform way, while the parameter p_d takes the scale into account. In order to account for small standard deviations due to noise, the threshold parameter τ_v is provided. Typical values for these parameters are $p_s = 0.5$, $p_d = 1.25$ and $\tau_v = 0.1$.

The calculation of the standard deviation is computationally expensive and therefore an approximation is used. During the computation of the anisotropic Kuwahara filter,



(a) *Pyramid of original image data* (b) *Upsampled approximation to local standard deviation* (c) *Merged and filtered result for each level*

Figure 5.26: *Original image, approximation to standard deviation and output of the anisotropic Kuwahara filter for the different pyramid levels.*

the standard deviations s_i for each sector are computed. Since the sum of all weighting functions w_i is equivalent to a Gaussian, the sum of the thresholded standard deviations

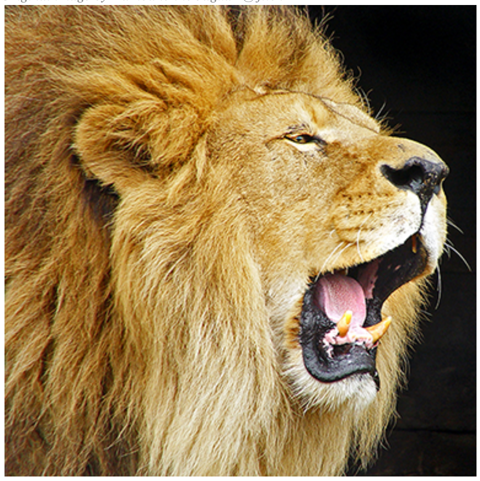
$$s_{\max} = \sum_{i=0}^{N-1} \max\left(\tau_w, \|s_i\|\right)$$

is an approximate for the local standard deviation that can be easily computed during the filtering at negligible computational cost. Since the standard deviation is required for the merging process before the actual computation of the anisotropic filter, the approximate standard deviation $s_{\max}$ is stored during the filtering as an additional result. This can be done, for example, by storing it in the alpha channel of the filter result. At each level of the pyramid, the approximate standard deviation from the previous level is then upsampled and used to calculate the weighting factor β_k (Figure 5.26).

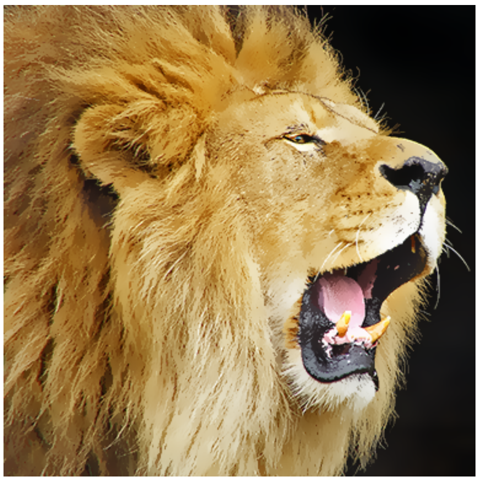
Upsampling is performed with bilinear interpolation. Typically, there is no benefit from using more sophisticated upsampling techniques. This might be surprising on the first glance. Especially, since one might expect that usage of a better upsampling filter should result in sharper color boundaries. However, for pixels close to edges, the approximate standard deviation is high, and therefore, the weight β^k is one. Thus, pixels close to edges will be overridden during the merging process.

5.6 Results

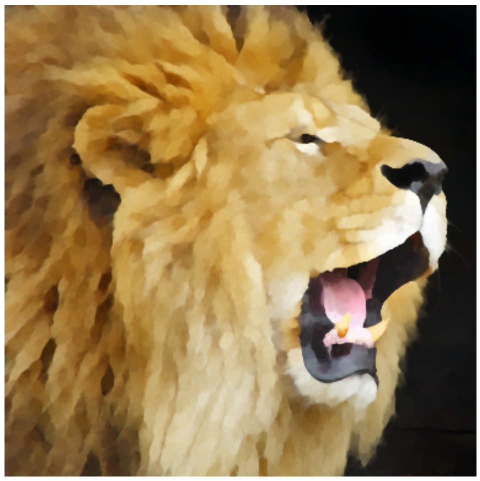
Figure 5.27 shows a comparison with the generalized Kuwahara filter by Papari et al. [PPC07]. Since this approach uses an isotropic static set of filter kernels, it fails to capture small directional image features. Moreover, shape boundaries are less preserved, because the filter kernels have predefined primary orientations. Due to the fixed isotropic filter



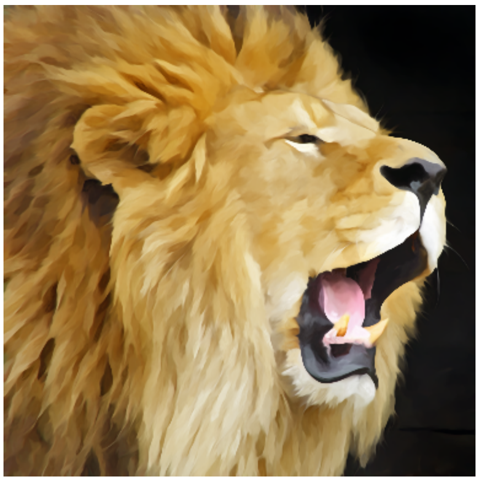
(a) *Original image*



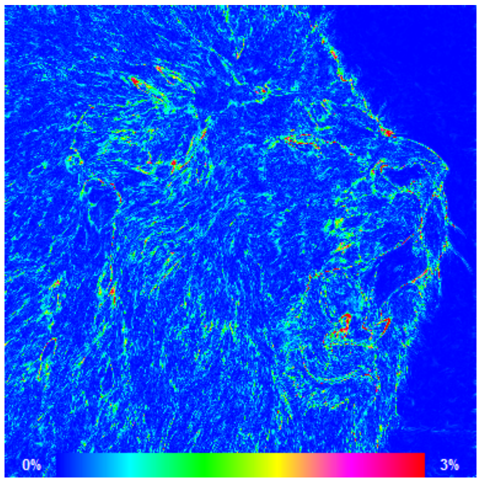
(b) *Bilateral filter*



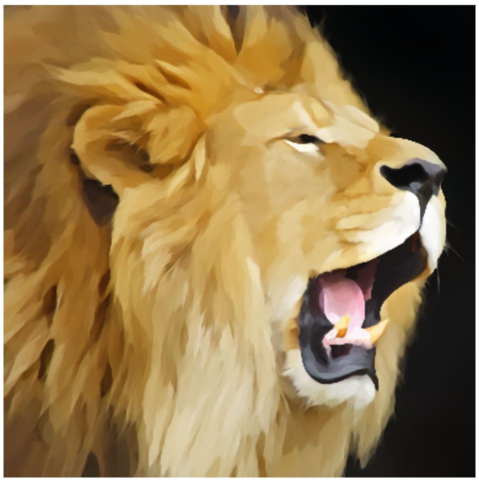
(c) *Generalized Kuwahara filter*



(d) *Single-scale anisotropic Kuwahara filter*



(e) *Difference of (d) to polynomial weighting functions (single-scale)*

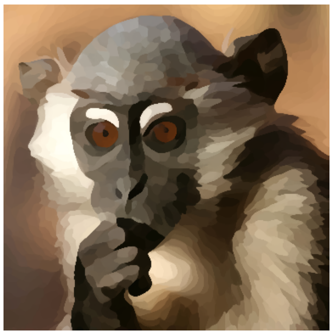


(f) *Multi-scale anisotropic Kuwahara filter*

Figure 5.27: *The anisotropic Kuwahara filter achieves a painterly look by preserving and enhancing directional image features.*

Original image by Keven Law @ flickr

(a) *Original image*


(b) *AKF (3 iterations)*

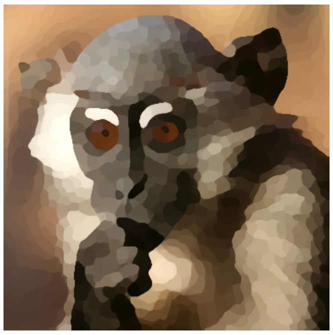
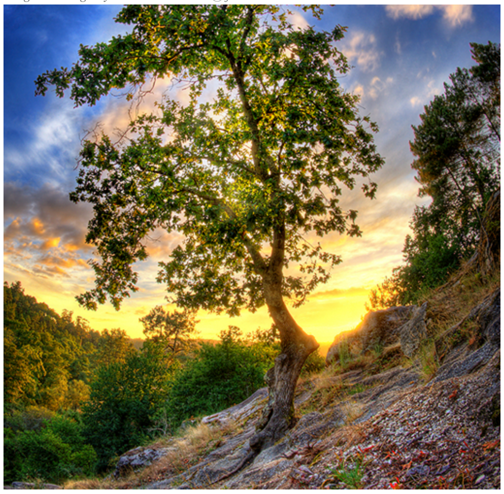
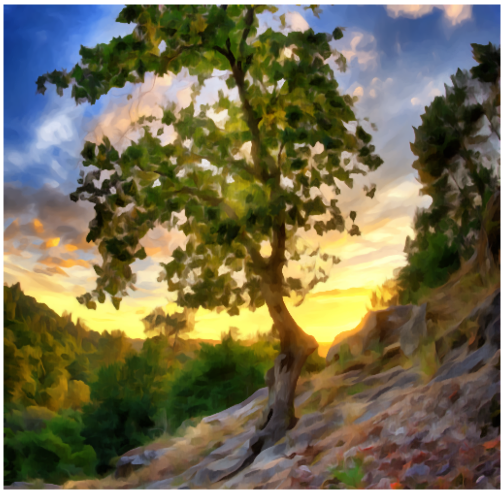
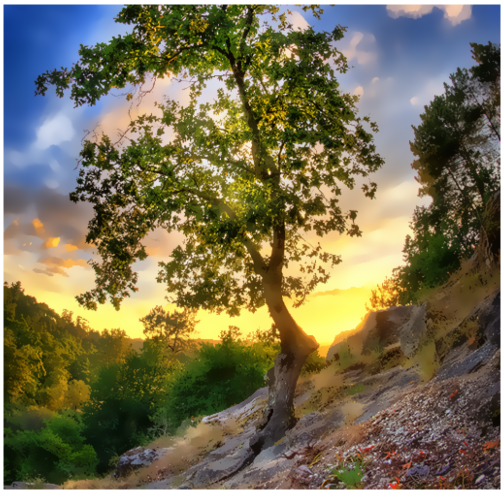

(c) *GKF (3 iterations)*

Figure 5.28: *The anisotropic Kuwahara filter creates, in contrast to the generalized Kuwahara filter, consistent color regions without showing clustering artifacts when applied iteratively.*

Original image by Paulo Brandão @ flickr


(a) *Original image*


(b) *Anisotropic Kuwahara filter*


(c) *Bilateral filter [TM98]*

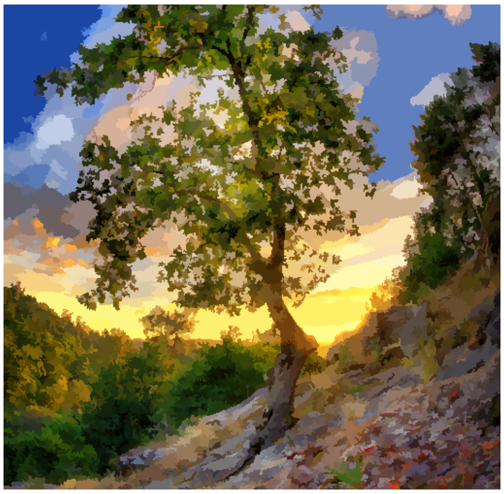
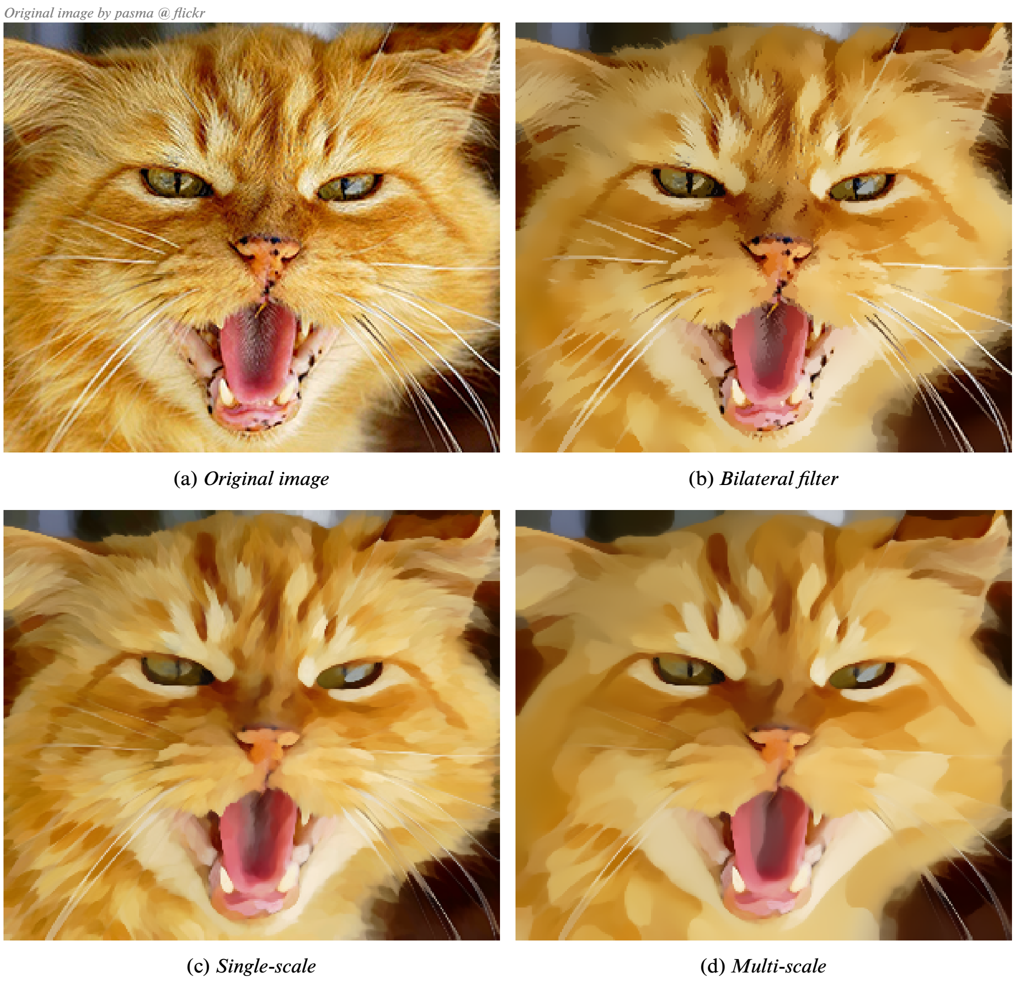

(d) *Mean shift segmentation [CM02]*

Figure 5.29: *Comparison with bilateral filter and mean shift segmentation.*



(a) *Original image* (b) *Bilateral filter*

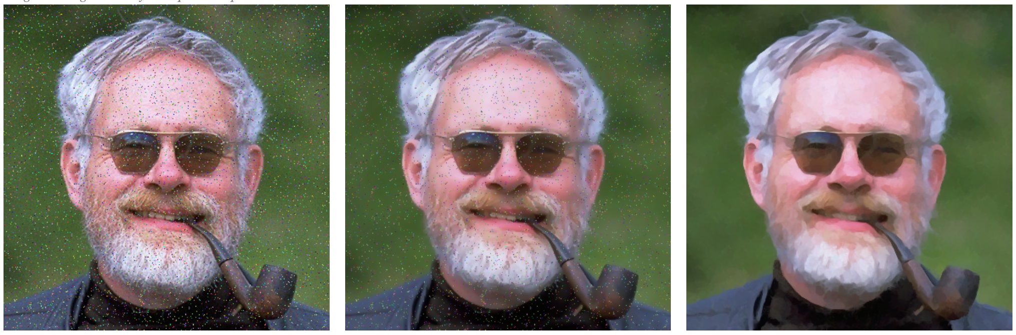
(c) *Single-scale* (d) *Multi-scale*

Figure 5.30: *Comparison of anisotropic Kuwahara filter with the bilateral filter.*

kernels, Papari et al.'s approach is prone to group pixels into clusters of circular shape. This becomes more problematic when the filter is applied iteratively to achieve further abstraction, as shown in Figure 5.28. By contrast, the anisotropic Kuwahara filter avoids this and creates feature-preserving color regions even after multiple applications. The results created by the implementation based on polynomial weighting functions are visually indistinguishable from the output of the anisotropic Kuwahara filter using the texture-based weighting functions. The same feature-preserving direction-enhancing look is created. Minor differences (Figure 5.27(e)) appear in high-contrast areas. This is not surprising, since both filters do not match exactly. As the primary aim of the filter is abstraction, these minor differences are irrelevant.

Figure 5.29 demonstrates the ability of the anisotropic Kuwahara filter to abstract high-contrast images. The bilateral filter is typically not effective in abstracting high-contrast details, and thus leaves many small and noisy regions untouched (see the ground). In case of mean shift segmentation, its density estimation in the multidimensional space results in rough region boundaries. The anisotropic Kuwahara filter, on the other hand, produces a clean and consistent look of abstraction across the image, without noisy pixels

Original image courtesy Philip Greenspun



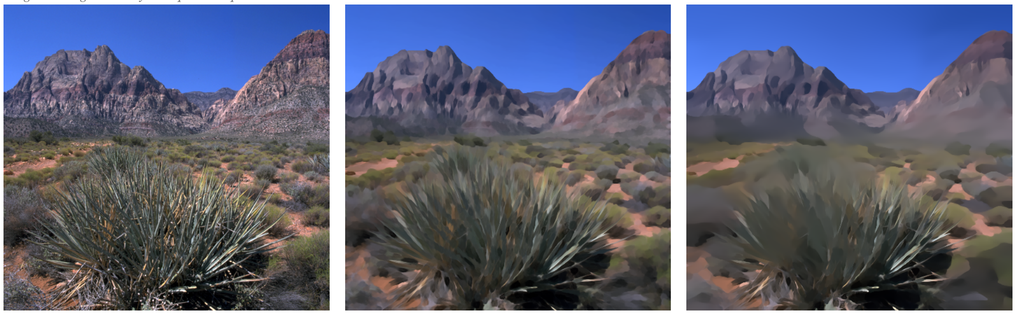
(a) *Original image* (b) *Flow-based bilateral filter* (c) *Proposed method*

Figure 5.31: *Application to an image with 2% Gaussian noise and 5% impulse noise.*

or rough boundaries. Figure 5.30 shows an example, where the bilateral filter smoothes away most of the interesting low-contrast information (the cat's fur) and leaves only high-contrast details (even the black spots around mouth and nose), whereas a consistent and feature-enhancing abstraction all over is provided by the anisotropic Kuwahara filter. The single-scale anisotropic Kuwahara filter, on the other hand, provides a very consistent level of abstraction over the whole image. The multi-scale version provides a much stronger abstraction. Above the nose, there is slightly less abstraction, but the overall look is also consistent. However, some of the cat's whiskers are lost. Moreover, the outputs of the single-scale and multi-scale method look a little bit washed out. Figure 5.31 shows the vulnerability of the bilateral filter to high-contrast noise even more clearly. The input image here is artificially corrupted with Gaussian and impulse noise. While the anisotropic Kuwahara filter successfully restores the image and creates a visually pleasing output, the flow-based bilateral filter, which is known to perform better on noisy input than the full-kernel bilateral filter, clearly fails to remove high-contrast impulse noise.

Two problematic cases, where the multi-scale anisotropic Kuwahara filter fails to produce good looking results for the default parameters are shown in Figures 5.32 and 5.33. In the first case, parts of the image look blurred. Also, the abstraction of the rocks in the background is inconsistent. Moreover, parts above the plant are smoothed with the ground. By adjusting the parameters p_s and p_d, the blurring can be removed. But in this case the

Original image courtesy Philip Greenspun



(a) *Original image* (b) *Single-scale* (c) *Multi-scale*

Figure 5.32: *Example showing a problematic case, where the multi-scale approach creates an inconsistent level of abstraction.*

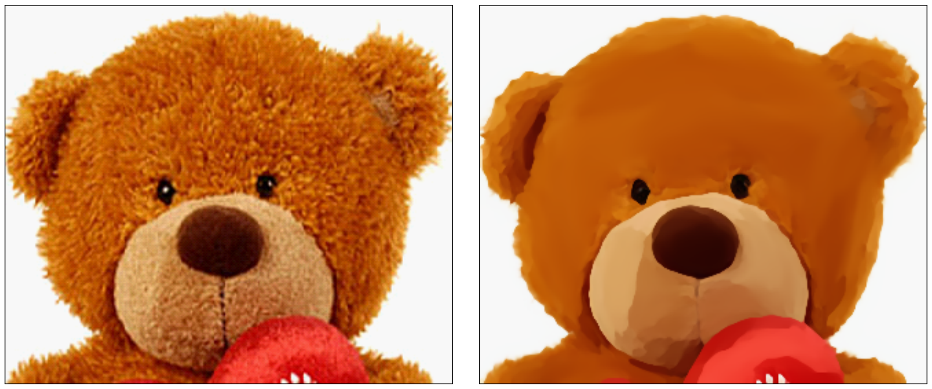

(a) *Original image* (b) *Multi-scale*

Figure 5.33: *The multi-scale anisotropic Kuwahara filter applied to an image with high-frequency texture.*

abstraction is also less strong. In the second case, the image is very difficult to abstract, due to its high-frequency texture. The multi-scale method performs decent. However, there are clearly noticeable artifacts near to color region boundaries, for example around the eyes and where the nose meets the face.

The single-scale and multi-scale anisotropic Kuwahara filter has been also applied to video abstraction and it was found that per-frame filtering alone provides excellent temporal coherence. Thus, no extra processing, such as motion estimation [Lit97; HE04] or adaptive color quantization is required. Providing a painterly look, it is an interesting alternative to video abstraction frameworks based on the bilateral filter, such as the ones discussed in Chapters 3 and 4. A limitation of the technique is that it is not suitable for creating a "rough and brushy" look as found in some oil paintings (e.g., "Starry Night" by Vincent van Gogh). To create such strong textured brush effects, a background paper texture or a directional cumulative alpha map [Her02] could be incorporated into the approach.

Chapter 6

Coherence-Enhancing Filtering

This chapter presents an image and video abstraction technique that places emphasis on enhancing the directional coherence of features. The most notable related work in this category is image abstraction and stylization based on PDEs, in particular, shape-simplifying image abstraction by Kang and Lee [KL08] and Weickert's coherence-enhancing shock filter [Wei03]. However, such PDE-based techniques may require a large number of iterations and tend to be unstable when used for video processing [Par08]. The technique presented in this chapter builds upon the idea of combining diffusion with shock filtering for image abstraction, but is, in a sense, contrary to that of Kang and Lee [KL08], which is outperformed in terms of speed, temporal coherence, and stability. Instead of simplifying the shape of the image features, the aim is to preserve the shape by using a curvature-preserving smoothing method that enhances coherence. More specifically, smoothing is performed in the direction of the smallest change, and sharpening in the orthogonal direction. Instead of modeling this process by a PDE and solving it, approximations that operate as local filters in a neighborhood of a pixel are used. Therefore, good abstraction results can be achieved in just a few iterations, making it possible to process images and video at real-time rates on a GPU. Moreover, the proposed method results in a much more stable algorithm, enabling temporally-coherent video processing. Compared to conventional abstraction approaches

Original images courtesy Tambako the Jaguar @ flickr and Philip Greenspun

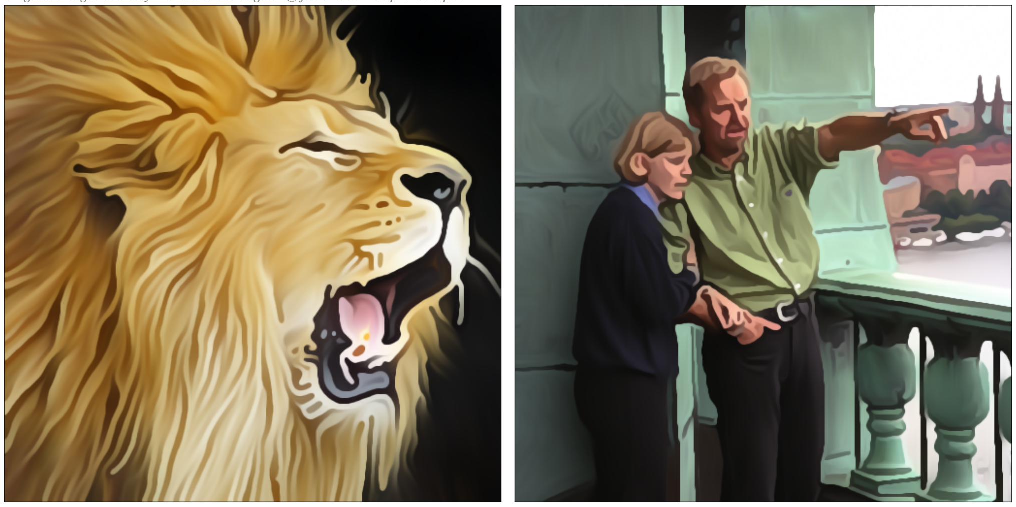
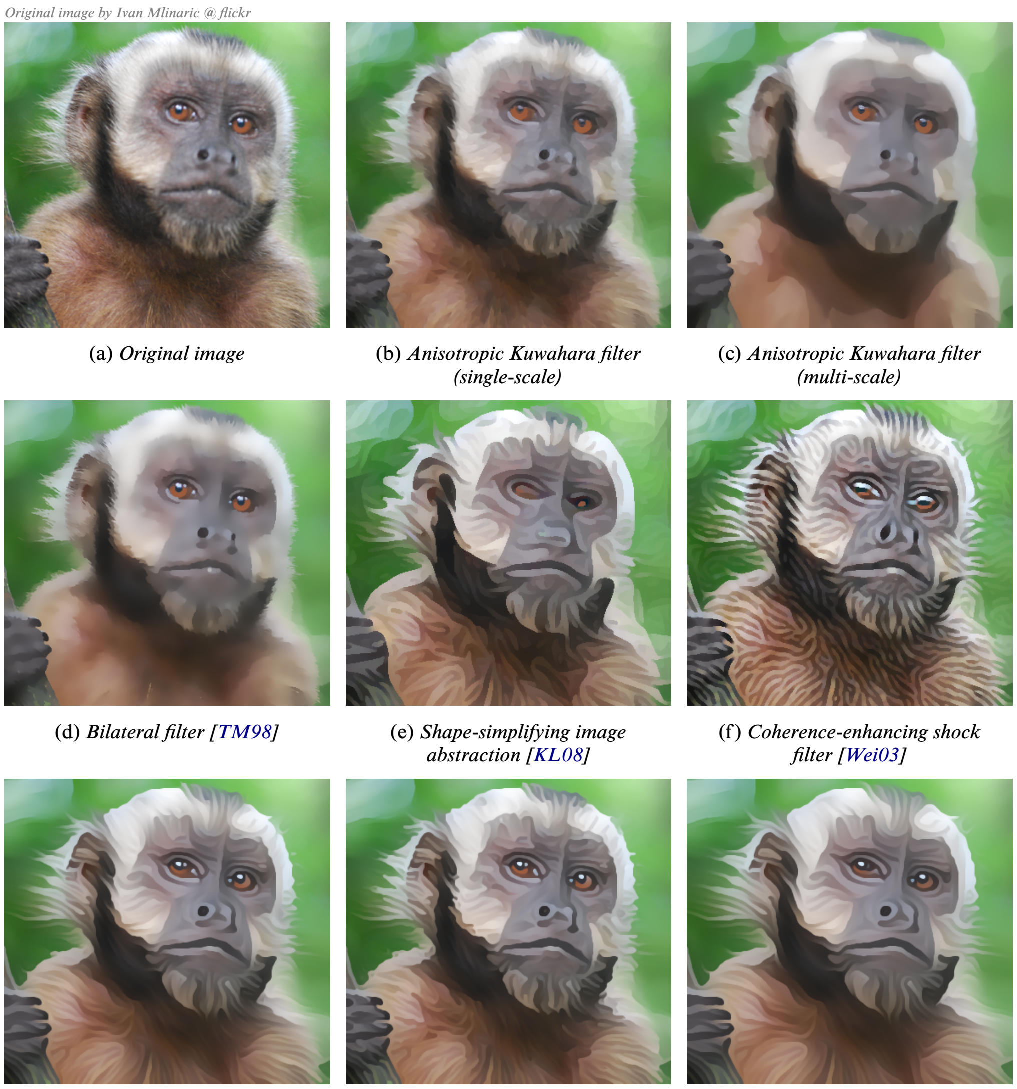


Figure 6.1: *Examples created by the technique described in this chapter.*



(a) *Original image*

(b) *Anisotropic Kuwahara filter (single-scale)*

(c) *Anisotropic Kuwahara filter (multi-scale)*

(d) *Bilateral filter [TM98]*

(e) *Shape-simplifying image abstraction [KL08]*

(f) *Coherence-enhancing shock filter [Wei03]*

(g) *Proposed method* $(N = 2)$

(h) *Proposed method* $(N = 5)$

(i) *Proposed method* $(N = 10)$

Figure 6.2: *Comparison of the presented approach with other popular image abstraction techniques. The bottom row shows results of the algorithm for different number of iterations* $(N = 2, 5, 10)$.

(Figure 6.2), such as the bilateral filter, the method provides a good balance between the enhancement of directional features and the smoothing of isotropic regions. As shown in Figure 6.1, the technique preserves and enhances directional features while increasing contrast, which helps to clarify boundaries and features.

A schematic overview of the presented technique is shown in Figure 6.3. Input is given by a grayscale or color image (RGB color space is used for all examples). The algorithm runs iteratively and stops after a user-defined number of iterations, controlling the strength of the abstraction. For each iteration, adaptive flow-guided smoothing (Figure 6.4(a)) and sharpening (Figure 6.4(b)) are performed. Both techniques require information about the local structure, which is obtained by an eigenvalue analysis of the smoothed structure tensor and computed twice for every iteration, once before the smoothing and again before the

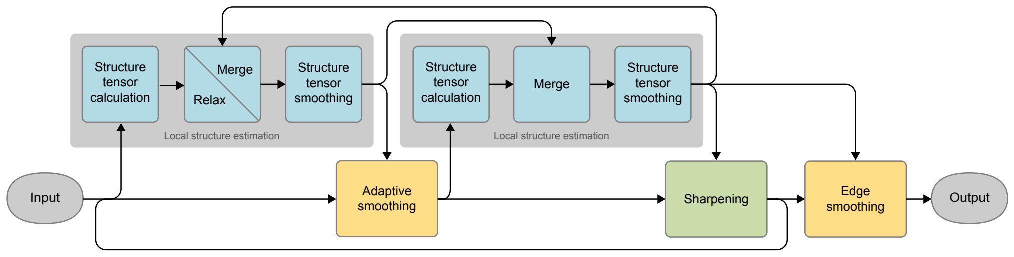

Structure tensor calculation
Merge
Relax
Structure tensor smoothing
Local structure estimation
Structure tensor calculation
Merge
Structure tensor smoothing
Local structure estimation
Input
Adaptive smoothing
Sharpening
Edge smoothing
Output

Figure 6.3: *Schematic overview of the proposed method.*

sharpening. With every iteration, the result becomes closer to a piecewise constant image, with large smooth or even flat image regions where no distinguished orientation is defined. Since having valid orientations defined for these regions is important for the stability of the algorithm, the structure tensor from the previous calculation is used. For the first iteration, where no result from a previous computation is available, a relaxation of the smoothed structure tensor is performed. As a final step, edges are smoothed by flow-guided smoothing with a small filter kernel (Figure 6.4(c)). The next section briefly reviews shape-simplifying image abstraction. The following sections then examine the different stages of the algorithm in detail.

6.1 Shape-simplifying Image Abstraction

Previously, Osher and Rudin [OR90], as well as Weickert [Wei03], made comments about the artistic look of shock filtered images, but the work of Kang and Lee [KL08] was the first to apply diffusion in combination with shock filtering for targeting IB-AR. They choosed mean curvature flow (MCF) as the diffusion method, which evolves isophote curves under curvature speed in normal direction, resulting in simplified isophote curves with regularized geometry. In contrast to other popular edge-preserving smoothing techniques, such as the bilateral or the Kuwahara filter, MCF smoothes not only irrelevant color variations while protecting region boundaries, but also simplifies the shape of those boundaries. The

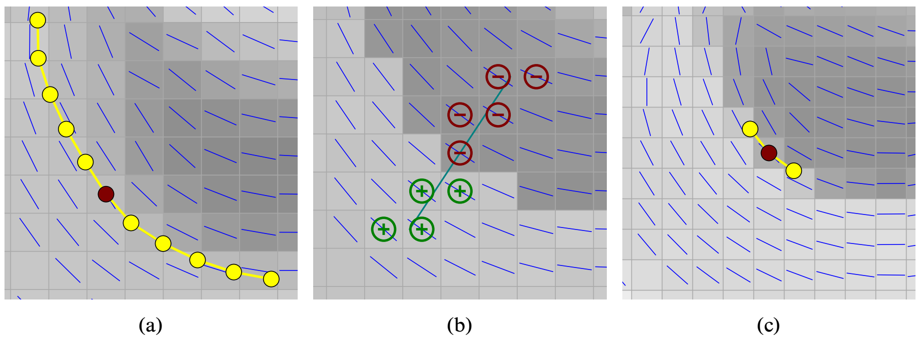
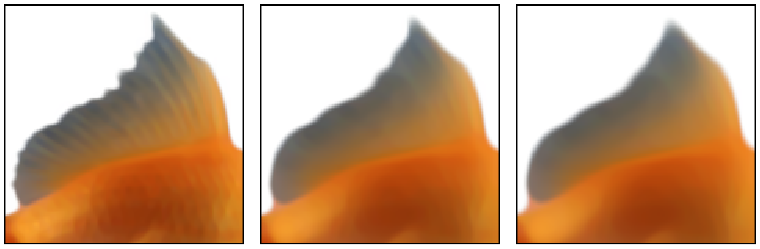
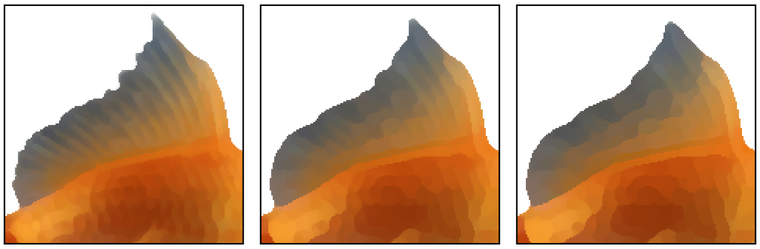


Figure 6.4: *Illustration of the different key techniques employed in the presented algorithm. (a) Adaptive flow-guided smoothing for simplification and abstraction. (b) Shock filtering to preserve and enhance sharp edge transitions. (c) Edge smoothing for anti-aliasing.*

(a) *Original image*


(b) *MCF (15 / 45 / 75 iterations)*


(c) *CMCF (15 / 45 / 75 iterations)*

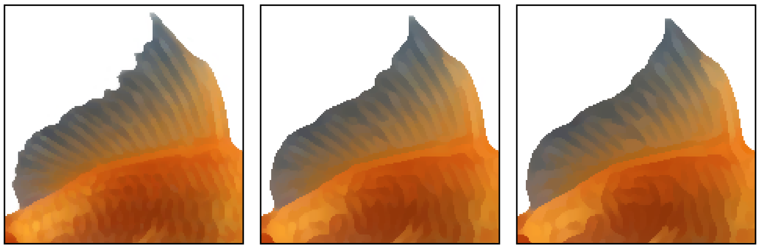

(d) *CMCF + Shock (15 / 45 / 75 iterations)*

Figure 6.5: *Comparison of mean curvature flow with/without shock filtering and constraint. (a) Original image. (b) Mean curvature flow. (c) Mean curvature flow with shock filtering after 15 iterations. (d) Constrained mean curvature flow with shock filtering after 15 iterations. In all cases, a time step of 0.25 was used.*

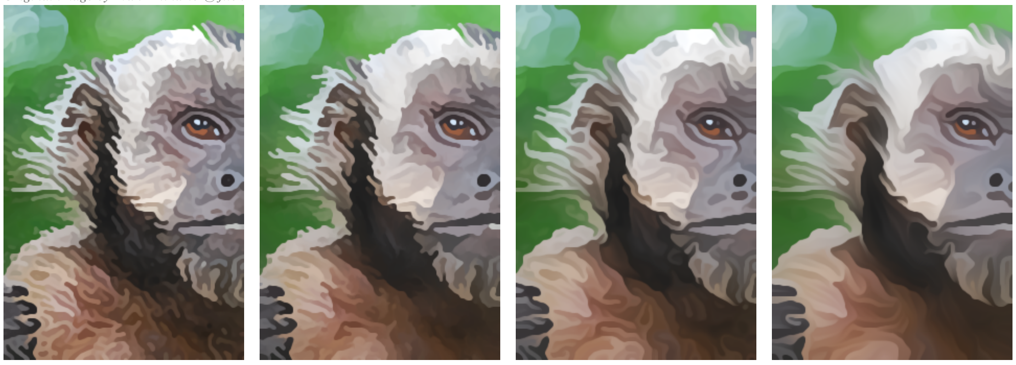
evolution equation of MCF is given by

$$\frac{\partial u}{\partial t} = \kappa |\nabla u| \quad \text{with} \quad \kappa = \frac{u_x^2 u_{xx} - 2u_x u_y u_{xy} + u_y^2 u_{yy}}{(u_x^2 + u_y^2)^{3/2}} \tag{6.1}$$

denoting the curvature. Equation (6.1) can be implemented using central differences. A better approach, however, is to use a finite difference scheme with harmonic averaging [DW07].

MCF performs strong simplification of an image but also creates blurred edges (Figure 6.5(b)). Therefore, Kang and Lee [KL08] performed deblurring with a shock filter (Section 6.4.1) after some MCF iterations, which helps to keep important edges during the evolution. From an artistic point of view, however, shock filtered MCF is typically still too aggressive and does not properly protect directional image features (Figure 6.5(c)). Similar to Equation (3.3), Kang and Lee therefore constrained the mean curvature flow by using the ETF to penalize diffusion that deviates from the local image structure (Figure 6.5(d)).

Original image by Ivan Mlinaric @ flickr



(a) *None* (b) *Relaxation* (c) *Gaussian smoothing* (d) *Both*

Figure 6.6: *Demonstration how relaxation and Gaussian smoothing of the structure influence the final result. In all cases, 10 iterations were performed. For the Gaussian smoothing* $\rho = 1$ *was used.*

More concretely, the evolution equation is given by

$$\frac{\partial u}{\partial t} = \left((1-r) + r \cdot \left| \left\langle \frac{E}{\|E\|}, \frac{\nabla u^{\perp}}{\|\nabla u\|} \right\rangle \right| \right) \|\kappa\| \,, \tag{6.2}$$

where $\langle \cdot, \cdot \rangle$ denotes the per-pixel scalar product of the ETF vectors and vectors perpendicular to the image gradients. The control parameter $r \in [0,1]$ allows for blending between the unconstrained and the constrained MCF. Alternatively, instead of the ETF, the minor eigenvector field of the SST can be used.

MCF and its constrained variant contract isophote curves to points [Gra87]. For this reason, important image features must be protected by a user-defined mask. A further limitation is that the technique is not stable against small changes in the input and therefore not suitable for per-frame video processing.

6.2 Local Structure Estimation

In low-contrast regions, the signal-to-noise ratio is low, making the gradient information unreliable. Accordingly, the estimated orientation is almost random and of little value. Since appropriate orientation information is critical for the presented algorithm, unreliable structure tensors are replaced using the approach discussed in Section 2.7.3. However, performing the relaxation for every computation of the structure tensor is expensive. Therefore, the relaxation is only performed for the first computation of the structure tensor. All subsequent computations use the structure tensor of the previous computation for points where the smoothed structure tensor's major eigenvalue is below a given threshold.

Figure 6.6 compares how relaxation and smoothing of the structure tensor influence the final result. As can be seen, without smoothing, the proposed algorithm performs rather poorly. Relaxation and smoothing perform much better, but even better results are obtained by combining both. When used in combination with the relaxation the standard deviation of the Gaussian smoothing can be kept comparatively small, while still achieving a strong stylization effect. This helps to preserve circular image features such as eyes.

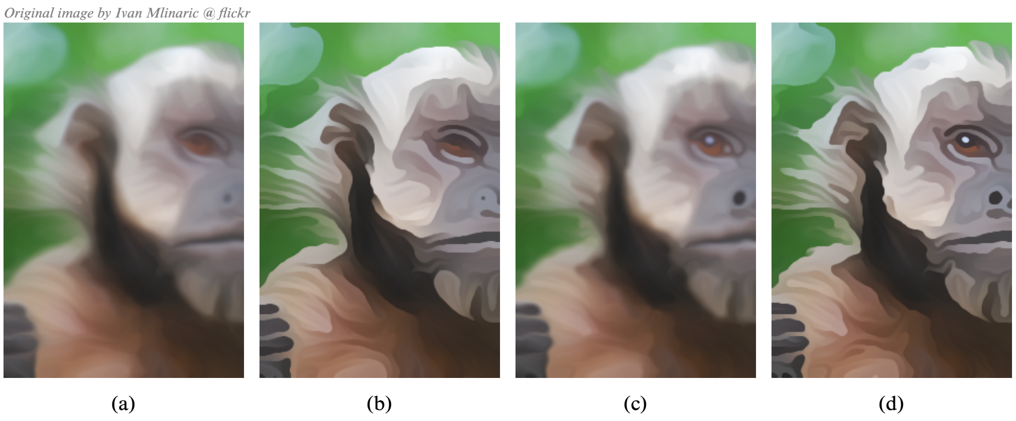


Figure 6.7: *Comparison of flow-guided smoothing using first-order Euler vs. second-order Runge-Kutta integration with and without shock filtering. (a) Euler. (b) Euler + shock filtering. (c) Runge-Kutta. (d) Runge-Kutta + shock filtering.*

6.3 Adaptive Smoothing

To simplify the image, line integral convolution along the integral curves defined by the minor eigenvector field of the smoothed structure tensor is performed (Section 2.9.3). In contrast to the isophote curves of the image, the integral curves defined by the smoothed structure tensor are much smoother. This has the effect that smoothing along them results in a regularization of the geometry of the isophote curves. To achieve high-quality results, the stream line integration must be performed using second-order Runge-Kutta method, since comparatively long integral curves are required to achieve a strong simplification effect. For such long curves, the Euler method is too imprecise (Figure 2.31). In Figure 6.7, it is demonstrated how the choice of the integration method influences the final result.

In Section 2.9.3, the length of the integral curves used for line integral convolution was expected to be globally defined; for instance, in case of a Gaussian filter kernel, to be proportional to its standard deviation. However, this may lead to issues in high curvature regions, as illustrated in Figure 6.8. If the stream lines are too long, they may wrap around, resulting in some pixels being sampled more often than others. Moreover, due to rounding errors and inaccuracies in the integral curve computation, adjacent isophotes may be sampled, which introduces additional blurring. To avoid these issues, the length of the stream lines and, correspondingly, the size of the Gaussian filter kernel must be controlled adaptively on a per-pixel basis. The next two sections discuss two approaches.

6.3.1 Anisotropy-based Length

To adaptively control the amount of smoothing, the anisotropy measure of the smoothed structure tensor discussed in Section 2.6.1 may be used. The anisotropy A ranges from zero to one, with zero corresponding to isotropic and one corresponding to entirely anisotropic regions. To control the length of the integral curve, the standard deviation of the one-dimensional Gaussian function is defined by

$$\widetilde{\sigma}_s = \frac{1}{4}\sigma_s\,(1 + A)^2 \,.$$

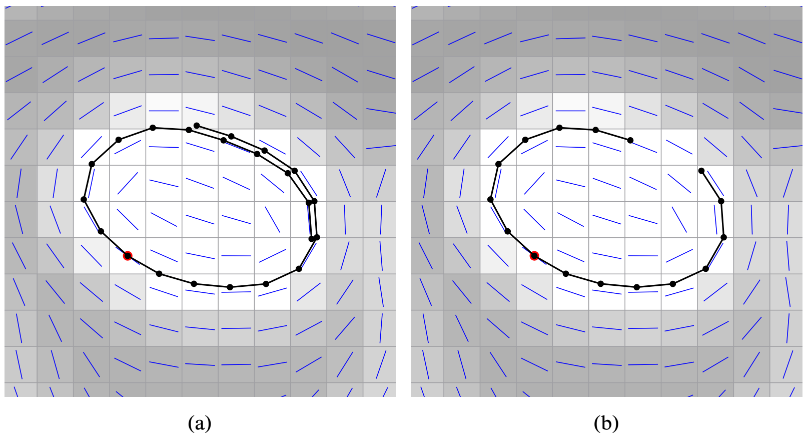


Figure 6.8: *(a) In high curvature regions, long integral curves may wrap around, resulting in blurring perpendicular to the integral curve. (b) Adaptively controlling the length of the integral curve avoids this problem.*

The global parameter σ_s thereby enables to fine tune the smoothing, and a typical choice is $\sigma_s = 6$, which is used for all examples. By definition, the adapted parameter $\widetilde{\sigma}_s$ lies in the range $\sigma_s/4 \leq \widetilde{\sigma}_s \leq \sigma_s$. Thus, in regions of high curvature, the anisotropy will be low and, therefore, σ_s small. In regions of low curvature, σ_s will be large, resulting in an enhancement and regularization of directional image features.

6.3.2 Adaptive Length

The approach discussed in the previous section works reasonably well in practice but is purely heuristic and its exact behavior difficult to analyze. Instead of adjusting the integral curve's length in advance, an alternative and more intuitive approach is to adjust its parameterization by slowing down the parameter's velocity if necessary. To this end, the angle between the minor eigenvectors of the previous step and the current step is computed:

$$\theta_k = \arccos \left\langle t^{k-1}(x^{k-1}), t^k(x^k) \right\rangle$$

Taking the sum $\Theta = \sum_{i=1}^{k} \theta_i$ of the angles measures the cumulated angular change of the stream line. If $\Theta \geq \pi$, the integral curve is likely to wrap around or to be comparatively noisy. In both cases, extending it further in the current direction is undesirable. However, simply stopping the tracing process corresponds to truncating the filtering operation and may introduce sampling artifacts. Instead, a better approach is to modify the parameter of the integral curve. To this end, the fraction traveled on a half-circle with arc length L can be considered for θ_k:

$$\Delta L_k = \frac{\theta_k}{\pi} L$$

If ΔL_k is larger than the step size h, this indicates that the parameter of the arc length parameterized integral curve moves too fast, such that a larger step size is required, for which ΔL_k is used:

$$u^{k \oplus 1} = u^k + \max(h, \Delta L_k)$$

The implementation is straightforward. In Listing 2.6, line 28 has to be replaced by:

```
float Lk = acosf(fminf(vt,1)) / CUDART_PI_F * f.radius();
u += fmaxf(step_size, Lk);
```

6.4 Sharpening

The adaptive flow-guided smoothing discussed in the previous section is very aggressive. As can been seen in Figures 6.7(a) and 6.7(c), the overall shape of the image features is well-preserved, but transitions between color regions are smoothed as well. In this section, deblurring by shock filtering to obtain sharp transitions at edges will be discussed.

6.4.1 Classical Shock Filter

In image processing, shock filters were first studied by [OR90]. The classical shock filter evolution equation is given by

$$\frac{\partial u}{\partial t} = -\operatorname{sign}\big(\mathfrak{L}(u)\big)|\nabla u| \ , \tag{6.3}$$

with initial condition $u(x,0) = I(x)$, and where $\mathfrak{L}$ is a suitable detector, such as the Laplacian Δu or the second derivative in direction of the gradient. In the influence zone of a maximum, $\mathfrak{L}(u)$ is negative, and therefore a local dilation with a disc as structuring element is performed. Similarly, in the influence zone of a minimum, $\mathfrak{L}(u)$ is positive, which results in a local erosion. This sharpens the edges at the zero-crossings of Δu, as shown in Figure 6.9. Shock filters have the attractive property of satisfying a maximum principle, and, in contrast to unsharp masking, therefore, do not suffer from ringing artifacts. Alvarez and Mazorra [AM94] later studied shock filter in conjunction with anisotropic diffusion.

Shock filters are also closely related to local neighborhood filters. As was shown by Guichard and Morel [GM03], the classical Osher-Rudin shock filter corresponds asymptotically to a filter that is similar to a filter originally proposed by Kramer and Bruckner [KB75]. This filter replaces the current gray level value by either the minimum or maximum of the filter region, depending on which is closer to the current value. This results in either an erosion or a dilation and thus sharpens the image. A systematic treatment of minimum and maximum filters was presented in [VVV88].

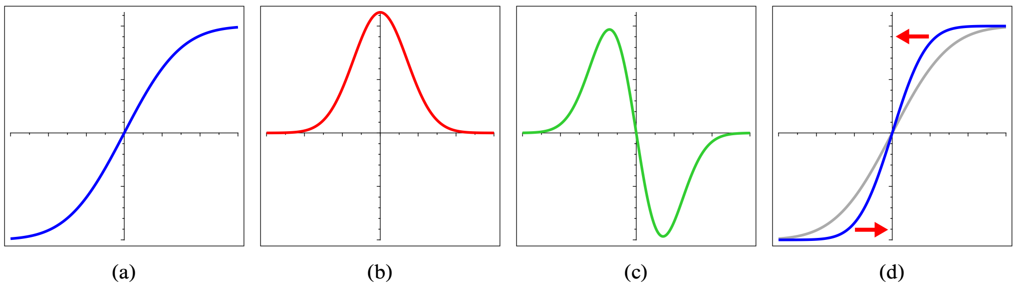
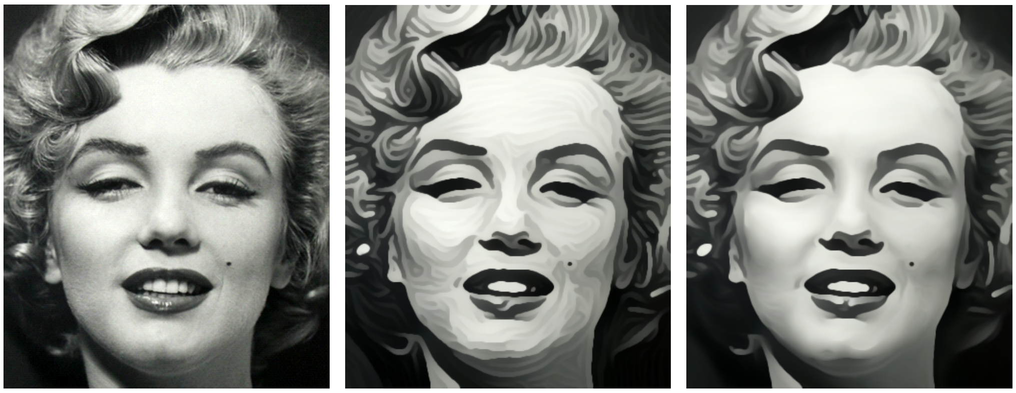


Figure 6.9: *Illustration of shock filtering. (a) A smooth step edge. (b) First derivative of the edge. (c) Second derivative of the edge. (d) A shock filter applies a dilation where the second derivative is positive, and an erosion where it is negative, resulting in a sharpening effect.*



(a) *Original image* (b) *Coherence-enhancing shock filter* (c) *Gradient-directed shock filter*

Figure 6.10: *Demonstration showing how the choice of the employed shock filter variant influences the final result of the proposed method.*

6.4.2 Coherence-Enhancing Shock Filter

A further modification is the coherence-enhancing shock filter by Weickert [Wei03]. The sign of the Laplacian is replaced by the sign of the second derivative in the direction of the major eigenvector of the smoothed structure tensor

$$u_t = -\operatorname{sign}(v_{\eta\eta})|\nabla u|\,,$$

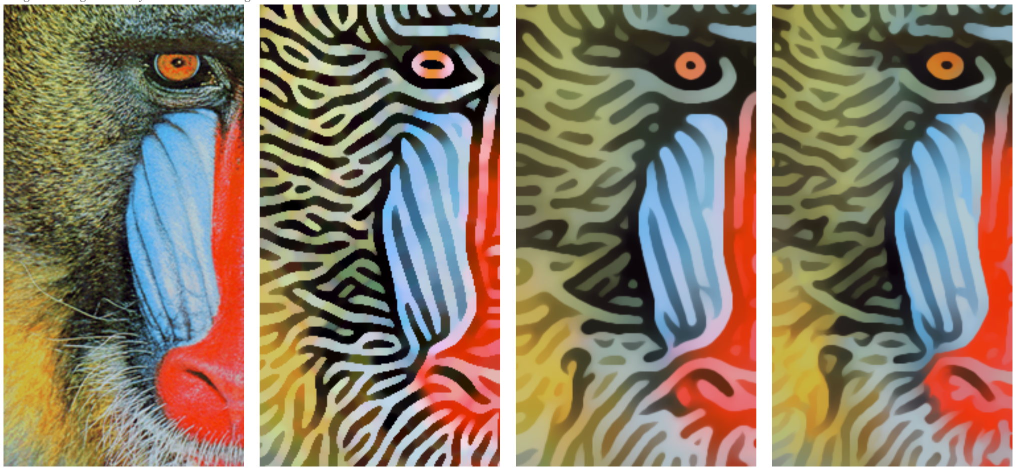
where $v = G_\sigma * u$ denotes a smoothed version of the evolving image and η is the major eigenvector derived from the structure tensor. To achieve higher robustness against small-scale image details, the input image may be regularized with a Gaussian filter prior to the second derivative or structure tensor computation.

In Figure 6.11(b) the coherence-enhancing shock filter applied to the popular mandrill image is shown. Clearly noticeable is the typical high-contrast of the output. Figure 6.11(c) shows the output of the proposed method, where the gradient-directed shock filter has been replaced by the coherence-enhancing shock filter. The color contrast is much more balanced and color regions have a consistent diffuse color variation. Weickert's shock filter achieves excellent results in combination with the proposed adaptive smoothing, but one limitation is its performance. The filter is typically implemented using the explicit Osher-Sethian [OS88] upwind scheme. In order to guarantee stability, the time step size has to be chosen ≤ 0.5 and, therefore, multiple iterations have to be performed. We use a time step size of 0.4 and three iterations. For each iteration, first and second order derivatives and the smoothed structure tensor have to be calculated. Another limitation is shown in Figure 6.10(b). Weickert's shock filter will introduce shocks in almost smooth regions, resulting in maze-like artifacts. The next section discusses an alternative approach.

6.4.3 Gradient-directed Shock Filter

The idea in obtaining a fast shock filter implementation is to approximate the general working principle discussed earlier and illustrated in Figure 6.9. First, whether a pixel is in the neighborhood of a minimum or maximum is detected. Then, correspondingly, either

Original image courtesy USC-SIPI Image Database

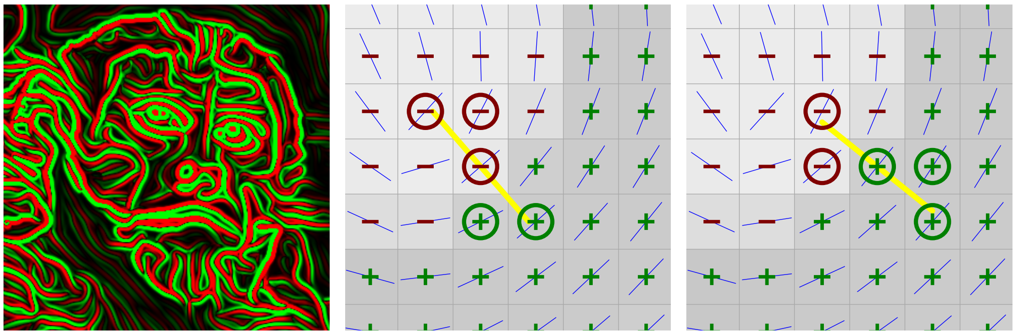


(a) *Original image* (b) *Coherence-enhancing shock filter [Wei03]* (c) *Proposed method with coherence-enhancing shock filter* (d) *Proposed method with gradient-directed shock filter*

Figure 6.11: *Comparison of the proposed method (10 iterations), using different shock filter variants, with the coherence-enhancing shock filter (20 iterations, $\rho = 5$, $\sigma = 3$, $\tau = 0.4$).*

an erosion or dilation is performed. Both operations are guided by the structure tensor, as illustrated in Figure 6.12.

Derivative operators are highly sensitive to noise, and sensitivity increases with order. Therefore, the second derivative operator used for the sign computation must be regularized to avoid artifacts. In addition, the regularization allows for artistic control over the resulting line thickness. Two strategies are at hand. First, the image can be isotropically smoothed prior to derivative computation, using a Gaussian filter with standard deviation σ_i. This helps to remove noise and allows for aggressive image simplification. Secondly, the smoothing and derivative operators can be consolidated into a single operator, since convolution and differentiation commute. Inspired by the flow-based difference of Gaussians filter and its separable implementation (Section 4.5), the second-order derivative in direction of the



(a) (b) (c)

Figure 6.12: *Approximation of a shock filter by a flow-guided local neighborhood filter. (a) Visualization of the sign of the FLoG filter. Negative values are red, positive values are green, and zero is shown black. (b) For a negative FLoG sign, the maximum of the marked pixels is computed. (c) For a positive FLoG sign, the minimum is computed.*

```
__global__ void flog( const gpu_plm2<float4> st, float sigma, gpu_plm2<float> dst )
{
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    float2 n = st_major_ev(st(ix, iy));
    float2 nabs = fabs(n);
    float ds = 1.0f / ((nabs.x > nabs.y)? nabs.x : nabs.y);
    float2 uv = make_float2(ix + 0.5f, iy + 0.5f);

    float halfWidth = 5 * sigma;
    float sigma2 = sigma * sigma;
    float twoSigma2 = 2 * sigma2;

    float sum = -sigma2 * tex2D(texL, ix + 0.5f, iy + 0.5f);
    for( float d = ds; d <= halfWidth; d += ds ) {
        float k = (d*d - sigma2) * __expf( -d*d / twoSigma2 );
        float2 o = d*n;
        float c = tex2D(texL, uv.x - o.x, uv.y - o.y) +
                  tex2D(texL, uv.x + o.x, uv.y + o.y);
        sum += k * c;
    }

    sum = sum / (sqrtf(2*CUDART_PI_F) * sigma2 * sigma);
    dst(ix, iy) = sum;
}
```

Listing 6.1: *Implementation of the flow-guided LoG filter.*

major eigenvector is implemented by convolving the image locally with a one-dimensional (scale-normalized) second-order Gaussian derivative,

$$\sigma_g^2 G''_{\sigma_g}(t) = \sigma_g^2 \frac{\mathrm{d}^2 G_{\sigma_g}(t)}{\mathrm{d}t^2} = \frac{x^2 - \sigma_g^2}{\sqrt{2\pi}\sigma_g^3} \exp\left(-\frac{t^2}{2\sigma_g^2}\right),$$

in the direction of the minor eigenvector. This operation will be referred to as *flow-based Laplacian of Gaussian* (FLoG). More specifically, let L be the input image converted to grayscale, let $v = G_{\sigma_i} * L$, and let x_0 be the current pixel; then the convolution is computed by

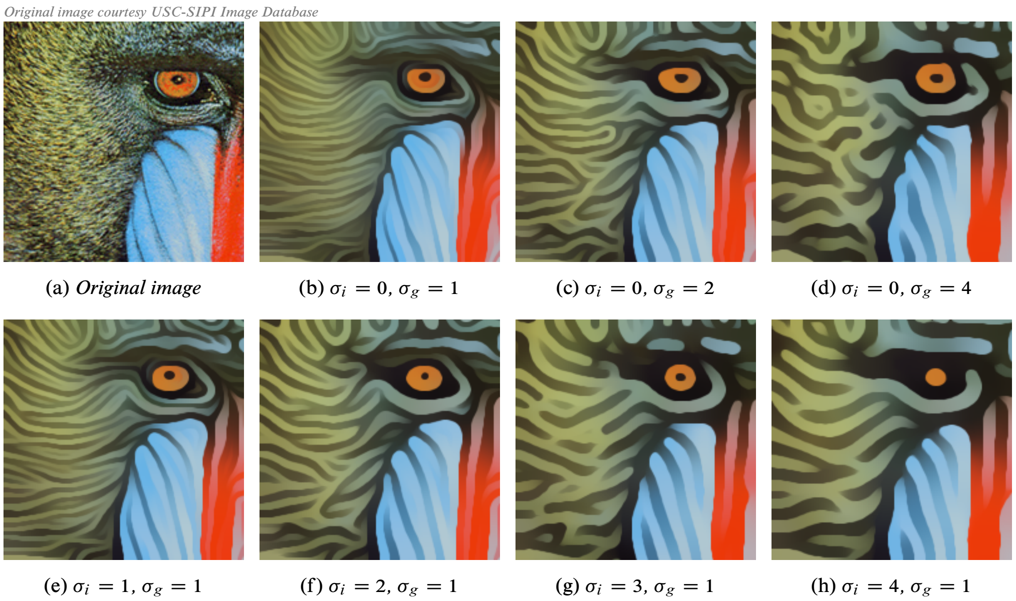
$$z(x_0) = \sigma_g^2 \int G''_{\sigma_g}(t)\, v\big(x_0 + t\,\eta(x_0)\big)\, \mathrm{d}t\,,$$

where $\eta(x_0)$ denotes the major eigenvector. The implementation is shown in Listing 6.1. As in case of the orientation-aligned bilateral filter (Section 3.4), the evaluation of the integral is performed using a constant step size with unit size along either the horizontal or the vertical axis (Figure 3.9), and using bilinear interpolation. Due to the unit step size, this results in a linear interpolation of two neighboring pixels, and allows for efficient implementation on GPUs using texturing hardware.

The erosion and dilation operations are implemented as directional neighborhood filter as well. Let f denote the input image and let x_0 be the current point, then the

```
enum minmax_t { MIN_FLT, MAX_FLT };

struct minmax_impl_t {
    float2 uv_, p_;
    float v_;

    __device__ minmax_impl_t(float2 uv) {
        uv_ = p_= uv;
        v_ = tex2D(texL, uv.x, uv.y);
    }

    template <minmax_t T> __device__ void add(float2 p) {
        float L = tex2D(texL, p.x, p.y);
        if ((T == MAX_FLT) && (L > v_)) {
            p_ = p;
            v_ = L;
        }
        if ((T == MIN_FLT) && (L < v_)) {
            p_ = p;
            v_ = L;
        }
    }

    template <minmax_t T> __device__  void run( float2 n, float radius ) {
        float ds;
        float2 dp;
        float2 nabs = fabs(n);
        if (nabs.x > nabs.y) {
            ds = 1.0f / nabs.x;
            dp = make_float2(0, 0.5f - 1e-3);
        } else {
            ds = 1.0f / nabs.y;
            dp = make_float2(0.5f - 1e-3, 0);
        }
        for( float d = ds; d <= radius; d += ds ) {
            float2 o = d*n;
            add<T>(uv_ + o + dp); add<T>(uv_ + o - dp);
            add<T>(uv_ - o + dp); add<T>(uv_ - o - dp);
        }
    }
};

__global__ void grad_shock( const gpu_plm2<float4> st, const gpu_plm2<float> sign,
                            float radius, gpu_plm2<float4> dst ) {
    const int ix = blockDim.x * blockIdx.x + threadIdx.x;
    const int iy = blockDim.y * blockIdx.y + threadIdx.y;
    if (ix >= dst.w || iy >= dst.h) return;

    minmax_impl_t mm(make_float2(ix + 0.5f, iy + 0.5f));
    float2 n = st_major_ev(st(ix, iy));
    float s = sign(ix, iy);
    if (s < 0) {
        mm.run<MAX_FLT>(n, radius);
    } else if (s > 0) {
        mm.run<MIN_FLT>(n, radius);
    }
    dst(ix, iy) = tex2D(texSRC, mm.p_.x, mm.p_.y);
}
```

Listing 6.2: *Implementation of the gradient directed min/max filter.*


Original image courtesy USC-SIPI Image Database

(b) $\sigma_i = 0, \sigma_g = 1$
(c) $\sigma_i = 0, \sigma_g = 2$
(d) $\sigma_i = 0, \sigma_g = 4$
(e) $\sigma_i = 1, \sigma_g = 1$
(f) $\sigma_i = 2, \sigma_g = 1$
(g) $\sigma_i = 3, \sigma_g = 1$
(h) $\sigma_i = 4, \sigma_g = 1$

Figure 6.13: *Different combinations of the isotropic and gradient directed smoothing parameters σ_i and σ_g, respectively, are shown for the proposed method.*

gradient-directed shock filter is defined as:

$$\begin{cases} \min\limits_{x \in \Lambda_r(x_0)} f(x) & \text{if } z(x_0) > +\tau_s \\ \max\limits_{x \in \Lambda_r(x_0)} f(x) & \text{if } z(x_0) < -\tau_s \\ f(x_0) & \text{otherwise} \end{cases}$$

Determination of the minimum and maximum is performed based on the corresponding gray values. The filter neighborhood Λ_r is defined as the set of pixels with a distance less than r from x_0 intersecting the line, $\{x_0 + \lambda\, \eta(x_0)\}$, defined by the major eigenvector. The implementation is shown in Listing 6.2. Again, a constant step size with unit size in either horizontal or vertical direction is used. Bilinear interpolation, however, is not appropriate for the computation of the minimum or maximum; therefore, the two neighboring pixels are sampled explicitly, using nearest-neighbor sampling. Through a small correction of the sampling offset, the correct sampling of horizontal, vertical, and diagonal lines is assured. For the radius, typically $r = 2$ is used. The parameter τ_s controls the sensitivity to noise and is typically set to $\tau_s \in [0, 0.01]$. Since a scale-normalized second-derivative is used, τ_s does not depend upon σ_g. The threshold effectively prevents the creation of shocks in almost smooth regions, as can be seen in Figure 6.10(c).

The quality of the output is comparable to that of the coherence-enhancing shock filter, but computationally the gradient-directed shock filter is much more efficient. It only requires the smoothed structure tensor and the input image converted to grayscale. For the sake of simplicity, the FLoG and the minimum and maximum filters have been implemented independently, but obviously they could be implemented in a single pass as well. Moreover, the gradient-directed shock filter provides finer artistic control. The parameter σ_g restricts smoothing to the major eigenvector direction. This is especially useful for preserving small

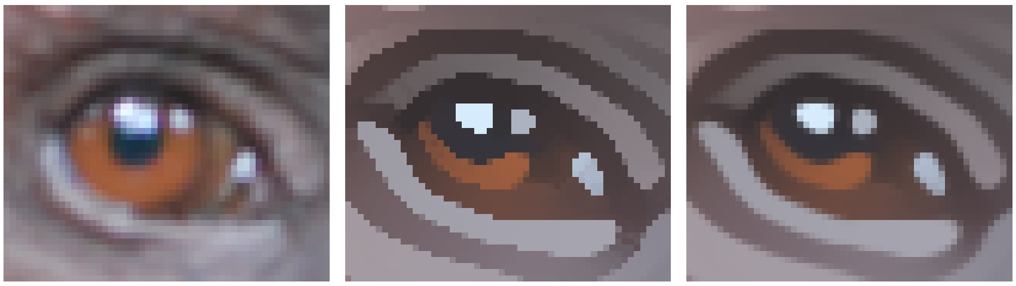

(a) *Original image* (b) *Result* (c) *Anti-aliased result*

Figure 6.14: *The final step of the proposed method anti-aliases sharp transitions created by the shock filter by performing flow-guided smoothing with a small kernel.*

image features. To achieve a stronger abstraction, the isotropic smoothing parameter σ_i is useful. For examples in this work, typically $\sigma_g = 1.5$ and $\sigma_i = 0$ were used. Figure 6.13 shows how the abstraction can be controlled using σ_g and σ_i.

6.5 Edge Smoothing

The shock filter creates very sharp transitions at region boundaries. To anti-alias the region boundaries, as demonstrated in Figure 6.14, a possible solution is to apply the adaptive smoothing with a small standard deviation $\sigma_a \in [1, 1.5]$ as a final pass. In this case no adaption of the standard deviation is performed. To this end, the structure tensor is not recomputed. Instead, the structure tensor that computed for the shock filter is used.

6.6 Results

Figure 6.15 demonstrates the ability of the proposed approach to preserve circular features like eyes. As shown in Figure 6.15(b), constrained mean curvature flow does not preserve curvature and contracts circular image features to points. By contrast, the proposed adaptive smoothing preserves the curvature of image features and, therefore, does not require manual assistance, such as masking, to protect important image features. In addition to preserving

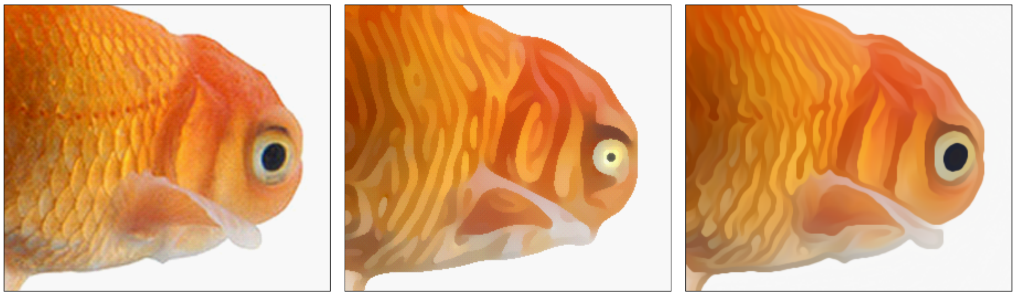
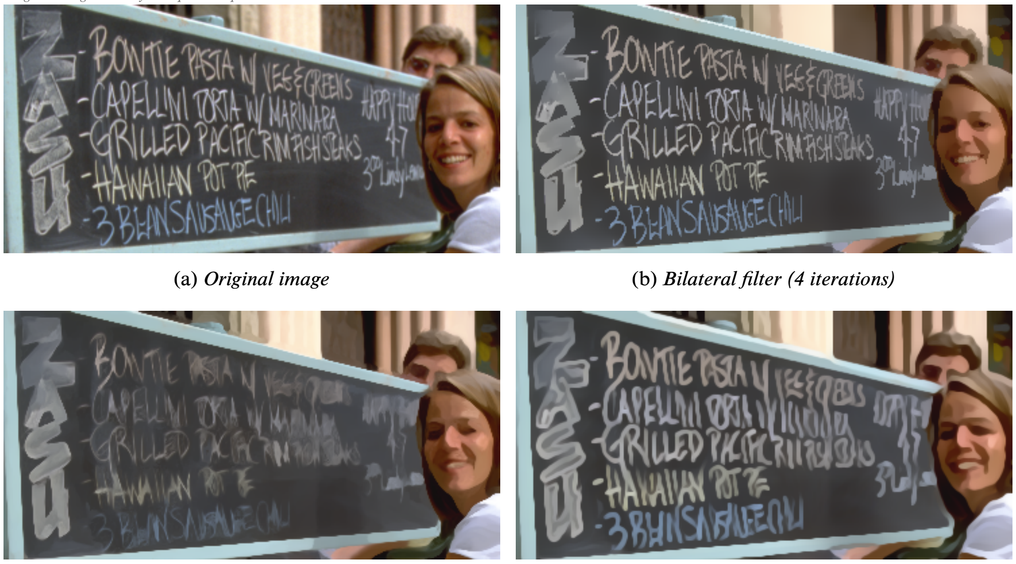

(a) *Original image* (b) *SSIA* (c) *Proposed method*

Figure 6.15: *In contrast to shape-simplifying image abstraction [KL08], the proposed approach does not require manual masking to protect small image features such as eyes.*

Original image courtesy Philip Greenspun


ZASU
-BOWTIE PASTA W/ VEG & GREENS
-CAPELLINI TORTA W/ MARINARA
-GRILLED PACIFIC RIM FISH STEAKS
-HAWAIIAN POT PIE
-3 BEAN SAUSAGE CHILI
HAPPY HOUR
4-7

(a) *Original image* (b) *Bilateral filter (4 iterations)*

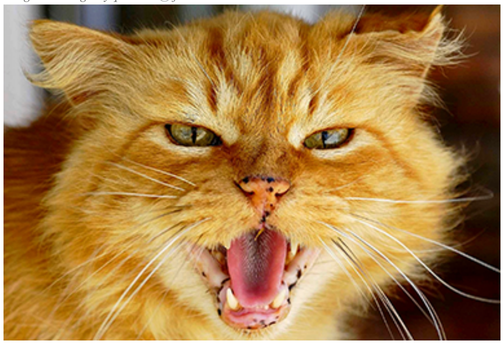
(c) *Anisotropic Kuwahara filter* (d) *Proposed method*

Figure 6.16: *In contrast to other techniques, the proposed approach emphasizes and enhances highly anisotropic image features.*

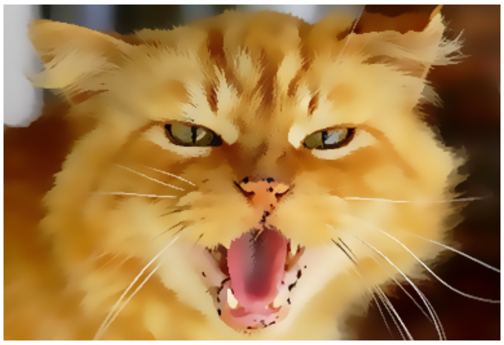
curvature, the adaptive smoothing effectively enhances small highly anisotropic image features. This is demonstrated in Figure 6.16. By contrast, the anisotropic Kuwahara filter either erodes or blurs thin image features.

Further examples are shown in Figure 6.17. In contrast to the bilateral filter (Figure 6.17(b)), the proposed approach creates a consistent abstraction across the entire image. The anisotropic Kuwahara filter (Figure 6.17(c)) also creates a consistent abstraction, but the result looks somewhat washed out because of missing contrast. For images with strong directional features, shape-simplifying image abstraction typically achieves very good results (Figure 6.17(d)). The proposed approach also performs excellently for these types of images and, by increasing the number of iterations, a strong abstraction effect can be obtained as well (Figure 6.17(f)). Figures 6.17(e) and 6.17(f) demonstrate the feasibility of the proposed approach to control the strength of abstraction by changing the number of iterations. This is also claimed for shape-simplifying image abstraction, but only given from a certain number of iterations. For a small number of iterations, shape-simplifying image abstraction typically shows artifacts, which makes a light abstraction effect impossible. The anisotropic Kuwahara filter is limited to a light abstraction effect, since the amount of abstraction depends upon the filter radius. Using larger filter radii will also increase the amount of blurring and may destroy important image features.

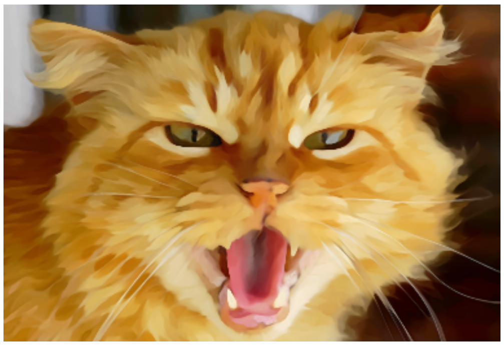
Figure 6.18 demonstrates the stability of the proposed approach. Different methods are applied to an image that has been artificially corrupted with Gaussian and impulse noise. Shape-simplifying image abstraction delivers the least satisfactory performance. The proposed approach performs decently, but is not able to remove the impulse noise. This is not surprising. Both filters are based on anisotropic smoothing, which will remove the Gaussian noise but also blur the impulse noise. Both techniques also use shock filtering, which enhances noise. The anisotropic Kuwahara filter successfully restores the image, *Original image by pasma @ flickr*



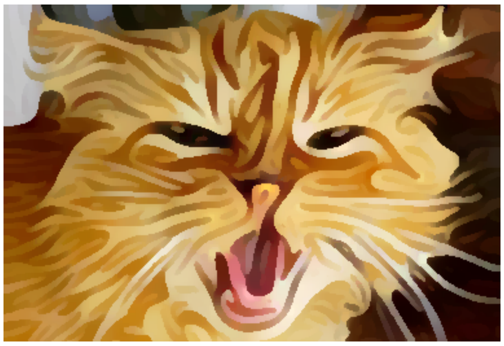
(a) *Original image*



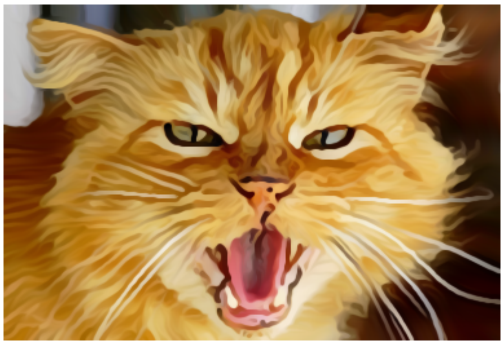
(b) *Bilateral filter (4 iterations)*



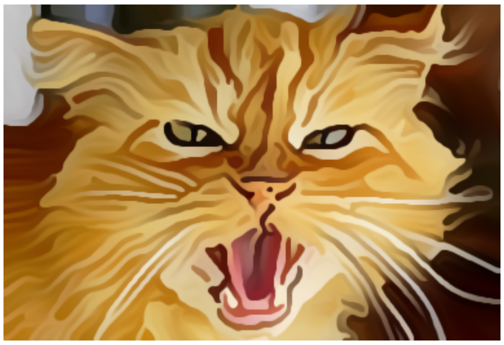
(c) *Anisotropic Kuwahara filter*



(d) *Shape-simplifying image abstraction [KL08]*



(e) *Proposed method ($N = 2$)*



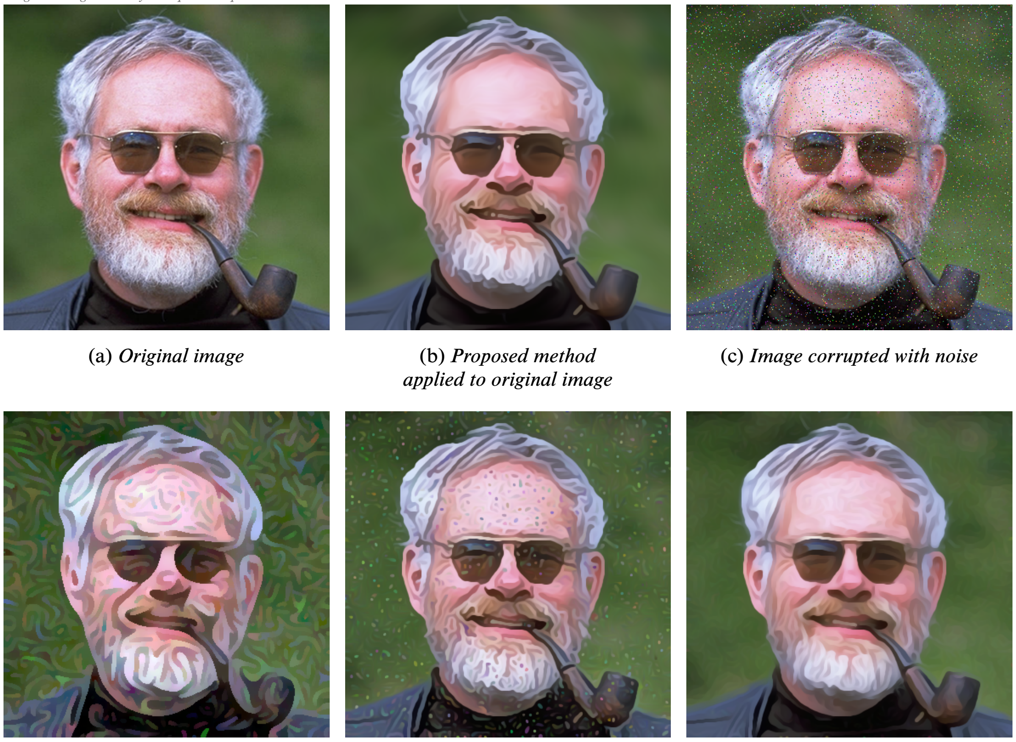
(f) *Proposed method ($N = 10$)*

Figure 6.17: *Comparison with anisotropic Kuwahara filter, shape-simplifying image abstraction, and bilateral filter.*

but the region boundaries are distorted. When the number of iterations is increased, the proposed approach performs well. It converges to a nearly steady state that looks a little odd (Figure 6.19(a)), but it does not blow up like shape-simplifying image abstraction (Figure 6.19(b)).

In Figure 6.20, the proposed approach is compared with GradientShop by Bhat et al. [Bha+10], a technique based on gradient domain image processing. In contrast to this technique, the proposed approach also performs a regularization of the image that results in smooth color boundaries. However, regions of the same color are not as smooth as the ones created by GradientShop. This is because the adaptive smoothing only performs smoothing in the tangential direction. On the other hand, this retains important visual detail, as for

Original image courtesy Philip Greenspun


(a) *Original image* (b) *Proposed method applied to original image* (c) *Image corrupted with noise*

(d) *Shape-simplifying image abstraction [KL08] applied to noisy image* (e) *Proposed method applied to noisy image* (f) *Anisotropic Kuwahara filter + proposed method applied to noisy image*

Figure 6.18: *Anisotropic Kuwahara filter, shape-simplifying image abstraction, and proposed method applied to an image corrupted with 2% Gaussian noise and 5% impulse noise.*

example, the specular reflection on the nose. Also notice that, unlike the proposed technique, GradientShop currently does not perform processing in real-time.

In Figure 6.21 a comparison with the single-scale and multi-scale anisotropic Kuwahara filters is shown. The level of abstraction is quite similar to the single-scale anisotropic Kuwahara filter, but the proposed approach creates output with stronger contrast. This is

Original image courtesy Philip Greenspun

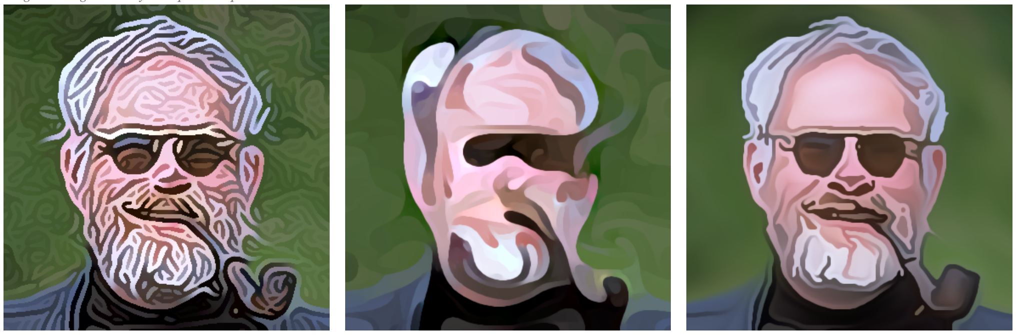

(a) *CESF* (b) *SSIA* (c) *CEF*

Figure 6.19: *Limit cases. (a) Coherence-enhancing shock filter [Wei03] with 50 iterations. (b) Shape-simplifying image abstraction [KL08] with 50 CMCF iterations. (c) Proposed method with 50 iterations.*

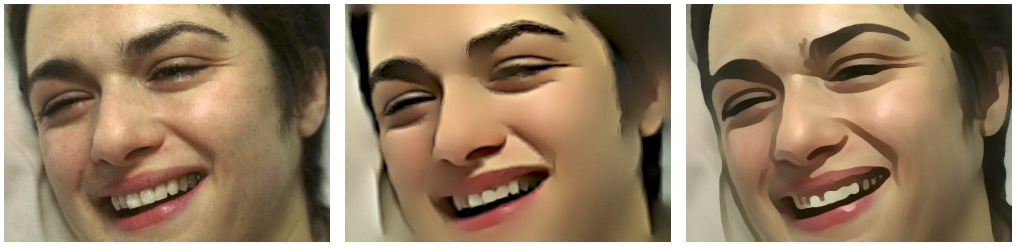

(a) *Original image* (b) *GradientShop [Bha+10]* (c) *Proposed method ($N = 10$)*

Figure 6.20: *Comparison with GradientShop by Bhat et al. [Bha+10].*

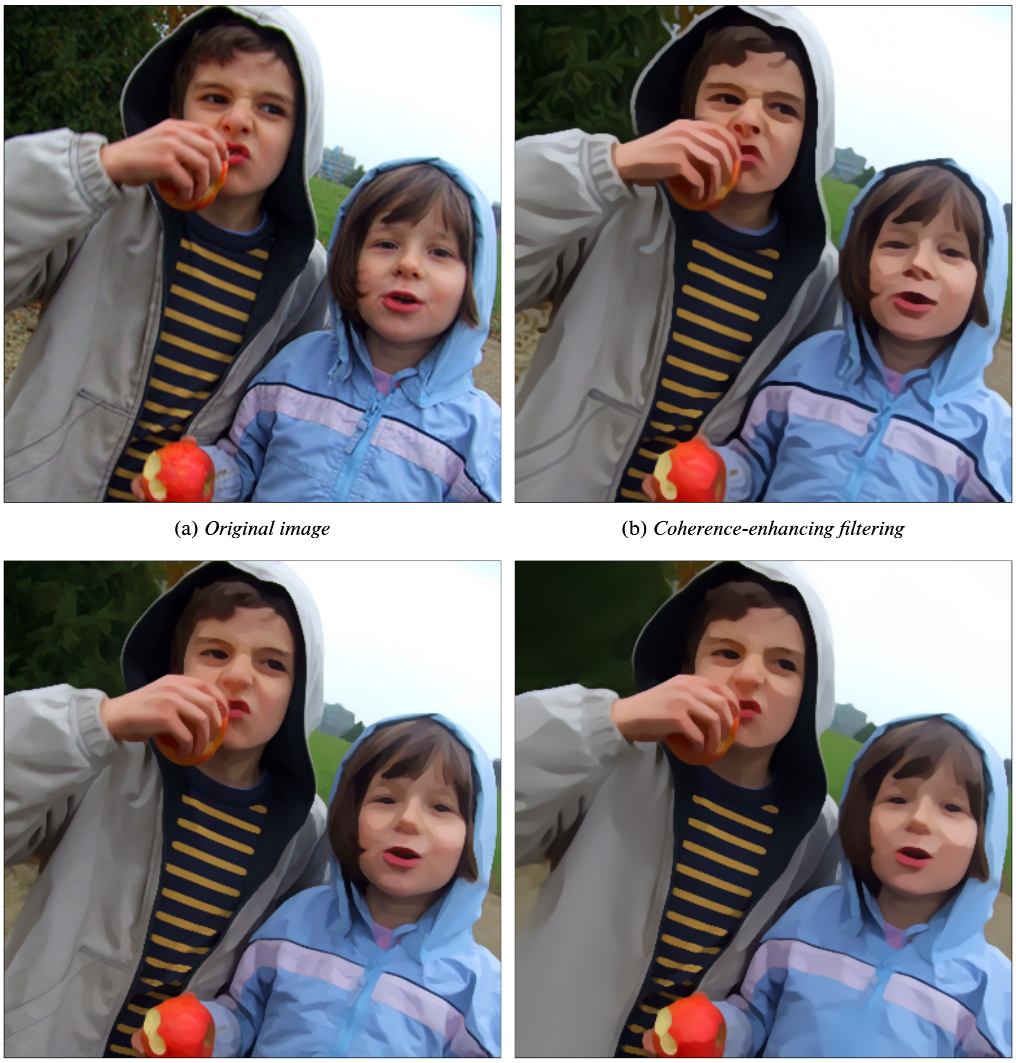
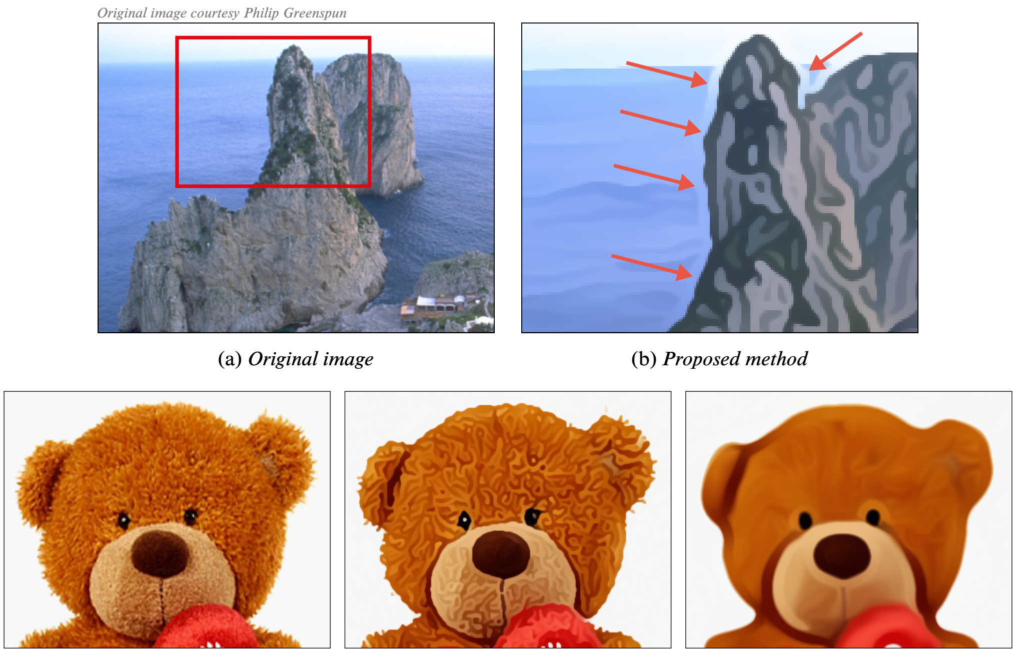

(a) *Original image* (b) *Coherence-enhancing filtering* (c) *AKF (single-scale)* (d) *AKF (multi-scale)*

Figure 6.21: *Comparison with single-scale and multi-scale anisotropic Kuwahara filter.*



(a) *Original image* (b) *Proposed method*

(c) *Original image* (d) *Proposed method* $(N = 10, \sigma_i = 0, \sigma_g = 1.5)$ (e) *Proposed method* $(N = 30, \sigma_i = 5, \sigma_g = 1.5)$

Figure 6.22: *Examples of images that are difficult to handle for the proposed approach.*

probably due to the shock filter. The output of the multi-scale approach looks very similar, but more detail has been removed from, for example, the jackets and the background.

An image that is difficult to handle for the proposed approach is shown in Figure 6.22(a). Here, two prevailing orientations exist. One of the water, which is almost horizontal, and one of the contour of the rock. Since the gradient magnitude is high at the rock's contour, the orientation of the contour will dominate in a small neighborhood of the contour, during the structure tensor smoothing. Therefore, the adaptive smoothing will perform smoothing on both sides of the contour. This results in a halo-like effect, as shown in Figure 6.22(b). In Figure 6.22(c), an image with high contrast texture is shown. If processed by the proposed method with typical parameters, the texture will be emphasized (Figure 6.22(d)). To obtain the result in Figure 6.22(e), a large number of iterations and $\sigma_i = 5$ had to be used to prevent the shock filter from regarding the texture as edges. However, flow-like structures are still clearly observable and are, again, due to the directional smoothing.

The proposed approach can also be applied to video using per-frame filtering without extra processing. Very good temporal coherence is provided for 1–3 iterations. Applying more iterations typically results in temporal artifacts becoming noticeable. However, compared to shape-simplifying image abstraction or segmentation based approaches these are rather minor.

Chapter 7

Conclusions

Measuring the success and evaluating the quality of an NPR technique is a difficult challenge. In fact, it is one of the challenges identified by David Salesin in his famous keynote at NPAR 2002. While some works have approached this subject—see the discussion on evaluation in [J1]—authors of IB-AR paper typically either innovate on new styles or demonstrate improvements over prior work; for instance, by an subjective evaluation using side-by-side comparisons of results, as was also done in this work, or by demonstrating performance improvements of the algorithm.

The orientation-bilateral filter improves over the xy-separable bilateral filter by avoiding horizontal and vertical artifacts. This comes at the price of additional computational overhead in form of the computation of the smoothed structure tensor. This has only a small influence on the overall performance of the generalized cartoon pipeline, since the structure tensor computation has to be performed only once and also may be shared with the flow-based DoG. The flow-based bilateral filter further improves on the ability to enhance directional image features but is also computationally more demanding. Moreover, it was shown that flow-based DoG filtering may be implemented in a separable way without sacrificing quality, thus yielding a significant performance improvement. The anisotropic Kuwahara filter generates a feature-preserving, direction-enhancing painterly look, without having to deal with individual brush strokes. Unlike existing nonlinear smoothing filters, such as the bilateral filter, it is robust against high-contrast noise and avoids overblurring in low-contrast areas, providing a consistent level of abstraction across the image. However, the anisotropic Kuwahara filter is also computationally expensive, since for every sector a full 2D kernel has to be computed. Coherence-enhancing filtering aggressively smoothes out unimportant regions but protects important features by enhancing contrast and directional coherence, providing a good balance between content abstraction and feature enhancement consistently across the image. Especially, for input with strong anisotropic structures it provides excellent results. However, the shock filter may introduce undesired edges in flat and homogeneous regions, due to the missing smoothing in the gradient direction. Moreover, highly textured image regions and region boundaries with contrary flow orientations on both sides are difficult to handle. For all the developed filters, the local orientation, and for some also the local anisotropy, obtained from the smoothed structure tensor were significant. Moreover, the experiment in Section 2.5 showed that the smoothed structure tensor constitutes not only an improvement over the edge tangent flow in terms of computationally efficiency, but also in terms of the quality of the orientation estimate.

There has been little discussion on video processing in this work, although the term video explicitly appeared in the title. This is because the techniques described in this work achieve good temporal coherent output off the shelf by simply processing each frame individually. This is significant, since temporal coherence is a long-standing challenge in video IB-AR.

List of Publications

Journal Papers

[J1] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg. "State of the 'art': A taxonomy of artistic stylization techniques for images and video". In: *IEEE Transactions on Visualization and Computer Graphics* (2013). DOI: 10.1109/TVCG.2012.160 (cit. on pp. 1, 145).

[J2] H. Winnemöller, J. E. Kyprianidis, and S. C. Olsen. "XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization". In: *Computers and Graphics* 36.6 (2012), pp. 740–753. DOI: 10.1016/j.cag.2012.03.004 (cit. on pp. 5, 81, 88, 149).

[J3] J. E. Kyprianidis and H. Kang. "Image and video abstraction by coherence-enhancing filtering". In: *Computer Graphics Forum* 30.2 (2011). Special issue on Eurographics 2011, pp. 593–602. DOI: 10.1111/j.1467-_8659.2011.01882.x (cit. on pp. 4, 6, 149).

[J4] J. E. Kyprianidis, H. Kang, and J. Döllner. "Image and video abstraction by anisotropic Kuwahara filtering". In: *Computer Graphics Forum* 28.7 (2009). Special issue on Pacific Graphics 2009, pp. 1955–1963. DOI: 10.1111/j.1467-_8659.2009.01574.x (cit. on pp. 5, 149).

Book Chapters

[B1] J. E. Kyprianidis and H. Kang. "Coherence-enhancing filtering on the GPU". In: *GPU Pro 4—Advanced Rendering Techniques*. Ed. by W. Engel. AK Peters, 2013 (cit. on pp. 6, 149).

[B2] J. E. Kyprianidis. "Artistic stylization by nonlinear filtering". In: *Image and Video-based Artistic Stylization*. Ed. by P. L. Rosin and J. Collomosse. Springer, 2013, pp. 77–101. DOI: 10.1007/978-_1-_4471-_4519-_6_5 (cit. on p. 1).

[B3] J. E. Kyprianidis, H. Kang, and J. Döllner. "Anisotropic Kuwahara filtering on the GPU". In: *GPU Pro—Advanced Rendering Techniques*. Ed. by W. Engel. AK Peters, 2010, pp. 247–264 (cit. on pp. 5, 149).

[B4] J. E. Kyprianidis and J. Döllner. "Real-time image abstraction by directed filtering". In: *ShaderX7—Advanced Rendering Techniques*. Ed. by W. Engel. Charles River Media, 2009, pp. 285–302 (cit. on pp. 5, 6, 149).

Tutorials

[T1] J. Collomosse and J. E. Kyprianidis. “Artistic stylization of images and video”. In: *Tutorial at Eurographics*. 2011.

Conference Papers

[C1] J. E. Kyprianidis. “Image and video abstraction by multi-scale anisotropic Kuwahara filtering”. In: *9th Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 2011, pp. 55–64. DOI: 10.1145/2024676.2024686 (cit. on pp. 4, 5).

[C2] J. E. Kyprianidis, A. Semmo, H. Kang, and J. Döllner. “Anisotropic Kuwahara filtering with polynomial weighting functions”. In: *Proceedings EG UK Theory and Practice of Computer Graphics*. 2010, pp. 25–30. DOI: 10.2312/LocalChapterEvents/TPCG/TPCG10/025-_030 (cit. on pp. 5, 149).

[C3] J. E. Kyprianidis and J. Döllner. “Image abstraction by structure adaptive filtering”. In: *Proceedings EG UK Theory and Practice of Computer Graphics*. 2008, pp. 51–58. DOI: 10.2312/LocalChapterEvents/TPCG/TPCG08/051-_058 (cit. on pp. 4–6, 89, 91, 149).

Posters

[P1] J. E. Kyprianidis, A. Semmo, H. Kang, and J. Döllner. “Anisotropic Kuwahara filtering with polynomial weighting functions”. In: *Poster at 8th Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 2010.

[P2] J. E. Kyprianidis, H. Kang, and J. Döllner. “Image and video abstraction by anisotropic Kuwahara filtering”. In: *Poster at 7th Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 2009.

[P3] J. E. Kyprianidis and J. Döllner. “Image abstraction by structure adaptive filtering”. In: *Poster at 6th Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 2008.

Miscellaneous

[M1] A. Semmo, M. Trapp, J. E. Kyprianidis, and J. Döllner. “Interactive visualization of generalized virtual 3D city models using level-of-abstraction transitions”. In: *Computer Graphics Forum* 31.3 (2012). Special issue on EuroVis 2012, pp. 885–894. DOI: 10.1111/j.1467-_8659.2012.03081.x (cit. on p. 8).

[M2] R. Richter, J. E. Kyprianidis, and J. Döllner. “Out-of-core GPU-based change detection in massive 3D point clouds”. In: *Transactions in GIS* (2013). DOI: 10.1111/j.1467-_9671.2012.01362.x.

[M3] A. Semmo, J. E. Kyprianidis, and J. Döllner. “Automated image-based abstraction of aerial images”. In: *Geospatial Thinking*. Ed. by M. Painho, M. Y. Santos, and H. Pundt. Lecture Notes in Geoinformation and Cartography. Proceedings of the 13th AGILE International Conference on Geographic Information Science. Springer, 2010, pp. 359–378. DOI: 10.1007/978-_3-_642-_12326-_9_19.

[M4] J. Döllner and J. E. Kyprianidis. “Approaches to image abstraction for photorealistic depictions of virtual 3D models”. In: *Cartography in Central and Eastern Europe*. Ed. by G. Gartner and F. Ortag. Lecture Notes in Geoinformation and Cartography. Selected papers of the ICA Symposium on Cartography for Central and Eastern Europe 2009. Springer, 2010, pp. 263–277. DOI: 10.1007/978-_3-_642-_03294-_3_17 (cit. on pp. 8, 9).

[M5] C. Richardt, J. E. Kyprianidis, and N. A. Dodgson. “Stereo coherence in watercolour rendering”. In: *Poster at Symposium on Computational Aesthetics (CAe) and Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 2010.

[M6] M. Jobst, J. E. Kyprianidis, and J. Döllner. “Mechanisms on graphical core variables in the design of cartographic 3D city presentations”. In: *Geospatial Vision*. Ed. by A. Moore and I. Drecki. Lecture Notes in Geoinformation and Cartography. Selected papers of GeoCart’2008. Springer, 2008, pp. 45–59. DOI: 10.1007/978-_3-_540-_70970-_1_3.

[M7] S. Maass, M. Trapp, J. E. Kyprianidis, J. Döllner, M Eichhorn, R. Pokorski, J. Bäuerlein, and H. v. Hesberg. “Techniques for the interactive exploration of high-detail 3D building reconstruction using the example of roman cologne”. In: *14th International Conference on Virtual Systems and Multimedia (VSMM 2008)*. Ed. by M Loannides, A Addison, A Georgopoulos, and L Kalisperis. Archaeolingua, 2008, pp. 223–229 (cit. on pp. 8, 9).

Open Source

[O1] *GLSL reference implementation of the generalized cartoon pipeline [C3; B4]*. URL: http://code.google.com/p/flowabs.

[O2] *GLSL reference implementation of the anisotropic Kuwahara filtering on the GPU [J4; B3]*. URL: http://code.google.com/p/gpuakf.

[O3] *GLSL reference implementation of the anisotropic Kuwahara filter with polynomial weighting functions [C2]*. URL: http://code.google.com/p/polyakf.

[O4] *CUDA reference implementation of coherence-enhancing filtering on the GPU [J3; B1]*. URL: http://code.google.com/p/cefabs.

[O5] *CUDA reference implementation of the XDoG filter, including the generalized cartoon pipeline [J2]*. URL: http://code.google.com/p/xdogdemo.

[O6] *CUDA library for image-based artistic stylization based on nonlinear filtering*. URL: http://code.google.com/p/liboz.

Bibliography

[ABD10] A. Adams, J. Baek, and M. A. Davis. "Fast high-dimensional filtering using the permutohedral lattice". In: *Computer Graphics Forum* 29.2 (2010), pp. 753–762. doi: 10.1111/j.1467-_8659.2009.01645.x (cit. on p. 70).

[Ada+09] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. "Gaussian KD-trees for fast high-dimensional filtering". In: *ACM Transactions on Graphics* 28.3 (2009), p. 1. doi: 10.1145/1531326.1531327 (cit. on p. 70).

[Aga+04] A. Agarwala, A. Hertzmann, D. Salesin, and S. M. Seitz. "Keyframe-based tracking for rotoscoping and animation". In: *ACM Transactions on Graphics* 23.3 (2004), pp. 584–591. doi: 10.1145/1015706.1015764 (cit. on p. 2).

[AK06] G. Aubert and P. Kornprobst. *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*. Springer, 2006 (cit. on p. 67).

[Ale06] M. Aleksic. "Novel bilateral filter approach: Image noise reduction with sharpening". In: *Proc. Digital Photography II*. Vol. 6069. SPIE, 2006, 60690F:1–7. doi: 10.1117/12.643880 (cit. on p. 65).

[AM94] L. Alvarez and L. Mazorra. "Signal and image restoration using shock filters and anisotropic diffusion". In: *SIAM Journal on Numerical Analysis* 31.2 (1994), pp. 590–605. doi: 10.1137/0731032 (cit. on p. 132).

[And+99] E. Anderson et al. *LAPACK Users' Guide*. 3rd ed. SIAM, 1999 (cit. on p. 30).

[AW95] V. Aurich and J. Weule. "Non-linear gaussian filters performing edge preserving diffusion". In: *Proc. DAGM-Symposium*. 1995, pp. 538–545 (cit. on pp. 68, 69).

[BA83] P. Burt and E. Adelson. "The laplacian pyramid as a compact image code". In: *IEEE Transactions on Communications* 31.4 (1983), pp. 532–540. doi: 10.1109/TCOM.1983.1095851 (cit. on p. 116).

[Ban+12] F. Banterle, M. Corsini, P. Cignoni, and R. Scopigno. "A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain". In: *Computer Graphics Forum* 31.1 (2012), pp. 19–32. doi: 10.1111/j.1467-_8659.2011.02078.x (cit. on p. 70).

[Bar02] D. Barash. "A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.6 (2002), pp. 844–847. doi: 10.1109/TPAMI.2002.1008390 (cit. on pp. 70, 71).

[BC04] D. Barash and D. Comaniciu. "A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift". In: *Image and Vision Computing* 22.1 (2004), pp. 73–81. doi: 10.1016/j.imavis.2003.08.005 (cit. on p. 68).

[BGH03] J. A. Bangham, S. E. Gibson, and R. Harvey. "The art of scale-space". In: *British Machine Vision Conference*. 2003, 58:1–10. doi: 10.5244/C.17.58 (cit. on p. 94).

[BGW91] J. Bigün, G. H. Granlund, and J Wiklund. "Multidimensional orientation estimation with applications to texture analysis and optical flow". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.8 (1991), pp. 775–790. doi: 10.1109/34.85668 (cit. on pp. 11, 38).

[Bha+10] P. Bhat, C. L. Zitnick, M. F. Cohen, and B. Curless. "GradientShop: A gradient-domain optimization framework for image and video filtering". In: *ACM Transactions on Graphics* 29.2 (2010), 10:1–14. doi: 10.1145/1731047.1731048 (cit. on pp. 140, 142).

[BHM00] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, 2000 (cit. on p. 54).

[Big06] J. Bigün. *Vision with Direction*. Springer, 2006 (cit. on pp. 11, 38).

[Bis07] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007 (cit. on p. 32).

[Bli05] J. F. Blinn. "Jim Blinn's corner: How to solve a quadratic equation". In: *IEEE Computer Graphics and Applications* 25.6 (2005), pp. 76–79. doi: 10.1109/MCG.2005.134 (cit. on p. 29).

[Bli89] J. F. Blinn. "Jim Blinn's corner: Return of the jaggy". In: *IEEE Computer Graphics and Applications* 9.2 (1989), pp. 82–89. doi: 10.1109/38.19054 (cit. on p. 117).

[Bli96] J. F. Blinn. "Jim Blinn's corner: Consider the lowly 2x2 matrix". In: *IEEE Computer Graphics and Applications* 16 (1996), pp. 82–88. doi: 10.1109/38.486688 (cit. on p. 22).

[BM05] E. P. Bennett and L. McMillan. "Video enhancement using per-pixel virtual exposures". In: *ACM Transactions on Graphics* 24.3 (2005), p. 845. doi: 10.1145/1073204.1073272 (cit. on p. 65).

[Boo02] R van den Boomgaard. "Decomposition of the Kuwahara-Nagao operator in terms of linear smoothing and morphological sharpening". In: *Proc. of the 6th Interational Symposium on Mathematical Morphology*. CSIRO Publishing, 2002, pp. 283–292 (cit. on p. 98).

[Bou+06] A. Bousseau, M. Kaplan, J. Thollot, and F. X. Sillion. "Interactive watercolor rendering with temporal coherence and abstraction". In: *Proc. NPAR*. 2006, pp. 141–149. doi: 10.1145/1124728.1124751 (cit. on pp. 94, 95).

[Bou+07] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin. "Video watercolorization using bidirectional texture advection". In: *ACM Transactions on Graphics* 26.3 (2007), 104:1–7. doi: 10.1145/1276377.1276507 (cit. on pp. 1, 94, 95).

[BPD06] S. Bae, S. Paris, and F. Durand. "Two-scale tone management for photographic look". In: *ACM Transactions on Graphics* 25.3 (2006), pp. 637–645. doi: 10.1145/1141911.1141935 (cit. on p. 65).

[BPT88] M. Bertero, T. Poggio, and V. Torre. "Ill-posed problems in early vision". In: *Proceedings of the IEEE* 76.8 (1988), pp. 869–889. doi: 10.1109/5.5962 (cit. on p. 15).

[Bro+06] T. Brox, R. van den Boomgaard, F. Lauze, J. van de Weijer, J. Weickert, P. Mrázek, and P. Kornprobst. "Adaptive structure tensors and their applications". In: *Visualization and Processing of Tensor Fields*. Springer, 2006, pp. 17–47. doi: 10.1007/3-_540-_31272-_2_2 (cit. on pp. 11, 35).

[Bro11] M. M. Bronstein. "Lazy sliding window implementation of the bilateral filter on parallel architectures". In: *IEEE Transactions on Image Processing* 20.6 (2011), pp. 1751–6. doi: 10.1109/TIP.2010.2095020 (cit. on p. 70).

[BVV99] P Bakker, L. J. van Vliet, and P. W. Verbeek. "Edge preserving orientation adaptive filtering". In: *Proc. CVPR*. Vol. 1. 1999, pp. 1535–1540. doi: 10.1109/CVPR.1999.786989 (cit. on pp. 45, 97).

[Can86] J. F. Canny. "A computational approach to edge detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.6 (1986), pp. 679–698. doi: 10.1109/TPAMI.1986.4767851 (cit. on pp. 82, 83, 88).

[CH05] J. Collomosse and P. M. Hall. "Genetic paint: A search for salient paintings". In: *Proc. EvoMUSART*. 2005, pp. 437–447. doi: 10.1007/978-_3-_540-_32003-_6_44 (cit. on p. 58).

[CHM87] J. S. Chen, A. Huertas, and G. Medioni. "Fast convolution with laplacian-of-gaussian masks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.4 (1987), pp. 584–590. doi: 10.1109/TPAMI.1987.4767946 (cit. on p. 85).

[CL93] B. Cabral and L. C. Leedom. “Imaging vector fields using line integral convolution”. In: *Proc. SIGGRAPH*. 1993, pp. 263–270. doi: 10.1145/166117.166151 (cit. on p. 61).

[CM02] D. Comaniciu and P. Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 603–619. doi: 10.1109/34.1000236 (cit. on p. 120).

[CPD07] J. Chen, S. Paris, and F. Durand. “Real-time edge-aware image processing with the bilateral grid”. In: *ACM Transactions on Graphics* 26.3 (2007), p. 103. doi: 10.1145/1276377.1276506 (cit. on p. 70).

[CRH05] J. P. Collomosse, D. Rowntree, and P. M. Hall. “Stroke surfaces: Temporally coherent non-photorealistic animations from video”. In: *IEEE Transactions on Visualization and Computer Graphics* 11.5 (2005), pp. 540–549. doi: 10.1109/TVCG.2005.85 (cit. on p. 2).

[Cri+10] A. Criminisi, T. Sharp, C. Rother, P. P’erez, and P. Pérez. “Geodesic image and video editing”. In: *ACM Transactions on Graphics* 29.5 (2010), 134:1–15. doi: 10.1145/1857907.1857910 (cit. on p. 95).

[CSU11] K Chaudhury, D Sage, and M Unser. “Fast O(1) bilateral filtering using trigonometric range kernels”. In: *IEEE Transactions on Image Processing* 20.12 (2011), pp. 3376–3382. doi: 10.1109/TIP.2011.2159234 (cit. on p. 70).

[Cum91] A. Cumani. “Edge detection in multispectral images”. In: *CVGIP: Graphical Models and Image Processing* 53.1 (1991), pp. 40–51. doi: 10.1016/1049-_9652(91)90018-_F (cit. on p. 22).

[Cur+97] C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. “Computer-generated watercolor”. In: *Proc. SIGGRAPH*. 1997, pp. 421–430. doi: 10.1145/258734.258896 (cit. on p. 2).

[DB08] P. Deuflhard and F. Bornemann. *Numerische Mathematik 2 - Gewöhnliche Differentialgleichungen*. De Gruyter, 2008 (cit. on p. 59).

[DD02] F. Durand and J. Dorsey. “Fast bilateral filtering for the display of high-dynamic-range images”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 257–266. doi: 10.1145/566654.566574 (cit. on pp. 65, 69, 70).

[DHK12] J. Döllner, B. Hagedorn, and J. Klimke. “Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps”. In: *Proc. International Conference on 3D Web Technology (Web3D)*. 2012, pp. 97–100. doi: 10.1145/2338714.2338729 (cit. on pp. 7, 9).

[DR07] D. DeCarlo and S. Rusinkiewicz. “Highlight lines for conveying shape”. In: *Proc. NPAR*. 2007, pp. 63–70. doi: 10.1145/1274871.1274881 (cit. on p. 1).

[Dra67] A. W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967 (cit. on p. 48).

[DS02] D. DeCarlo and A. Santella. “Stylization and abstraction of photographs”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 769–776. doi: 10.1145/566654.566650 (cit. on p. 83).

[DW07] S. Didas and J. Weickert. “Combining curvature motion and edge-preserving denoising”. In: *Scale Space and Variational Methods in Computer Vision*. Ed. by F. Sgallari, A. Murli, and N. Paragios. Vol. 4485. LNCS. Springer, 2007, pp. 568–579. doi: 10.1007/978-_3-_540-_72823-_8 (cit. on p. 128).

[ED04] E. Eisemann and F. Durand. “Flash photography enhancement via intrinsic relighting”. In: *ACM Transactions on Graphics* 23.3 (2004), p. 673. doi: 10.1145/1015706.1015778 (cit. on p. 65).

[Ede88] A. Edelman. “Eigenvalues and condition numbers of random matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 9.4 (1988), p. 543. doi: 10.1137/0609045 (cit. on p. 46).

[Ede89] A. Edelman. “Eigenvalues and condition numbers of random matrices”. PhD thesis. Massachusetts Institute of Technology, 1989 (cit. on p. 46).

[EHN96] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Springer, 1996 (cit. on p. 15).

[Ela05] M. Elad. “Scale space and PDE methods in computer vision”. In: *Scale Space and PDE Methods in Computer Vision*. Vol. 3459. LNCS. Springer, 2005, pp. 217–229. doi: 10.1007/11408031_19 (cit. on p. 65).

[Eng+12] J. Engel, S. Pasewaldt, M. Trapp, and J. Döllner. “An immersive visualization system for virtual 3D city models”. In: *Proc. International Conference on Geoinformatics*. 2012, pp. 1–7. doi: 10.1109/Geoinformatics.2012.6270289 (cit. on pp. 8, 9).

[ER05] A. Edelman and N. R. Rao. “Random matrix theory”. In: *Acta Numerica* 14 (2005), pp. 233–297. doi: 10.1017/S0962492904000236 (cit. on p. 46).

[FA91] W. T. Freeman and E. H. Adelson. “The design and use of steerable filters”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.9 (1991), pp. 891–906. doi: 10.1109/34.93808 (cit. on p. 11).

[Fab+08] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno. “2D Euclidean distance transform algorithms”. In: *ACM Computing Surveys* 40.1 (2008), 2:1–44. doi: 10.1145/1322432.1322434 (cit. on p. 95).

[Far02] G. Farnebäck. “Polynomial expansion for orientation and motion estimation”. PhD Thesis. Linköping University, Sweden, 2002 (cit. on p. 11).

[FBSe05] J. Fischer, D. Bartz, and W. Straß er. “Stylized augmented reality for improved immersion”. In: *Proc. IEEE Virtual Reality*. 2005, pp. 195–202. doi: 10.1109/VR.2005.71 (cit. on pp. 66, 91).

[Fen03] X. Feng. “Analysis and approaches to image local orientation estimation”. MA thesis. University of California Santa Cruz, 2003 (cit. on pp. 38, 45, 48).

[FFS06] M. Felsberg, P.-E. Forssén, and H. Scharr. “Channel smoothing: Efficient robust smoothing of low-level signal features”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.2 (2006), pp. 209–22. doi: 10.1109/TPAMI.2006.29 (cit. on p. 70).

[FG87] W. Förstner and E Gülch. “A fast operator for detection and precise location of distinct points, corners and centers of circular features”. In: *Proceedings of the ISPRS Intercommission Workshop on Fast Processing of Photogrammetric Data*. 1987, pp. 281–305 (cit. on pp. 45, 50).

[Fis08] G. Fischer. *Lineare Algebra*. 16th ed. Vieweg+Teubner, 2008 (cit. on p. 22).

[FM02] X. Feng and P. Milanfar. “Multiscale principal components analysis for image local orientation estimation”. In: *Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*. IEEE, 2002, pp. 478–482. doi: 10.1109/ACSSC.2002.1197228 (cit. on pp. 11, 38, 45, 48, 55).

[FS04] H. Farid and E. Simoncelli. “Differentiation of discrete multidimensional signals”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 496–508. doi: 10.1109/TIP.2004.823819 (cit. on p. 19).

[FS97] H. Farid and E. Simoncelli. “Optimally rotation-equivariant directional derivative kernels”. In: *Computer Analysis of Images and Patterns, 7th International Conference, CAIP ’97*. Vol. 1296. Lecture Notes in Computer Science. Springer, 1997, pp. 207–214. doi: 10.1007/3-_540-_63460-_6 (cit. on p. 19).

[För86] W. Förstner. “A feature based correspondence algorithm for image matching”. In: *International Archives of Photogrammetry and Remote Sensing* 26.3 (1986), pp. 150–166 (cit. on pp. 11, 31, 45, 50).

[Gau02] K. F. Gauss. *General investigations of curved surfaces of 1827 and 1825*. Trans. by J. C. Morehead and A. M. Hiltebeitel. The Princeton University Library, 1902 (cit. on p. 25).

[GG01] B. Gooch and A. A. Gooch. *Non-photorealistic Rendering*. AK Peters, 2001 (cit. on p. 1).

[Gin+99] M. van Ginkel, J. van de Weijer, L. van Vliet, and P. Verbeek. "Curvature estimation from orientation fields". In: *Proc. Scandinavian Conference on Image Analysis (SCIA)*. 1999, pp. 545–551 (cit. on p. 71).

[GK95] G. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer, 1995 (cit. on pp. 11, 39).

[GM03] F. Guichard and J. M. Morel. "A note on two classical enhancement filters and their associated PDE's". In: *International Journal of Computer Vision* 52.2 (2003), pp. 153–160. doi: 10.1023/A:1022904124348 (cit. on p. 132).

[Gol91] D. Goldberg. "What every computer scientist should know about floating-point arithmetic". In: *ACM Computing Surveys* 23.1 (1991), pp. 5–48. doi: 10.1145/103162.103163 (cit. on p. 29).

[Gra78] G. Granlund. "In search of a general picture processing operator". In: *Computer Graphics and Image Processing* 8.2 (1978), pp. 155–173. doi: 10.1016/0146-_664X(78)90047-_3 (cit. on p. 33).

[Gra87] M. A. Grayson. "The heat equation shrinks embedded plane curves to round points". In: *Journal of Differential Geometry* 26.2 (1987), pp. 285–314 (cit. on p. 129).

[GRG04] B. Gooch, E. Reinhard, and A. A. Gooch. "Human facial illustrations: Creation and psychophysical evaluation". In: *ACM Transactions on Graphics* 23.1 (2004), pp. 27–44. doi: 10.1145/966131.966133 (cit. on p. 81).

[Gun11] B. K. Gunturk. "Fast bilateral filter with arbitrary range and domain kernels." In: *IEEE Transactions on Image Processing* 20.9 (2011), pp. 2690–2696. doi: 10.1109/TIP.2011.2126585 (cit. on p. 70).

[GVL96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. 3rd ed. Johns Hopkins University Press, 1996 (cit. on p. 37).

[GW06] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2006 (cit. on pp. 31, 84).

[Hae90] P. Haeberli. "Paint by numbers: Abstract image representations". In: *Proc. SIGGRAPH*. Vol. 24. 3. ACM SIGGRAPH. 1990, pp. 207–214. doi: 10.1145/97880.97902 (cit. on pp. 2, 57).

[Har84] R. M. Haralick. "Digital step edges from zero crossing of second directional derivatives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.1 (1984), pp. 58–68. doi: 10.1109/TPAMI.1984.4767475 (cit. on p. 88).

[HE04] J. Hays and I. Essa. "Image and video based painterly animation". In: *Proc. NPAR*. 2004, pp. 113–120. doi: 10.1145/987657.987676 (cit. on pp. 2, 33, 57, 123).

[Her02] A. Hertzmann. "Fast paint texture". In: *Proc. NPAR*. Ed. by A. Finkelstein. 2002, pp. 91–96. doi: 10.1145/508530.508546 (cit. on p. 123).

[Her98] A. Hertzmann. "Painterly rendering with curved brush strokes of multiple sizes". In: *Proc. SIGGRAPH*. 1998, pp. 453–460. doi: 10.1145/280814.280951 (cit. on pp. 2, 57).

[Hig96] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996 (cit. on p. 29).

[HJ85] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985 (cit. on pp. 22, 25, 27, 37).

[HS88] C Harris and M Stephens. "A combined corner and edge detector". In: *Proceedings of The Fourth Alvey Vision Conference*. 1988, pp. 147–151 (cit. on pp. 31, 49).

[HSZ87] R. M. Haralick, S. R. Sternberg, and X. Zhuang. "Image analysis using mathematical morphology". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 9.4 (1987), pp. 532–550. doi: 10.1109/TPAMI.1987.4767941 (cit. on p. 94).

[IU04] K. Inoue and K. Urahama. "Anisotropic shock filter for enhancement and non-photorealistic generation of flow patterns". In: *Proc. IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*. 2004, pp. 277–280. doi: 10.1109/MWSCAS.2004.1353981 (cit. on p. 6).

[JG97] J. S. Jin and Y. Gao. “Recursive implementation of LoG filtering”. In: *Real-Time Imaging* 3.1 (1997), p. 7. doi: 10.1006/rtim.1996.0045 (cit. on p. 85).

[Job08] M. Jobst. “Ein semiotisches Modell für die kartografische Kommunikation mit 3D”. PhD thesis. Technische Universität Wien, 2008 (cit. on p. 9).

[JSK99] B. Jähne, H. Scharr, and S. Körkel. “Principles of filter design”. In: *Handbook of Computer Vision and Applications*. Vol. 2. Academic Press, 1999, pp. 125–151 (cit. on pp. 19, 20).

[Jäh05] B. Jähne. *Digital Image Processing*. 6th ed. Springer, 2005 (cit. on pp. 19, 39).

[Jän08] K. Jänich. *Lineare Algebra*. 11th ed. Springer, 2008. doi: 10.1007/978-_3-_540-_75502-_9 (cit. on p. 22).

[Kag+11] M. Kagaya, W. Brendel, Q. Deng, T. Kesterson, S. Todorovic, P. J. Neill, and E. Zhang. “Video painting with space-time-varying style parameters.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.1 (2011), pp. 74–87. doi: 10.1109/TVCG.2010.25 (cit. on p. 61).

[KB75] H. P. Kramer and J. B. Bruckner. “Iterations of a non-linear transformation for enhancement of digital images”. In: *Pattern recognition* 7.1–2 (1975), pp. 53–58. doi: 10.1016/0031-_3203(75)90013-_8 (cit. on pp. 98, 132).

[KDA97] P Kornprobst, R Deriche, and G Aubert. “Non-linear operators in image restoration”. In: *Proc. CVPR*. 1997, pp. 325–331. doi: 10.1109/CVPR.1997.609344 (cit. on p. 67).

[Kin82] D. King. *Implementation of the Marr-Hildreth theory of edge detection*. Tech. rep. Univ. Southern California, 1982 (cit. on p. 85).

[KL08] H. Kang and S. Lee. “Shape-simplifying image abstraction”. In: *Computer Graphics Forum* 27.7 (2008), pp. 1773–1780. doi: 10.1111/j.1467-_8659.2008.01322.x (cit. on pp. 125–128, 138, 140, 141).

[KLC07] H. Kang, S. Lee, and C. K. Chui. “Coherent line drawing”. In: *Proc. NPAR*. 2007, pp. 43–50. doi: 10.1145/1274871.1274878 (cit. on pp. 11, 43, 88, 89, 91).

[KLC09] H. Kang, S. Lee, and C. K. Chui. “Flow-based image abstraction”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.1 (2009), pp. 62–76. doi: 10.1109/TVCG.2008.81 (cit. on pp. 5, 11, 43, 44, 89).

[KN09] H. Knutsson and K. Nordberg. *EDUPACK: Orientation*. Tech. rep. Linköping University, Sweden, 2009 (cit. on p. 39).

[Knu89] H Knutsson. “Representing local structure using tensors”. In: *Proc. Scandinavian Conference on Image Analysis (SCIA)*. 1989, pp. 244–251 (cit. on pp. 11, 39).

[Kop+07] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. “Joint bilateral upsampling”. In: *ACM Transactions on Graphics* 26.3 (2007), p. 96. doi: 10.1145/1276377.1276497 (cit. on p. 65).

[Kuw+76] M Kuwahara, K Hachimura, S Ehiu, and M Kinoshita. “Processing of ri-angiocardiographic images”. In: *Digital Processing of Biomedical Images*. Ed. by K. Preston and M. Onoe. Plenum Press, 1976, pp. 187–203 (cit. on pp. 93, 96).

[KW87] M. Kass and A. Witkin. “Analyzing oriented patterns”. In: *Computer Vision, Graphics, and Image Processing* 37.3 (1987), pp. 362–385. doi: 10.1016/0734-_189X(87)90043-_0 (cit. on pp. 31, 39).

[Kön04] K. Königsberger. *Analysis 1+2*. Springer, 2004 (cit. on pp. 14, 16).

[Köt03a] U. Köthe. “Edge and junction detection with an improved structure tensor”. In: *Pattern Recognition*. Vol. 2781. Springer, 2003, pp. 25–32. doi: 10.1007/978-_3-_540-_45243-_0_4 (cit. on p. 41).

[Köt03b] U. Köthe. *Gradient-based segmentation requires doubling of the sampling rate*. Tech. rep. FBI-HH-M 326/03. University of Hamburg, 2003 (cit. on p. 41).

[Köt08] U. Köthe. *Reliable low-level image analysis*. University of Hamburg, 2008 (cit. on p. 41).

[Lan74] M. J. Langford. *Advanced Photography: A Grammar of Techniques*. Focal Press, 1974 (cit. on p. 88).

[Lan87] S. Lang. *Linear Algebra*. 3rd ed. Springer, 1987 (cit. on p. 22).

[Lee03] J. M. Lee. *Introduction to smooth manifolds*. Springer, 2003 (cit. on p. 14).

[Lee80] J.-S. Lee. "Digital image enhancement and noise filtering by use of local statistics". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.2 (1980), pp. 165–168. doi: 10.1109/TPAMI.1980.4766994 (cit. on p. 93).

[Lee81] J.-S. Lee. "Refined filtering of image noise using local statistics". In: *Computer Graphics and Image Processing* 15 (1981), pp. 380–389. doi: 10.1016/S0146-_664X(81)80018-_4 (cit. on p. 93).

[Li+08] W. Li, M. Agrawala, B. Curless, and D. Salesin. "Automated generation of interactive 3D exploded view diagrams". In: *ACM Transactions on Graphics* 27.3 (2008), 101:1–7. doi: 10.1145/1360612.1360700 (cit. on p. 1).

[Lim90] J. S. Lim. *Two-dimensional Signal and Image Processing*. Prentice Hall, 1990 (cit. on p. 85).

[Lin94a] T. Lindeberg. "Scale-space theory: A basic tool for analyzing structures at different scales". In: *Journal of Applied Statistics* 21.1 (1994), pp. 225–270. doi: 10.1080/757582976 (cit. on p. 85).

[Lin94b] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994 (cit. on pp. 116, 117).

[Lit97] P. Litwinowicz. "Processing images and video for an impressionist effect". In: *Proc. SIGGRAPH*. 1997, pp. 407–414. doi: 10.1145/258734.258893 (cit. on pp. 2, 57, 123).

[Liu+06] C. Liu, W. T. Freeman, R. Szeliski, and S. B. Kang. "Noise estimation from a single image". In: *Proc. CVPR*. 2006, pp. 901–908. doi: 10.1109/CVPR.2006.207 (cit. on p. 65).

[LKL06] H. Lee, S. Kwon, and S. Lee. "Real-time pencil rendering". In: *Proc. NPAR*. 2006, pp. 37–45. doi: 10.1145/1124728.1124735 (cit. on p. 1).

[Low04] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. doi: 10.1023/B:VISI.0000029664.99615.94 (cit. on p. 85).

[Mar+01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A Database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Proc. ICCV*. 2001, pp. 416–423. doi: 10.1109/ICCV.2001.937655 (cit. on p. 46).

[Mei96] B. J. Meier. "Painterly rendering for animation". In: *Proc. SIGGRAPH*. 1996, pp. 477–484. doi: 10.1145/237170.237288 (cit. on p. 1).

[MG08] D. Mould and K. Grant. "Stylized black and white images from photographs". In: *Proc. NPAR*. 2008, pp. 49–58. doi: 10.1145/1377980.1377991 (cit. on p. 1).

[MH80] D Marr and R. C. Hildreth. "Theory of edge detection". In: *Proc. Royal Society London* 207.1167 (1980), pp. 187–217. doi: 10.1098/rspb.1980.0020 (cit. on pp. 81, 83, 85).

[MJ99] K. V. Mardia and P. E. Jupp. *Directional Statistics*. 2nd. Wiley, 1999 (cit. on pp. 32, 33).

[MS87a] P. Maragos and R. Schafer. "Morphological filters—Part I: Their set-theoretic analysis and relations to linear shift-invariant filters". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.8 (1987), pp. 1153–1169. doi: 10.1109/TASSP.1987.1165259 (cit. on p. 94).

[MS87b] P. Maragos and R. Schafer. "Morphological filters—Part II: Their relations to median, order-statistic, and stack filters". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.8 (1987), pp. 1170–1184. doi: 10.1109/TASSP.1987.1165254 (cit. on p. 94).

[MWB06] P. Mrázek, J. Weickert, and A. Bruhn. "On robust estimation and smoothing with spatial and tonal kernels". In: *Geometric Properties for Incomplete Data*. Ed. by R. Klette, R. Kozera, L. Noakes, and J. Weickert. Springer, 2006, pp. 335–352. doi: 10.1007/1-_4020-_3858-_8_18 (cit. on p. 68).

[NM79] M Nagao and T Matsuyama. “Edge preserving smoothing”. In: *Computer Graphics and Image Processing* 9 (1979), pp. 394–407. doi: 10.1016/0146-_664X(79)90102-_3 (cit. on pp. 93, 97).

[Nob88] J. A. Noble. “Finding corners”. In: *Image and Vision Computing* 6.2 (1988), pp. 121–128. doi: 10.1016/0262-_8856(88)90007-_8 (cit. on p. 50).

[Oh+01] B. M. Oh, M. Chen, J. Dorsey, and F. Durand. “Image-based modeling and photo editing”. In: *Proc. SIGGRAPH*. 2001, pp. 433–442. doi: 10.1145/383259.383310 (cit. on p. 65).

[OR90] S. Osher and L. I. Rudin. “Feature-oriented image enhancement using shock filters”. In: *SIAM Journal on Numerical Analysis* 27.4 (1990), pp. 919–940. doi: 10.1137/0727053 (cit. on pp. 127, 132).

[Orz+07] A. Orzan, A. Bousseau, P. Barla, and J. Thollot. “Structure-preserving manipulation of photographs”. In: *Proc. NPAR*. 2007, pp. 103–110. doi: 10.1145/1274871.1274888 (cit. on p. 83).

[OS75] A. V. Oppenheim and R. W. Schafer. *Digital signal processing*. Prentice-Hall, 1975 (cit. on p. 85).

[OS88] S Osher and J. A. Sethian. “Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations”. In: *Journal of Computational Physics* 79.1 (1988), pp. 12–49. doi: 10.1016/0021-_9991(88)90002-_2 (cit. on p. 133).

[Par+09] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand. “Bilateral filtering: Theory and applications”. In: *Foundations and Trends in Computer Graphics and Vision* 4.1 (2009), pp. 1–73. doi: 10.1561/0600000020 (cit. on pp. 65, 69).

[Par08] S. Paris. “Edge-preserving smoothing and mean-shift segmentation of video streams”. In: *Proc. ECCV*. Springer, 2008, pp. 460–473. doi: 10.1007/978-_3-_540-_88688-_4_34 (cit. on p. 125).

[PD06] S. Paris and F. Durand. “A fast approximation of the bilateral filter using a signal processing approach”. In: *European Conference on Computer Vision*. Vol. 4. 2006, pp. 568–580 (cit. on p. 70).

[Pen04] X. Pennec. “Probabilities and statistics on Riemannian manifolds: Basic tools for geometric measurements”. In: *Proc. Nonlinear Signal and Image Processing (NSIP)*. 2004, pp. 194–198 (cit. on p. 32).

[PH02] S.-C. Pei and J.-H. Horng. “Design of FIR bilevel Laplacian-of-Gaussian filter”. In: *Signal Processing* 82.4 (2002), pp. 677–691. doi: 10.1016/S0165-_1684(02)00136-_6 (cit. on p. 85).

[Pha06] T. Q. Pham. “Spatiotonal adaptivity in super-resolution of undersampled image sequences”. PhD thesis. Quantitative Imaging Group, Delft University of Technology, 2006 (cit. on pp. 45, 70, 105).

[PM90] P Perona and J Malik. “Scale-space and edge detection using anisotropic diffusion”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pp. 629–639. doi: 10.1109/34.56205 (cit. on pp. 66, 67).

[Pod07] V. Podlozhnyuk. *FFT-based 2D convolution*. Tech. rep. NVIDIA, 2007 (cit. on p. 101).

[Por08] F. Porikli. “Constant time O(1) bilateral filtering”. In: *Proc. CVPR*. 2008, pp. 1–8. doi: 10.1109/CVPR.2008.4587843 (cit. on p. 70).

[PP09] G. Papari and N. Petkov. “Continuous glass patterns for painterly rendering”. In: *IEEE Transactions on Image Processing* 18.3 (2009), pp. 652–64. doi: 10.1109/TIP.2008.2009800 (cit. on p. 94).

[PPC07] G. Papari, N. Petkov, and P. Campisi. “Artistic edge and corner enhancing smoothing”. In: *IEEE Transactions on Image Processing* 16.10 (2007), pp. 2449–2462. doi: 10.1109/TIP.2007.903912 (cit. on pp. 93, 94, 98, 101, 118).

[PR11] T. Pouli and E. Reinhard. “Progressive color transfer for images of arbitrary dynamic range”. In: *Computers & Graphics* 35.1 (2011), pp. 67–80. doi: 10.1016/j.cag.2010.11.003 (cit. on p. 1).

[Pra01] W. K. Pratt. *Digital Image Processing*. 3rd ed. John Wiley & Sons, Inc., 2001. doi: 10.1002/0471221325 (cit. on pp. 18, 82).

[Pre+07] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes*. 3rd ed. Cambridge Unversity Press, 2007 (cit. on pp. 29, 38, 59).

[PV05] T. Q. Pham and L. J. van Vliet. "Separable bilateral filtering for fast video preprocessing". In: *IEEE International Conference on Multimedia and Expo*. 2005. doi: 10.1109/ICME.2005.1521458 (cit. on p. 70).

[PVS06] T. Q. Pham, L. J. van Vliet, and K. Schutte. "Robust fusion of irregularly sampled data using adaptive normalized convolution". In: *EURASIP Journal on Advances in Signal Processing* 2006 (2006), pp. 1–13. doi: 10.1155/ASP/2006/83268 (cit. on p. 105).

[Rao90] A. R. Rao. *A Taxonomy for Texture Description and Identification*. Springer, 1990 (cit. on pp. 31, 37).

[RC12] P. L. Rosin and J. Collomosse, eds. *Image and Video-based Artistic Stylization*. Springer, 2012. doi: 10.1007/978-_1-_4471-_4519-_6 (cit. on p. 1).

[Rei+01] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley. "Colour transfer between images". In: *IEEE Computer Graphics and Applications* 21 (2001), pp. 34–41. doi: 10.1109/38.946629 (cit. on p. 1).

[Ric08] S. Richter. "Computergenerierte Aquarellillustration von 3D-Stadtmodellen". Bachelor's Thesis. Hasso-Plattner-Institut, 2008 (cit. on pp. 8, 9).

[Roh87] K. Rohr. "Untersuchung von grauwertabhängigen Transformationen zur Ermittlung der optischen Flusses in Bildfolgen". mastersthesis. Universität Karlsruhe, 1987 (cit. on p. 50).

[Roh94] K. Rohr. "Localization properties of direct corner detectors". In: *Journal of Mathematical Imaging and Vision* 4.2 (1994), pp. 139–150. doi: 10.1007/BF01249893 (cit. on p. 50).

[RS89] A. Rao and B. Schunck. "Computing oriented texture fields". In: *Proc. CVPR*. 1989, pp. 61–68. doi: 10.1109/CVPR.1989.37829 (cit. on p. 31).

[RS91] A. Rao and B. G. Schunck. "Computing oriented texture fields". In: *CVGIP: Graphical Models and Image Processing* 53.2 (1991), pp. 157–185. doi: 10.1016/1049-_9652(91)90059-_S (cit. on p. 37).

[Sal+97] M. Salisbury, M. T. Wong, J. F. Hughes, and D. Salesin. "Orientable textures for image-based pen-and-ink illustration". In: *Proc. SIGGRAPH*. 1997, pp. 401–406. doi: 10.1145/258734.258890 (cit. on p. 2).

[SB97] S. M. Smith and J. M. Brady. "SUSAN—A new approach to low level image processing". In: *International Journal of Computer Vision* 23.1 (1997), pp. 45–78. doi: 10.1023/A:1007963824710 (cit. on p. 68).

[SBv05] D. Sýkora, J. Buriánek, and J. Žára. "Colorization of black-and-white cartoons". In: *Image and Vision Computing* 23.9 (2005), pp. 767–782. doi: 10.1016/j.imavis.2005.05.010 (cit. on p. 81).

[Sch00] H. Scharr. "Optimale Operatoren in der Digitalen Bildverarbeitung". PhD thesis. Ruprecht-Karls-Universität Heidelberg, 2000 (cit. on p. 19).

[Sch92] D. Schumacher. "General filtered image rescaling". In: *Graphics Gems III*. Ed. by D. Kirk. Academic Press, 1992, pp. 8–16 (cit. on pp. 42, 116).

[SH95] D. Stalling and H.-C. Hege. "Fast and resolution independent line integral convolution". In: *Proc. SIGGRAPH*. Ed. by R. Cook. 1995, pp. 249–256. doi: 10.1145/218380.218448 (cit. on p. 61).

[Sim94] E. Simoncelli. "Design of multi-dimensional derivative filters". In: *Proc. ICIP*. Vol. 1. 1994, pp. 790–794. doi: 10.1109/ICIP.1994.413423 (cit. on p. 19).

[SKJ97] H Scharr, S Körkel, and B Jähne. "Numerische Isotropieoptimierung von FIR-Filtern mittels Querglättung". In: *Mustererkennung 1997*. 1997, pp. 367–374 (cit. on p. 19).

[SP93] M. A. Schulze and J. A. Pearce. “Value-and-criterion filters: A new filter structure based on morphological opening and closing”. In: *Proc. of SPIE (Nonlinear Image Processing IV)*. Vol. 1902. SPIE, 1993, pp. 106–115. doi: 10.1117/12.144746 (cit. on p. 95).

[SP94] M. Schulze and J. Pearce. “A morphology-based filter structure for edge-enhancing smoothing”. In: *Proc. ICIP*. 1994, pp. 530–534. doi: 10.1109/ICIP.1994.413627 (cit. on pp. 93, 95).

[SS02] T. Strothotte and S. Schlechtweg. *Non-photorealistic Computer Graphics*. Morgan Kaufmann, 2002. doi: 10.1145/544522 (cit. on p. 1).

[ST08] P. Sand and S. Teller. “Particle video: Long-range motion estimation using point trajectories”. In: *International Journal of Computer Vision* 80.1 (2008), pp. 72–91. doi: 10.1007/s11263-_008-_0136-_6 (cit. on p. 65).

[TC97] S. M. F. Treavett and M. Chen. “Statistical techniques for the automated synthesis of non-photorealistic images”. In: *Proc. EGUK*. 1997, pp. 201–210 (cit. on p. 57).

[TK91] C. Tomasi and T. Kanade. *Shape And motion from image streams: A factorization method - part 3 detection and tracking of point features*. Tech. rep. Carnegie Mellon University, 1991 (cit. on p. 50).

[TM98] C Tomasi and R Manduchi. “Bilateral filtering for gray and color images”. In: *Proc. ICCV*. 1998, pp. 839–846. doi: 10.1109/ICCV.1998.710815 (cit. on pp. 68, 69, 120, 126).

[TP84] V. Torre and T. Poggio. *On edge detection*. Tech. rep. Massachusetts Institute of Technology, 1984 (cit. on p. 18).

[TP86] V. Torre and T. A. Poggio. “On edge detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.2 (1986), pp. 147–163. doi: 10.1109/TPAMI.1986.4767769 (cit. on p. 88).

[Tsc06] D Tschumperlé. “Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE’s”. In: *International Journal of Computer Vision* 68.1 (2006), pp. 65–82. doi: 10.1007/s11263-_006-_5631-_z (cit. on p. 67).

[TT77] F Tomita and S Tsuji. “Extraction of multiple regions by smoothing in selected neighborhoods”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 7.2 (1977), pp. 107–109. doi: 10.1109/TSMC.1977.4309664 (cit. on p. 93).

[VVV88] P. W. Verbeek, H. A. Vrooman, and L. J. van Vliet. “Low-level image processing by max-min filters”. In: *Signal Processing* 15.3 (1988), pp. 249–258. doi: 10.1016/0165-_1684(88)90015-_1 (cit. on p. 132).

[Wan+04] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. “Video tooning”. In: *ACM Transactions on Graphics* 23.3 (2004), pp. 574–583. doi: 10.1145/1015706.1015763 (cit. on p. 2).

[WB05] J. van de Weijer and R. van den Boomgaard. “Least squares and robust estimation of local image structure”. In: *International Journal of Computer Vision* 64.2–3 (2005), pp. 143–155. doi: 10.1007/s11263-_005-_1840-_0 (cit. on p. 35).

[WCB90] R. Wilson, S. C. Clippingdale, and A. H. Bhalerao. “Robust estimation of local orientations in images using a multiresolution approach”. In: *Visual Communications and Image Processing ’90*. SPIE, 1990, pp. 1393–1403. doi: 10.1117/12.24154 (cit. on p. 55).

[Wei03] J. Weickert. “Coherence-enhancing shock filters”. In: *DAGM-Symposium*. Springer, 2003, pp. 1–8. doi: 10.1007/978-_3-_540-_45243-_0_1 (cit. on pp. 125–127, 133, 134, 141).

[Wei06] B. Weiss. “Fast median and bilateral filtering”. In: *ACM Transactions on Graphics* 25.3 (2006), pp. 519–526. doi: 10.1145/1141911.1141918 (cit. on p. 70).

[Wei98] J. Weickert. *Anisotropic Diffusion in Image Processing*. Teubner-Verlag, Germany, 1998 (cit. on pp. 66, 67).

[Wei99] J. Weickert. “Coherence-enhancing diffusion of colour images”. In: *Image and Vision Computing* 17.3–4 (1999), pp. 201–212. doi: 10.1016/S0262-_8856(98)00102-_4 (cit. on pp. 4, 45, 67, 71).

[Wij02] J. J. van Wijk. "Image based flow visualization". In: *ACM Transactions on Graphics* 21.3 (2002), pp. 745–754. doi: 10.1145/566654.566646 (cit. on p. 61).

[Wik12] Wikipedia. *Expressionism*. 2012 (cit. on p. 4).

[Win12] H. Winnemöller. "NPR in the wild". In: *Image and Video-based Artistic Stylization*. Ed. by P. L. Rosin and J. Collomosse. Springer, 2012. doi: 10.1007/978-_1-_4471-_4519-_6_17 (cit. on p. 7).

[WOG06] H. Winnemöller, S. C. Olsen, and B. Gooch. "Real-time video abstraction". In: *ACM Transactions on Graphics* 25.3 (2006), pp. 1221–1226. doi: 10.1145/1141911.1142018 (cit. on pp. 2, 5, 66, 69, 70, 81, 83, 86, 91).

[WS82] G Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley Interscience Publishers, New York, 1982 (cit. on p. 69).

[Xia+06] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi. "Bilateral filtering-based optical flow estimation with occlusion detection". In: *Proc. ECCV*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Vol. 3951. Lecture Notes in Computer Science. Springer, 2006, pp. 211–224. doi: 10.1007/11744023 (cit. on p. 65).

[XK08] J. Xu and C. S. Kaplan. "Artistic thresholding". In: *Proc. NPAR*. 2008, pp. 39–47. doi: 10.1145/1377980.1377990 (cit. on p. 1).

[Yan+96] G. Z. Yang, P Burger, D. N. Firmin, and S. R. Underwood. "Structure adaptive anisotropic image filtering". In: *Image and Vision Computing* 14.2 (1996), pp. 135–145. doi: 10.1016/0262-_8856(95)01047-_5 (cit. on p. 45).

[Yar85] L. P. Yaroslavsky. *Digital Picture Processing*. Springer, 1985 (cit. on p. 69).

[YG05] R. D. Yates and D. Goodman. *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*. 2nd ed. Wiley, 2005 (cit. on p. 48).

[You87] R. A. Young. "The Gaussian derivative model for spatial vision: I. Retinal mechanisms". In: *Spatial Vision* 2.4 (1987), pp. 273–293. doi: 10.1163/156856887X00222 (cit. on pp. 85, 86).

[YTA09] Q. Yang, K.-H. Tan, and N. Ahuja. "Real-time O(1) bilateral filtering". In: *Proc. CVPR*. 2009, pp. 557–564. doi: 10.1109/CVPR.2009.5206542 (cit. on p. 70).

[Zen+09] K. Zeng, M. Zhao, C. Xiong, and S.-C. Zhu. "From image parsing to painterly rendering". In: *ACM Transactions on Graphics* 29.1 (2009), 2:1–11. doi: 10.1145/1640443.1640445 (cit. on pp. 1, 3, 58).

[ZHT07] E. Zhang, J. Hays, and G. Turk. "Interactive tensor field design and visualization on surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 13.1 (2007), pp. 94–107. doi: 10.1109/TVCG.2007.16 (cit. on p. 61).

[Di 86] S. Di Zenzo. "A note on the gradient of a multi-image". In: *Computer Vision, Graphics, and Image Processing* 33.1 (1986), pp. 116–125. doi: 10.1016/0734-_189X(86)90223-_9 (cit. on pp. 20, 31, 67).