Saarland University
Faculty of Natural Sciences
and Technology I
Department of Computer Science
Master's Program in Computer Science

Master's Thesis

PDE-based Image Compression using Corner Information

submitted by
Henning Lars Zimmer

on September 17, 2007

Supervisor
Prof. Dr. Joachim Weickert

Advisor
Prof. Dr. Joachim Weickert

Reviewers
Prof. Dr. Joachim Weickert
Dr.-Ing. Bodo Rosenhahn

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbruecken, September 17, 2007

Acknowledgments

First of all I have to thank Prof. Dr. Joachim Weickert. On the one hand for providing me with a really interesting and challenging topic for my thesis and on the other hand for being always friendly, helpful and accessible to me.

I am also very thankful to all the nice people at our chair for providing me with a nice working atmosphere and being kind and helpful all the time.

Of course, I also have to thank Dr.-Ing. Bodo Rosenhahn for being so kind and accepting my request to be my second reviewer.

Last but not least, I have to express my special thanks to my friends and family for their continuous support.

Contents

Abstract

A new general approach for image compression codecs – compared to existing ones like JPEG – is to store in the compression step only important image features like edges or corners. In the decompression step, one hence has to fill in the missing information, discarded in the compression step. We do this in our case by PDE-based inpainting.
In this thesis, we will present a variant of the above, where we store small neighborhoods of image corners in the compression step. The reasons for this approach are that these information are on the one hand easy to obtain by well-known corner detection algorithms and on the other hand quite efficient to store. The problem is now that the very limited number of corners in an image poses quite some challenges for the decompression step. So, it is difficult to strive for favorable decompression results, which may compete with codecs like JPEG or even its successor JPEG2000.
To enhance decompression results, we invest in a new kind of PDE-based inpainting, which does not only rely on Edge-Enhancing Diffusion (EED), which was up to now considered to be the best choice. We will actually interleave EED with Mean Curvature Motion (MCM), a morphological filter, which can also be written in PDE form. This interleaving will result in noticeably better decompression results, compared to pure EED inpainting.

Chapter 1

Introduction

To start this thesis, we first want to motivate our work and introduce the problem addressed to the reader. Finally, we shortly summarize the further organization of the upcoming chapters.

1.1 Need for Image Compression

In recent years, digital images are becoming more and more important. Digital cameras are now rather cheap and technically mature. As a consequence, digital images are replacing conventional analog images in almost every field. Examples range from holiday pictures to medical images, like x-ray tomography. So, there is a natural need to store images on a computer and also to transmit them over the internet, to share them with other persons.
The big problem is that digital images are quite memory-consuming, and so a need to compress images is also quite natural if one wants to store lots of images with a rather high resolution or wants to transmit these images via a channel with limited bandwidth. As an example, consider a RGB-color image with a resolution of 3072×2304 pixels, corresponding to 7 Megapixels, which is standard for modern digital cameras. With the usual byte-coding this image consumes $3 \cdot 3072 \cdot 2304 = 21233664$ byte $= 20.25$ MByte, if stored in a raw format. Although memory is getting cheaper and cheaper these days it is still not unlimited and also the number and –even more severe– the resolution of images one may want to store has increased in the last years. Furthermore, we should not forget that transferring images is still an issue, as the bandwidth of networks, may they be wireless or wired, is still quite limited, actually too limited to transfer images in a raw format, where they consume about 20 MByte of size.
Considering that, the importance of image compression methods should be directly obvious, and indeed there already exist quite some well-founded and established approaches, like the de-facto standard codec, the JPEG codec [43]. This codec is – like almost all practical useable compression algorithms – a so called *lossy compression algorithm*, i.e., the decompressed image will not be identical to the initial uncompressed image, its

subjective quality is usually poorer. Of course, there exist approaches for lossless compression, but still up to now the achieved compression rates are not competitive there.
Another thing to mention in this context is that image compression algorithms usually won't yield reasonable results for "random images", i.e., images where we consider the color value of each pixel as the outcome of a random experiment. The main reason why compression algorithms need "natural images" is the presence of image boundaries (edges) and flat regions (plateaus) there. The edges kind of subdivide the image into regions and hence are the semantically important parts. The idea is then to code only these for a human observer important parts and neglect the rest, which reduces the amount of data needed. We should note in this context that for random images, the raw format is actually the best coding one could achieve, but in practice such images will not play an important role.
Our idea – on which we will detail in the following section – for a new image compression codec is then loosely spoken not to store all color values in a flat region, which is kind of redundant, but to mainly store the semantically valuable information at the boundaries.

1.2 Problems and our Approach

As mentioned, there already exist well established approaches like JPEG, so one could ask why one should strive for developing a new image compression algorithm. The reason is that most of the classical image compressions are relatively "blind". For JPEG, this means for example that in the compression step, the image is chopped into 8×8 pixel blocks, which are transferred into a frequency domain (via a discrete cosine-transform, DCT, cf. Section 2.1.1). In this domain, one quantizes high frequencies, which are most likely less important than lower ones. This step also introduces the loss of information. The "blindness" of this approach is visible in the block-building step. Inside a flat region, this is a very good idea, but if the flat region would be larger, then building larger blocks would be more clever. The problem is even more severe when a block crosses an image boundary. Here, they actually destroy valuable image information and the infamous blocky artifacts of the JPEG compression appear.
For a more detailed discussion of the JPEG-standard and other compression algorithms, the reader may refer to Section 2.1.

So, a logical consequence in improving such algorithms is to be less blind. Therefore one uses semantic image information, the so called *image features*, like edges or corners, to decide which are the vital information contents of the image one wants to preserve in the compression step.
The approach this thesis is mainly inspired from is the one of Galic et al. [26], which relies on *PDE-based inpainting* using diffusion filters. The basic idea here is to only store the image features of interest –which introduces a loss of information, of course–, which is called *sparsification* of the image. This finally yields a so called *sparse image* (cf. Figure 1.1), which is black everywhere except in the locations of the stored features.

In the decompression step, one then has to smoothly (using *diffusion filters*, cf. Section 3.3.1) fill in the missing information, which is done via PDE-based inpainting algorithms (cf. Section 3.3.4), which were actually invented to restore images with small defects, like scratches. So, one sees that one kind of pushes the idea of inpainting to a new dimension, instead of just filling in missing information in a small damaged area of an image, one has to fill in most of the sparse image.
One big issue with these approaches is how to efficiently store the image features, which will be image corners in the case of this thesis. The main reason for this decision is that corners are easy to obtain from well-known corner detection algorithms (cf. Section 3.2.1) and on the other hand they are quite easy to store efficiently (cf. Section 4.2.3). In this context, we should stress that the sparse image is not compressed at all, so we have to find a way to efficiently store the features to be able to reconstruct the sparse image before inpainting. For a more detailed discussion, the reader may refer to the related work chapter, Section 2.2.
One thing we should explain now is how we measure *compression rates*, as we will talk about them quite often in the upcoming sections and chapters of this thesis. There are actually two ways to measure compression rates. The first one are *absolute compression rates*, which are given in the form $1 : f,\ f > 0$. For example, an absolute compression rate of $1 : 10$ states that one only uses a fraction of $\frac{1}{10}$ of the initial image data in the compressed image. The other option are compression rates given in *bpp*, *bits per pixel*, which state how many bits one has to store per image pixel. So, an uncompressed image with byte-coding has a "compression rate" of 8 bpp.
At last, let us mention that it is actually quite easy to convert compression rates from absolute to bpp values, and vice versa. From bpp to absolute values one just has to divide by a factor of 8 and convert the resulting fraction to the form $\frac{1}{f} = 1 : f$. Consequently, the opposite direction works by multiplying the fraction $\frac{1}{f}$ by a factor of 8.

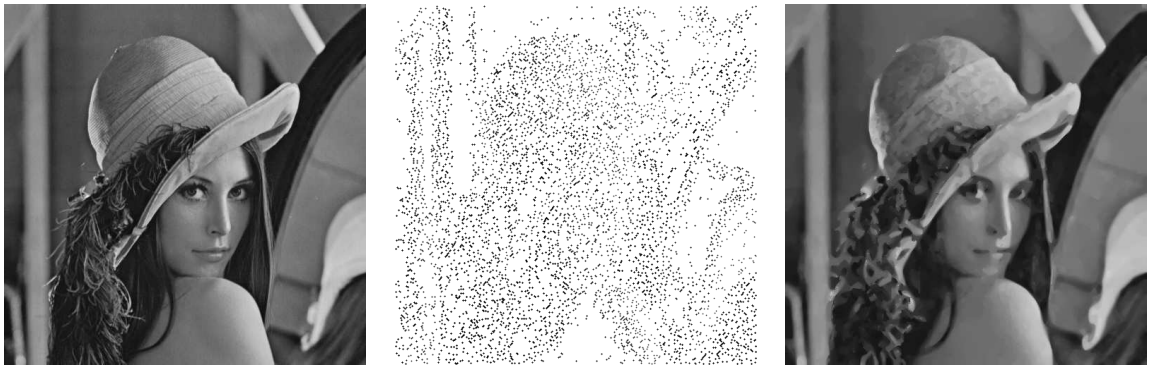

Figure 1.1: **Left:** Initial image, *lena* (512×512 pixels). **Middle:** Sparse image, basis for decompression. Restored from stored compressed data with a compression rate of approximately $1 : 10$ (0.83 bpp). *Note*: For printing reasons we replaced the usual black background with a white one. **Right:** Inpainting of the sparse image gives the decompression result.

To conclude this section, we now want to shortly discuss approaches using different kinds of image features to be stored in the compression step. More details on them can be found in Section 2.2.

- **BTTC**
  In [26], the authors use a simple, adaptive triangulation method based on *B-tree triangular coding (BTTC)*. This method allows on the one hand to extract important image features, and on the other hand to store them in an efficient way. Basically, this method decomposes a given image into triangular regions, which should be recoverable in a sufficiently quality from just the vertices of the triangles. The vertices are finally stored in a binary tree structure, hence the name.
  More details can be found in Section 2.2.1.

- **Edges**
  Edges are considered to be kind of the most important image features [38], so the decompression via inpainting would yield quite nice results. The big caveat here is that the efficient coding of edges is very hard. In principle, one has to store every point on the edge with its coordinates. This yields a really big overhead and makes the storing of edges quite expensive. If one is interested in a discussion on the importance of edges for image compression, one should refer to [19, 18], but we will also discuss this approach in Section 2.2.2.
  The main reason for the mentioned overhead is *redundancy*, which comes from storing also the information of "straight" parts of the edges. This is redundant as a decompression using sophisticated PDE-based inpainting methods, like EED inpainting (cf. Section 5.2.1), will anyway connect interrupted straight edges.

- **Corners**
  So, a logical improvement of the edge-storing approach is to store only exactly these parts of the edges, where the edge changes its direction, which are just the *corners* of the image.
  The implementation of this idea constitutes the main part of this thesis, to be more exact, the approach to store a small neighborhood of image corners, which we will call *corner region* (cf. Figure 3.2) in the remainder of this thesis. The storage of the neighborhood is very important, as it in fact stores vital gradient information.
  As mentioned, corners are just those elements of edges where the edge "abruptly" changes its direction. So, the information content of corners and hence corner regions cannot be as high as the one of edges, but the nice thing is that one avoids the mentioned redundancy. It is even more important that one can efficiently store corner regions by just storing the coordinate of the corner point and the color values of its neighboring points in some fixed order.
  However, there is also a drawback: corners are a quite seldom image feature.

Hence, the decompression using PDE-based inpainting is quite challenging. Because of this, we invested in an extension of the common PDE-based inpainting, namely in *interleaved inpainting*, which we will discuss in Section 3.3.4.

1.3 Organization

After this introductory chapter, we continue this thesis with an overview of **related work** in Chapter 2, where we will on the one hand discuss classical compression algorithms, like JPEG, and on the other hand more recent approaches which rely on storing vital image features, like edges. Finally, we discuss inpainting approaches based on PDEs, as they will constitute the main part of our decompression step.
Chapter 3 deals with the **theoretical background** of our codec and hence consists of two parts, namely compression and decompression. In the first one, we mainly focus on corner detection using the structure tensor and in the latter, we deal with PDE-based diffusion filtering and PDE-based morphological filters, their extensions to vector-valued images and finally with PDE-based inpainting.
After these considerations, we can turn our attention on the **implementation** of the approaches we have shown in the previous chapter, which we will do in Chapter 4, again divided in a compression and a decompression part. As our theoretical considerations will be done in a continuous domain, we will mainly deal with discretization issues, especially with the finite difference method. This method will be used for discretizing derivatives occurring in the PDEs of our inpainting approach for decompression.
Of course, we will also show some results of **experiments** with our codec, which we do in Chapter 5, where we will also discuss parameter selection issues.
Finally, we **conclude** the thesis in Chapter 6, where we show what we have done and learned. We will also give pointers on **future work** which may improve and extend our ideas.

1.4 Notation

Before we go on, we should shortly present the mathematical notation used in the remainder of this thesis.

1.4.1 Images as Functions

It is common practice in the field of image processing to describe images as functions, e.g., $f : \Omega \to R$, where $\Omega \subset \mathbb{R}^n$ is an n-dimensional *image domain*. This means we usually have $\Omega \subset \mathbb{R}^2$ for an *image plain*. With R, we denote the *image co-domain*, usually $R = \mathbb{R}$ for *grayscale images* and $R = \mathbb{R}^3$ for *color images*. Most of the time, we will introduce our theories on grayscale images, because of notational convenience and because the adaption to color images is usually not too problematic.

A last thing to mention is the following notational issue: Often we will denote elements of the image domain Ω simply with $x \in \Omega$, i.e., here we have $x \in \mathbb{R}^2$. On the other hand, one sometimes has to talk about image derivatives in x- and y-direction. For this, we need the longer notation $(x, y) \in \Omega$, i.e., here we have $x \in \mathbb{R}$ and $y \in \mathbb{R}$.

1.4.2 Image Derivatives

We will also heavily use *image derivatives*. Sometimes, we will only consider 1D-signals $v : \mathbb{R} \to \mathbb{R}$, where we will use the usual abbreviation

$$v' := \frac{\mathrm{d}}{\mathrm{d}x} v. \tag{1.1}$$

Partial Derivatives

However, most of the time we will have to use *partial derivatives* of an image $u : \Omega \to \mathbb{R}$, where we use the following abbreviations to ease notation:

$$\frac{\partial}{\partial x} u =: \partial_x u =: u_x. \tag{1.2}$$

The *gradient* of u, we will denote by ∇u, which is defined as

$$\nabla u = \begin{pmatrix} \partial_x u \\ \partial_y u \end{pmatrix}. \tag{1.3}$$

At some points, we will also have to use a *directional derivative* of u in direction $\xi \in \mathbb{R}^2$, which is defined as

$$\partial_\xi u := \xi^\top \cdot \nabla u. \tag{1.4}$$

1.4.3 Linear Algebra Basics

Most of the time, we will use concepts from the field of mathematical analysis. Nevertheless, we will sometimes have to use concepts from linear algebra, which we want to present now.

Vectors and Matrices

Assume, we have a *vector* $v \in \mathbb{R}^n$ and another vector $w \in \mathbb{R}^n$. We will usually denote them with

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = (v_1, \ldots, v_n)^\top \tag{1.5}$$

and

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = (w_1, \ldots, w_n)^\top, \tag{1.6}$$

respectively.
If we deal with a *matrix* $A \in \mathbb{R}^{n \times m}$, we will denote it with

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}. \tag{1.7}$$

A special case is the *unit matrix* $I \in \mathbb{R}^{n \times n}$, which is a *diagonal matrix*, i.e., it has only non-zero entries on its diagonal, which we denote by

$$I := \operatorname{diag}(\underbrace{1, \ldots, 1}_{n-\text{times}}) = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}. \tag{1.8}$$

Multiplying Vectors

When multiplying vectors $v, w \in \mathbb{R}^n$, we will use two possibilities:

Scalar Product: The first one is the *scalar product* $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, which we define as

$$\langle v, w \rangle = v^\top w = \sum_{i=1}^{n} v_i \, w_i. \tag{1.9}$$

Tensor Product: The second possibility is given by the *tensor product* $\otimes : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n \times n}$, defined as

$$v \otimes w = v \, w^\top = \begin{pmatrix} v_1 \, w_1 & v_1 \, w_2 & \cdots & v_1 \, w_n \\ v_2 \, w_1 & v_2 \, w_2 & \cdots & v_2 \, w_n \\ \vdots & \vdots & \ddots & \vdots \\ v_n \, w_1 & v_n \, w_2 & \cdots & v_n \, w_n \end{pmatrix}. \tag{1.10}$$

Chapter 2

Related Work

In this chapter, we discuss classical image compression methods, like JPEG, and present more recent methods which inspired our PDE-based approach for image compression. The latter rely on only storing vital image features, like edges, to compress a given image.
Finally, we give some references to the origins of PDE-based inpainting, which we use in our decompression step in a quite unusual way.
We should note that we do *not* mention corner detection approaches in this chapter, although they constitute the main part of our compression step. The reason for this is that we will discuss them anyway in Section 3.2.1, where we introduce the theoretical background of our compression step.

2.1 Classical Image Compression Methods

The "classical" image compression methods, like JPEG, mainly rely on a *basis transformation*, i.e., they transform the image from the canonical, i.e., spatial, representation to another one and encode only the "important" parts of the transformation, whatever that may mean.
This quite abstract idea will become clear in the upcoming sections, where we shortly describe the approach of the JPEG codec and its successor JPEG2000.

2.1.1 JPEG

Maybe the most famous compression codec around is *JPEG* [43], which was established in 1992 by the Joint Photographic Experts Group (hence the name JPEG) and became an ISO standard in 1994 (ISO 10918-1). The notion JPEG specifies both the *codec* (*co*ding raw images into a compressed stream of bytes and extracting, i.e., *dec*oding, an image from such a stream) and the file format to store the compressed byte stream. Although there are lossless versions of the codec, it became famous for its fast lossless compression part, which yields quite favorable results for not too high compression rates.

Because of this, it became the de-facto standard for storing and transmitting images on the world-wide web. The main reason is that it is fast and in addition it yields a better quality than GIF and smaller files than the PNG-format. However, as JPEG is relatively "blind" w.r.t. image features like edges (cf. Section 1.2), it is not well-suited for line drawings or images mainly containing text or letters, which is actually also the point where our codec will give ideas on how to improve this.

In the following we will shortly sketch the coding and decoding step of JPEG and talk about advantages and disadvantages. The complete official sources on JPEG are quite hard to read, and so we will only give a brief overview, oriented on the concise and easy to understand Wikipedia page [69].
The *coding step* for the most common case (lossy compression of raw color images with byte-wise coding, i.e., $3 \cdot 8 = 24$ bits per pixel) consists of several steps.
First, the image is transferred from the RGB color space into the $YCbCr$ color space, where the Y component represents the brightness, which is most important for a human observer. The other channels carry the chrominance information. The next step then downsamples the Y, Cb and Cr components with a given ratio, usually $4:2:0$, i.e., we reduce the components by a factor of 2 in horizontal and vertical directions, which already allows to save up to 50% of space.
All the upcoming steps will now treat the color channels separately, which makes sense as the transformation from RGB to $YCbCr$ introduces an important coupling of the three RGB color channels. The very next step splits every channel into 8×8 pixel blocks. Dummy pixels have to be filled in if the image size is not a multiple of 8, which is usually done by mirroring (aka. reflecting boundary conditions, cf. Section 4.3.1). Note that this splitting also involves the "blindness" of the codec. On these blocks, the most important, the transformation step is performed, in which every block of every channel is by a forward discrete cosine transform (DCT, type II, cf. Section 2.1.1) transformed into its frequency domain.
Actually, all of the small 8×8 images are transformed into a matrix of $8 \times 8 = 64$ frequency components, where the coefficient at $(0, 0)$ represents the average gray value (DC-coefficient) and the other coefficients (AC-coefficients) correspond to increasing frequencies in x- and y-direction, respectively. Note that this alone actually involves a blowup in space rather than a compression as 11 to 12 bits are needed for each coefficient, compared to the 8 bits for each color value in every channel. But if one exploits the fact that the most significant part for the human eye are the low frequencies, one can in the next step quantize the DCT coefficients. This leads to the fact that the amount of information in the high frequency parts is reduced, which yields the main loss of information in the codec. To this end, one just divides each coefficient by a constant given by a quantization matrix, which usually has larger constants for higher frequencies. This explains that the magnitude of the quantization constants in the quantization matrix mainly determines the achieved compression rate.
If one now rounds the quantization result, one gets almost only coefficients equal to 0

in the higher frequency range and also quite small coefficients in the middle frequency range. This motivates the approach to store these coefficients as a Huffman coded [30] sequence by just traversing the coefficient matrix in a zig-zag manner. This sequence will contain only 0 entries at the end and so we can stop storing the sequence at this point and indicate the end by a special code in the Huffman code. Another thing is that one can efficiently store the frequently occurring small coefficients in the middle frequency range by assigning smaller codes to them. This step is known as the entropy coding step and concludes the coding.
The *decoding step* just does all the above in reverse. First, a DCT coefficient matrix is created from the Huffman-coded sequence, then it is multiplied by the quantization matrix and finally an inverse DCT (type III, cf. Section 2.1.1) is performed.

The JPEG codec also has some other features, besides the pure compression of images. For example, it provides *progressive transmission*, which allows to see a preview of the decompression result altough not the whole coded image has been transmitted. This is quite useful when transmitting images through a slow communication channel.

To conclude this section, we want to have a look at the performance of JPEG. We see that both parts of the codec are very fast, especially the decompression step is much faster than the one we used, namely the PDE-based inpainting. However, JPEG also has some disadvantages, which mainly lie in its mentioned blindness. If we look at Figure 2.1, we see that especially at edges, JPEG does perform quite bad, and remember that edges are one of the most important carriers of the image semantics. For small compression rates this is only hardy visible, but for high rates above $1 : 100$, we clearly see the infamous blocky artifacts, which mainly disturb the image at the boundaries. (cf. Figure 2.2).

Figure 2.1: Accentuated differences between an initial image and the JPEG compression of it. Note especially the changes occurring near sharp edges. **Source:** Wikipedia [69].

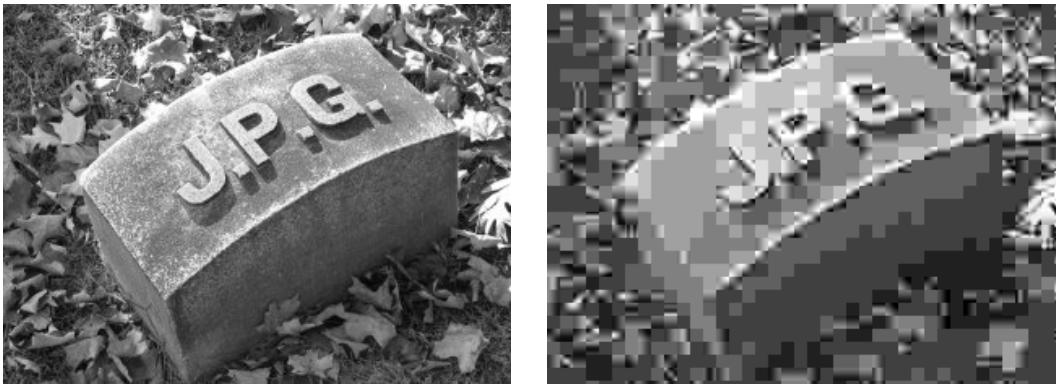

Figure 2.2: **Left:** Initial image. **Right:** JPEG compression with compression rate 1 : 100 (0.08 bpp) with visible blocky artifacts. **Source:** Wikipedia [69].

The Discrete Cosine Transform (DCT)

As mentioned above, the *discrete cosine transform* (DCT) constitutes one of the main parts of the JPEG codec. It was advocated in 1974 by Ahmed et al. [1], but a newer overview can be found in [34]. For a very concise overview, one may again refer to Wikipedia [68].
The DCT is a Fourier-related transform very similar to the better known *discrete Fourier transform* (DFT). The difference is that a DCT works only with real numbers, whereas a DFT works with complex exponents, representing sine and cosine frequencies. More precisely, a DCT is a special case of a DFT operating on real data with double length and even symmetry.
Because of its similarity to the DFT, also the DCT expresses a discrete function or signal $f = (f_0, \ldots, f_{N-1}) \in \mathbb{R}^N$ of length N as a sum of N sinusoids (sine and cosine functions) with different frequencies and amplitudes. This means that it transforms f from the spatial domain into its frequency domain. The interesting property of this transform is the *energy compaction*, which means that most of the (for a human observer) important part of the signal is concentrated in the low-frequency coefficients of the DCT. Hence, this transform may be useful for the compression of $1D$ signals, e.g., audio signals and also $2D$ signals, i.e., grayscale images.
As the DCT works like all discrete Fourier-related transformations on discrete functions with a finite domain, one has to implicitly define an extension outside this domain, as the sine and cosine function are defined over the whole $\mathbb{R}$. These *boundary conditions*, specifying how to extend the function domain give rise to 16 different Fourier related transformations, where 8 of them constitute the different DCTs and the rest are DSTs, discrete sine transforms. The interesting ones yielding DCTs are the ones where the left boundary is even, i.e., $f(x) = f(-x)$, in contrast to an odd boundary, which is given by $f(x) = -f(-x)$, where x is a boundary point. The different DCTs are then created by considering around which point the function is even at the left boundary. Here, one has two possibilities. Consider a discrete function $f = (a, b, c, d) \in \mathbb{R}^4$ of length $N = 4$.

In the first case we create an even left boundary about the point a, which yields the extension $f_{ext} = (\ldots, d, c, b, a, b, c, d)$. In the second case one uses the halfway between a and the previous point, yielding $f_{ext} = (\ldots, d, c, b, a, a, b, c, d)$. Interestingly, these boundary conditions are responsible for the energy compaction property stated above. However, the usage of Fourier-related transformations also has a clear disadvantage. One can show that discontinuities in the function reduce the rate of convergence of the Fourier series, the continuous counterpart of the DFT. Hence, we need more sinusoids to accurately represent the discontinuous function. To put it short, the smoother a function, the fewer coefficients are needed and hence the higher the compression rate.
But let us now formally define the DCT and its inverse transformation, the *inverse discrete cosine transform* (*IDCT*), where the latter is used in the decompression step of the JPEG codec. We should note that we mentioned that there are 8 different types of the DCT, but for computational reasons it is common to use the DCT-II as "the DCT" and the DCT-III as "the IDCT". The problem is that different authors will use slightly different definitions of the DCT and the IDCT, which we will discuss in our definitions, too.
The discrete cosine transform of a discrete signal $f = (f_0, \ldots, f_{N-1}) \in \mathbb{R}^N$ of length N is a linear, invertible function $\mathcal{C} : \mathbb{R}^N \to \mathbb{R}^N$, i.e., it can also be expressed by a system matrix $A \in \mathbb{R}^{N \times N}$. If we define

$$\mathcal{C}(f) = \mathcal{C}(f_0, \ldots, f_{N-1}) = (\hat{f}_0, \ldots, \hat{f}_{N-1}), \tag{2.1}$$

then the DCT (DCT-II) reads as

$$\hat{f}_k = \sum_{i=0}^{N-1} f_i \cdot \cos\left(\frac{\pi}{N}\left(i + \frac{1}{2}\right)k\right), \quad \text{for } k = 0, \ldots, N-1. \tag{2.2}$$

This transform is then up to a scale factor equivalent to a DFT of length $4N$ with even symmetry and where all even indexed entries of f are equal to zero. In some contexts, one will encounter that the first coefficient of the DCT will read $\hat{f}_0/\sqrt{2}$. If one furthermore multiplies all DCT coefficients with a scale factor of $\sqrt{2/N}$, one ends up with a favorable orthogonal system matrix A, but also looses the direct correspondence to the DFT.
Concerning boundary conditions, we have for DCT-II: The function values f_i have to be even around the points $-\frac{1}{2}$ and $N - \frac{1}{2}$. For the DCT coefficients $\hat{f}_i$, we require even symmetry around 0 and N.
The inverse DCT (IDCT) is given by the DCT-III, which reads

$$f_k = \frac{1}{2}\hat{f}_0 + \sum_{i=0}^{N-1} \hat{f}_i \cdot \cos\left(i \cdot \frac{\pi}{N}\left(k + \frac{1}{2}\right)\right), \quad \text{for } k = 0, \ldots, N-1. \tag{2.3}$$

Here, we have the corresponding scale factors as for the DCT-II, i.e., $\sqrt{2}f_0$ and an overall scale factor of $\sqrt{2/N}$. Also the boundary conditions are equivalent, i.e., even symmetry of $\hat{f}_i$ around 0 and N and even symmetry of f_i around $-\frac{1}{2}$ and $N - \frac{1}{2}$.

Multidimensional DCTs: Until now, we have considered the DCT of discrete $1D$ signals $f \in \mathbb{R}^N$ of length N. For our purposes and the ones of JPEG one has to deal with $2D$ signal, i.e., grayscale images $f \in \mathbb{R}^{N_1 \times N_2}$. To be more precise, JPEG is concerned with 8×8 pixel blocks of an image, i.e., $N_1 = N_2 = 8$. The extension from the $1D$ DCT to the $2D$ DCT is very straight forward, as the DCT is *separable*, i.e., one can perform a $1D$ DCT in x-direction (for all rows of the image), followed by a $1D$ DCT in y-direction (for all columns of the image).
If we define a discrete $2D$ signal $f \in \mathbb{R}^{N_1 \times N_2}$ as

$$f = \begin{pmatrix} f_{0,0} & \cdots & f_{0,N_2-1} \\ \vdots & \ddots & \vdots \\ f_{N_1-1,0} & \cdots & f_{N_1-1,N_2-1} \end{pmatrix}, \tag{2.4}$$

we can write the $2D$ DCT-II of f as

$$\mathcal{C}(f) = \begin{pmatrix} \hat{f}_{0,0} & \cdots & \hat{f}_{0,N_2-1} \\ \vdots & \ddots & \vdots \\ \hat{f}_{N_1-1,0} & \cdots & \hat{f}_{N_1-1,N_2-1} \end{pmatrix}, \tag{2.5}$$

with

$$\hat{f}_{k_1,k_2} = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} f_{i,j} \cdot \cos\left(\frac{\pi}{N_1}\left(i+\frac{1}{2}\right)k_1\right) \cdot \cos\left(\frac{\pi}{N_2}\left(j+\frac{1}{2}\right)k_2\right), \tag{2.6}$$

$$\text{for} \quad k_1 = 0, \ldots, N_1 - 1 \text{ and } k_2 = 0, \ldots, N_2 - 1.$$

Altough the direct application of this scheme would lead to a computational complexity of $\mathcal{O}(N^2)$ (for $N = N_1 = N_2$), there are clever algorithms which compute the $2D$ DCT in $\mathcal{O}(N \log N)$, using factorization ideas, as in the well known *fast Fourier transform* (*FFT*). If one is interested in the ideas of a fast DCT, one may refer to [23].

2.1.2 JPEG2000

The Joint Photographic Experts Group was recently working on an improvement of the JPEG codec, which they called *JPEG2000* [54]. To get in touch with this really sophisticated codec, we used again the Wikipedia page [70]. For a deeper discussion, the overview article of Marcellin et al. [37] is recommendable.

After the development of the JPEG codec in 1992, a lot of research was done to improve the codec. The result was a *wavelet*-based codec, which allows higher compression rates (less than 0.25 bpp) without the generation of the infamous blocky artifacts of JPEG. Altough JPEG2000 became an ISO standard (ISO/IEC 15444-1:2000) it still up to now not widely supported, i.e., it has not replaced JPEG, yet.

One should note that the main improvements in JPEG2000 do *not* only lie in better compression results for high compression rates, but in other features, like *random codestream access*, also referred as *Region Of Interest* (*ROI*). This feature allows to store different parts of the same image with different quality. Also quite some research was done to improve robustness w.r.t. noisy transmission channels.
We now want to give a coarse overview over the technical details of JPEG2000. For more details, we refer the reader to the references stated above.
Like with JPEG, the image to be compressed is first transformed into a more adequate color space, which is in this case either the $YCbCR$ or the YUV color space, which then again allow to process the color channels independently.
After that, the image is split into so called tiles. In JPEG, these were squares of size 8×8 pixels. Here, rectangular regions of an arbitrary size are allowed, but in one image, each one has to have the same size. It is hence even possible to just use one large tile for the whole image, i.e., actually no splitting is performed then. In this context there is a certain tradeoff. Using more, smaller tiles makes coding more efficient, but also provokes blocky artifacts.
The next step constitutes the main part of the codec. The tiles undergo a wavelet transform, actually a discrete wavelet transform (DWT, cf. Section 2.1.2). Here, two different transformations are possible. The first one is the *CDF 9/7* wavelet transform [6], which is called *irreversible*, as it introduces quantization noise. The other one is the *CDF 5/3* wavelet transform [11], which only uses integer values, which do not have to be rounded and hence introduce no quantization noise. Because of this, this transformation is called *reversible* and used for *lossless compression*. The favorable thing about the wavelet coefficients is that they represent features of the image corresponding to a certain frequency (cf. DCT coefficients of the JPEG codec) *and* a spatial area of the image. The latter introduces a big advantage compared to the "spatially blind" DCT of JPEG.
For a lossy compression, the real-valued wavelet coefficients of the CDF 9/7 transform are then scalar quantized. This reduces the number of bits needed to store them, and also introduces the loss of information. Furthermore, the quantization scale constitutes the most important factor for fixing the achieved compression rate.
The final step is to efficiently code the quantized wavelet coefficients. This step encompasses different sub-steps, which mainly do a subsequent splitting and merging of the image regions. These steps are quite involved, as they have to provide all the additional features, like the mentioned random codestream access. Therefore, we only sketch these steps here and again refer the reader to the given references. The result of the wavelet transform and the quantization are a collection of so called sub-bands (sets of wavelet coefficients), which represent several approximation scales of the initial image. The sub-bands are split into precincts, rectangular regions in the wavelet domain. These are further split into code-blocks, which are equally sized and located in a single sub-band. An exception form code-blocks located at edges, i.e., here JPEG2000 also takes into account semantic image information coded by the edges. The bits of the code-blocks are

then coded starting with the most significant ones, which is called an *Embedded Block Coding with Optimal Truncation*, *EBCOT* scheme [53], consiting of three different coding passes. The bits that are selected by these coding passes are then coded using a context-driven binary arithmetic coder, the well-known binary MQ-coder, to be more precise. This also distinguishes JPEG2000 from JPEG, where a Huffman coding was used. The result of the coding steps is then a bit-stream, which is split into packets. These packets group selected passes from the EBCOT scheme of all code-blocks from a certain precinct together. The packets are then collected in layers, s.t. the image quality increases with each layer, allowing a progressive transmission. The problem that arises from this is to find the optimal packet length which minimizes overall distortions s.t. the desired compression rate is achieved. This is a constrained optimization problem and can be solved using classical approaches, like *Lagrangian multipliers*.

If one evaluates the pure compression performance of JPEG2000 one sees that the improvement compared to JPEG is rather modest, actually about 20%. The larger, but for us not so important improvements lie in the mentioned additional features. A comparison of JPEG2000 with JPEG can be inspected in Figure 2.3.
However, one has to admit that for large images and images with low contrast edges (which occur for example in medical applications like MRI or x-ray imaging) JPEG2000 gives noticeably better results than JPEG.
Also the infamous blocky artifacts of JPEG are not such a big problem with JPEG2000. Up to a compression rate of about 0.4 bpp, no visible artifacts occur and if so, artifacts of JPEG2000 rather consist in a blurring (like with our PDE-based approach) than in blocky artifacts.

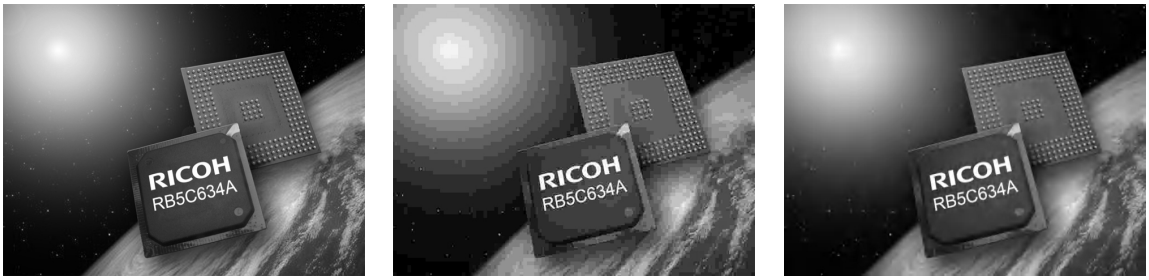

Figure 2.3: Comparing JPEG2000 with JPEG. **Left:** Initial image (640×480 pixels). **Middle:** Compressed with JPEG at a compression rate of about $1 : 200$ (0.04 bpp). **Right:** Same with JPEG2000. **Source:** *http://www.s-t-e.de/content/Articles/Articles_11.php*.

The Wavelet Transform

As done before, we will now shortly present the basic ideas of the *wavelet transform*, which constitutes the main part of the JPEG2000 codec, in contrast to the discrete

cosine transform used in JPEG. The $2D$ wavelet transform one has to use for images was defined by Meyer and Lemarié in 1986 [35, 40].
Actually, there are different variants of wavelet transforms, like for the DCT, but the ones used in JPEG2000, namely CDF 9/7 and CDF 5/3, are not the standard ones. Furthermore, a detailled discussion of wavelet transforms would exceed the scope of this related work chapter and so we decided to only sketch the very basic ideas of the wavelet transform here and refer the interested reader to the given references or text books on wavelets.

For writing this section, we used the first sections of the article of Antonini et al. [6] (CDF 9/7), which describes the use of the wavelet transform for image compression and gives a pleasant theoretical introduction to wavelets.
The reason to use the wavelet transform for an image compression algorithm relying on a basis transformation, are three important features. First, it accepts nonstationarity and is well localized in the space and frequency domain. This is important for a correspondence to the human visual system. Second, it is biorthogonal, which avoids redundancies and last but not least there exist efficient implementations of the wavelet transform [36].
So let us now define the basic notions of the wavelet transform. *Wavelets* are just a family of functions generated from a single function, the *mother wavelet* Ψ, by dilations and translations, i.e.,

$$\Psi^{a,b}(t) = \frac{1}{\sqrt{|a|}}\,\Psi\left(\frac{t-b}{a}\right), \tag{2.7}$$

where $\Psi(t)$ has to satisfy

$$\int_t \Psi(t)\,\mathrm{d}t = 0, \tag{2.8}$$

which practically implies that Ψ has to oscillate. The definition of Ψ also implies that we have narrow (high frequency) wavelets for $a < 1$ and wide (low frequency) wavelets for $a > 1$.
With this family we can represent an arbitrary function $f(t)$ as a superposition of different scaled and translated wavelets. Such a decomposition decomposes f into different scale levels, where each level is further decomposed with a specific resolution. The *discrete wavelet transform* (DWT) uses a discrete superposition, i.e., a finite sum instead of an integral, discretized parameters $a = a_0^m$ and $b = n\,b_0\,a_0^m$ for $m, n \in \mathbb{Z}$ and fixed $a_0 > 1$ and $b_0 > 1$. With this, we can write the (discrete) wavelet decomposition of a discrete function $f : \{0, \ldots, N-1\} \to \mathbb{N}$ of length N (and grid size $h = 1$) as

$$f(t) = \sum_{t \in \{0,\ldots,N-1\}} c_{m,n}\big(f(t)\big)\;\Psi_{m,n}(t), \tag{2.9}$$

with the discrete wavelets

$$\Psi_{m,n}(t) := \Psi^{a_0^m,\,n\,b_0\,a_0^m}(t) = a_0^{-m/2}\,\Psi(a_0^{-m}t - n\,b_0). \tag{2.10}$$

If one chooses $a_0 = 2$ and $b_0 = 1$, one ends up with an orthonormal basis built by the $\Psi_{m,n}$, so that we have

$$c_{m,n}\big(f(t)\big) = \langle \Psi_{m,n}(t), f(t) \rangle = \int_t \Psi_{m,n}(t)\, f(t)\, \mathrm{d}t \tag{2.11}$$

in this case. Such a basis corresponds to a *multiresolution analysis* invented by Mallat [36], and which is well adapted to the use of wavelet bases for image analysis.
For such a multiresolution analysis one actually uses two functions, namely the mother wavelet Ψ and a dilated and translated *scaling function* $\Phi_{m,n}(x) = 2^{-\frac{m}{2}} \Phi(2^{-m}x - n)$. If we fix m, then one can show that the $\Phi_{m,n}$ are orthonormal and hence span spaces V_m, which describe successive approximation spaces, i.e., $\cdots V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \cdots$ with resolution 2^m, explaining the name multiresolution analysis. Furthermore, it should be clear that for fixed m the $\Psi_{m,n}$ span spaces W_m which are the orthogonal complement of V_m in V_{m-1}. This means that the coefficients $\langle \Psi_{m,n}, f \rangle = c_{m,n}(f)$ describe the loss of information when going from resolution 2^{m-1} to the coarser scale of 2^m. From this one can derive an algorithm [36] for the computation of $c_{m,n}$ and $a_{m,n}$, where the latter coefficients describe the projection of f onto V_m. This also means that if we have given a discrete, sampled version of f (which will be the case for the discrete wavelet transform used in JPEG2000, as we deal with discrete images) then one can take these samples as initialization, i.e., $a_{0,n}$.
The algorithm then reads

$$c_{m,n}(f) = \sum_k g_{2n-k}\, a_{m-1,k}(f) \tag{2.12}$$

$$a_{m,n}(f) = \sum_k h_{2n-k}\, a_{m-1,k}(f), \tag{2.13}$$

where $g_j = (-1)^j\, h_{-j+1}$ and $h_n = 2^{\frac{1}{2}} \int \Phi(x-n)\, \Phi(2x)\, \mathrm{d}x$.
Because of the association of the presented algorithm to orthonormal wavelet bases, we have an exact reconstruction of the initial discrete function f, given by the samples $a_{0,n}$, via

$$a_{m-1,j}(f) = \sum_n \Big(h_{2n-j}\, a_{m,n}(f) + g_{2n-j}\, c_{m,n}(f) \Big). \tag{2.14}$$

Until now, we only gave theoretical considerations for *ideal* wavelet filters, which allow an exact reconstruction.
The problem with wavelet filters that should be applicable to images is the following: Images are mostly smooth (except at edges, of course). So one should aim for an exact reconstruction coding scheme which corresponds to an orthonormal basis with reasonably smooth mother wavelets. Furthermore, one imposes that filters are linear phase, because this allows to cascade them in a pyramidal filter structure without a need for phase compensation. Unfortunately, there exist no (non-trivial) filters with these properties.

However, if one only requires orthogonal bases, one actually can achieve the before mentioned properties. In such a *biorthogonal basis*, one uses the same decomposition as in equations 2.12 and 2.13, but the following reconstruction scheme:

$$a_{m-1,j}(f) = \sum_n \Big(\tilde{h}_{2n-j}\, a_{m,n}(f) + \tilde{g}_{2n-j}\, c_{m,n}(f) \Big), \tag{2.15}$$

i.e., one uses filters $\tilde{h}$ and $\tilde{g}$ different from h and g. For them, one imposes

$$\tilde{g}_n = (-1)^n\, h_{-n+1} \quad \text{and} \quad g_n = (-1)^n\, \tilde{h}_{-n+1} \tag{2.16}$$

$$\sum_n h_n\, \tilde{h}_{n+2k} \quad = \quad \delta_{k,0}. \tag{2.17}$$

With this, one defines functions Φ and $\tilde{\Phi}$ by

$$\Phi(x) = \sum_n h_n\, \Phi(2x-n) \;\text{ and }\; \tilde{\Phi} = \sum_n \tilde{h}_n\, \tilde{\Phi}(2x-n), \tag{2.18}$$

and functions Ψ and $\tilde{\Psi}$ by

$$\Psi(x) = \sum_n g_n\, \Phi(2x-n) \;\text{ and }\; \tilde{\Psi} = \sum_n \tilde{g}_n\, \tilde{\Phi}(2x-n). \tag{2.19}$$

The decomposition then reads

$$a_{m,n}(f) \;=\; \langle \Phi_{m,n}, f \rangle = 2^{-\frac{m}{2}} \int \Phi_{m,n}(x)\, f(x)\, \mathrm{d}x \tag{2.20}$$

$$c_{m,n}(f) \;=\; \langle \Psi_{m,n}, f \rangle = 2^{-\frac{m}{2}} \int \Psi_{m,n}(x)\, f(x)\, \mathrm{d}x. \tag{2.21}$$

And the corresponding reconstruction is then simply

$$f = \sum_{m,n} \langle \Psi_{m,n}, f \rangle\, \tilde{\Psi}_{m,n}. \tag{2.22}$$

The associated filter structure is given in Figure 2.4 and a sample wavelet basis for biorthogonal filters close to an orthonormal wavelet filter can be inspected in Figure 2.5.

2.2 Compression using Image Features

In this section, we want to present approaches that inspired our image compression algorithm, which only stores important image features in the compression step and tries to recover the image by PDE-based inpainting. So, let us first define what we actually understand under the notion *inpainting*.

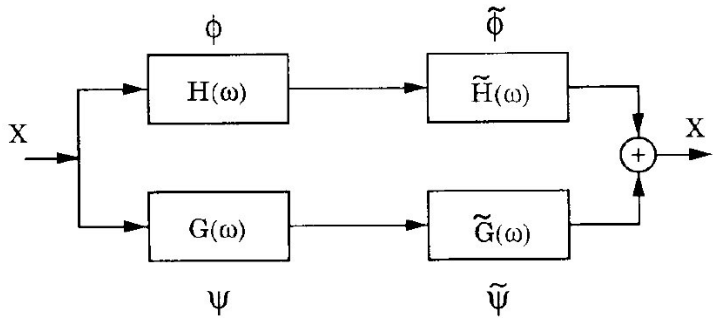

Figure 2.4: Filter structure and associating wavelets. **Author:** Antonini et al. [6].

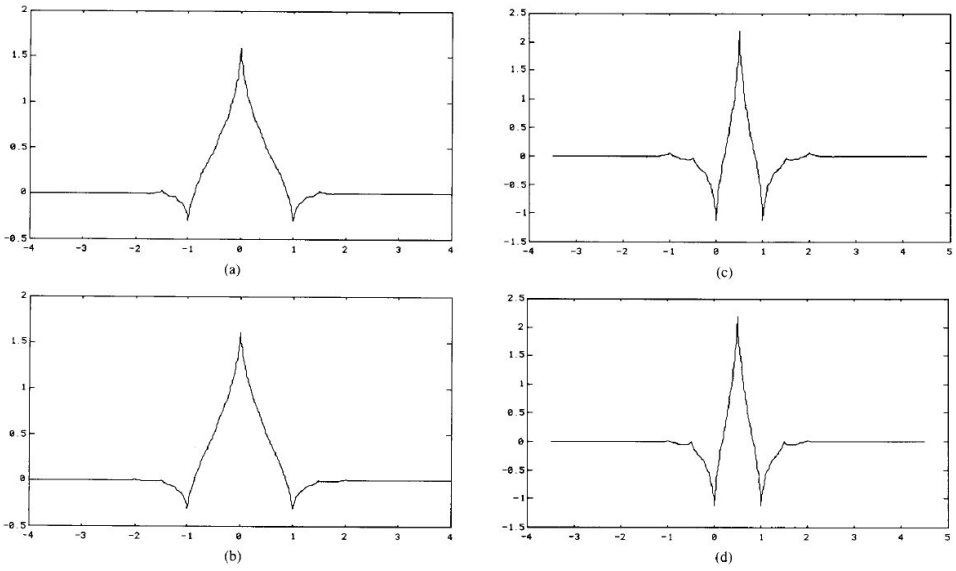

Figure 2.5: Example of a wavelet basis for biorthogonal filters close to an orthonormal wavelet filter. **Left column:** Scaling functions Φ (top) and $\tilde{\Phi}$ (bottom). **Right column:** Wavelets Ψ (top) and $\tilde{\Psi}$ (bottom). **Author:** Antonini et al. [6].

The image *inpainting technique* (also called *retouching*) describes the modification of images, s.t. it is not detectable by a human observer.
It was already used before the invention of computers to denote the restoration of old and damaged paintings by professional restorators [22, 56]. So, the aim is to fill in missing portions of work.
The most striking difference between classical image inpainting and our approach is that the classical inpainting was used for image restoration, i.e., to repair *small* defects in images or to remove *small* objects from images. For our purpose, it has to do a lot more, as the sparsified images are very sparse indeed. These problems are discussed in more detail in Section 5.2.1.

Disocclusion: To avoid misunderstandings, we will now briefly review a technique that is related to inpainting, namely *disocclusion*. In accordance to the definition of [17], it describes the recovery of hidden parts of objects in (2D–) digital images, so it tries to imitate the human vision system which has exactly this ability. Usually, the output of such algorithms is hence a 3D ordering of the scene. An article concerning disocclusion can be found in the work of Masnou et al. [39].
This also stresses the difference to inpainting, whose output is only a 2D image again. These approaches are mainly based on *level lines*, i.e., isophotes, as described in [39]. The reason for that is that edges are not contrast invariant, and so one hopes to be better able to restore the occluded objects using level lines.

Another, not so common application of inpainting is digital zooming, which is described in [18].

2.2.1 BTTC

This method, named *B-tree triangular coding (BTTC)*, is used in the work of Galic et al. [26] and originates from [21]. It is useful for adaptive sparse image creation and coding, hence for the compression step. The final result is a sparse image that only contains the important image features.

The basic idea is to consecutively subdivide a given image of size $(2^m + 1) \times (2^m + 1)$ pixels into triangular regions, for which one only stores the vertices of the triangles, hence the sparsification. From these vertices, the regions should then be recoverable in a sufficient quality, i.e., the absolute difference between the recovered data and the initial data has to be below a specified threshold. Finally, these triangles (or to be more exact, the vertices) are stored in a binary tree structure, which explains the name BTTC.
In fact, the initialization of the *compression step* consists in splitting the image into two triangles, where the 4 resulting vertices (including 2 shared vertices) are just the 4 corners of our image. In the consecutive refinement steps, the image is approximated by

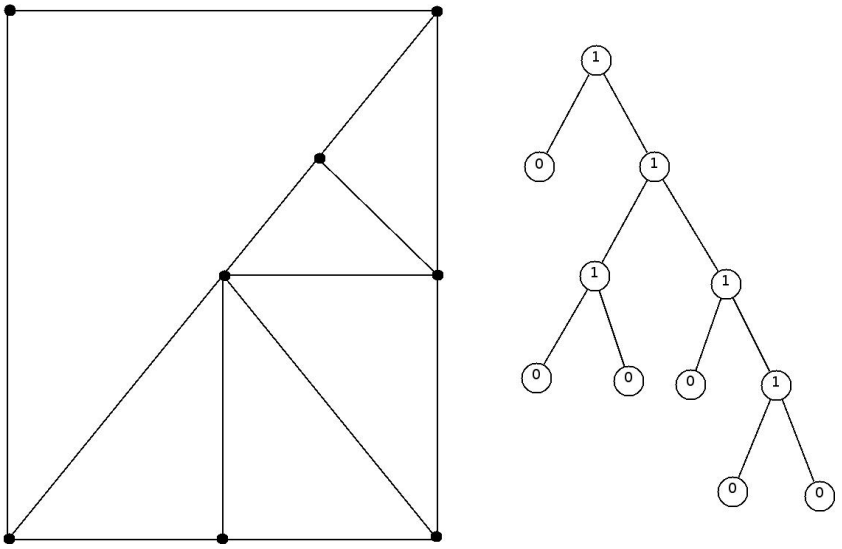

Figure 2.6: Example of a splitted image and the corresponding tree for BTTC.

linear interpolation of each triangular region from its 3 vertices. If for all pixels (i, j) of a region, the approximation error is below a given error threshold ε, the subdivision is stopped for this triangle. All triangles which contain a pixel (i, j) whose error is above ε, are split further. The splitting just creates two new triangles, s.t. the center of the hypotenuse of the old triangle becomes the new common vertex of the new triangles. If it happens that a pixel (i, j) whose error is above the threshold lies exactly on a side of a triangle, then just one of the adjacent triangles is chosen to be split. The final result of this process is a so called vertex mask, which just stores the positions of all vertices. So, in the first part of the upcoming *coding step*, one has to efficiently code the vertex positions. To this end, a binary tree structure is introduced, which represents the hierarchical splitting of the triangles. All splitted triangles are intermediate nodes of the tree, where the two subtriangles are its children. Undivided triangles are represented as the leaves of the tree. Note that the root of the tree then just represents the initial image, and its two children are the two triangles of the first splitting step. To code this tree, it is traversed in a fixed order and one bit is stored per node: A '1' for a node with children and a '0' for a leaf. From this tree, one can reconstruct the vertex mask, but one of course also has to store the color values of the masked pixels in the original image. This is done by traversing the mask in a fixed order and storing the color values in a sequence, which is efficiently coded by a Huffman coding [30], like in the coding step of JPEG (cf. Section 2.1.1).
As a last improvement the authors propose to introduce a lossy quantization of the color values from 256 (1 byte = 8 bit per channel) to 64 (6 bit per channel).
This process is quite complex, so we refer the reader to Figure 2.6, which gives an example of a splitted image and the corresponding tree.

Figure 2.7: Image compression with edge information. **Left:** Initial image. **Middle:** Edge tube E from Canny edge detector. **Right:** Decompression via TV inpainting. **Authors:** Chan and Shen [18].

2.2.2 Edges

In [19, 18] Chan and Shen discuss a very interesting approach to PDE-based image compression using edge information. Edges are very important image features (see [38]) and hence are very promising for image compression.
In the compression step, the authors first apply an edge detector (Canny edge detection [12]), which gives the edge collection E of the image. Then they generate a small ε-neighborhood of E, the so called edge tube T. In the implementation, they just thicken the edges by 1 or 2 pixels. With this, the authors can store vital gradient information. The problem is now the coding, as they have to store the address of all tube pixels and then code the corresponding color values of the initial image at these locations. Obviously, this is not very efficient.
For the decompression step, the authors use their TV inpainting scheme. This approach wants to find a minimizer of the (discrete) energy

$$\min_u \left[\sum_{(i,j)\in\Omega_{disc}} |\nabla_{(i,j)} u| + \frac{\lambda_T(i,j)}{2}(u_{(i,j)} - f_{(i,j)})^2 \right], \tag{2.23}$$

where $(i,j) \in \Omega_{disc}$ denotes a discrete pixel of the discrete image domain Ω_{disc} and $\nabla_{(i,j)} u$ is a discrete version of the gradient in (i,j), e.g., a finite difference approximation, which will be described in Section 4.1.1. The parameter $\lambda_T(i,j)$ is an extended Lagrange multiplier, which is equal to a Lagrange multiplier λ in the edge tube T and equal to 0 elsewhere.
The motivation for using TV diffusion in the decompression step is that the coded edges will not be regular and TV diffusion can straighten stored wiggled edges in the decompression step.
The result of this approach can be inspected in Figure 2.7.

2.3 Inpainting

Now, we want to present the "classical" inpainting approaches. There already exist several approaches for inpainting of images, some based on diffusion filters and some based on *variational approaches*.
To make the notation match our definitions concerning PDE-based inpainting in Section 3.3.4, we will denote the inpainting domain of an image $u : \Omega \to \mathbb{R}$ with $\Omega \setminus \Omega'$, where Ω' is consequently the area which should not be changed by the inpainting.

2.3.1 Variational Approaches

First, we want to present the *variational approaches*, which are less important to us. However, one may also rewrite them in a for us more appropriate PDE form.

Approach of Chan and Shen

In [16, 19, 18] Chan and Shen developed a nice approach for inpainting. They used a so called total variation (TV) approach, which is a variational approach and strives to minimize an energy

$$J_\lambda(u) := \int\limits_{\Omega\setminus\Omega'} |\nabla u| \, \mathrm{d}x + \int\limits_{\Omega'} \frac{\lambda}{2} |u - f|^2 \, \mathrm{d}x, \tag{2.24}$$

where f denotes the given initial image and λ is a Lagrange multiplier.
A sufficient criterion for a minimizer u of J is given by the so called *Euler-Lagrange equation*, which reads in this case

$$-\mathrm{div}\left(\frac{1}{|\nabla u|} \cdot \nabla u\right) + \lambda_e(u - f) = 0, \tag{2.25}$$

where $\lambda_e := \mathcal{X}_{\Omega'} \cdot \lambda$ is an extended Lagrange multiplier using the characteristic function $\mathcal{X}_{\Omega'}$ of Ω'. Hence λ_e is equal to the normal λ outside the inpainting domain, i.e., in Ω' and zero inside $\Omega \setminus \Omega'$. If we now want to fulfill this constraint, we employ a gradient descent method, which will then read

$$\partial_t u = \mathrm{div}\left(\frac{1}{|\nabla u|} \cdot \nabla u\right) + \lambda_e(u - f). \tag{2.26}$$

This is then finally the governing PDE of our PDE-based inpainting approach.
In this context, we should note the relation between *mean curvature motion* (*MCM*) and TV. MCM is a morphological image filter, which can be written in PDE form and

which we will discuss in Section 3.3.2. For MCM, we have the governing PDE

$$\partial_t u = \kappa \cdot \nabla u + \lambda_e(u - f) \tag{2.27}$$

$$= \operatorname{div}\left(\frac{\nabla u}{|\nabla u|} \cdot \nabla u\right) + \lambda_e(u - f), \tag{2.28}$$

with the *curvature* κ, which is defined as

$$\kappa := \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right). \tag{2.29}$$

Approach of Masnou and Morel

The approach of Masnou et al. [39] is also a variational approach, or to be more precise a total variation (TV) approach, where one minimizes the energy functional

$$E(v) = \int_{\Omega' \setminus \Omega} |\nabla v|(1 + |\operatorname{curv}(v)|)\, \mathrm{d}(x, y), \tag{2.30}$$

inside the occluded area, which is the inpainting domain $\Omega' \setminus \Omega$, where $\operatorname{curv}(v) := \kappa$ denotes the curvature of the level lines of image v.

2.3.2 PDE-based Inpainting

Approach of Bertalmio, Sapiro, Caselles and Ballester

In [8], Bertalmio et al. were the first to present an algorithm for automatic digital inpainting, based on the techniques used by professional image restorators. It only requires the user to specify an inpainting mask, determining where to fill in missing information. The goal was to be able to fill in regions sourrounded by different backgrounds, but they still faced problems with very textured images (which is actually always the case if one uses diffusion filters for inpainting).
Let us denote by $\partial(\Omega \setminus \Omega')$ the boundary of the inpainting domain. Then the basic idea is to prolong *isophote* lines (lines of the same gray value) arriving at $\partial(\Omega \setminus \Omega')$, taking into account their angle to $\partial(\Omega \setminus \Omega')$ and curving them in order to avoid crossings of isophotes. To reach this, the authors tried to mimic the two basic steps of human image restorators.

1. Continue the structure of the surrounding inside the gap and

2. Fill the resulting regions inside the gap with color matching the boundary.

Here we should note that usually human restorators will also propgate texture from outside the gap, which is not done in this algorithm, but an extension to textured images is mentioned in [8].
In the implementation, these two steps are performed iteratively to propagate more and more information from the boundaries to the interior of $\Omega \setminus \Omega'$. Formally, a family of images $u(x, y, t)$ with time parameter t arises, where the final result is given by the *steady state*, i.e., $\lim_{t\to\infty} u(x, y, t)$. The evolution can be written as

$$u(x, y, t + 1) = u(x, y, t) + \Delta n \cdot u_n(x, y, t), \qquad \forall (x, y) \in \Omega \setminus \Omega', \tag{2.31}$$

where Δn is the rate of change and $u_n(x, y, t)$ is the change of information inside $\Omega \setminus \Omega'$. As the authors wanted to mimic human restorators, they had to smoothly propagate information from outside of $\Omega \setminus \Omega'$ to the inside, so they defined

$$u_n(x, y, t) := \delta\vec{L}(x, y, t) \cdot \vec{N}(x, y, t), \tag{2.32}$$

where $\delta\vec{L}$ measures the change of propagated information, i.e., we can write $\delta\vec{L}(x, y, t) := \nabla L$, and $\vec{N}$ is the propagation direction. For smoothness reasons, they decided to use an image smoothness measure, the Laplacian, for $\vec{L}$, so $\vec{L}(x, y, t) := \triangle u(x, y, t) = \partial_{xx} u(x, y, t) + \partial_{yy} u(x, y, t)$. For the direction $\vec{N}$, they used an estimation of the isophote direction, which is the normal to the gradient direction, so $\vec{N}(x, y, t) := (\nabla u(x, y, t))^{\perp}$. A last thing to mention is that the authors interleaved this isophote-based inpainting with a diffusion process, in relation 15 times inpainting to 2 times diffusion, to avoid the crossing of isophotes.

To better understand this approach, we now want to rewrite things a little bit to make it match our general inpainting approach which will be described in Section 3.3.4. This means we strive for a PDE describing the evolution inside $\Omega \setminus \Omega'$.
As a first observation, we note that the interleaving of two different approaches, which are governed by the two elliptic differential operators $L_1(u)$ and $L_2(u)$, respectively, can be written as

$$\partial_t u = (1 - \varepsilon) \cdot L_1(u) + \varepsilon \cdot L_2(u), \tag{2.33}$$

with a relation factor $0 < \varepsilon < 1$. For a more detailled discussion on this, we refer the reader to Section 3.3.4.
What remains is to fill in the mentioned diffusion process, as the inpainting process is already described above. According to [8], the diffusion is an "anisotropic" diffusion given by the PDE

$$\partial_t u = g \cdot \kappa \cdot |\nabla u|, \tag{2.34}$$

where g is a scalar-valued Perona–Malik diffusivity (cf. Section 3.3.1), and κ is the curvature.
We see that this does *not* match our notion of anisotropic diffusion, as will be described

in Section 3.3.1. Now, we can finally assemble the PDE driving the interleaved inpainting in $\Omega \setminus \Omega'$ as

$$\partial_t u = (1 - \mathcal{X}_{\Omega'}) \cdot \left(\left[(1-\varepsilon) \cdot (\delta \vec{L} \cdot \vec{N}) \right] + \varepsilon \cdot (g \cdot \kappa \cdot |\nabla u|) \right) \tag{2.35}$$

$$= (1 - \mathcal{X}_{\Omega'}) \cdot \left(\left[(1-\varepsilon) \cdot \left\langle \nabla \triangle u, \nabla^{\perp} u \right\rangle \right] + \varepsilon \cdot \left(g \cdot |\nabla u| \cdot \operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right) \right) \right), \tag{2.36}$$

where $\mathcal{X}_{\Omega'}$ denotes the characteristic function of Ω'.

If we analyze our deduced evolution, we see that the right diffusion term is something like a Perona–Malik-like dampended mean curvature motion (MCM) (cf. Section 3.3.2), as we can write $|\nabla u| \cdot \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) = u_{\xi\xi}$. The left inpainting term is a third order PDE, as $\nabla \triangle u = \begin{pmatrix} u_{xxx} + u_{xyy} \\ u_{xxy} + u_{yyy} \end{pmatrix}$.

Approach of Chan and Shen

CDD Inpainting: In 2001 [17], Chan and Shen revised their former [18, 16] PDE-based total variation (TV) approach (see below) for image inpainting. There, they used the governing equation

$$\partial_t u = \operatorname{div} \left(\frac{1}{|\nabla u|} \cdot \nabla u \right) + \lambda_e (u - f), \tag{2.37}$$

with a scalar-valued diffusivity $\frac{1}{|\nabla u|} =: D$ and where $\lambda_e = \mathcal{X}_{\Omega'} \cdot \lambda$ is an extended Lagrange multiplier, as already described above in Section 2.3.1. $\lambda \in [0, 1]$ is a data parameter, determining the importance of the non-inpainted part of the given image f.
The major drawback of this method was that it does not follow the *connectivity principle* for single objects which are largely disconnected by the inpainting domain. The before mentioned principle was discovered by the psychologist G. Kanizsa in 1979 [33] and (simply spoken) says that human perception prefers connected results, even if large parts of an object are occluded (see also Figure 3.13 in Section 3.3.4). The main reason is that in the presented TV model, only the *contrast* (edges, visible in ∇u) of the image is considered, but not the *geometry*, which is visible in the scalar curvature κ.
This inspired the authors to come up with a new diffusion model, replacing the diffusivity $D = \frac{1}{|\nabla u|}$ by a new, curvature dependent diffusivity, yielding a *curvature-driven diffusion* (CDD) approach with

$$D' := \frac{g(|\kappa|)}{|\nabla u|}. \tag{2.38}$$

Here, $g(s)$ is an "annihilator" of large curvatures which has to satisfy

$$g(s) := \begin{cases} 0, & \text{if } s = 0 \\ \infty, & \text{if } s = \infty \\ c \in (0, \infty), & \text{if } 0 < s < \infty. \end{cases} \tag{2.39}$$

The reason for this choice is that at corners with $|\kappa| = \infty$, they want to have a strong diffusion, which tends to connect even largely interrupted objects (cf. Figure 3.13, again). The actual choice of the authors was

$$g(s) = \begin{cases} 0, & \text{if } s = 0 \\ s^p, \quad p \geq 1, & \text{if } s > 0, \end{cases} \tag{2.40}$$

which finally leads to the CDD-scheme

$$\partial_t u = (1 - \mathcal{X}_{\Omega'}) \cdot \Big(\operatorname{div} (D' \cdot \nabla u) \Big) + \mathcal{X}_{\Omega'} \cdot (u - f) \tag{2.41}$$

$$= (1 - \mathcal{X}_{\Omega'}) \cdot \left(\operatorname{div} \left(\frac{g(|\kappa|)}{|\nabla u|} \cdot \nabla u \right) \right) + \mathcal{X}_{\Omega'} \cdot (u - f). \tag{2.42}$$

2.4 Conclusions

Considering the above, one can say that the present image compression approaches mainly use direct transformations and their inverse transformations, for compression and decompression, respectively. As an example, we examined the cosine transform of JPEG [43] or the wavelet-transform of JPEG2000 [54].

PDE-based image compression is still in an early stage. The BTTC-based approach of Galic et al. [26] uses a quite sophisticated triangulation algorithm in the compression step and can hence rely on (comparably) straight-forward PDE-based inpainting in the decompression step.
The edge coding approach of Chan and Shen [19, 18] uses a simple but powerful edge detection in the compression step, allowing a quite simple TV inpainting in the decompression step. The caveat here is that the efficient storing of edges is not trivial at all.

Chapter 3

Theoretical Background

This chapter deals with the theoretical foundations of our codec, but what exactly do we understand under the notion *codec*?

A *codec* just encompasses two algorithms for image compression and decompression (*co*ding and *dec*oding), respectively. The compression step transforms a given raw-image into a compressed format, which should use significantly less memory. In the decompression step, we restore a raw-data image from the stored, compressed format, which usually is not identical to the initial image. In this case we have a *lossy compression* technique. The objective quality of the decompression result is usually poorer, but one aims at only loosing these kind of information which may not be too perturbing for a human observer.

In the upcoming chapter, we hence want to present on the one hand the corner detection approaches of the compression step, as well as the PDE-based inpainting techniques, applied to the sparse images in the decompression step. In this context, we will also give an introduction to PDE-based diffusion filtering and PDE-based continuous morphology. Large parts of these sections will be based on the lecture notes of the "Differential Equations in Image Processing and Computer Vision" lecture of Prof. Dr. Joachim Weickert [64], but we also used further references, which we will state.

Note: We will first introduce our concepts on **grayscale images** $u : \Omega \to \mathbb{R}$, which eases notation and understanding. Later on, we present extensions to color images, if needed.
In some cases we will not even have to do this, as we can just treat a color image $v : \Omega \to \mathbb{R}^3$ as three grayscale images, which we process independently. Note that this is only adequate if our filters are assumed to be independent of image features, like edges, as they are actually present in all three color channels together. In such cases, we will present the so-called *coupled approaches*, solving this problem.

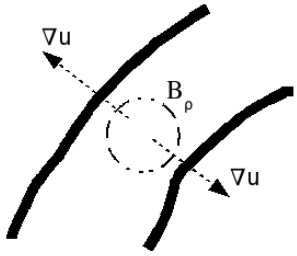

Figure 3.1: For flow-like structures, the local gradients ∇u cancel in a neighborhood B_ρ, which is larger than the flow structure.

3.1 Analyzing Local Structure – the Structure Tensor J_ρ

To be able to do a robust corner detection (cf. Section 3.2.1), but also for performing coherence-enhancing nonlinear anisotropic diffusion (cf. Section 3.3.1), one needs to analyze the local structure of a given image $u : \Omega \to \mathbb{R}$. So, we need a robust descriptor of local image structure for our compression step (corner detection part) and maybe also for our decompression step (inpainting using PDE-based diffusion).
The most common descriptor for the local image structure is the so called *structure tensor* J_ρ (also known as *second-moment matrix*, *scatter matrix* or *interest operator*) [24, 45], which we will introduce once now, as we will need it in Sections 3.2.1 and 3.3.1.

The structure tensor $J_\rho \equiv J_\rho(\nabla u(x, y))$ gives us a gradient-like descriptor of local image structure at point $(x, y) \in \Omega$. Local means in this case that we average the structures in some neighborhood B_ρ, which is just a circular neighborhood of radius ρ around point (x, y), formally:

$$B_\rho \equiv B_\rho(x, y) := \Big\{(x', y') \in \Omega \mid \sqrt{(x - x')^2 + (y - y')^2} \leq \rho\Big\}. \tag{3.1}$$

Note: For the sake of notational convenience, we will usually skip the explicit mentioning of points (x, y), i.e., for example we will write J_ρ instead of $J_\rho(\nabla u(x, y))$.

As one can show, the structure tensor is especially useful for analyzing flow-like structures, which usually suffer from cancellation errors when averaging gradients in opposite direction (cf. Figure 3.1), which will be for example useful for coherence-enhancing nonlinear anisotropic diffusion.

3.1.1 Definition

The structure tensor J_ρ is defined as

$$J_\rho \equiv J_\rho(\nabla u) := K_\rho * (\nabla u \nabla u^\top) = \begin{pmatrix} K_\rho * u_x^2 & K_\rho * u_x u_y \\ K_\rho * u_x u_y & K_\rho * u_y^2 \end{pmatrix}, \tag{3.2}$$

where K_ρ is just a Gaussian with standard deviation ρ (and standard deviation 0) defined as

$$K_\sigma(x) := \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{|x|^2}{2\sigma^2}\Big), \tag{3.3}$$

for $x \in \mathbb{R}^n, n \in \mathbb{N}$.
The operator $*$ is the *convolution* operator, defined for two functions $f, g : \mathbb{R} \to \mathbb{R}$ as

$$(f * g)(x) := \int_{\mathbb{R}} f(x - x') \cdot g(x') \ \mathrm{d}x'. \tag{3.4}$$

So, one can see that $J_\rho(\nabla u(x, y))$ is a symmetric, positive definite matrix for each point $(x, y) \in \Omega$. Its orthonormal eigenvectors v_1 and v_2 give the preferred local structure within $B_\rho \equiv B_\rho(x, y)$ and the corresponding eigenvalues λ_1 and λ_2 give the average contrast along v_1 and v_2, respectively. This means that v_1 is the orientation with the highest fluctuations and v_2 the preferred local orientation:
Let w.l.o.g. $\lambda_1 \geq \lambda_2$, then we can distinguish the following cases:

- If $\lambda_1 = \lambda_2 = 0$, then we have a *flat region* in point (x, y),
- if $\lambda_1 \gg \lambda_2 = 0$, then we have a *straight edge* in v_2-direction in point (x, y),
- if $\lambda_1 \geq \lambda_2 \gg 0$, then we have a *corner* in point (x, y).
- Finally, the expression $(\lambda_1 - \lambda_2)^2$ is a measure of the *anisotropy* (coherence) in point (x, y).

3.1.2 Regularization

For a more robust version of the structure tensor w.r.t. noise, it may be advisable to use a pre-smoothed version of the gradient ∇u, which we will denote by $\nabla u_\sigma := K_\sigma * \nabla u$. To be more exact, we just convolve the gradient with a Gaussian K_σ with standard deviation σ. This allows a spatial regularization, cf. regularization of the Perona–Malik model in Section 3.3.1.

3.1.3 Usage for Corner Detection

From the above, we see that analyzing the eigenvectors of J_ρ is a prefect mean to do a corner detection. We just have to do a thresholding w.r.t. λ_2. A detailed discussion of approaches to corner detection using the structure tensor can be found in Section 3.2.1.

3.1.4 Usage for Nonlinear Anisotropic Diffusion

The main goal of coherence-enhancing nonlinear anisotropic diffusion filters is to close interrupted lines of flow-like structures. To reach this aim, we have to take into account

the direction of local image structures, which is visible in the eigenvector v_2 of the structure tensor. For a more detailed discussion of nonlinear anisotropic diffusion, the reader may refer to Section 3.3.1.

3.2 Compression

In this section, we want to present the theoretical ideas behind the compression part of our codec.
As mentioned in Section 1.2, we compress our images by just storing a certain neighborhood of some corners, which we will call *corner regions*. This will result in a so called *sparse image*, constituting the basis for decompression using PDE-based inpainting, which we will describe in Section 3.3.
So, the first thing to do in the compression step is a *corner detection*, and we will introduce the theory behind this technique in this section.

Definition of a Corner

A neat definition of what a *corner* actually is can be found in Wikipedia [67]:

"A corner is a point for which there are two dominant edge directions in a local neighborhood of the point."

Definition of a Corner Region

In this context we should define now what we understand under the mentioned corner regions. It is important to note that this concept is a **discrete** one, a continuous description which we usually have in this chapter is less appropriate here. So, we will define corner regions in terms of discrete images, on which we will detail in Section 4.1. For now, it should suffice to know that if we have a continuous image $u : \mathbb{R}^2 \to \mathbb{R}$, where we address the image points (x, y) with $u(x, y)$, we have its discrete equivalent $\tilde{u} : \{0, \dots, h(N_1 - 1)\} \times \{0, \dots, h(N_2 - 1)\} \to \{0, 255\}$, with grid size h and image dimensions N_1 and N_2. Here, we use the notation $\tilde{u}_{i,j}$ to address the *pixels* (i, j) of $\tilde{u}$.
Basically, a *corner region* is just a (rectangular) neighborhood $\mathcal{N}$ of a corner pixel $(i, j) \in \Omega$. The classification of corner pixels is the output of a corner detector, which we will describe below. For $\mathcal{N}$, we can use several types of neighborhoods, like 4-neighborhods or 8-neighborhods (cf. Figure 3.2), but one may also think of using larger ones like 24-neighborhoods, or even non-rectangular ones. In this context, we note that we actually did *not* consider the corner itself at pixel (i, j). The reason is that it suffices to consider only the neighboring pixels, as they will not be too different from the corner itself, and they also suffice to store the vital gradient information.

Note: One may now notice that just storing the sparse image yields no compression at all, we would just store another image of the same size, but with a much poorer visual quality. Of course, one may develop clever approaches to exploit the sparsity for storing the sparse images in a space-efficient manner. These ideas we will present in Section 4.2.3.

	$(i, j-1)$	
$(i-1, j)$		$(i+1, j)$
	$(i, j+1)$	

$(i-1, j-1)$	$(i, j-1)$	$(i+1, j-1)$
$(i-1, j)$		$(i+1, j)$
$(i-1, j+1)$	$(i, j+1)$	$(i+1, j+1)$

Figure 3.2: **Top:** Corner region for pixel (i, j) and $\mathcal{N} = 4$. **Bottom:** Same for $\mathcal{N} = 8$.

Figure 3.3: Output of a corner detection for *lena* (512×512 pixels). Used Förstner–Harris corner detector with pre-smoothing scale $\sigma = 1$, integration scale $\rho = 4$ and threshold $\Theta = 2$ yielding 467 corners.

Definition of a Corner Detector

Note: From now on, we can go on with a continuous description, as announced.

The last thing to define is what we understand under the notion of a *corner detector*. Basically, this is just an algorithm that given an input image $u : \Omega \to \mathbb{R}$ classifies each point as being a corner, or not. Its output is hence just a binary image, v say, with

$$v(x, y) = \begin{cases} 1, & \text{if corner in pixel } (x, y) \\ 0, & \text{else.} \end{cases} \tag{3.5}$$

For a sample output of a corner detector, the reader may refer to Figure 3.3. Note that for illustration purposes, we did *not* give the binary image v, but the initial image with the detected corners marked with small white dots.

3.2.1 Corner Detection using the Structure Tensor

As mentioned in Section 3.1, the structure tensor J_ρ gives us a powerful mean to do corner detection.
If we define

$$J_\rho(\nabla u(x,y)) = K_\rho * (\nabla u(x,y)\nabla u(x,y)^\top) \tag{3.6}$$

as the structure tensor at point (x,y), we know that for its eigenvalues $\lambda_1 \geq \lambda_2$ we have: If $\lambda_1 \geq \lambda_2 \gg 0$, then there is a corner in point (x,y).
So, corner detection can be done by thresholding λ_2 w.r.t. a threshold parameter Θ. For this, several approaches exist, we will present in a minute.
But before we start we should note that for the actual implementation, we will use the regularized version of the structure tensor, as described in Section 3.1.2, namely $J_\rho(\nabla u_\sigma)$.

Förstner / Harris

Corner detection using the structure tensor originates on the one hand from Förstner and Gülch in 1986 [24] and from Harris and Stephens in 1988 [29].
The actual corner classification is shown below, where we should note that the authors did *not* use the eigenvalues of J_ρ, but the determinant and the trace, which are much easier to compute.

$$v(x,y) = 1 \iff \mathrm{tr}J_\rho(x,y) = K_\rho * u_x^2(x,y) + K_\rho * u_y^2(x,y) \overset{!}{>} \Theta \text{ and } \frac{\det J_\rho(x,y)}{\mathrm{tr}J_\rho(x,y)} \overset{!}{=} \max, \tag{3.7}$$

Note: max stands for a local maximum in this case.

Rohr

Quite similar to Förstner and Harris, Karl Rohr proposed in 1987 [46] the following approach, which only relies on the determinant of J_ρ :

$$v(x,y) = 1 \iff \det J_\rho(x,y) = K_\rho * u_x^2(x,y) \cdot K_\rho * u_y^2(x,y) - (K_\rho * u_x(x,y)u_y(x,y))^2 \overset{!}{=} \max, \tag{3.8}$$

Tomasi / Kanade

Another quite nice and robust approach originates from Tomasi and Kanade in 1991 [55], where they considered the eigenvalues of J_ρ as a corner measure, which is quite robust, but computationally expensive :

$$v(x,y) = 1 \iff \lambda_2 \overset{!}{>} \Theta \text{ and } \lambda_2 \overset{!}{=} \max, \tag{3.9}$$

where we recall that we assumed $\lambda_1 \geq \lambda_2$.

3.2.2 Extension to Vector-valued Images

Assume we now want to do a corner detection on a vector-valued, i.e., a color image $u : \Omega \to \mathbb{R}^3$. As corners are *not* equally present in all of the three channels, we have to couple local structure information from all channels in a *joint structure tensor* J_ρ. To compute it, we just sum up all the structure tensors of the channels, i.e., if we denote by $u_i : \Omega \to \mathbb{R}$ for $i = 1, \ldots, 3$ the i-th channel of a color image, we get

$$J_\rho \equiv J_\rho(\nabla u_1, \nabla u_2, \nabla u_3) = K_\rho * \sum_{i=1}^{3} (\nabla u_i \nabla u_i^\top). \tag{3.10}$$

In this tensor, we get the "average" structure direction in all the channels, and can use it as shown above for corner detection on vector-valued images.

3.3 Decompression

In this section, we want to present the theoretical ideas behind the decompression part of our codec. Remember that in our compression step, we stored corner information. From this we can obtain a sparse image, which is the basis for our decompression using PDE-based inpainting. How we actually store the information in the compression step will be explained in Section 4.2.
As we will use PDE-based inpainting of the sparse image (mainly using diffusion filters), we will first introduce basic concepts of PDE-based diffusion filters. Then we explain how to smoothly fill in missing information in the sparse image, using inpainting. We will also talk about PDE-based morphology, as we will also use these kind of filters for inpainting, which will be described in Section 3.3.4.

3.3.1 Diffusion using PDEs

First, we want to introduce the reader to PDE-based diffusion filters, as they are the heart of our inpainting methods, which are themselves the main part of our decoding step. This introduction mainly follows the review paper of Weickert [60], but for more details (especially theoretical results, like *well-posedness*, *existence, uniqueness and regularity of solutions*, *continuous dependency on initial image* and *scale-space properties* like *average gray level invariance*, *maximum-minimum-principle*, *Lyapunov functionals* and *convergence* to a constant steady state) the reader may refer to his book [61], which offers more information. For a visualization of the outcomes of these filters, one should have a look at Figure 3.9, at the end of this section.

Physical Background and General Approach

We start our introduction by describing how the notion of *diffusion*, which will be known to most persons from a physical context, carries over to image processing tasks.
In physics, diffusion is a process that equalizes concentration differences, while preserving the total mass, e.g., a metal surface which is heated at distinct points will after some time be equally warm in every spot, and in an ideal case the temperature will then be equal to the average temperature from the given initial temperature distribution.
The phenomenon of concentration equilibrium is expressed in *Fick's Law*:

$$j = -D \cdot \nabla u, \tag{3.11}$$

which states that a concentration gradient ∇u will cause a flux j of concentrations, which aims at compensating (hence the minus-sign) the gradient. The actual proportional relation between the strength of the flux j and the gradient ∇u is given by the so called *diffusion tensor* D, a positive symmetric matrix. Considering D, the following nomenclature is common:
If D is constant, we speak of *homogeneous, linear diffusion*, if it is a function of the differential structure of u, we are facing *nonlinear diffusion*. Considering j and ∇u, we

have *isotropic* diffusion if j and ∇u are parallel, and *anisotropic* diffusion if they are not parallel.
Remember that Fick's law only talks about compensating concentrations, but the preservation of mass is not expressed, which is treated by the so called *continuity equation*

$$\partial_t u = -\operatorname{div} j, \tag{3.12}$$

where t denotes the time, the diffusion process has evolved. The fact that no mass is created or lost here becomes more clear when rewriting equation 3.12 as $\partial_t u + \operatorname{div} j = 0$.

If we now plug equations 3.11 and 3.12 together, we finally obtain the *diffusion equation*, which will be our general model for all upcoming considerations of diffusion filters:

$$\partial_t u = \operatorname{div} (D \cdot \nabla u). \tag{3.13}$$

This equation is also known as the *heat equation*.

One last issue is how all this physical theory carries over to image processing. The answer is quite simple: We just consider an image $u : \Omega \to \mathbb{R}$ as the concentration distribution over the image domain. The favorable property here is that we can guarantee preservation of the average gray value (cf. preservation of total mass) during the evolution ($t \to \infty$) when using *reflecting* or *periodic boundary conditions*.

Boundary conditions: In the remainder of this section we assume homogeneous Neumann boundary conditions on $u(x, y, t)$, i.e.,

$$\partial_n u = 0 \qquad \text{on } \partial\Omega \times \mathbb{R}, \tag{3.14}$$

where $\partial\Omega$ denotes the image boundary of the image domain Ω and n the outer normal vector of $\partial\Omega$.

General Approach: Now, we can come to the general approach for all our diffusion filters, based on a *partial differential equation*, *PDE*, like the diffusion equation 3.13. If we are given an image $f(x, y) : \Omega \to \mathbb{R}$, we consider it as the *initialization* of a diffusion process of a function $u(x, y, t) : \Omega \times \mathbb{R} \to \mathbb{R}$ given by

$$\partial_t u = \operatorname{div} (D \cdot \nabla u), \tag{3.15}$$
$$u(x, y, 0) = f(x, y). \tag{3.16}$$

This means that we embed f into a continuous family

$$\{T_t f \mid t \geq 0\} \tag{3.17}$$

of gradually smoothed and simplified versions of it, where for $t = 0$ we have the initial image f. This also implies that we have $u(\cdot, \cdot, t) = T_t f$. Here we should note that unfortunately, it is common practice to denote with T also the stopping time of diffusion processes. This is the time point after which we stop the evolution. Hence, the result is then given by $u(x, y, T)$.
This concept is known as the famous *scale-space* concept, which was first introduced in the western world by Witkin in 1983 [71] and was –among many others– axiomatized by Alvarez, Guichard, Lions and Morel in 1993 [4]. Here, it is interesting to note that Iijima already introduced the scale-space ideas in 1962 in Japan [65]. The main importance of the scale-space concept lies in the fact that it introduces a hierarchy of image features, i.e., less important features vanish first, which constitutes an important step from pixelwise image representation to a semantical description. Here, we should also note that our evolution converges to the average gray value of f, i.e.,

$$\lim_{t \to \infty} u(x, y, t) = \mu, \tag{3.18}$$

where μ denotes the average gray value of f, defined as

$$\mu := \frac{1}{|\Omega|} \int_{\Omega} f(x, y) \ \mathrm{d}(x, y).$$

Now we are ready to come to the different diffusion processes, which we will discriminate by the choice of the diffusion tensor $D \in \mathbb{R}^{2\times 2}$.

Linear Isotropic Diffusion

This is the easiest case, where flux and gradient are directly proportional, so we choose

$$D = I := \mathrm{diag}(1, 1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{3.19}$$

Plugging this into the diffusion equation 3.13, we get

$$\begin{aligned} \partial_t u &= \mathrm{div}\,(I \cdot \nabla u) && (3.20)\\ &= \mathrm{div}\,(\nabla u) && (3.21)\\ &= \triangle u && (3.22)\\ &= \partial_{xx} u + \partial_{yy} u. && (3.23) \end{aligned}$$

Its (analytical) solution is given by the *convolution integral* of f with K_σ, a Gaussian with standard deviation σ, i.e.,

$$u(x, y, t) = \begin{cases} f(x, y), & \text{if } \ t = 0 \\ \big(K_{\sqrt{2t}} * f\big)(x, y), & \text{if } \ t > 0. \end{cases} \tag{3.24}$$

The main problem of linear isotropic diffusion, also called *homogeneous diffusion*, is the Gaussian smoothing it results in (see equation 3.24). So, it not only performs well in removing noise and small-scale details, but also in removing the image itself, i.e., also important image features like edges.
To overcome this problem, the idea is now to make the diffusion process inhomogeneous and steer it by the geometry of the image. This brings us to nonlinear isotropic diffusion.

Nonlinear Isotropic Diffusion

Inhomogeneous Linear Diffusion: Now, we want to include a-priori knowledge into our model, i.e., we actually want to reduce the diffusivity at edges, which we consider as important image features to be preserved, cf. [38].
To reach this, we use $|\nabla f|$ as a fuzzy edge detector of the initial image f. Points $(x, y) \in \Omega$ with a large $|\nabla f|(x, y)$ are more likely to belong to an edge, so we reduce diffusivity at these very points. To reach this, we use a scalar-valued diffusivity $g(|\nabla f|^2) : \mathbb{R} \to \mathbb{R}$. This yields a diffusion tensor D defined as follows:

$$D = \operatorname{diag}\Big(g(|\nabla f|^2),\, g(|\nabla f|^2)\Big) = \begin{pmatrix} g(|\nabla f|^2) & 0 \\ 0 & g(|\nabla f|^2) \end{pmatrix}, \tag{3.25}$$

with

$$g(|\nabla f|^2) := \frac{1}{\sqrt{1 + \frac{|\nabla f|^2}{\lambda^2}}} \quad , \qquad (\lambda > 0). \tag{3.26}$$

The diffusivity g defined in equation 3.26 is called *Charbonnier diffusivity* [20], with a *contrast parameter* λ.
If we plug this into the diffusion equation 3.13, we get

$$\partial_t u = \operatorname{div}\left(\operatorname{diag}\Big(g(|\nabla f|^2),\, g(|\nabla f|^2)\Big) \cdot \nabla u\right) \tag{3.27}$$

$$= \operatorname{div}\big(g(|\nabla f|^2) \cdot \nabla u\big). \tag{3.28}$$

Here, we should note that altough the adaption (which originates from Fritsch [25]) is nonlinear, the resulting diffusion equation is still linear, which will lead to artifacts reflecting the differential structure of f for large t.

Another thing to mention is that from now on, every resulting diffusion equation won't possess a nice analytical solution as it was the case for linear isotropic diffusion with the convolution integral (cf. equation 3.24). In fact, we will have to approximately solve the equations *numerically*, which we will describe in Section 4.3.1.

Nonlinear Isotropic Diffusion with the Perona–Malik Model: So, to come to a really nonlinear diffusion process which will cope with the artifact problem, we take

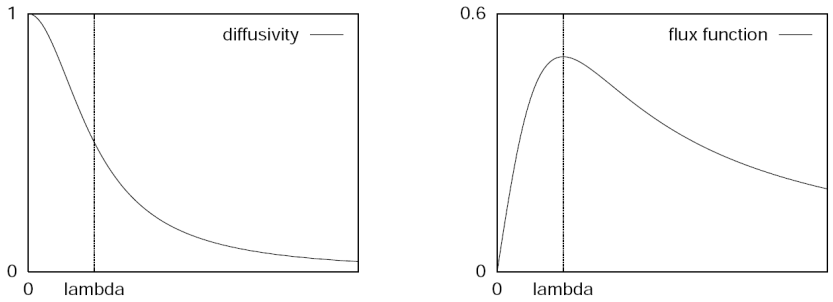

Figure 3.4: **Left:** Diffusivity $g(s^2) = \frac{1}{1+(s^2/\lambda^2)}$. **Right:** Corresponding flux function $\Phi(s) = g(s^2) \cdot s = \frac{s}{1+(s^2/\lambda^2)}$. **Author:** Weickert [60].

a look at the *Perona-Malik diffusivity* [44], which was the first nonlinear diffusion filter. It looks quite similar to the Charbonnier diffusivity, but to reduce artifacts, they adapt the diffusivity g to the gradient of the *evolving image* $u(x, y, t)$, instead of the one of the initial image f. This actually introduces the nonlinearity in the *nonlinear diffusion equation*:

$$\partial_t u = \operatorname{div}\big(g(|\nabla u|^2) \cdot \nabla u\big). \tag{3.29}$$

As a difference to Charbonnier, Perona and Malik also used a more rapidly decreasing diffusivity, by skipping the usage of the square root:

$$g(|\nabla u|^2) := \frac{1}{1 + \frac{|\nabla u|^2}{\lambda^2}}\ , \qquad (\lambda > 0). \tag{3.30}$$

One important property of the Perona–Malik model is that it is *edge-enhancing*, which we want to shortly study now. Let $\Psi(|\nabla u|)$ be a potential function whose gradient is given by the flux $\Phi(\nabla u)$, i.e.,

$$\nabla\Big(\Psi(|\nabla u|)\Big) = \Phi(\nabla u). \tag{3.31}$$

This yields for the flux $\Phi(\nabla u)$:

$$\Phi(\nabla u) = g(|\nabla u|^2) \cdot |\nabla u|. \tag{3.32}$$

For notational convenience, we define $s := |\nabla u|$ and restrict to the *1D-case*, i.e., $u = u(x, t)$, $s = |\partial_x u|$ and $\Phi(s) = g(s^2) \cdot s$.
After this notational issues, we can turn our attention the the edge enhancing property. If we consider the diffusivity in 3.30, we can show that the potential function $\Psi(s)$ is only convex for $|s| \leq \lambda$, and so for the flux $\Phi(s)$ we have that $\Phi'(s) \geq 0$ for $|s| \leq \lambda$ and $\Phi'(s) < 0$ for $|s| > \lambda$, which is illustrated in Figure 3.4.
Let us now consider and rewrite the nonlinear diffusion equation (cf. equation 3.29) for

the 1D-case:

$$\begin{aligned}
\partial_t u &= \operatorname{div}\big(g(s^2)\cdot\nabla u\big) && (3.33)\\
&= \operatorname{div}\big(g(s^2)\cdot\partial_x u\big)\\
&= \partial_x\big(g(s^2)\cdot\partial_x u\big)\\
&= \partial_x\big(g(s^2)\cdot s\big)\\
&= \partial_x\big(\Phi(s)\big)\\
&= \Phi'(s)\cdot\partial_x s\\
&= \Phi'\big(\partial_x u\big)\cdot\partial_{xx} u. && (3.34)
\end{aligned}$$

From the last lines, we can see that although $g(s^2) > 0$, for all s, the Perona–Malik model is of *forward parabolic type* for $|s| \leq \lambda$, as here $\Phi'(s) \geq 0$. A *backward parabolic type*, we have for $|s| > \lambda$, as here $\Phi'(s) < 0$, which means that the Perona–Malik model resembles the backward diffusion equation $\partial_t u = -\partial_{xx} u$ in this region. But what does these observations actually mean? To put it straight, forward diffusion for $|s| \leq \lambda$ smoothes the contrast, whereas backward diffusion for $|s| > \lambda$ enhances contrast, i.e., it enhances edges in our image if their gradient is larger than the contrast parameter λ.

Regularization: As one may imagine, the advantages of forward-backward diffusion also bring some drawbacks. The main problem is that it has big problems with image noise, as it will misinterpret noise as small-scale edges and hence enhance the noise. Furthermore, for the used smooth, nonmonotone flux functions $\Phi(s)$, there is no theory that guarantees *well-posedness* of the problem, i.e., there is no guarantee for a unique solution of the resulting diffusion equation, that depends continuously on the initial image. To overcome this, a *regularization* is needed.
One approach could be to use an implicit regularization in the discretization step, which leads to the so called *staircaising effect* as the only remaining shortcoming (see [60] and the references therein for more details on this).

But a maybe more elegant solution would be to regularize the continuous diffusion equation already. One attempt, we want to discuss is a spatial regularization (cf. Section 3.1.2), which averages the gradient within the diffusivity.
A first mathematical sound formulation of this was given by Catte, Lions, Morel and Coll [14] in 1992, where they proposed to replace the diffusivity $g(|\nabla u|^2)$ by $g(|\nabla u_\sigma|^2)$, where ∇u_σ is just the averaged gradient, or to be more exact, the convolution of the gradient with a Gaussian: $\nabla u_\sigma := K_\sigma * \nabla u$.
For the resulting *regularized nonlinear diffusion equation*

$$\partial_t u = \operatorname{div}\big(g(|\nabla u_\sigma|^2)\cdot\nabla u\big), \tag{3.35}$$

they could prove existence, uniqueness and regularity of a solution.

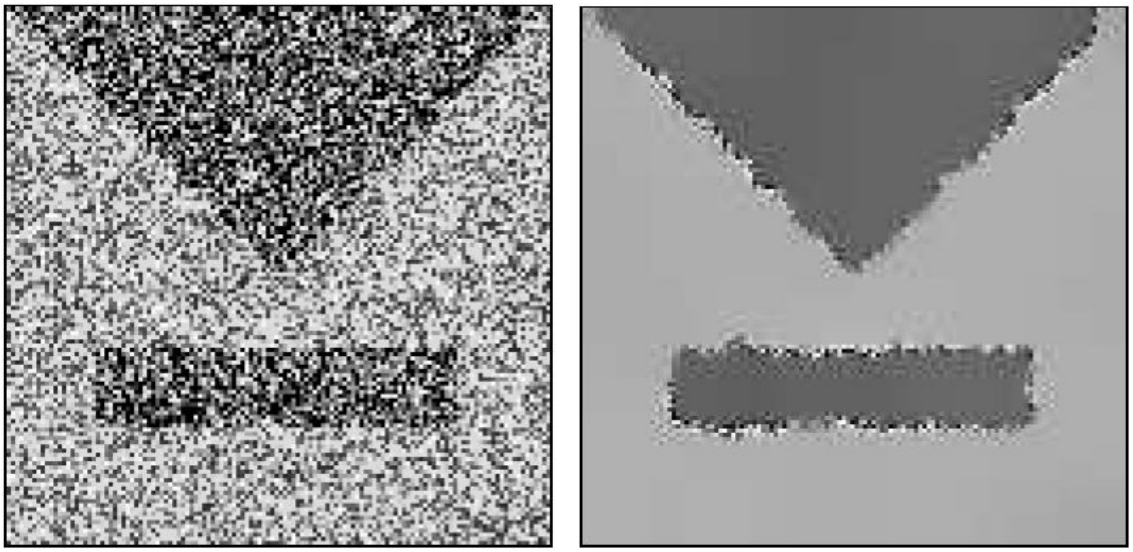

Figure 3.5: Behavior of Perona-Malik diffusion at a noisy test image. Note especially the unpleasant preservation of noise at the edges. **Left:** Initial Image (129×129 pixels). **Right:** Nonlinear isotropic diffusion (Perona–Malik) with $\lambda = 3.5, \sigma = 3$ and stopping time $T = 80$. **Author:** Weickert [61].

Nonlinear Anisotropic Diffusion

The initial question may be why one wants to further improve the process of nonlinear isotropic diffusion. The answer is again quite easy: Imagine the behavior of the Perona–Malik filter at a noisy edge. It will reduce diffusivity here and hence the edge will remain noisy (cf. Figure 3.5).

To overcome this, we need a real *anisotropic* behavior, i.e., we rotate the flux j towards the orientation of image features, like edges, which we could not do with the scalar-valued diffusivity $g(s^2)$. Here, the flux $j = -g(s^2) \cdot \nabla u$ was always parallel to ∇u. The result of such an anisotropic filter will be that it will smooth along discontinuities (edges), but not across them, which solves the noisy edges problem.

In the upcoming section, we will study two cases of nonlinear anisotropic diffusion: First, we describe *edge-enhancing diffusion (EED)*, which does exactly what we described, and secondly we will even go a step further and discuss *coherence-enhancing diffusion (CED)*, which can in addition enhance arbitrary flow- and line-like structures.

EED: As mentioned, our main goal will be to smooth along edges, but reduce smoothing across them, by rotating the flux towards the orientation of edges.
To reach this, we construct an orthonormal system of eigenvectors v_1, v_2 of our positive, symmetric diffusion tensor $D \in \mathbb{R}^{2\times 2}$, s.t., $v_1 \parallel \nabla u_\sigma$ and $v_2 \perp \nabla u_\sigma$, which actually rotates the flux in direction of the edges. If we now want to smooth along, but not

across edges, we choose the corresponding eigenvalues λ_1, λ_2 as shown in [58], namely

$$\lambda_1 := g(|\nabla u_\sigma|^2) \leq 1, \tag{3.36}$$
$$\lambda_2 := 1, \tag{3.37}$$

where $g(|\nabla u_\sigma|^2)$ is just the regularized Perona–Malik diffusivity. The reduction of λ_1 via the function g can be explained by the fact that the corresponding eigenvector $v_1 \parallel \nabla u_\sigma$ is perpendicular to the edge, as the gradient is itself is always perpendicular to an edge. Finally, to obtain our diffusion tensor D, we first have to figure out how to get v_1 and v_2. From above and the fact that we have to create an *orthonormal system*, we get

$$v_1 = \frac{\nabla u_\sigma}{|\nabla u_\sigma|}, \tag{3.38}$$

$$v_2 = \frac{\nabla u_\sigma^\perp}{|\nabla u_\sigma|}, \tag{3.39}$$

where, if we have $\nabla u_\sigma = K_\sigma * \begin{pmatrix} u_x \\ u_y \end{pmatrix}$ then $\nabla u_\sigma^\perp := K_\sigma * \begin{pmatrix} -u_y \\ u_x \end{pmatrix}$, for example.

If we put this together, we obtain by the *spectral theorem* (also known as *principal axis theorem*, *spectral decomposition* or *eigendecomposition*) the *edge-enhancing diffusion tensor* as

$$D = D(\nabla u_\sigma) \tag{3.40}$$

$$:= (v_1 \mid v_2) \cdot \operatorname{diag}(\lambda_1, \lambda_2) \cdot \begin{pmatrix} v_1^\top \\ v_2^\top \end{pmatrix} \tag{3.41}$$

$$= (v_1 \mid v_2) \cdot \begin{pmatrix} g(|\nabla u_\sigma|^2) & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1^\top \\ v_2^\top \end{pmatrix}. \tag{3.42}$$

Note: The reason why we can actually apply the spectral theorem here, is that we defined the diffusion tensor as a symmetric matrix, which can, using the spectral theorem, be decomposed as

$$D = V \cdot \Lambda \cdot V^\top. \tag{3.43}$$

Furthermore, this also explains why we only have to apply the Perona–Malik diffusivity $g(|s|^2)$ to the diagonal entries of Λ, as for applying a function $g : \mathbb{R} \to \mathbb{R}$ to a symmetric matrix D, we do a spectral decomposition of D, apply g just to the diagonal entries of

Λ, and then recompose $g(D)$. Formally,

$$\begin{aligned} g(D) &= g\big(V \cdot \Lambda \cdot V^\top\big) && (3.44)\\ &= V \cdot g(\Lambda) \cdot V^\top && (3.45)\\ &= V \cdot g\big(\mathrm{diag}(\lambda_1, \lambda_2)\big) \cdot V^\top && (3.46)\\ &= V \cdot \mathrm{diag}\big(g(\lambda_1), g(\lambda_2)\big) \cdot V^\top. && (3.47) \end{aligned}$$

Finally, the *edge-enhancing diffusion equation* then consequently reads

$$\partial_t u = \mathrm{div}\,\big(D(\nabla u_\sigma) \cdot \nabla u\big), \tag{3.48}$$

which is short for

$$\partial_t u = \mathrm{div}\left(\left((v_1 \mid v_2) \cdot \begin{pmatrix} g(|\nabla u_\sigma|^2) & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1^\top \\ v_2^\top \end{pmatrix}\right) \cdot \nabla u\right). \tag{3.49}$$

CED: One last thing that one actually *cannot* achieve with EED is the closing of locally interrupted line-like structures via enhancing flow-like structures [59]. But why one needs such a behavior? One nice example is the enhancing of a digital fingerprint image, as shown in Figure 3.6.

If we now want to reach the aim of enhancing flow-like structures, we need a more sophisticated structure descriptor than ∇u_σ, as such a purely *local* analysis will never be able to close interrupted lines. The problem here is namely that for large σ, cancellation effects occur at parallel lines which are quite close together. In such areas, the gradients will have the same length, but opposite orientation, and so averaging will cancel the gradients (cf. Figure 3.1). If we try to simply reduce σ, our filter will be very noise sensitive, and so we cannot circumvent to use a more sophisticated, *semilocal* structure descriptor, which will be the structure tensor J_ρ (cf. Section 3.1), which averages image structures in a given neighborhood B_ρ, which is semilocal. As a small modification, we will also use a Gaussian pre-smoothing of our image u, as seen in the regularized Perona–Malik model (cf. Section 3.3.1). We then consequently have

$$J_\rho(\nabla u_\sigma) = K_\rho * (\nabla u_\sigma \nabla u_\sigma^\top), \tag{3.50}$$

where we usually will use a relatively small regularization parameter σ and a larger integration scale ρ, which determines the area in which we average orientation information. As mentioned in Section 3.1, the eigenvectors v_1, v_2 of J_ρ give the preferred local image structure in a neighborhood B_ρ, to be more exact, v_2 gives the preferred local orientation, the so called *coherence direction*, in contrast to v_1, which is perpendicular to

Figure 3.6: Coherence-enhancing anisotropic diffusion of a fingerprint image. **Left:** Initial image (257×257 pixels). **Right:** Filtered with CED with $\sigma = 0.5, \rho = 4$ and stopping time $T = 20$. **Author:** Weickert [60].

v_2. Furthermore the corresponding eigenvalues $\mu_1 \geq \mu_2 \geq 0$ give a measure of local coherence, expressed in the expression $(\mu_1 - \mu_2)^2$. So, to enhance coherent structures, our filter should smooth mainly in coherence-, i.e., v_2 direction, with a diffusivity λ_2, which increases w.r.t. $(\mu_1 - \mu_2)^2$. This gives us our *coherence-enhancing diffusion tensor* D as

$$\begin{aligned} D &= D(J_\rho) && (3.51) \\ &:= (v_1 \mid v_2) \cdot \operatorname{diag}(\lambda_1, \lambda_2) \cdot \begin{pmatrix} v_1^\top \\ v_2^\top \end{pmatrix}. && (3.52) \end{aligned}$$

where

$$\lambda_1 := \alpha, \tag{3.53}$$

$$\lambda_2 := \begin{cases} \alpha, & \text{if } \mu_1 = \mu_2, \\ \alpha + (1-\alpha)\exp\Big(\frac{-C}{(\mu_1-\mu_2)^{2m}}\Big), & \text{else,} \end{cases} \tag{3.54}$$

with $C > 0$, $m \in \mathbb{N}$ and a small positive parameter $\alpha \in (0, 1)$, which is used to keep D uniformly positive definite.
From this, we can simply derive the *coherence-enhancing diffusion equation*, which reads

$$\partial_t u = \operatorname{div}\big(D(J_\rho) \cdot \nabla u\big), \tag{3.55}$$

with the definition of $D(J_\rho)$ as given above.

3.3.2 PDE-based Morphology

Another image filter we will use in the inpainting step during decompression is a so called *morphological filter*, namely *mean curvature motion* (*MCM*). It evolves image features in accordance to their curvature, hence the name. But before we come to this, we will first give a short introduction to *PDE-based morphology*, or *continuous-scale morphology*.
By definition, morphological filters are just image filters which analyze the shape of objects in images. All of these approaches go back to the year 1965 and the work of Serra and Matheron at the ENS des Mines de Paris in Fontainbleau. In the next years, their ideas became a very successful class of image analysis methods, and they still are up to now.
In the beginning, morphological filters were used for analyzing objects in *binary* images and later extensions to grayscale images were proposed, which use the level sets of the image as the shapes under consideration. A *level set* $L_\lambda(f)$ of an image $f : \Omega \to \mathbb{R}$, given a gray value λ, is simply defined as

$$L_\lambda(f) := \{x \in \Omega \mid f(x) \geq \lambda\}, \tag{3.56}$$

so

$$[L_\lambda(f)](x) := \begin{cases} 1, & \text{if } f(x) \geq \lambda \\ 0, & \text{else.} \end{cases} \tag{3.57}$$

Note that we can reconstruct a discrete and quantized image $\tilde{f} : \Omega_{disc} \to \{0, \ldots, 255\}$ given its level sets via

$$\tilde{f}_{i,j} = \sum_{\lambda=1}^{255} [L_\lambda(\tilde{f})]_{i,j}, \tag{3.58}$$

i.e., that level sets completely characterize an image.

Morphological Basics

Basically, morphological filters match the image pixels $f(x)$ with so called structuring elements $B \subset \mathbb{R}^2$. What they actually do is to replace the value of $f(x)$ by the supremum or infimum within B, called *dilation* and *erosion*, respectively. Their formal definition is given by

$$\text{Dilation}: \quad [\delta_B f](x) := \sup\{f(x-y) \mid y \in B\}, \tag{3.59}$$

$$\text{Erosion}: \quad [\varepsilon_B f](x) := \inf\{f(x+y) \mid y \in B\}. \tag{3.60}$$

Note that hence, dilation and erosion are dual operators. A sample application of these filters to a shape in a binary image is depicted in Figure 3.7. Surprisingly, just these two operations are the building blocks of all morphological filters. Their most important property is called *morphological invariance* and states that they are invariant under monotonously increasing gray level rescalings, i.e., contrast invariant.

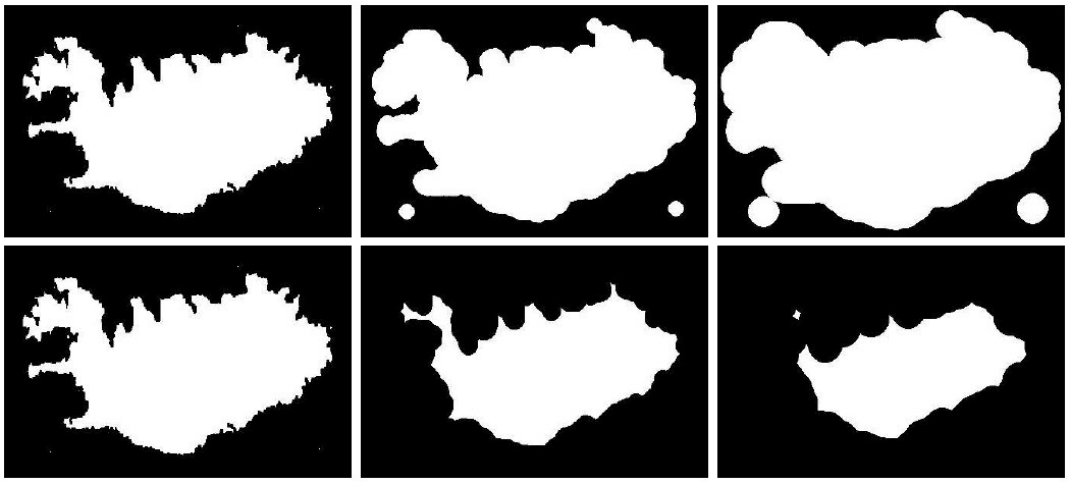

Figure 3.7: **Top row:** Dilation of the contour of Iceland with a disc shaped structuring element of size 0, 10 and 20. **Bottom row:** Same for erosion. **Author:** Weickert [64].

In the classical setting of above, the theory is described in terms of algebraic set theory [50, 51], but also PDE formulations [4, 49, 10] based on wave propagation (Huygen's principle) exist, which are naturally of more interest for our application to PDE-based inpainting.

Continuous-scale Morphology: Now, we want to describe how the –in principle discrete– ideas of dilation and erosion carry over to a (for us more useful) continuous framework. Note that the "discrete versions" of dilation and erosion simply use a discrete structuring element $B_{disc} \subset \mathbb{N}^2$ and replace sup by max and inf by min, respectively.
Now, to come to the *continuous-scale morphology*, we will introduce a convex structuring element tB with a scaling parameter t. Then the dilation $\delta_{tB} f =: u(t)$ of f and the erosion $\varepsilon_{tB} f =: u(t)$ boil down to solving the PDEs

$$\partial_t u(x,t) = \sup_{y \in tB} \langle y, \nabla u(x,t) \rangle, \tag{3.61}$$

$$\partial_t u(x,t) = \inf_{y \in tB} \langle y, \nabla u(x,t) \rangle, \tag{3.62}$$

respectively. As initial condition we use as suspected $u(x,0) = f(x)$, i.e. the initial image u.

Our next step will be to transfer the ideas of dilation and erosion to so called *curve evolutions*. If we consider a binary image object with its boundary curve $C_0(t)$, dilation with a disc-shaped structuring element results in the curve evolution

$$\partial_t C = n, \tag{3.63}$$

where n is a vector in outernormal direction of C and the initial condition is given by $C(0) = C_0$. This process propagates level lines (*isophotes*) with constant speed in outer

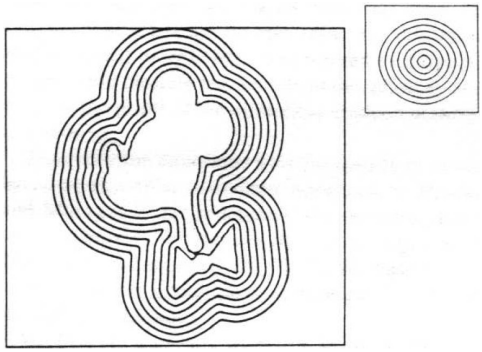

Figure 3.8: Contour evolution of shape "Mickey" with a disc shaped structuring element of increasing size. **Author:** Sapiro et al. [49].

normal direction (cf. Figure 3.8). In this case, isophotes are just the boundaries of level sets, i.e., for each gray level λ we define the corresponding isophote I_λ (or level line) as

$$I_\lambda := \{x \in \Omega \mid u(x) = \lambda\}. \tag{3.64}$$

Note that one can show that also the isophotes completely characterize an image, as shown for the level sets, above.
As erosion is dual to dilation, erosion is obtained by simply using $-n$ instead of n in equation 3.63.
If we now consider a more general curve evolution with a speed function $\beta(\kappa)$ depending on the *curvature* κ of C, we end up with

$$\partial_t C = \beta(\kappa) \cdot n, \tag{3.65}$$

where

$$\kappa := \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right), \tag{3.66}$$

as usual. We can now embed C as a *level set* in a smooth image $u(x,t)$ via the *signed distance function* [42]

$$u(x,t) := \begin{cases} \operatorname{dist}(x, C(t)), & \text{if x inside C,} \\ -\operatorname{dist}(x, C(t)), & \text{else,} \end{cases} \tag{3.67}$$

which yields the curve C as the zero-crossings of u. With this, an *image evolution*

$$\partial_t u = \beta(\kappa) \cdot |\nabla u|, \tag{3.68}$$

can be obtained, which gives a PDE for continuous-scale dilation.
After this introductory section, we can finally come to the MCM filter, we will actually use in our decompression step.

Mean Curvature Motion (MCM)

The goal of mean curvature motion is to smooth the isophotes I_λ of an image u. To reach this, let us again consider and rewrite the linear isotropic diffusion equation 3.23 as

$$\begin{aligned} \partial_t u &= \triangle u && (3.69)\\ &= \partial_{xx} u + \partial_{yy} u && (3.70)\\ &= \partial_{\xi\xi} u + \partial_{\eta\eta} u, && (3.71) \end{aligned}$$

where $\xi \perp \nabla u$ and $\eta \parallel \nabla u$. We realize that $\partial_{\eta\eta} u$ smoothes along flowlines, while $\partial_{\xi\xi} u$ smoothes along isophotes, so we can define the PDE for *mean curvature motion* (*MCM*) by only considering $\partial_{\xi\xi} u$ [4], i.e.,

$$\partial_t u = \partial_{\xi\xi} u. \tag{3.72}$$

But why does equation 3.72 describe a PDE for a morphological, curvature-dependent image evolution? To answer this we rewrite the equation as

$$\begin{aligned} \partial_t u &= \partial_{\xi\xi} u && (3.73)\\ &= \frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2} && (3.74)\\ &= |\nabla u| \cdot \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) && (3.75)\\ &= |\nabla u| \cdot \kappa, && (3.76) \end{aligned}$$

which is actually dependent on the curvature κ.
Note that equation 3.74 also gives a good starting point for a numerical implementation of MCM, as will be described in Section 4.3.1.

3.3.3 Extension to Vector-valued Images

In our considerations above, we were talking about scalar-valued images, i.e., grayscale images $u : \Omega \to \mathbb{R}$. Now, we want to turn our attention on how to process vector-valued images, i.e., in our case RGB color images $u : \Omega \to \mathbb{R}^3$. Here, we denote with $u_i : \Omega \to \mathbb{R}$ for $i = 1, \ldots, 3$ the three color channels, i.e., the red, green and blue channel.
In the remainder, we will distinguish two cases: The first one is to assume that the color channels are independent and process each of them separately, called uncoupled processing, which is very easy but rarely adequate. The second case will be formed by so called coupled approaches, which take into account dependencies between the color channels.

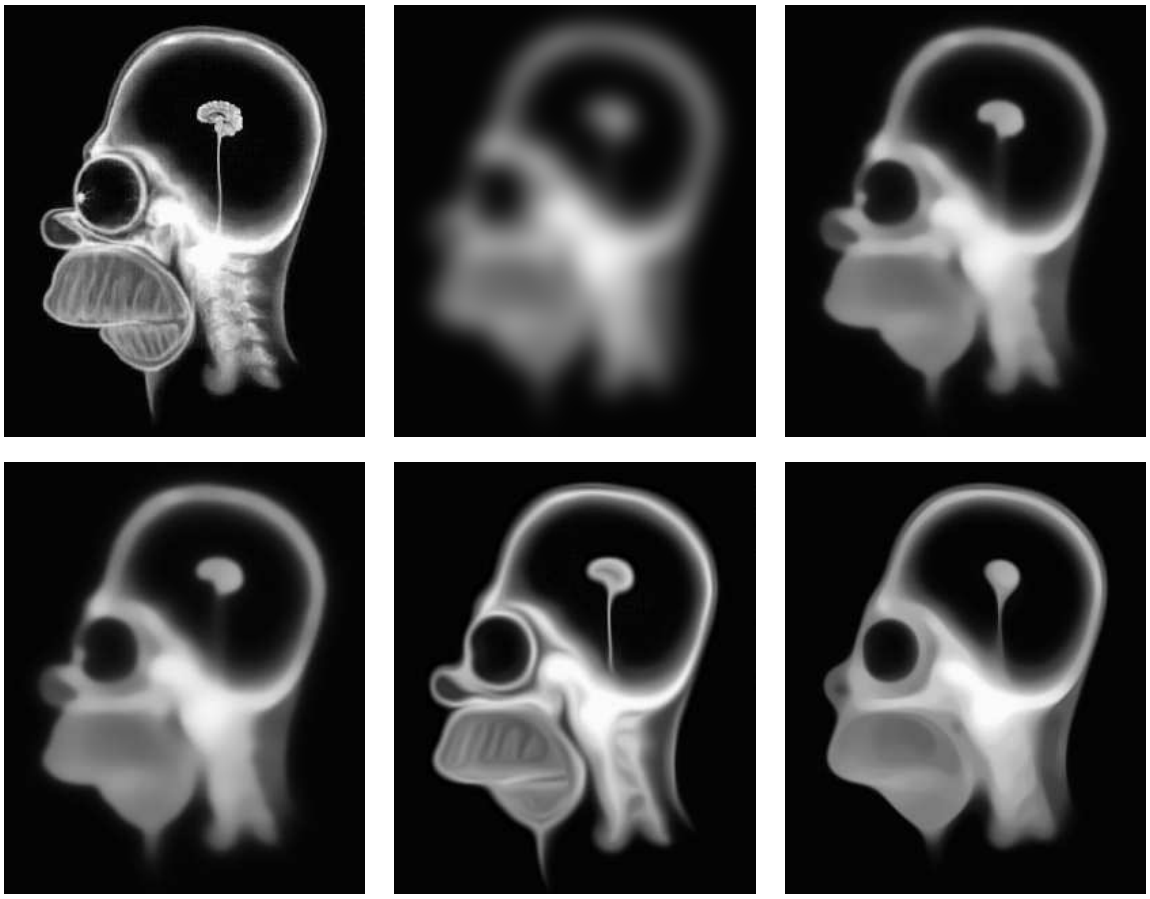

Figure 3.9: Illustration of the presented filters. **Top left:** Initial image, *brain* (266×318 pixels). **Top middle:** Homogeneous linear isotropic diffusion for stopping time $T = 40$, realized via a Gaussian convolution with $\sigma = \sqrt{2T} \approx 8.94$. **Top right:** Nonlinear isotropic diffusion (Perona–Malik) for $T = 40$ with $\lambda = 2.5$. **Bottom left:** Edge-enhancing diffusion (EED) for $T = 40$ with $\lambda = 3$ and $\sigma = 1$. **Bottom middle:** Coherence-enhancing diffusion (CED) for $T = 40$ with $C = 1, \sigma = 1, \rho = 3$ and $\alpha = 0.001$. **Bottom right:** Mean-curvature motion (MCM) for $T = 40$.

Uncoupled Approaches

These very simple approaches are useful if the used diffusivity of a diffusion filter is *not* space-variant, like for homogeneous diffusion. As said, we process every channel independently there and obtain for vector-valued linear isotropic diffusion

$$\partial_t u_i = \triangle u_i, \quad \text{for } i = 1, \ldots, 3. \tag{3.77}$$

Coupled Approaches

These are the method of choice for space-variant filters, which are indeed most of our filters.
For vector-valued regularized isotropic nonlinear diffusion (Perona–Malik model), Gerig et al. proposed in 1992 [27]

$$\partial_t u_i = \operatorname{div}\left(g\left(\sum_{j=1}^{3} |\nabla u_{j,\sigma}|^2 \right) \cdot \nabla u_i \right), \quad \text{for } i = 1, \ldots, 3, \tag{3.78}$$

which couples the three channels via a *joint diffusivity* $g(s_i^2)$, which depends on the sum of the gradient magnitudes in all three channels. This idea is also used for vector-valued EED and CED, which go back to Weickert in 1994 [57] and 1999 [62], respectively. He used a joint diffusion tensor and obtained for vector-valued EED

$$\partial_t u_i = \operatorname{div}\left(g\left(\sum_{j=1}^{3} \nabla u_{j,\sigma} \cdot \nabla u_{j,\sigma}^\top \right) \cdot \nabla u_i \right), \quad \text{for } i = 1, \ldots, 3, \tag{3.79}$$

and for vector-valued CED

$$\partial_t u_i = \operatorname{div}\left(D\left(\sum_{j=1}^{3} J_\rho(\nabla u_{j,\sigma}) \right) \cdot \nabla u_i \right), \quad \text{for } i = 1, \ldots, 3. \tag{3.80}$$

As a last consideration, we will now deal with vector-valued MCM. Chambolle [15] proposes two coupled approaches, although he mentions that for pure denoising an uncoupled approach

$$\partial_t u_i = \partial_{\xi_i \xi_i} u_i, \quad \text{for } i = 1, \ldots, 3, \tag{3.81}$$

may also perform quite well, where $\xi_i \perp \nabla u_i$. But let us now come to the coupled approaches, where one strives for defining an isophote direction ξ, which takes into account the direction of all isophotes ξ_i. This follows the idea to stop diffusion in the direction which maximizes the color change, which yields

$$\partial_t u_i = \partial_{\xi\xi} u_i, \quad \text{for } i = 1, \ldots, 3. \tag{3.82}$$

The first proposal is now to define ξ as the unit eigenvector to the minimal eigenvalue of the tensor

$$\sum_{i=1}^{3} \nabla u_i \cdot \nabla u_i^\top.$$

The second one is to choose $\xi \perp \eta$ with $\eta := (u_1 - u_2)\nabla u_3 + (u_2 - u_3)\nabla u_1 + (u_3 - u_1)\nabla u_2$.

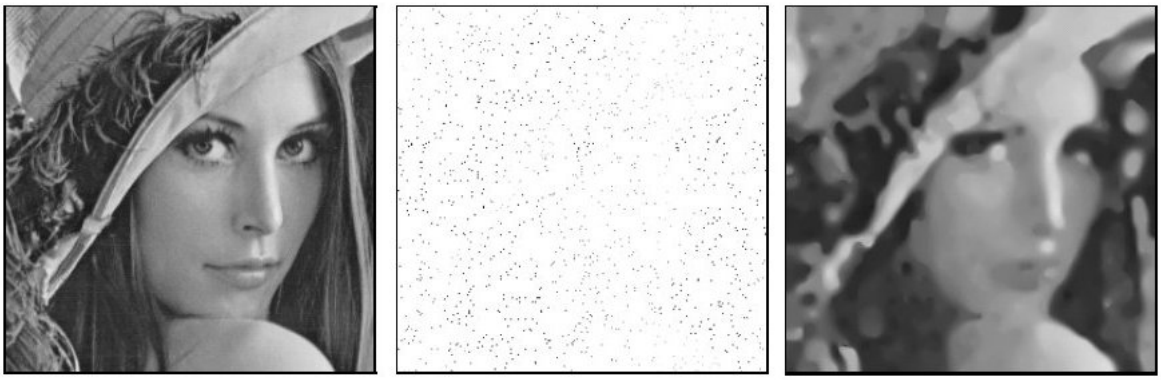

Figure 3.10: Strength of PDE-based inpainting using EED. **Left:** Initial image, *lena*. **Middle:** Sparse image, where only 2 % of all pixels were randomly chosen. **Right:** Decompression result, using EED in the inpainting step. **Author:** Galic et al. [26].

3.3.4 PDE-based Inpainting

In general, inpainting techniques were invented to restore images with small defects, like scratches, as proposed in [8, 17]. In the context of image compression, only PDE-based *diffusion* was used as a pre-processing step [52], or as a post-processing step [2, 73, 72, 28] to remove coding artifacts.
But as one can see in the work of Galic et al. [26], its usage for image compression, or to be more exact, for smoothly filling in missing information in sparse images, is quite fruitful. Nevertheless, the very sparse image data is quite a challenge.
Another insight we used from [26] is that anisotropic edge-enhancing diffusion (EED) is maybe the best choice as diffusion filter to use in the inpainting algorithm. This is the case if one aims at high compression rates and assumes that the images to compress are not too textured. The advantages of EED mainly lie in the fact that it respects discontinuities (EED smoothes along discontinuities, not across them) and the satisfaction of a max-min-principle. In [26], the strength of PDE-based inpainting using EED is nicely demonstrated: In Figure 3.10, we see an EED inpainting of the *lena* image, where only 2 percent of all pixels were randomly chosen, which is basically the most easy sparsification approach.

So, let us now investigate in how to inpaint the missing information, where we oriented our approach at the one of [26].

But before we actually start, one last remark concerning *textured images*. For those kind of images, diffusion-based inpainting is not a good choice, as its smoothing property will erase the statistical fluctuations which constitute the texture pattern. This problem can be nicely illustrated when considering the difference image between our decompression result and the initial image, depicted in Figure 3.11.

Figure 3.11: **Left:** Initial image, *lena* (512 × 512 pixels). **Right:** Difference image between *lena* and our decompression result (*Inverted*, i.e., darker values correspond to higher differences or errors.) Note especially the large difference at the strongly textured plumes of the hat.

General Approach

Assume we compress an initial given image $v : \Omega \to \mathbb{R}$ (usually $\Omega \subseteq \mathbb{R}^2$). From the resulting compressed data stream, we should be able to recover a sparse image $f : \Omega \to \mathbb{R}$, that is identical to v on some subset $\Omega' \subset \Omega$ and 0 elsewhere, i.e,

$$f(x) := \begin{cases} v(x), & \text{if } x \in \Omega' \\ 0, & \text{else, i.e., } x \in \Omega \setminus \Omega'. \end{cases} \tag{3.83}$$

Note that Ω' covers exactly our extracted corner regions.
We are now striving for our decompressed image $u : \Omega \to \mathbb{R}$, which is in fact an *interpolating function*, identical to v in Ω' (*interpolation condition*) and smooth and close to v in $\Omega \setminus \Omega'$. In a little bit informal manner, one may write

$$u(x) = \begin{cases} f(x), & \text{if } x \in \Omega' \\ \text{“smooth”}, & \text{else.} \end{cases} \tag{3.84}$$

For the “smooth” part, diffusion enters the game. We embed our problem in an evolution setting $u(x,t)$ with “time” parameter t, as seen for the diffusion equations above in Section 3.3.1. Our desired decompressed image u is then given by the steady state of this evolution, i.e., for $t \to \infty$. In accordance to the general diffusion equation 3.13, we then get the *general inpainting equation*

$$\partial_t u = \left(1 - \mathcal{X}_{\Omega'}\right) \cdot \underbrace{\operatorname{div}\left(D \cdot \nabla u\right)}_{\text{fill by diffusion}} - \mathcal{X}_{\Omega'} \cdot \underbrace{(u - f)}_{\text{unchanged}}, \tag{3.85}$$

with initial condition $u(x,0) = f(x)$ and where $\mathcal{X}_{\Omega'}$ denotes the characteristic function of the set Ω', i.e,

$$\mathcal{X}_{\Omega'}(x) = \begin{cases} 1, & \text{if } x \in \Omega' \\ 0, & \text{else, i.e., } x \in \Omega \setminus \Omega'. \end{cases} \tag{3.86}$$

Note that with this, we could also write $f(x)$ as $f(x) = v(x) \cdot \mathcal{X}_{\Omega'}(x)$.

If we now consider equation 3.85, we see that the left term of the sum, namely $\left(1-\mathcal{X}_{\Omega'}\right) \cdot \operatorname{div}(D \cdot \nabla u)$, does the smooth filling in of missing information via a diffusion process specified by the diffusion tensor D. But this is only done for $x \in \Omega \setminus \Omega' \Leftrightarrow \mathcal{X}_{\Omega'}(x) = 0$. In the known corner regions, i.e., for $x \in \Omega' \Leftrightarrow \mathcal{X}_{\Omega'}(x) = 1$, we do not change anything which is observable in the left term $\mathcal{X}_{\Omega'} \cdot (u - f)$.
From equation 3.85, we can derive the steady state equation by just setting $\partial_t u = 0$, as the scale-space property *convergence to a constant steady state* (cf. Section 3.3.1) guarantees that there exists a time t, where our evolution does not change $u(x,t)$ anymore, i.e., where $\partial_t u = 0$ holds. Hence, we get

$$\left(1 - \mathcal{X}_{\Omega'}\right) \cdot \operatorname{div}(D \cdot \nabla u) - \mathcal{X}_{\Omega'} \cdot (u - f) = 0, \tag{3.87}$$

with reflecting boundary conditions. Here, we also see that for $x \in \Omega' \Leftrightarrow \mathcal{X}_{\Omega'}(x) = 1$, the interpolation condition $u(x) = f(x) = v(x)$ is satisfied. In $x \in \Omega \setminus \Omega' \Leftrightarrow \mathcal{X}_{\Omega'}(x) = 0$, we have to satisfy $\operatorname{div}(D \cdot \nabla u) = 0$, which is the steady sate of the evolution

$$\partial_t u = \operatorname{div}(D \cdot \nabla u), \tag{3.88}$$

with Dirichlet boundary conditions. This equation should be known to us as our general diffusion equation 3.13.

The choice of D is now the important factor for a good decompression result. As mentioned in [26], $D = D(\nabla u_\sigma)$, so EED, is a good choice, but one may also investigate in using other approaches. For a more detailed discussion on which diffusion filter to use for the inpainting step, we refer to Section 5.2.1.

Other possibilities for the sparse image $f(x)$

At first, we adopted the approach from [26] to set the pixels of the inpainting domain $\Omega \setminus \Omega'$ to 0, as can be seen in the top right part of Figure 3.12. However, in the literature exist other ideas to initially fill the inpainting domain, which we will present now.

Before we start, one should note that *theoretically*, this choice should not make any difference as the desired steady state for $t \to \infty$ should be *in our case* independent of the initial guess for the inpainting domain $\Omega \setminus \Omega'$. The reason is that for EED, one assumes that there exists an equivalent variational approach based on minimizing a convex energy functional (cf. Section 2.3.1). The minimization can in this case be done by a

simple gradient descent algorithm, for which the initialization does not matter because of the convexity.
Interestingly, this assumption of an equivalent variational approach is only supported by empirical experiments. No one could come up with the concrete energy functional until today. For the much easier case of linear isotropic diffusion with the steady state equation $\triangle u = 0$, one can show that minimizing the energy functional

$$E(u) = \frac{1}{2} \int_{\Omega} \|\nabla u(x)\|_2^2 \, \mathrm{d}x \tag{3.89}$$

leads to a solution u which has to satisfy the mentioned steady state equation.
Practically, one may not iterate the evolution until one reaches a steady state, as this will take quite a long time, cf. Section 5.2.1. So, one may hope that a better initialization of the inpainting domain with a somewhat clever guess will help to faster (i.e., with less iterations) reach a favorable result. The reason is simply that for a convex setting, the initialization does not matter theoretically, but if one starts close to the unique global minimum of a convex energy functional, one will of course faster approach the minimum.
But let us now come to the different approaches for the initial filling of the inpainting domain.

First, we tried to initially fill the inpainting region with the average gray value μ of the initial image $v(x)$ (cf. Figure 3.12, bottom left), but this was not successfull, as will be shown in Section 5.2.4.
Inspired from the work of Chan and Shen [19, 18], we also tried to fill the inpainting domain with a random guess (cf. Figure 3.12, bottom right), i.e., we just set $f(x) = r_{[0,255]}$ in $\Omega \setminus \Omega'$, where $r_{[0,255]}$ denotes an uniformly distributed random number from the interval $[0, 255]$. As will be presented in Section 5.2.4, the objective results were a little bit worse than with our first approach (setting the pixels to 0), but the visual, subjective quality, we judged a little bit better, so one may prefer to use this method for generating "nicer" results for the human observer.

Interleaved Inpainting

As mentioned above, (pure) EED inpainting of the sparse images will work quite well for decompression, but the results can still be improved (cf. Section 5.2.2). The main reason is that we –in this thesis– maybe not used an optimal point set for creating the sparse images, namely corner regions. This assumption is justified by the fact that corners are a quite seldom image feature. So, we were striving to improve this by using a more sophisticated inpainting technique. Our idea was to not only use *one* diffusion filter for inpainting, but two ones. From an implementation point of view this will mean that we just interleave, hence the name *interleaved inpainting*, the two diffusion evolutions with a certain relation. This idea was inspired from the work of Bertalmio et al. [8] and

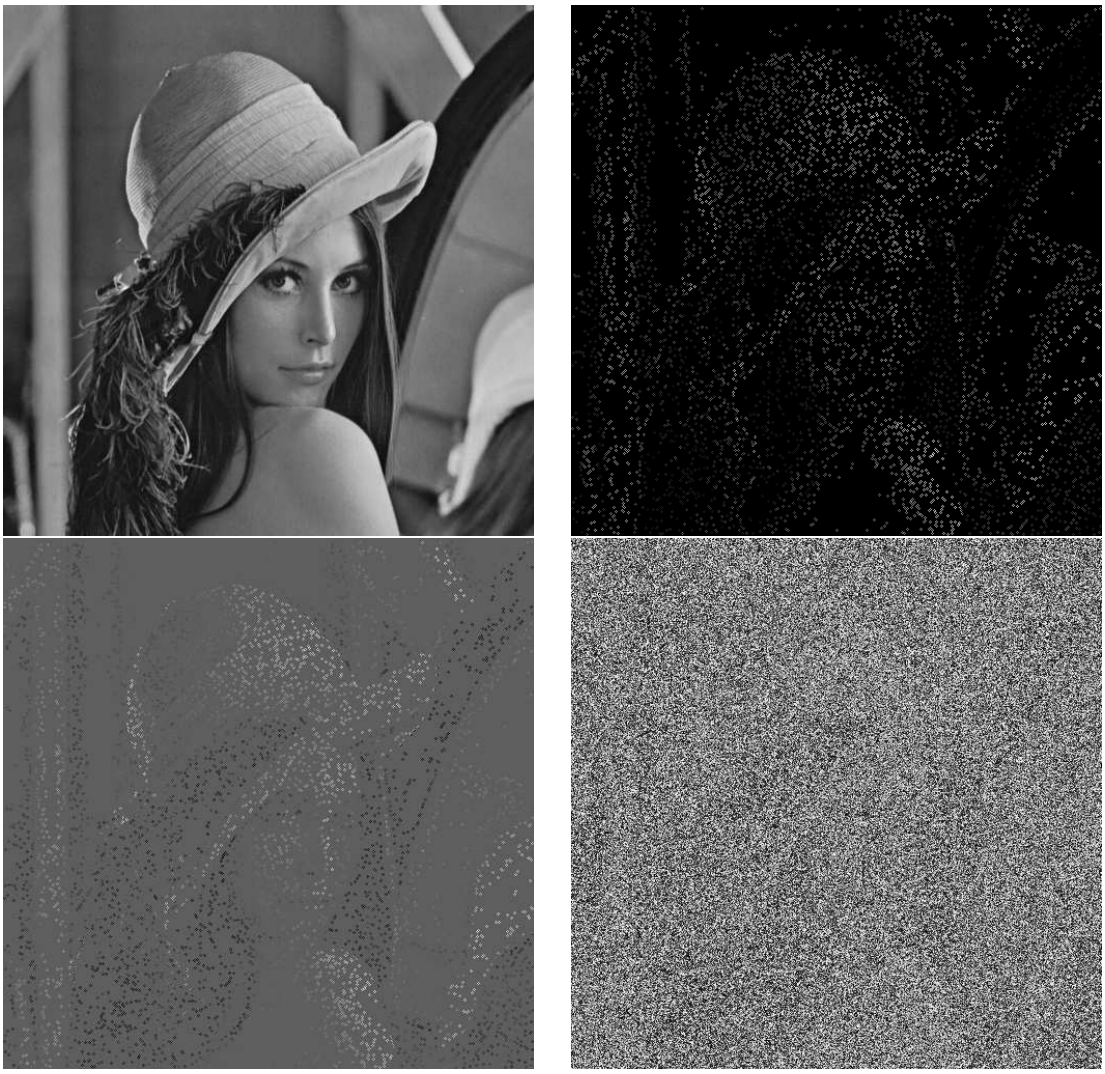

Figure 3.12: Different possibilities to fill the inpainting domain $\Omega \backslash \Omega'$ in the sparse image $f(x)$. **Top left:** Initial image $v(x)$, *lena* (512×512 pixels). **Top right:** Sparse image $f(x)$ for compression rate of 0.83 bpp, filled with black pixels. **Bottom left:** Same, but filled with average gray value μ of initial image $v(x)$. **Bottom right:** Same, but filled with random values in $[0, 255]$.

Caselles [13]. In our case, it led to an interleaving of EED as a diffusion process with MCM, a morphological process, which can also be written in PDE form.

The question that may arise is why one should use a morphological filter, like MCM, for the PDE-based inpainting. Usually (cf. [26]), one uses diffusion filters as presented above in section 3.3.1. However, in accordance to Chan and Shen [17], one major drawback of diffusion filters is that they only take into account the *strength* of the isophotes of an image, i.e., they only consider edge information visible in ∇u. What they lack is to consider the *geometry* of the image data, which is visible in the scalar curvature κ. But what is the actual problem when not considering the geometric image information? If we take a look at Figure 3.13 (taken from [17]), we see that for single objects where large parts are not given (i.e., have to be inpainted), pure diffusion inpainting techniques fail to restore these single objects, as –simply spoken– they do not strive for restoring isophotes, but "only" take into account edges. This result is against human intuition, as human perception follows a so called *connectivity principle*, which was discovered by the great psychologist Kanisza [33] in 1979. It mainly says that humans will mostly prefer connected results, as can be seen in Figure 3.13. Another reason for considering filters working on the level lines of images can be found in the article [13], which we will present in a minute. Here, the authors mention that "level lines [...] contain all of the image information invariant with respect to contrast changes" (cf. Section 3.3.2).
As mentioned, the idea to use MCM was further supported by the work of Caselles et al. in 1995 [13]. Here, the authors used the *junctions* and *atoms* of an image –which are closely related to our corner regions– for a kind of sparsification. To be more exact, the junctions are simply these points of an image where two level lines L_{λ_1} and L_{λ_2} with $\lambda_1 \neq \lambda_2$ meet. The atoms are then a neighborhood of the level lines arriving at junctions. With this, the authors proposed an adaption of a morphological filter (MCM), which was only applied to the atoms, and *not* to the junctions of an image.

To put above considerations more formal, we could say the following: Recall our usual diffusion equation

$$\partial_t u = \operatorname{div}(D \cdot \nabla u). \tag{3.90}$$

We see that the diffusivity D which "steers" the diffusion process is only dependent on ∇u, i.e., the isophote strength. To add geometric information, we have to consider the curvature κ, which is done in MCM, which we know is governed by the PDE

$$\partial_t u = |\nabla u| \cdot \kappa. \tag{3.91}$$

General Interleaving Framework: Motivated from above considerations, we want to develop a general, theoretical framework for interleaved PDE-based inpainting.
If we look at the general inpainting equation 3.85, we realize that we can rewrite the diffusion PDE which drives the actual inpainting evolution in $\Omega \setminus \Omega'$ in a more general

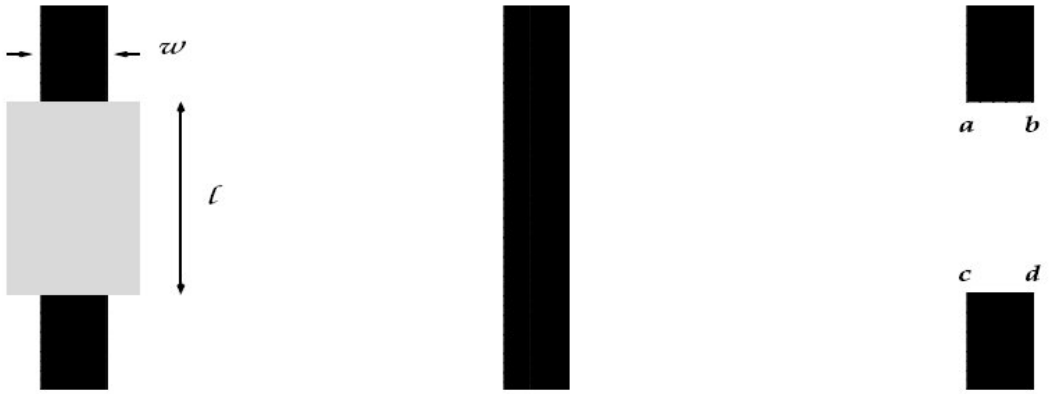

Figure 3.13: Illustration of the connectivity principle originating from Kanisza [33]. **Left:** Given image with occluded object (gray: inpainting domain $\Omega \setminus \Omega'$). **Middle:** Desired result for most human observers (*connected*!). **Right:** Result from most diffusion-based inpainting techniques (unfortunately *not connected*!) . **Author:** Chan and Shen [17].

form as

$$\begin{aligned} \partial_t u &= \operatorname{div}(D \cdot \nabla u) && (3.92) \\ &= L(u), && (3.93) \end{aligned}$$

with an elliptic differential operator L. For homogeneous diffusion we have $L(u) = \triangle u$, for example.
Now, we strive to interlave *two* PDE-based image filters, given by L_1 and L_2, respectively. This interleaving should be performed with a certain relation given by a factor $0 < \varepsilon < 1$. The question is now how to combine them in one PDE, which resembles the interlaving on implementation level.

Locally One-dimensional (LOD) Schemes: To answer this, we should have a look at the following simple homogeneous diffusion, which will be separated in x- and y-direction, which enables us to view it as an interleaving between the two directions (cf. Janenko's work from 1969, [31]). Following the approaches we will present in Section 4.3, we can discretize and rewrite the homogeneous diffusion equation

$$\partial_t u = \triangle u \tag{3.94}$$

as an *implicit scheme*

$$u^k = (I - \tau A) \cdot u^{k+1}, \tag{3.95}$$

with a system matrix A, which we split in $A_1 + A_2 := A$, with a matrix A_1 for the evolution in x-direction and A_2 for the y-direction, respectively. From this we can

perform a *LOD splitting* [31] of

$$\big(I - \tau(A_1 + A_2)\big) \cdot u^{k+1} = u^k, \tag{3.96}$$

which yields

$$\begin{aligned} (I - \tau A_1) \cdot u^{k+\frac{1}{2}} &= u^k & (3.97)\\ (I - \tau A_2) \cdot u^{k+1} &= u^{k+\frac{1}{2}}. & (3.98) \end{aligned}$$

If we want to quantify the error involved in this LOD scheme, we plug the above two equations together and obtain the implicit scheme

$$(I - \tau A_1) \cdot (I - \tau A_2) \cdot u^{k+1} = u^k, \tag{3.99}$$

which we can rewrite as

$$(I - \tau(A_2 - A_1) + \tau^2 A_1 A_2) \cdot u^{k+1} = u^k, \tag{3.100}$$

which shows that the erroneous term is $\tau^2 A_1 A_2$, which is in the size of τ^2 and hence vanishes if $\tau \to 0$.

To summarize our considerations, we see that the interleaving of two PDE-based image filters, can be written as a *sum*, which finally enables us to write the driving PDE for our interleaved inpainting of L_1 and L_2 with relation $0 < \varepsilon < 1$ as

$$\partial_t u = \varepsilon \cdot L_1(u) + (1 - \varepsilon) \cdot L_2(u). \tag{3.101}$$

Our Approach: Interleaving EED with MCM: Bertalmio et al. [8], interleaved a third order inpainting PDE with a Perona–Malik-like dampened MCM filtering (cf. Section 2.3).
Of course, our "main" diffusion type will be EED, but we decided to interleave it with MCM, following the ideas of Bertalmio, where they mentioned that the isophote direction ξ is a good direction to steer an inpainting process. Following our idea of above, where we wrote the interleaving as a sum of the two participants, we can write the driving PDE for our inpainting as

$$\partial_t u = \big(1 - \mathcal{X}_{\Omega'}\big) \cdot \Big(\varepsilon \cdot L_1(u) + (1 - \varepsilon) \cdot L_2(u)\Big) - \mathcal{X}_{\Omega'} \cdot (u - f). \tag{3.102}$$

Note that MCM is actually not a diffusion filter, but it can also be written in PDE form

$$\begin{aligned} \partial_t u &= \partial_{\xi\xi} u & (3.103)\\ &= |\nabla u| \cdot \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) & (3.104)\\ &= |\nabla u| \cdot \kappa & (3.105)\\ &=: L_{\mathrm{MCM}}(u), & (3.106) \end{aligned}$$

which enables us to use it in our interlaving framework (given by equation 3.101), which will then look for interleaving of EED with MCM like

$$\partial_t u = \varepsilon \cdot \underbrace{\partial_{\xi\xi} u}_{=L_{\mathrm{MCM}}} + (1-\varepsilon) \cdot \underbrace{\operatorname{div}\left(D(\nabla u_\sigma) \cdot \nabla u\right)}_{=:L_{\mathrm{EED}}} \tag{3.107}$$

$$= \varepsilon \cdot |\nabla u| \cdot \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) + (1-\varepsilon) \cdot \operatorname{div}\left(D(\nabla u_\sigma) \cdot \nabla u\right). \tag{3.108}$$

If we plug this into our interleaved inpainting framework from equation 3.102, we finally obtain

$$\partial_t u = \left(1-\mathcal{X}_{\Omega'}\right) \cdot \left(\varepsilon \cdot |\nabla u| \cdot \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) + (1-\varepsilon) \cdot \operatorname{div}\left(D(\nabla u_\sigma) \cdot \nabla u\right)\right) - \mathcal{X}_{\Omega'} \cdot (u-f). \tag{3.109}$$

3.3.5 Error Measures

One important evaluation tool for our codec will be the error between the initial, uncompressed image v and the decompression result u. This means we somehow have to define a measure for the deviation of u from v.
There exist several ideas for computing such a measure, but we decided to use the *Average Absolute Difference (AAD)*, which is defined as

$$AAD(u,v) := \frac{1}{|\Omega|} \cdot \int_\Omega |u(x) - v(x)| \, \mathrm{d}x. \tag{3.110}$$

Chapter 4

Implementation Details

In this chapter, we will describe the implementation of our ideas for compression and decompression, given in Section 3.2 and 3.3, respectively.
We will first concern ourselves with discrete images and describe our approach for discretizing derivatives using the finite difference method. Then we detail on the implementation of the compression and the decompression part of our codec.
For the first part, we will shortly talk about straight forward discretizations of corner detectors and define a format for efficiently coding the sparse images. More involved is the discretization of the PDE-based image filters used for inpainting in the decompression step, which will constitute the main part of this chapter. Finally, we will explain how to use the deduced discrete PDE-based filters for the inpainting needed in the decompression step.

4.1 Discretization of Images

In our theoretical considerations, we developed a continuous framework for compression (corner detection using the structure tensor J_ρ, cf. Section 3.2.1) and decompression (PDE-based inpainting, cf. Section 3.3.4).
For this, we used images represented as real-valued functions with a real-valued domain, $f : \Omega \to \mathbb{R}$ with $\Omega \subset \mathbb{R}^2$. This is favorable for developing a theoretical framework, but it should be clear that the pixel structure of digital images poses a need for discretizing the continuous framework, to be able to implement our ideas on a computer.

So, in this chapter we will deal with *discretized images* defined on a *discrete pixel grid*, which is N_1 pixels wide, N_2 pixels high and uses a *grid size* of h_1 in x-direction and h_2 in y-direction. We have $h_1 = a_1/N_1,\ h_2 = a_2/N_2$, if we assume that the support of Ω lies on the rectangle $(0, a_1) \times (0, a_2)$. Usually, we will use $h_1 = h_2 = 1$, as this is common practice in image processing, but our approaches will be written in a general manner, using the arbitrary grid sizes h_1 and h_2.
This actually means that we transfer the continuous image domain $\Omega \subseteq \mathbb{R}^2$ of our images

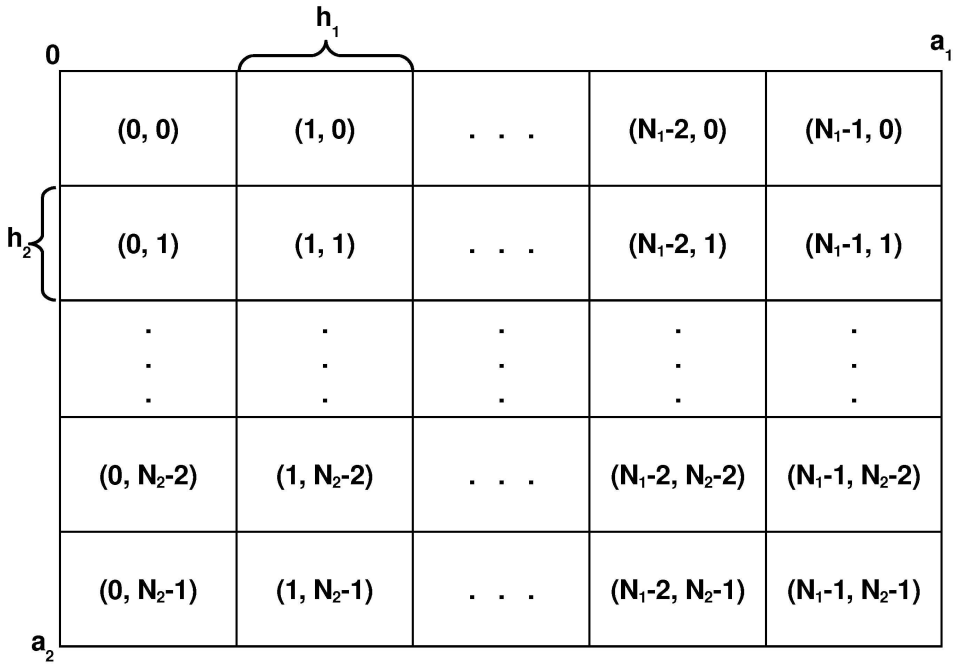

Figure 4.1: Discrete pixel grid of a continuous image of width a_1 and height a_2, discretized using the grid sizes $h_1 = a_1/N_1$ and $h_2 = a_2/N_2$ in x-, respectively y-direction. This yields N_1 pixels ($\{0, \dots, N_1 - 1\}$) in x-direction and N_2 pixels ($\{0, \dots, N_2 - 1\}$) in y-direction.

to a rectangular, discrete domain $\Omega_{disc} = \{0, \dots, N_1 - 1\} \times \{0, \dots, N_2 - 1\}$ (assuming $h_1 = h_2 = 1$), which is known as *sampling*. But we will also discretize the co-domain (the gray values) to usually $\{0, \dots, 255\}$, representing 256 different gray values, which is known as *quantization*. Putting all of this together we get images

$$\tilde{f} : \Omega_{disc} \to \{0, \dots, 255\},$$

where we denote by $\tilde{f}_{i,j}$ the gray value of $\tilde{f}$ at pixel (i, j) with $0 \leq i \leq N_1 - 1$ and $0 \leq j \leq N_2 - 1$. To be more exact, we can say that the pixel $(i, j) \in \Omega_{disc}$ represents the coordinates $(x_i, y_j) \in \Omega$ via

$$x_i = \left(i - \frac{1}{2}\right) h_1 \tag{4.1}$$

$$y_j = \left(j - \frac{1}{2}\right) h_2. \tag{4.2}$$

To clarify our theoretical considerations one can inspect a sketch of a discrete pixel grid in Figure 4.1.
As a last note, we want to point out that sometimes, we will not define a discrete

image in terms of a function, but as a vector from the $\mathbb{R}^N$ with $N := N_1 \cdot N_2$, i.e., $\tilde{u} = (\tilde{u}_1, \ldots, \tilde{u}_N)^\top$. Note that one may also skip the $\top$-sign.

Now we should recall that for our diffusion filters, we embedded the resulting image into a continuous family of gradually smoothed images $\{u(x, y, t) \mid t \geq 0\}$ (*scale-space concept*), i.e., we introduced a notion of continuous time. In the discrete case, we will just consider fixed timepoints $t_k = k \cdot \tau,\ k \in \mathbb{N}$ with a *time step size* τ. To denote the gray values of an image u at pixel (i, j) and at time t_k, we will from now on write $u_{i,j}^k$. Another thing to note is that we were heavily using the concept of image derivatives, as the gradient $\nabla u := (\partial_x u, \partial_y u)^\top$ was used in the structure tensor $J_\rho(\nabla u_\sigma)$ for corner detection in the compression step, as well as in the PDEs of the decompression step, which look in general like $\partial_t u = \operatorname{div}(D \cdot \nabla u)$. This has the consequence that we unfortunately will also have to discretize the notion of derivatives, which are obviously a continuous concept. In this context we should also note that for almost all diffusion filters (except for homogeneous diffusion), we cannot find an analytical solution, i.e., we have to work with numerical approximations anyway, which need "discretized derivatives".

4.1.1 The Finite Difference Method

The discretization of (image) derivatives is an essential step in implementing our ideas. The method we are actually using for this end is the so called *finite difference method*, which is for example described in the book of Morton and Mayers [41]. This approach basically aims at replacing the continuous concept of *differential quotients*

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{4.3}$$

by the discrete concept of (finite) *difference quotients*, which we will deduce in a minute. But before that, we should note that there also exist other approaches to discretize derivatives, but the pixel grid structure of digital images makes finite differences the method of choice for our purposes. If one is interested in alternative approaches, one should have a look at the *finite element* methods [7, 32, 47], which are also quite popular.

But let us now come to the finite difference method. The idea here is to approximate image derivatives in a pixel $(i, j) \in \Omega_{disc}$ by taking into account the values of its neighboring pixels, which we will now illustrate by a 1-D example, taken from [64].
Let us assume that we want to approximate the second derivative u'' of u in point i by using the three-neighborhood $i-1, i, i+1$ (which actually uses two neighbors plus the point itself). The question is now how the gray values u_{i-1}, u_i and u_{i+1} have to be weighted.
To answer this we do a *Taylor expansion* of u_{i+h} around point i, where h denotes the grid size in x-direction. If we plug in the definition of the Taylor series

$$f(x) = T_m(x, \xi) + \mathcal{O}(h^{m+1}) = \sum_{k=0}^{m} \frac{(x-\xi)^k}{k!} f^{(k)} + \mathcal{O}(h^{m+1}). \tag{4.4}$$

If we denote by $u_i^{(k)}$ the k-th derivative of u_i, we get

$$\begin{aligned} u_i + h &= T_m(i+h, i) + \mathcal{O}(h^{m+1}) && (4.5)\\ &= \sum_{k=0}^{m} \frac{(i+h-i)^k}{k!}\, u_i^{(k)} + \mathcal{O}(h^{m+1}) && (4.6)\\ &= \sum_{k=0}^{m} \frac{h^k}{k!}\, u_i^{(k)} + \mathcal{O}(h^{m+1}). && (4.7) \end{aligned}$$

For our example we then get

$$\begin{aligned} u_{i-1} &= u_i - h u_i^{'} + \frac{h^2}{2} u_i^{''} - \frac{h^3}{6} u_i^{'''} + \frac{h^4}{24} u_i^{''''} - \frac{h^5}{120} u_i^{'''''} + \mathcal{O}(h^6) && (4.8)\\ u_i &= u_i && (4.9)\\ u_{i+1} &= u_i + h u_i^{'} + \frac{h^2}{2} u_i^{''} + \frac{h^3}{6} u_i^{'''} + \frac{h^4}{24} u_i^{''''} + \frac{h^5}{120} u_i^{'''''} + \mathcal{O}(h^6). && (4.10) \end{aligned}$$

From this we can do a *comparison of coefficients* to finally approximate $u^{''}$ using u_{i-1}, u_i and u_{i+1}:

$$u_i^{''} \overset{!}{\approx} \alpha_{-1} \cdot u_{i-1} + \alpha_0 \cdot u_i + \alpha_1 \cdot u_{i+1}, \tag{4.11}$$

with real-valued weights α_{-1}, α_0 and α_1. We can rewrite this equation as

$$\begin{aligned} 0 \cdot u_i + 0 \cdot u_i^{'} + 1 \cdot u_i^{''} &\overset{!}{\approx} \alpha_{-1} \cdot u_{i-1} + \alpha_0 \cdot u_i + \alpha_1 \cdot u_{i+1} && (4.12)\\ &= (1 \cdot \alpha_{-1} + 1 \cdot \alpha_0 + 1 \cdot \alpha_1) \cdot u_i && (4.13)\\ &\quad + (-h \cdot \alpha_{-1} + 0 \cdot \alpha_0 + h \cdot \alpha_1) \cdot u_i^{'}\\ &\quad + \left(\frac{h^2}{2} \cdot \alpha_{-1} + 0 \cdot \alpha_0 + \frac{h^2}{2} \cdot \alpha_1 \right) \cdot u_i^{''}\\ &\quad + \mathcal{O}(h^3), \end{aligned}$$

which gives us the sought-after weights α_{-1}, α_0 and α_1 by solving the follwing linear system of equations.

$$\begin{pmatrix} 1 & 1 & 1 \\ -h & 0 & h \\ \frac{h^2}{2} & 0 & \frac{h^2}{2} \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{4.14}$$

One can easily verify that its solution is given by

$$\alpha_{-1} = \frac{1}{h^2}, \quad \alpha_0 = -\frac{2}{h^2} \quad \text{and} \quad \alpha_1 = \frac{1}{h^2}. \tag{4.15}$$

This yields for our approximation

$$u_i^{''} \approx \alpha_{-1} \cdot u_{i-1} + \alpha_0 \cdot u_i + \alpha_1 \cdot u_{i+1} \tag{4.16}$$

$$= \frac{1}{h^2} \cdot u_{i-1} - \frac{2}{h^2} \cdot u_i + \frac{1}{h^2} \cdot u_{i+1} \tag{4.17}$$

$$= \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}. \tag{4.18}$$

The question that should immediately arise is "how large is the error involved in this approximation?"
To answer this, we plug the Taylor expansions (4.8), (4.9) and (4.10) into the solution (4.18) and get after some calculations

$$u_i^{''} \approx \frac{1}{h^2} \cdot u_{i-1} - \frac{2}{h^2} \cdot u_i + \frac{1}{h^2} \cdot u_{i+1} = \cdots = u_i^{''} + \frac{h^2}{12} \cdot u_i^{''''} + \mathcal{O}(h^4), \tag{4.19}$$

where the first non-vanishing coefficient $\frac{h^2}{12}$ shows that the approximation error lies in $\mathcal{O}(h^2)$, which finally leads to

$$u_i^{''} = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + \mathcal{O}(h^2). \tag{4.20}$$

In the same manner one can show the follwing approximations of first derivatives in 1D:

$$u_i^{'} = \frac{u_{i+1} - u_i}{h} + \mathcal{O}(h), \tag{4.21}$$

$$u_i^{'} = \frac{u_i - u_{i-1}}{h} + \mathcal{O}(h), \tag{4.22}$$

$$u_i^{'} = \frac{u_{i+1} - u_{i-1}}{h} + \mathcal{O}(h^2). \tag{4.23}$$

which we call *forward difference*, *backward difference* and *central difference*, respectively. Note that central differences have a smaller error, and so we will mainly use them.

4.2 Compression

The main part of the compression step is constituted by the corner detection, whose implementation we will describe now.

4.2.1 Corner Detection

The discretization of the corner detectors described in Section 3.2.1 is quite straight forward. Remember that we mainly used the structure tensor $J_\rho(\nabla u_\sigma)$ to determine the cornerness of pixels. This means that we can just compute for every pixel (i, j) the corresponding

$$[J_\rho(\nabla u_\sigma)]_{(i,j)} := K_\rho * (\nabla u^\sigma_{i,j} \nabla u^{\sigma^\top}_{i,j}) \tag{4.24}$$

$$= K_\rho * \begin{pmatrix} (\partial_x u^\sigma_{i,j})^2 & \partial_x u^\sigma_{i,j} \cdot \partial_y u^\sigma_{i,j} \\ \partial_x u^\sigma_{i,j} \cdot \partial_y u^\sigma_{i,j} & (\partial_y u^\sigma_{i,j})^2 \end{pmatrix}, \tag{4.25}$$

where $\nabla u^\sigma_{i,j}$ denotes the gradient of the pre-smoothed image u at pixel (i, j). The occurring partial derivatives can be approximated by central differences as described above. Recall that some of our corner detectors used the eigenvalues or the determinant of J_ρ, i.e., we have to compute these values for all matrices $[J_\rho(\nabla u_\sigma)]_{(i,j)}$ of all pixels $(i, j) \in \Omega_{disc}$. These computations are straight-forward and can be found in any textbook on linear algebra, e.g., in [5].

4.2.2 Compression Rates

An important issue is what compression rates we can achieve for compressing a given discrete image $v : \Omega_{disc} \to \{0, \ldots, 255\}$ into a compressed image $u : \Omega_{disc} \to \{0, \ldots, 255\}$. To answer this, we will now present a formula to compute the compression rates F and F_{bpp}. Here, F denotes the *absolute compression rate*, whereas F_{bpp} gives the compression rates in *bpp*, *bits per pixel*.
Before we start, we have to fix some notation.

- N_1 and N_2 are the image dimensions in width and height, respectively,
- c is the number of color channels, so $c = 1$ for grayscale images and $c = 3$ for color images,
- n is the number of extracted corner regions,
- $\mathcal{N}$ is the neighborhood size of the corner regions, usually $\mathcal{N} \in \{4, 8\}$,
- $space(f)$ gives the space consumption of an image f measured in bytes and
- $space$(Corner Region) denotes the space consumption of a single stored corner region.

With this notation, the formula for computing the absolute compression rate F reads

$$\begin{aligned} F &\equiv F(c, \mathcal{N}, n, N_1, N_2) && (4.26)\\ &:= \frac{space(u)}{space(f)} && (4.27)\\ &= \frac{n \cdot space(\text{Corner Region})}{c \cdot N_1 \cdot N_2} && (4.28)\\ &= \frac{n \cdot \left(\frac{3}{4} \cdot \mathcal{N} \cdot c + 2\right)}{c \cdot N_1 \cdot N_2} && (4.29) \end{aligned}$$

The space consumption of a corner region $space(\text{Corner Region}) = \frac{3}{4} \cdot \mathcal{N} \cdot c + 2$ can be explained as follows. First, forget about the $\frac{3}{4}$ (we will deal with it in a minute), then every corner region consists of $\mathcal{N}$ pixels for all color channels c. In addition, we have to store the coordinate of the corner which will need 2 bytes and can be explained as follows. We basically index all pixels by a single index k, which we –for a reasonable image size– would have to store as `long` datatype, consuming 4 bytes. However, our idea is to only store an offset o_n from the last read index k_{n-1}, which we can do with a `short` datatype, consuming only 2 bytes.

Single Indexing: The single index k is actually computed by a function $p : \mathbb{N}^2 \to \mathbb{N}$ which just indexes all pixels (i, j) starting in the upper left corner at $(0, 0)$ and ending at the lower right corner $(N_1 - 1, N_2 - 1)$ of our image, so that we have

$$p(i, j) := i \cdot N_1 + j =: k. \tag{4.30}$$

For the decompression, we have to compute $p^{-1} : \mathbb{N} \to \mathbb{N}^2$ via

$$p^{-1}(i, j) = \begin{pmatrix} k & \text{div } w \\ k & \text{mod } w \end{pmatrix}, \tag{4.31}$$

where k div w denotes integer division, k div $w := \lfloor k/w \rfloor$, for example 9 div 2 = 4. With k mod $w := k - (k \text{ div } w) \cdot w$, we denote the modulo operation, for example 9 mod 2 = 1.

The n-th offset o_n is easily computed from the indices k_n and k_{n-1} via

$$o_n := k_n - k_{n-1}, \tag{4.32}$$

and for decompression we can compute the index k_n from the stored offset o_n and k_{n-1} as

$$k_n = k_{n-1} + o_n. \tag{4.33}$$

Quantization: Do you recall that we forgot about the $\frac{3}{4}$-factor above? So we should explain this now. The reason why we only need to store $\frac{3}{4} \cdot \mathcal{N}$ bytes for every corner region is *quantization* of the color values. This means we only store 6 bits per color value in each channel instead of the usual 8 bits = 1 byte, which explains the factor of $\frac{3}{4} = \frac{6}{8}$.
This also means that we can only store values in $\{0, \ldots, 2^6 - 1\} = \{0, \ldots, 63\}$ instead of $\{0, \ldots, 2^8 - 1\} = \{0, \ldots, 255\}$. So, we have to scale every color value down by a factor of $1/\big(2 \cdot (8-6)\big) = \frac{1}{4}$ when we store it. Of course, we have then to rescale it by a factor of 4 in the decompression step.
In this context, we should note that quantization is of course a *lossy* compression step, as we discard information. If we consider our used quantization from 8 to 6 bits, we can show that the error (the deviation from the restored to the original color value) involved is at most 3:
Assume we quantize a color value $x \in \{0, \ldots, 255\}$. Then x is quantized (and stored) as $\tilde{x} := \lfloor \frac{x}{4} \rfloor \in \{0, \ldots, 63\}$. In the decompression step we compute the restored color value $x' \in \{0, \ldots, 255\}$ as $x' = 4\tilde{x}$, which yields an error of $error(x, x') := |x' - x| = |4\tilde{x} - x| = |4\lfloor \frac{x}{4} \rfloor - x| = x \mod 4 \leq 3$.

Implementation Details: The question is now how to implement our quantization ideas? The approach we used is to exploit the fact that we will actually use $\mathcal{N} = 4$, i.e., we store 4 color values with 6 bits per corner region, yielding $4 \cdot 6 = 24$ bits = 3 bytes per corner region. So we only store 3 bytes as an `unsigned char` datatype. These bytes we will denote by `b0,b1,b2`. The 4 color values in the initial image we will denote by `a0, a1, a2, a3`. First, we divide each `ai` by a factor of 4 and then store `a0,a1,a2` in `b0,b1,b2`. Because of our division by 4 we will only have to store values in $\{0, \ldots, 63\}$, i.e., the two upper bits of `b0,b1,b2` are not touched. These $2 \cdot 3 = 6$ bits we use to store `a3`, i.e., we distribute the 6 bits of `a3` in the two upper bits of `b0,b1,b2`.
In the decompression step, we just read the three stored bytes `b0,b1,b2` and restore the 4 color values of our corner region `a0,a1,a2,a3` by storing the first 6 bits of `b0,b1,b2` in `a0,a1,a2`, respectively and assembling the two upper bits of `b0,b1,b2` to `a3`.

Computing F_{bpp}: Recall, that we also wanted to compute a compression rate in bpp, i.e., F_{bpp}. Fortunately, it can be easily computed from F via

$$\begin{aligned} F_{bpp} &= F_{bpp}(c, \mathcal{N}, n, N_1, N_2) && (4.34) \\ &:= 8 \cdot \frac{space(u)}{N_1 \cdot N_2} && (4.35) \\ &= 8 \cdot \frac{n \cdot (c \cdot \frac{3}{4} \cdot \mathcal{N} + 2)}{N_1 \cdot N_2} && (4.36) \\ &= 8 \cdot c \cdot F. && (4.37) \end{aligned}$$

Last but not least, we should mention that the computation of the compression rate is only an approximation, as we for example did not take into account the header of our compressed files, which we store in the *CRF-format* we invented and want to present now.

4.2.3 CRF-Format

After corner detection is finished, we somehow have to store the extracted information. This is not too difficult, we just store for each corner region the coordinate of its center and the color values of all pixels in the neighborhood in some fixed order. Files of this format we will call *CRF-files*(**C**orner **R**egion **F**ormat). This format –we invented– is specified in Figure 4.2.
From a given CRF-file, it is quite easy to recover the sparse image and a mask for PDE-based inpainting, like described in 3.3.4. We just have to consecutively read the corner regions and then put the color values in the fixed order in the sparse image, accordingly to the read coordinates. In addition, for every put corner pixel, we will also set the according pixel in the mask.

4.3 Decompression

The decompression step mainly uses PDE-based inpainting with diffusion filters or MCM. So, we will first describe their implementation, based on the theoretical background presented in Section 3.3.1. Then we will step over to the implementation of the PDE-based inpainting using the previously described filters.

4.3.1 Discretizations of PDE-based Diffusion Filters

Now, we present the implementation of the diffusion filters from Section 3.3.1, i.e., we will deduce discrete schemes which allow us to compute the *discrete image evolution* $\{u_{i,j}^k \mid k \geq 0\}$ for discrete timepoints $t_k = k \cdot \tau$ and a fixed time step size τ.
Before we start, we should note that the upcoming sections are quite concise and tailored towards our needs. The reader not so familiar with these topics, or the one interested in general or theoretical explanations should refer to the book of Weickert [61].

Linear Isotropic Diffusion

This is the easiest case, as we recall that here, there actually exists an analytical solution $u_{i,j}^k = K_{\sqrt{2t_k}} * f$. This means that we can just compute the convolution of the initial image f with a Gaussian K of corresponding standard deviation $\sigma = \sqrt{2t_k}$. However, to get in touch with the discretization of diffusion filters resulting in a discrete scheme, we will now deduce this scheme for linear isotropic diffusion, as this is quite easy to

CRF
Comments
⋮
Comments
$< N_1 >< N_2 >$
$< c >$
$< n >$
$< \mathcal{N} >$
$< o_1 > (c_1^1.1)(c_1^2.1)(c_1^3.1)\dots(c_1^{\mathcal{N}}.1) \ \dots\dots\dots \ (c_1^1.c)(c_1^2.c)(c_1^3.c)\dots(c_1^{\mathcal{N}}.c) < o_2 > (c_1^1.1)(c_1^2.1)(c_1^3.1)\dots(c_1^{\mathcal{N}}.1) \ \dots\dots\dots \ (c_1^1.c)(c_1^2.c)(c_1^3.c)\dots(c_1^{\mathcal{N}}.c) \ \dots\dots$
⋮
⋮

$< o_n > (c_n^1.1)(c_n^2.1)(c_n^3.1)\dots(c_n^{\mathcal{N}}.1) \ \dots\dots\dots \ (c_n^1.c)(c_n^2.c)(c_n^3.c)\dots(c_n^{\mathcal{N}}.c)$

Figure 4.2: CRF-file specification.

Explanations:
$< N_1 >< N_2 >$ are the image dimensions in width and height, respectively.
$< c >$ is the number of color channels, so $c = 1$ for grayscale images and $c = 3$ for color images.
$< n >$ is the number of corner regions.
$< \mathcal{N} >$ is the neighborhood size of the corner regions, usually $\mathcal{N} = 4$.
$< o_i >$ for $i = 1, \dots N$ is the offset of the single indexed coordinates of the center of corner region i, where $N := N_1 \cdot N_2$.
$(c_i^j.k)$ denotes the k-th color channel of the j-th element of corner region i.

$(\cdot)$ denotes that the value is stores as a *byte* $\in [0, 255]$, i.e., `unsigned char`-datatype consuming 1 byte and
$< \cdot >$ denotes that the value is stored in ASCII-format, as datatype `unsigned short`, consuming 2 bytes.

understand.
Recall that the corresponding diffusion equation reads

$$\partial_t u = \triangle u = \partial_{xx} u + \partial_{yy} u \tag{4.38}$$

in this case. If we approximate $\partial_t u$ by the forward difference

$$\partial_t u \approx \frac{u_{i,j}^{k+1} - u_{i,j}^k}{\tau} \tag{4.39}$$

and the second spatial derivatives by

$$\partial_{xx} u \approx \frac{u_{i-1,j}^k - 2u_{i,j}^k + u_{i+1,j}^k}{h_1^2} \tag{4.40}$$

for the x-direction with grid size h_1, and the equivalent for the y-direction, we obtain

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\tau} = \frac{u_{i-1,j}^k - 2u_{i,j}^k + u_{i+1,j}^k}{h_1^2} + \frac{u_{i,j-1}^k - 2u_{i,j}^k + u_{i,j+1}^k}{h_2^2}. \tag{4.41}$$

The favorable thing here is that we can *explicitly* compute the sought-after $u_{i,j}^{k+1}$, i.e., the image at the next time step $k+1$, as we can verify that

$$u_{i,j}^{k+1} = \Big(1 - 2\frac{\tau}{h_1^2} - 2\frac{\tau}{h_2^2}\Big) u_{i,j}^k + \frac{\tau}{h_1^2} u_{i+1,j}^k + \frac{\tau}{h_1^2} u_{i-1,j} + \frac{\tau}{h_2^2} u_{i,j+1}^k + \frac{\tau}{h_2^2} u_{i,j-1}^k \tag{4.42}$$

holds, which we will call *explicit finite difference scheme*. As we apply this scheme to all pixels (i,j), it is convenient to write the weights in *stencil notation*, cf. Figure 4.3.

0	$\frac{\tau}{h_2^2}$	0
$\frac{\tau}{h_1^2}$	$1 - 2\frac{\tau}{h_1^2} - 2\frac{\tau}{h_2^2}$	$\frac{\tau}{h_1^2}$
0	$\frac{\tau}{h_2^2}$	0

Figure 4.3: Stencil notation for the explicit finite difference scheme for linear isotropic diffusion.

Boundaries: One question that should arise to the astute reader is what to do at image boundaries, e.g., in pixel $(0,0)$, as we cannot directly apply our stencil there. Usually, we will use *Homogeneous Neumann boundary conditions*, i.e., we just mirror our image at the boundaries, we say that we introduce *dummy pixels*, which look in a 1D-case with $\Omega_{disc} = \{0, \dots, N_1 - 1\}$ like $u_{-1}^k := u_0^k$ and $u_{N_1}^k := u_{N_1-1}^k$.
This can be explained as follows: Remember that usually $\Omega \subset \mathbb{R}^2$, so we have a real subset of the $\mathbb{R}^2$, which gives rise to the question what our filters do at the *image boundaries*, denoted $\partial\Omega$, of an image $u : \Omega \to \mathbb{R}$.

A first, simple approach are so called *Dirichlet boundary conditions*, which set all values outside of Ω to 0, i.e., $u(x, y) = 0$ on $\partial\Omega$.
A second approach, which is also the approach we use, are *Homogeneous Neumann boundary conditions*, which do the mirroring of the image pixels at the boundaries. Because of this, these boundary conditions are also known as *reflecting boundary conditions*. Formally, this means $n^\top \cdot \nabla u(x, y) = 0$ on $\partial\Omega$, where n is the outer normal vector of the boundary $\partial\Omega$ and $n^\top \cdot \nabla u = \partial_n u$ is the directional derivative w.r.t. n (cf. Section 1.4). The favorable consequence of the usage of Neumann boundary conditions are that no flux of gray values across the boundaries occurs, yielding a preservation of the average gray value. This is the reason why they are usually more preferable than Dirichlet boundary conditions.

Well-posedness and scale-space properties: These properties of diffusion filters are (theoretically) quite important, but also hard to digest in a short way. We decided to just give the results and refer the interested reader once more to the book of Weickert [61], where one can find proofs and in-depth explanations for all the well-posedness (stability) and scale-space conditions, like average gray level invariance, maximum-minimum principle, Lyapunov functions and convergence to a constant steady state.
Furthermore, we have to mention that we will only present the for us more important semidiscrete and discrete theory. The continuous theory is completely skipped here, but can also be found in [61].

For the linear isotropic case, we have to impose that the stencil weights sum up to 1 and that they are all non-negative, yielding a *discrete maximum-minimum principle*. This just means that during the evolution the smallest gray value won't become smaller than the smallest gray value of the initial image and the same for the largest gray value, formally

$$\min_{n,m} f_{n,m} \leq u_{i,j}^k \leq \max_{n,m} f_{n,m} \ , \quad \forall k \geq 0, \ \ \forall i, j \in \Omega_{disc}. \tag{4.43}$$

As we can see from Figure 4.3, the only crucial stencil weight for the non-negativity condition is the central one, and we can verify that for

$$\tau \leq \frac{1}{\frac{2}{h_1^2} + \frac{2}{h_2^2}}, \tag{4.44}$$

we can achieve non-negativity. For the common case with $h_1 = h_2 = 1$, we obtain

$$\tau \leq \frac{1}{4}. \tag{4.45}$$

Extension to Vector-Valued Images: If we have to process vector-valued images, i.e. RGB-color images with three channels in our case, we can use for linear isotropic diffusion an uncoupled approach, as described in Section 3.3.3. The reason for that is that homogeneous filters, like linear isotropic diffusion, are space-invariant, and hence we can simply apply the deduced stencil separately for all three color channels.

Nonlinear Isotropic Diffusion

Let us now come to the discretization of the presented nonlinear isotropic diffusion filter, i.e., in our case the *regularized Perona–Malik model*.
We start by rewriting the diffusion equation as

$$\begin{aligned} \partial_t u &= \operatorname{div}\left(g(|\nabla u_\sigma|^2)\cdot\nabla u\right) && (4.46)\\ &= \operatorname{div}\begin{pmatrix} g(|\nabla u_\sigma|^2)\cdot\partial_x u \\ g(|\nabla u_\sigma|^2)\cdot\partial_y u \end{pmatrix} && (4.47)\\ &= \partial_x\Big(g(|\nabla u_\sigma|^2)\cdot\partial_x u\Big) + \partial_y\Big(g(|\nabla u_\sigma|^2)\cdot\partial_y u\Big). && (4.48) \end{aligned}$$

The problem is how to discretize this term, as it contains nested derivatives.
For this discretization, we only focus on the first term of the sum in equation 4.48, i.e., $\partial_x\Big(g(|\nabla u_\sigma|^2)\cdot\partial_x u\Big)$, as the second term follows in the same fashion.

For convenience, we define

$$\partial_x\Big(g(|\nabla u_\sigma|^2)\cdot\partial_x u\Big) =: \partial_x\Big[\phi(x,u,\nabla u_\sigma)\Big]. \tag{4.49}$$

At first, we will now discretize $\partial_x\Big[\phi(x,u,\nabla u_\sigma\Big]$ using forward differences (4.22), which yields

$$\partial_x\Big[\phi(x,u,\nabla u_\sigma\Big] \approx \frac{\Big[\phi(x,u,\nabla u_\sigma)\Big]\Big|_{i+1,j} - \Big[\phi(x,u,\nabla u_\sigma)\Big]\Big|_{i,j}}{h_1}. \tag{4.50}$$

Now, we can discretize the term $\phi(x,u,\nabla u_\sigma)$ using backward-differences (4.23), yielding

$$\begin{aligned} \phi(x,u,\nabla u_\sigma) &= g(|\nabla u_\sigma|^2)\cdot\partial_x u && (4.51)\\ &\approx \frac{g_{i,j}+g_{i-1,j}}{2}\cdot\frac{u_{i,j}-u_{i-1,j}}{h_1}. && (4.52) \end{aligned}$$

Plugging (4.50) and (4.52) together, we obtain

$$\begin{aligned} \partial_x\Big(g(|\nabla u_\sigma|^2)\cdot\partial_x u\Big) &\approx \left[\frac{g_{i,j}+g_{i-1,j}}{2}\cdot\frac{u_{i,j}-u_{i-1,j}}{h_1}\right]\Bigg|_{i+1,j} \\ &\quad -\left[\frac{g_{i,j}+g_{i-1,j}}{2}\cdot\frac{u_{i,j}-u_{i-1,j}}{h_1}\right]\Bigg|_{i,j} \end{aligned} \tag{4.53}$$

$$\begin{aligned} &= \frac{1}{h_1} \\ &\quad \cdot\Big(\frac{g_{i+1,j}+g_{i,j}}{2}\cdot\frac{u_{i+1,j}-u_{i,j}}{h_1} - \frac{g_{i,j}+g_{i-1,j}}{2}\cdot\frac{u_{i,j}-u_{i-1,j}}{h_1}\Big). \end{aligned} \tag{4.54}$$

If we have this, the second term of the sum in (4.48) follows in the same manner and so we can finally discretize the right-hand side of the diffusion equation, which we will call *spatial discretization*, or *semidiscretization*, and which looks like

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}u_{i,j} &= \frac{1}{h_1}\Big(\frac{g_{i+1,j}+g_{i,j}}{2}\cdot\frac{u_{i+1,j}-u_{i,j}}{h_1} - \frac{g_{i,j}+g_{i-1,j}}{2}\cdot\frac{u_{i,j}-u_{i-1,j}}{h_1}\Big) \\ &+ \frac{1}{h_2}\Big(\frac{g_{i,j+1}+g_{i,j}}{2}\cdot\frac{u_{i,j+1}-u_{i,j}}{h_2} - \frac{g_{i,j}+g_{i,j-1}}{2}\cdot\frac{u_{i,j}-u_{i,j-1}}{h_2}\Big), \end{aligned} \tag{4.55}$$

where we can discretize $g_{i,j}$ using central differences as

$$g_{i,j} = g_{i,j}(|\nabla u_\sigma|^2) \tag{4.56}$$

$$= g\Big(\sqrt{(\partial_x u_\sigma)^2+(\partial_y u_\sigma)^2}\Big)\Bigg|_{i,j} \tag{4.57}$$

$$\approx g\left(\sqrt{\Big(\frac{u_{i+1,j}-u_{i-1,j}}{2h_1}\Big)^2+\Big(\frac{u_{i,j+1}-u_{i,j-1}}{2h_2}\Big)^2}\right). \tag{4.58}$$

This scheme can be written in a quite elegant way, using a *matrix-vector form*, if we allow to represent a pixel (i,j) by a single index $k(i,j)$.
For an image of width N_1 and height N_2, we obtain $k(i,j) = N_1\cdot i + h$, leading to a traversal of the image pixels from top left to bottom right. If we further denote with $\mathcal{N}_n(k)$ the neighborhood of pixel $k(i,j)$ in direction $n\in\{1,2\}$, where $n=1$ represents

neighbors in x-direction and $n = 2$ neighbors in y-direction, we end up with

$$\frac{\mathrm{d}}{\mathrm{d}t} u_k = \sum_{n=1}^{2} \sum_{l \in \mathcal{N}_n(k)} \frac{g_l + g_k}{2h_n^2} \cdot (u_l - u_k) \tag{4.59}$$

$$=: A(u) \cdot u, \tag{4.60}$$

where $u = (u_1, \ldots, u_N)^\top, A \in \mathbb{R}^{N \times N}$ with $N = N_1 \cdot N_2$ and $A(u) = (a_{kl}(u))$ satisfying

$$a_{kl} := \begin{cases} \frac{g_k + g_l}{2h_n^2}, & \text{if } l \in \mathcal{N}_n(k) \\ -\sum_{n=1}^{2} \sum_{l \in \mathcal{N}_n(k)} \frac{g_k + g_l}{2h_n^2}, & \text{if } l = k \\ 0 & \text{else.} \end{cases} \tag{4.61}$$

Now, it is a logical consequence to strive for a complete discretization of equation 4.60, which requires to discretize $\frac{\mathrm{d}}{\mathrm{d}t} u$, which we will do by using forward differences

$$\frac{\mathrm{d}}{\mathrm{d}t} u \approx \frac{u^{k+1} - u^k}{\tau}, \tag{4.62}$$

with time step size τ. Interstingly, the resulting equation

$$\frac{u^{k+1} - u^k}{\tau} = A(u^k) \cdot u^k \tag{4.63}$$

can be solved in various ways.

The **first** one we will discuss is again an explicit scheme

$$u^{k+1} = \big(I + \tau \cdot A(u^k)\big) \cdot u^k, \tag{4.64}$$

directly solvable for u^{k+1}.

The **second** possibility arises, if we replace u^k on the right-hand side by the image at the new time step u^{k+1}:

$$\frac{u^{k+1} - u^k}{\tau} = A(u^k) \cdot u^{\mathbf{k+1}}, \tag{4.65}$$

which requires to solve the linear system of equations

$$u^k = \Big(I - \tau \cdot A(u^k)\Big) \cdot u^{k+1}, \tag{4.66}$$

which we can again rewrite in a matrix-vector form

$$u^{k+1} = \Big(I - \tau \cdot A(u^k)\Big)^{-1} \cdot u^k \tag{4.67}$$

$$=: Q(u^k) \cdot u^k, \tag{4.68}$$

resulting in a so called *semi-implicit scheme*.

Well-posedness and scale-space properties: For the semi-implicit scheme using the matrix $Q(u) = \big(q_{i,j}(u)\big)$, we have to pose the following restrictions:

- **Smoothness** : $q_{i,j}(u^k)$ is continuous in u^k
- **Symmetry** : $q_{i,j}(u^k) = q_{j,i}(u^k)$
- **Unit row sums** : $\sum_j q_{i,j}(u^k) = 1, \ \ \forall i$
- **Nonnegativity** : $q_{i,j}(u^k) \geq 0, \ \ \forall i, j$
- **Positive Diagonal** : $q_{i,i}(u^k) > 0, \ \ \forall i$
- **Irreducibility** : $\forall i, j \ \ \exists k_0, \ldots, k_r$ with $k_r = j$ s.t. $q_{k_p k_{p+1}} \neq 0$ for $p = 0. \ldots, r-1$.

Interestingly, we can prove that the semi-implicit scheme is stable for *arbitrary* large time steps τ, whereas for the explicit scheme using the matrix $A(u^k)$, we only get stability for $\tau < \frac{1}{4}$ (see [61] for details).

Conclusions: If we can solve the linear system of equations which arises from using the semi-implicit scheme with matrix $Q(u^k)$ in an efficient way, this scheme will outperform the explicit one. The reason is that because of the arbitrary large time step size, we do not have to do so many iterations for the same diffusion time, compared to the explicit scheme.
The basic idea for such an efficient algorithm is based on the sparsity of the used system matrix, which is for the linear system $\big(I - \tau \cdot A(u^k)\big) \cdot u^{k+1}$, the matrix $B := I - \tau \cdot A(u^k)$. If we recall the definition of $A(u^k)$ (equation 4.61), we see that this is a *pentadiagonal matrix* (a matrix with non-zero entries only on 5 diagonals), where the main diagonal holds the weights for all the N pixels of the image $u = (u_1, \ldots, u_N)^\top$, the first two off-diagonals hold the weights for the two neighbors of each pixel in x-direction and the last two off-diagonals hold the weights for the two neighbors in y-direction. For such sparse matrices, direct methods fail, as they fill in the zero-entries, s.t. one should use classical iterative algorithms [74, 48] like *Jacobi*, *Gauss-Seidel* or *SOR*.
To increase the performance even more, one could head for *multigrid methods* [9], which are very efficient, but also very hard to implement. So, a good compromise can be a splitting-based technique, like *Additive Operator Splitting (AOS)*, as described in [66]. The idea here is to decompose an m-D problem into simpler 1D ones. If we define $A(u^k) \cdot u^k = \sum_{l=1}^m A_l(u^k) \cdot u^k$, we can rewrite our semi-implicit scheme as the follwing sum

$$u^{k+1} = \Big(I - \tau \cdot \sum_{l=1}^{m} A_l(u^k)\Big)^{-1} \cdot u^k, \tag{4.69}$$

which we can split into

$$u^{k+1} = \frac{1}{m} \cdot \sum_{l=1}^{m} \Big(I - m \cdot \tau \cdot A_l(u^k)\Big)^{-1} \cdot u^k, \tag{4.70}$$

which is the corresponding AOS-scheme for our semi-implicit scheme.

Extension to Vector-Valued Images: Unfortunately, all upcoming filters will be space-variant, and so we have to use coupled approaches as described in Section 3.3.3. For nonlinear isotropic diffusion, this boils down to coupling the three channels via a joint diffusivity $g(s_i^2)$, which depends on the sum of the gradient magnitudes in all three channels, so

$$g(s_i^2) := g\Big(\sum_{j=1}^{3} |\nabla u_{j,\sigma}|^2\Big), \quad \text{for } i = 1, \dots, 3. \tag{4.71}$$

From an implementation point of view this just means that we *first* have to compute the coupled diffusivity, using information from all three channels and then assemble the matrix A using $g(s_i^2)$, i.e., we may also speak of a *coupled system matrix* here. Then we can use this very diffusivity to evolve all three channels separately.

Nonlinear Anisotropic Diffusion

Recall that we were dealing with two kinds of nonlinear anisotropic diffusion filters, namely EED and CED (cf. Section 3.3.1). Fortunately, we can cast their governing PDEs in a general form

$$\partial_t u = \operatorname{div}\Big(D\big(J_\rho(\nabla u_\sigma)\big) \cdot \nabla u\Big), \tag{4.72}$$

where we have $D = D\big(J_\rho(\nabla u_\sigma)\big)$ for CED. For the easier case of EED, we choose $\rho = 0$ and set $D = g\big(J_0(\nabla u_\sigma)\big) = g(\nabla u_\sigma \cdot \nabla u_\sigma^\top)$.
By the way, we can also use this general form for the other classes of diffusion filters. For nonlinear isotropic diffusion we set $D = g(|\nabla u_\sigma|^2) \cdot I$ and for linear isotropic diffusion we just have $D = I$.

When it comes to stability and scale-space conditions, we can use the results for nonlinear isotropic diffusion, which we will summarize in Figure 4.4.

Framework **Requirement**	*Semidiscrete* $\frac{du}{dt} = A(u)u$	*Discrete* $u^{k+1} = Q(u^k)u^k$
Smoothness	A Lipschitz continuous	Q continuous
Symmetry	A symmetric	Q symmetric
Conservation	$\sum_i a_{i,j} = 0$	$\sum_i q_{i,j} = 1$
Nonnegativity	A nonnegative off-diagonals	Q nonnegative elements
Connectivity	A irreducible	Q irreducible and positive diagonal

Figure 4.4: Five stability and scale-space requirements for the two frameworks (semidiscrete and discrete). **Adapted from:** Weickert [60].

Now we should proceed with the discretization of the general diffusion equation 4.72. Therefore, let the symmetric diffusion tensor D be given as

$$D := \begin{pmatrix} a & b \\ b & c \end{pmatrix}. \tag{4.73}$$

Then we have

$$\begin{aligned} \partial_t u &= \operatorname{div}(D \cdot \nabla u) = \operatorname{div} \begin{pmatrix} a \cdot \partial_x u + b \cdot \partial_y u \\ b \cdot \partial_x u + c \cdot \partial_y u \end{pmatrix} && (4.74) \\ &= \partial_x(a \cdot \partial_x u) + \partial_x(b \cdot \partial_y u) + \partial_y(b \cdot \partial_x u) + \partial_y(c \cdot \partial_y u). && (4.75) \end{aligned}$$

As $\partial_x(a \cdot \partial_x u) + \partial_y(c \cdot \partial_y u) = a \cdot \partial_{xx} u + c \cdot \partial_{yy} u$, we realize that the only novelty compared to nonlinear isotropic diffusion (cf. Section 4.3.1) are the *mixed terms* of the form $\partial_x(b \cdot \partial_y u) + \partial_y(b \cdot \partial_x u)$, which do not occur for nonlinear isotropic diffusion. For them, we can do an approximation by central differences

$$\begin{aligned} \partial_x(b \cdot \partial_y u) &\overset{\partial_x}{\approx} \frac{1}{2h_1}\Big(b \cdot \partial_y u\big|_{i+1,j} - b \cdot \partial_y u\big|_{i-1,j}\Big) && (4.76) \\ &\overset{\partial_y}{\approx} \frac{1}{2h_1}\Big(b_{i+1,j}\frac{u_{i+1,j+1} - u_{i+1,j-1}}{2h_2} - b_{i-1,j}\frac{u_{i-1,j+1} - u_{i-1,j-1}}{2h_2}\Big). && (4.77) \end{aligned}$$

Accordingly, we get

$$\partial_y(b \cdot \partial_x u) \approx \frac{1}{2h_2}\Big(b_{i,j+1}\frac{u_{i+1,j+1} - u_{i-1,j+1}}{2h_1} - b_{i,j-1}\frac{u_{i+1,j-1} - u_{i-1,j-1}}{2h_1}\Big). \tag{4.78}$$

If we now deduce the stencil form of the discretized filter, we see that it has signs as depicted in Figure 4.5. The stencil itself is quite tedious to write down, and so we refer

?	+	?
+	-	+
?	+	?

Figure 4.5: Sign pattern of the stencil for semidiscrete nonlinear anisotropic diffusion. **Author:** Weickert [63].

to [63], if one is interested in the actual stencil weights.
The problem now comes from the ?-entries, which are the result of the mixed terms and were not present in the stencil for nonlinear isotropic diffusion. The actual problem is that for the ?-entries, we cannot determine a sign in general, as altough D is positive semidefinite, we can only assure that $a > 0$ and $c > 0$, but b may be positive or negative. So, if the ?-entries are negative, we will violate the nonnegativity requirement stated in

Figure 4.4 and will hence violate the maximum-minimum-principle.
However, if we require the *spectral condition number* κ of D to satisfy

$$\kappa = \operatorname{cond}(D) \leq 3 + 2\sqrt{2} \approx 5.8284, \tag{4.79}$$

we can guarantee nonnegativity for a very specific discretization. However, another problem is that κ corresponds to the ratio between the two eigenvalues of D, i.e., our upper bound for κ limits the anisotropy of our approach.
The resulting nonnegative stencil is now *really* tedious to write down and we refer the interested reader to [61], where one can find the stencil on page 82.
On the other hand, in practical situations, one can often get along without nonnegative semidiscretizations, if one is satisfied with a weaker stability, like stability in the *Euclidian norm*, formally

$$\|u^{k+1}\|_2 \leq \|u^k\|_2, \tag{4.80}$$

where

$$\|u\|_2 := \sqrt{\sum_{i=1}^{N} |u_i|^2}, \tag{4.81}$$

and $u = (u_1, \ldots, u_N)^\top$.

Extension to Vector-Valued Images: Also here we have to use a coupled approach, cf. Section 3.3.3, which means that we use a joint diffusion tensor D, taking into account all three color channels, which reads for EED as

$$D := g\left(\sum_{j=1}^{3}\left(\nabla u_{j,\sigma} \cdot \nabla u_{j,\sigma}^\top\right)\right), \quad \text{for } i = 1, \ldots, 3, \tag{4.82}$$

and for CED as

$$D := D\left(\sum_{j=1}^{3} J_\rho(\nabla u_{j,\sigma})\right), \quad \text{for } i = 1, \ldots, 3. \tag{4.83}$$

To implement this, we proceed as presented for nonlinear isotropic diffusion, i.e., we first compute the joint tensor, assemble the joint matrix Q and finally evolve the three channels separately using the stencil deduced from Q.

4.3.2 Discretization of Morphological Filters

Recall that we will also use a non-diffusion PDE-based image filter, namely mean curvature motion (MCM) (cf. Section 3.3.2), which we interleave with EED for better inpainting results. Therefore, we will now present the implementation of MCM.

Mean Curvature Motion

Remember that equation 3.74 in Section 3.3.2 already gives a promising approach for implementing MCM, namely

$$\partial_t u = \partial_{\xi\xi} u \tag{4.84}$$

$$= \frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2}, \tag{4.85}$$

which we can discretize using forward differences for the time derivatives $\partial_t u$ (cf. Section 4.3.1, above), central differences for the simple spatial derivatives $\partial_y u, \partial_x u, \partial_{xx} u, \partial_{yy} u$ (cf. Section 4.1.1, above) and for the mixed derivatives $\partial_{xy} u$ we use [64]

$$\partial_{xy} u \approx \operatorname{sgn}(u_x \cdot u_y) \cdot \frac{-u_{i+1,j+1} + u_{i,j+1} + u_{i+1,j} - u_{i-1,j-1} + u_{i,j-1} + u_{i-1,j} - 2u_{i,j}}{2h_1 h_2}, \tag{4.86}$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1, \text{ if } x \geq 0 \\ 0, \text{ else.} \end{cases} \tag{4.87}$$

Putting all this together we obtain

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\tau} = \frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2}, \tag{4.88}$$

which we can *explicitly* solve for $u_{i,j}^{k+1}$:

$$u_{i,j}^{k+1} = \tau \left(\frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2} \right) + u_{i,j}^k. \tag{4.89}$$

Implicit Scheme: For implementing our interleaved inpainting of EED with MCM, we used the explicit MCM discretization, as shown above. The reason is that MCM will only play a minor part for our interleaved inpainting. EED will play the major role, as proposed in [26]. However, there exists a implicit MCM discretization. It can be found in the article of Alvarez [3] in Section 4.3.

Extension to Vector-Valued Images: If we have a look at Section 3.3.3, we see that one should implement MCM for vector-valued images using a coupled approach, which uses a *common isophote direction* ξ, taking into account the direction of isophotes ξ_m of all the three color channels. To compute ξ, we use a tensor $C_{i,j}$ for every pixel (i, j), which is given by

$$C_{i,j} = \sum_{m=1}^{3} \left(\nabla u_{m;i,j} \cdot \nabla u_{m;i,j}^\top \right), \tag{4.90}$$

where $u_{m;i,j}$ denotes the pixel (i,j) of color channel m.
Now we have to compute the eigenvector v_2 to the *smallest* eigenvalue λ_2 of $C_{i,j}$, which gives the common isophote direction ξ via

$$\xi := \frac{v_2}{||v_2||}. \tag{4.91}$$

One can easily verify that if we define

$$C_{i,j} := \begin{pmatrix} a_{i,j} & b_{i,j} \\ b_{i,j} & c_{i,j} \end{pmatrix}, \tag{4.92}$$

we get

$$v_2 = \left(\frac{\sqrt{a_{i,j}^2 - 2a_{i,j}c_{i,j} + 4b_{i,j}^2 + c_{i,j}^2} - a_{i,j} + c_{i,j}}{2b_{i,j}}, -1 \right)^\top. \tag{4.93}$$

What remains is to show how to evolve u w.r.t. ξ.
If we recall the general MCM equation

$$\partial_t u = \partial_{\xi\xi} u, \tag{4.94}$$

we get by plugging in the definition of the directional derivative (cf. Section 1.4) and defining $\xi := (\xi_1, \xi_2)^\top$, for the three color channels $m = 1, \ldots, 3$:

$$\begin{aligned}
\partial_t u_m &= \partial_{\xi\xi} u_m && (4.95)\\
&= \partial_\xi \Big(\xi^\top \cdot \nabla u_m \Big) && (4.96)\\
&= \partial_\xi (\xi_1 u_{m;x} + \xi_2 u_{m;y}) && (4.97)\\
&= \xi^\top \cdot \nabla (\xi_1 u_{m;x} + \xi_2 u_{m;y}) && (4.98)\\
&= \xi^\top \cdot \begin{pmatrix} \xi_1 u_{m;xx} + \xi_2 u_{m;xy} \\ \xi_1 u_{m;xy} + \xi_2 u_{m;yy} \end{pmatrix} && (4.99)\\
&= \xi_1^2 u_{m;xx} + 2\xi_1\xi_2 u_{m;xy} + \xi_2^2 u_{m;yy}, && (4.100)
\end{aligned}$$

where $u_{m;x}$ denotes the x-derivative of the m-th color channel of u, for example. If we now discretize the time derivative again by forward differences, we end up with

$$\frac{u_{m;i,j}^{k+1} - u_{m;i,j}^k}{\tau} = \xi_1^2 u_{m;xx} + 2\xi_1\xi_2 u_{m;xy} + \xi_2^2 u_{m;yy}, \tag{4.101}$$

which we can explicitly solve for $u_{m;i,j}^{k+1}$ as

$$u_{m;i,j}^{k+1} = \tau(\xi_1^2 u_{m;xx} + 2\xi_1\xi_2 u_{m;xy} + \xi_2^2 u_{m;yy}) + u_{m;i,j}^k, \tag{4.102}$$

where we simply discretize the occurring spatial derivatives by central differences, as shown above for scalar-valued MCM, for example.

4.3.3 Inpainting using Diffusion Filters

After we have finished presenting the implementation of our diffusion filters and MCM, we can finally turn our attention to the implementation of the inpainting algorithm using these filters.
Let us first recall the general inpainting equation 3.85, which reads

$$\partial_t u = \big(1 - \mathcal{X}_{\Omega'}\big) \cdot \operatorname{div}\left(D \cdot \nabla u\right) - \mathcal{X}_{\Omega'} \cdot (u - f). \tag{4.103}$$

We realize that, basically, we just have to prevent evolution at points $(x, y) \in \Omega'$, i.e., in the corner regions, corresponding to the uncompressed image where $\mathcal{X}_{\Omega'}(x, y) = 1$. For all other points, we can do our "normal" evolution, using the deduced stencils from above.
To make concepts discrete, we should consider a characteristic function $\mathcal{X}_{\Omega'_{disc}}$ which satisfies $\mathcal{X}_{\Omega'_{disc}}(i, j) = 1$ if $(i, j) \in \Omega'_{disc}$ and 0 else, i.e., $\mathcal{X}_{\Omega'_{disc}}$ is the discrete counterpart of $\mathcal{X}_{\Omega'}$. With this, we can formally write the evolution of $u^k_{i,j}$ with initialization $u^0_{i,j} = f_{i,j}$ as

$$u^{k+1}_{i,j} = \begin{cases} s^\top \cdot \mathcal{N}(u^k_{i,j}), & \text{if } \mathcal{X}_{\Omega'_{disc}}(i, j) = 0 \\ u^k_{i,j}, & \text{else,} \end{cases} \tag{4.104}$$

where $s^\top \cdot \mathcal{N}(u^k_{i,j})$ represents the evolution in pixel (i, j) w.r.t. the stencil s and the neighborhood $\mathcal{N}$ of $u^k_{i,j}$. So, if the stencil weights are given according to Figure 4.6, we get

$$s := (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{i})^\top, \tag{4.105}$$

and

$$\begin{aligned} \mathcal{N}(u^k_{i,j}) \;:=\; & (u_{i-1,j-1}, u_{i-1,j}, u_{i-1,j+1}, \\ & u_{i,j-1}, u_{i,j}, u_{i,j+1}, \\ & u_{i+1,j-1}, u_{i+1,j}, u_{i+1,j+1})^\top. \end{aligned} \tag{4.106}$$

a	**b**	**c**
d	**e**	**f**
g	**h**	**i**

Figure 4.6: Stencil weights.

Interleaved Inpainting

Recall that our sparsification approach using corner regions poses quite some challenges for the inpainting in the decompression step. To overcome this, we invented interleaved

inpainting of EED with MCM, cf. Section 3.3.4. This gave better results than the usual (pure) EED inpainting. The question is now, how to implement this.
Let us first recall the general framework for interleaved inpainting with two PDEs, given by two elliptic differential operators L_1 and L_2, respectively:

$$\partial_t u = \left(1 - \mathcal{X}_{\Omega'}\right) \cdot \Big(\varepsilon \cdot L_1(u) + (1-\varepsilon) \cdot L_2(u)\Big) - \mathcal{X}_{\Omega'} \cdot (u - f), \tag{4.107}$$

where $0 < \varepsilon < 1$ defines the relation between L_1 and L_2. One can easily see that the only difference to the usual inpainting framework, given by equation 4.103, lies in the term $\varepsilon \cdot L_1(u) + (1-\varepsilon) \cdot L_2(u)$. This term describes the interleaving of L_1 and L_2 with relation ε. It replaces the lonely diffusion PDE given by the term $\operatorname{div}(D \cdot \nabla u)$ in equation 4.103. For the sake of easier understanding, we define

$$\varepsilon =: \frac{1}{p}, \quad p \in \mathbb{N} \setminus \{0\}. \tag{4.108}$$

We can do this, as we imposed $0 < \varepsilon < 1$. This yields

$$1 - \varepsilon = 1 - \frac{1}{p} = \frac{p-1}{p}. \tag{4.109}$$

From this, it becomes clear that we can –on implementation level– achieve interleaving of L_1 and L_2 with relation ε by the following scheme:

1. Evolve the image u for a time $t_1 = 1$ w.r.t. the evolution equation 4.104 corresponding to L_1.
   This means that we use the stencil $s^\top$ corresponding to the PDE given by L_1.
2. Evolve u for a time of $t_2 = p - 1$ in accordance to equation 4.104 corresponding to L_2.
3. Go to step 1. until a stopping criterion is reached.

Our Approach: Interleaving EED with MCM: We actually used interleaving of EED with MCM (cf. Section 3.3.4). This means for L_1, we use the right-hand side of the PDE for MCM and for L_2 the same for EED. In Section 4.3.2, we have seen that we have to use a time step size of $\tau_1 = 0.2$ for MCM, because of the explicit scheme. In contrast, for EED, we have an implicit scheme (cf. Section 4.3.1 and 4.3.1), for which we can use an arbitrary time step size τ_2. Because of this, we have to do for step 1. from above $5 \cdot t_1$ iterations of MCM with $\tau_1 = 0.2$. For step 2., we do one iteration of EED with $\tau_2 = t_2$.

4.4 Actual Implementation

The actual implementation of the ideas stated above can be done in a straight-forward manner in any programming language. We used `C`, mainly because of performance

reasons.
The reader should note that code for some parts was kindly provided by Prof. Joachim Weickert and M.Sc. Irena Galic. In detail:

- Prof. Joachim Weickert kindly provided the code for
  - Corner detection using the structure tensor J_ρ
  - PDE-based inpainting using diffusion filters
  - Different diffusion filters, e.g., EED
  - Mean curvature motion (MCM)
- M.Sc. Irena Galic kindly provided the code for
  - Computing the AAD error measure
- For this thesis, we newly implemented the
  - Extraction and efficient storing of corner regions in a CRF file
  - Restoration of a sparse image and a mask from a CRF file
  - Interleaved inpainting, in our case of EED with MCM
  - Coupled MCM approach for vector-valued images

Chapter 5

Experiments

In this chapter we want to present results of experiments with our codec. This means we will start with shortly discussing parameters for corner detectors used in the compression step and how to find parameters corresponding to a desired compression rate. After that we will move on to the more interesting experiments concerning our decompression step. This step uses PDE-based inpainting and hence we will shortly discuss parameter choices, also considering the choice of an optimal stopping time T. The main part of this chapter then presents and compares different decompression results, e.g., for different stopping times, for different compression rates or for artificial images, just to state some. We will also do a short comparison to JPEG, the de-facto standard codec. To evaluate results, we will use the AAD (Average Absolute Difference) error measure, as presented in Section 3.3.5. With this, we can measure how large the difference between our decompression results and the initial, uncompressed image is.

5.1 Compression

In the compression step of our codec, we extracted corner regions to create a sparse image. Hence, we will describe some experiments concerning corner detection in this section.

5.1.1 Tradeoff $\mathcal{N} \leftrightarrow n$

Recall the computation of our achieved compression rate

$$F = \frac{n \cdot (\frac{3}{4} \cdot \mathcal{N} \cdot c + 2)}{c \cdot N_1 \cdot N_2}. \tag{5.1}$$

We can rewrite this as

$$n = \frac{F \cdot c \cdot N_1 \cdot N_2}{\frac{3}{4} \cdot \mathcal{N} \cdot c + 2}. \tag{5.2}$$

Let us now consider a given image with dimensions N_1 and N_2 and c color channels. Further, we fix the compression rate to F, so F, c, N_1, N_2 can be considered as constant

factors. From equation 5.2, we see that the neighborhood size $\mathcal{N}$ and n, the number of corner regions we can store, are inversely proportional, i.e., if we use a larger $\mathcal{N}$, we have to get along with less corner regions for achieving the same compression rate. For our decompression step using PDE-based inpainting, we should strive for a sparse image which is not *too* sparse, i.e., we should keep n as large as possible. So why shouldn't we then just take the smallest $\mathcal{N} = 1$, i.e., we only store the corner pixel itself? There are actually two reasons for that.
First, we have to store at least some neighborhood of a corner to code vital gradient information. Secondly, larger $\mathcal{N}$ have the advantage that we can store them more efficiently. Here, we just have to store the coordinates of the corner pixel and then can store all the neighboring pixels in a fixed order without a need to store their coordinates. So, the relation between $\mathcal{N}$ and n is crucial, where usually $\mathcal{N} \in \{1, 4, 8, 24\}$.
As an example, one may have a look at Figure 5.1, where the sparse images and the corresponding decompression results of *lena* are depicted for different choices of $\mathcal{N}$. The corresponding error measures in AAD are given in Figure 5.2.

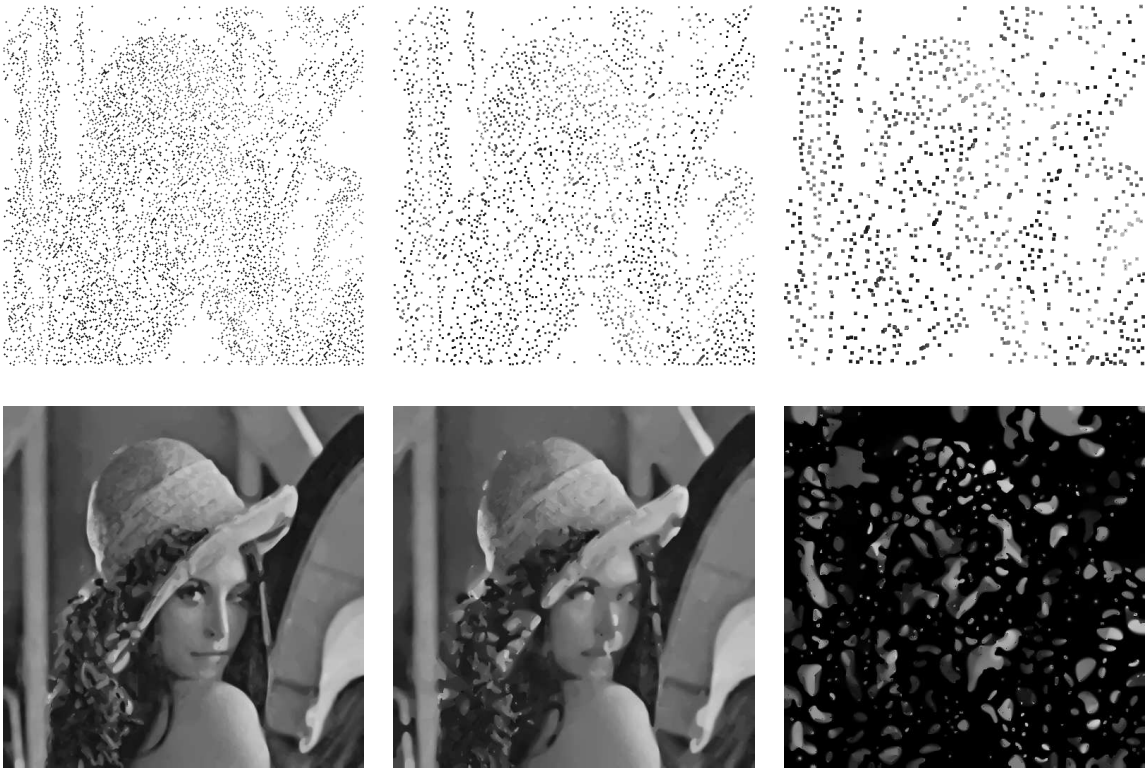

Figure 5.1: Decompression of *lena* (512×512 pixels) for different neighborhood sizes $\mathcal{N}$ and a fixed compression rate of 1 : 9.6 (0.83 bpp). **Top row:** Sparse images for $\mathcal{N} = 4$, $\mathcal{N} = 8$ and $\mathcal{N} = 24$ (with white background for printing reasons). **Bottom row:** Corresponding decompression results using interleaved inpainting of EED/MCM with relation 100 : 5, noise scale $\sigma = 1.0$, contrast parameter $\lambda = 0.1$ (stopped after 7000 iterations).

Neighborhood size $\mathcal{N}$	AAD error in decompression result
4	6.37
8	7.75
24	74.13

Figure 5.2: Error measures in AAD of the decompression results from Figure 5.1.

Considering the above, we decided to use a neighborhood size of $\mathcal{N} = 4$, as this is the smallest one which allows to store gradient information in x- and y-direction. The benefit is that with this small $\mathcal{N}$, we can store a maximal amount of corners, helping to improve decompression results.

5.1.2 Used Corner Detector

The choice of the corner detector for the compression step is rather unproblematic. All tested corner detectors worked fairly well, and so we decided to use the one of Förstner/Harris (cf. Section 3.2.1), which computes the "cornerness" w.r.t. the term '$\det J_\rho / \mathrm{tr} J_\rho$'. This was a good compromise between quality (compared to the Rohr corner detector) and computational effort (compared to the Tomasi/Kanade corner detector).

Parameters for Corner Detector

The following "standard" parameter settings worked well for our used Förstner/Harris corner detector:

- Integration scale ρ for structure tensor J_ρ : $\rho \in [1, 3]$
- Noise scale σ for robustness increasing pre-smoothing : $\sigma = 1$
- Threshold Θ for thresholding corner points : $\Theta = 1$

Here, it is interesting to note that the actual threshold parameter Θ is not so crucial for the reached compression rate. More important is the integration scale ρ, where higher ρ correspond to higher compression rates.

Finding ρ

As a consequence from above considerations, our compression algorithm will only need one variable parameter, namely ρ. The problem is that specifying ρ is not very intuitive. One may rather want to directly specify the desired compression rate F^*_{bpp} or F^*, depending on wether the user specified the desired compression rate in *bpp* or as an absolute value.
The problem that arises now is how to compute an appropriate ρ for a given F^*_{bpp}. (Note

that we only have to deal with this case, as if we are given F^*, we can easily compute $F^*_{bpp} = 8 \cdot \ c \cdot F^*$, c.f. Section 4.2.2).
Empirically, we found that

$$\rho = \frac{2}{3 \cdot F^*_{bpp}} \tag{5.3}$$

gives a good initial estimation for ρ. From this, we can continuously adapt ρ and compute the achieved compression rate F_{bpp} from the actual ρ until we have that

$$|F^*_{bpp} - F_{bpp}| < \varepsilon, \tag{5.4}$$

where we used $\varepsilon = 0.04$. The actual adaption is quite straight-forward, as we just have to increase or decrease ρ by a certain factor γ. To be more exact, we use the fact that the number of used corners n is inversely proportional to ρ and from the fact that n is proportional to F_{bpp}, we obtain the following rule for computing the new ρ':

$$\rho' = \begin{cases} \rho + \gamma, & \text{if } F_{bpp} > F^*_{bpp} \\ \rho - \gamma, & \text{else.} \end{cases} \tag{5.5}$$

Actually, we initially used $\gamma = 0.1$ and then decreased γ by a factor of 0.5 each time the error $|F^*_{bpp} - F_{bpp}|$ started to increase again.

In this context we should note that this algorithm will not work all the time. For example if we face an artificial image, like depicted in Figure 5.26, and want to have a high bit rate, e.g., one larger than 1 bpp. The reason is that for such images, there are simply not enough pixels which may be classified as corners, no matter how we choose the parameters of our corner detector. So, there actually does not exist a ρ, matching the desired bit rate, and logically, our algorithm will fail to find one.

5.1.3 Other Parameter Choices

As mentioned above in Section 5.1.2, we used a fixed threshold $\Theta = 1$ and adapted the integration scale ρ to achieve different compression rates. This worked out well, given that one only wants to have a single parameter determining the compression rate. Of course, varying all three parameters Θ, ρ and σ might be better but is quite hard, so one idea may be to invest in just fixing $\sigma = 1$ and varying Θ and ρ, as the noise scale σ is not so crucial for the compression result. A detailed discussion on this can be found in Section 6.2.1.
Another experiment we did was to fix $\sigma = 1$ and $\rho = 1$ and adapt Θ in accordance to the desired compression rate. This also sounds more natural, as a threshold parameter is more likely to be variable than an integration scale. The problem is that it is actually harder to find a matching Θ, especially if one aims at high compression rates.
As an example we again compressed the lena image at a compression rate of 1 : 9.6 (0.83 bpp), which we achieved with fixed $\rho = 1$ and variable $\Theta = 3$. With our common

approach we had to use a fixed $\Theta = 1$ and a variable $\rho = 1.25$. The corresponding sparse images and the decompression result using interleaved EED/MCM inpainting are depicted in Figure 5.3.

Here, we should note that the fixing of the noise scale $\sigma = 1$ is maybe not optimal. If one is given a noisy initial image, one should use a higher σ to avoid the corner detector to misinterpret noise as corners. A more detailed discussion on the robustness of our approach w.r.t. noise will be given in Section 5.2.5. Furthermore, one *usually* uses an integration scale ρ which is larger than σ. But as one can see, in our approach with fixing Θ and varying ρ this will be the case. Here, we will choose $\rho \in [1, 3]$ corresponding to the desired compression rate. However, let us come back to Figure 5.3. If one first compares the sparse images, one sees that fixing $\rho = 1$ gives a more natural

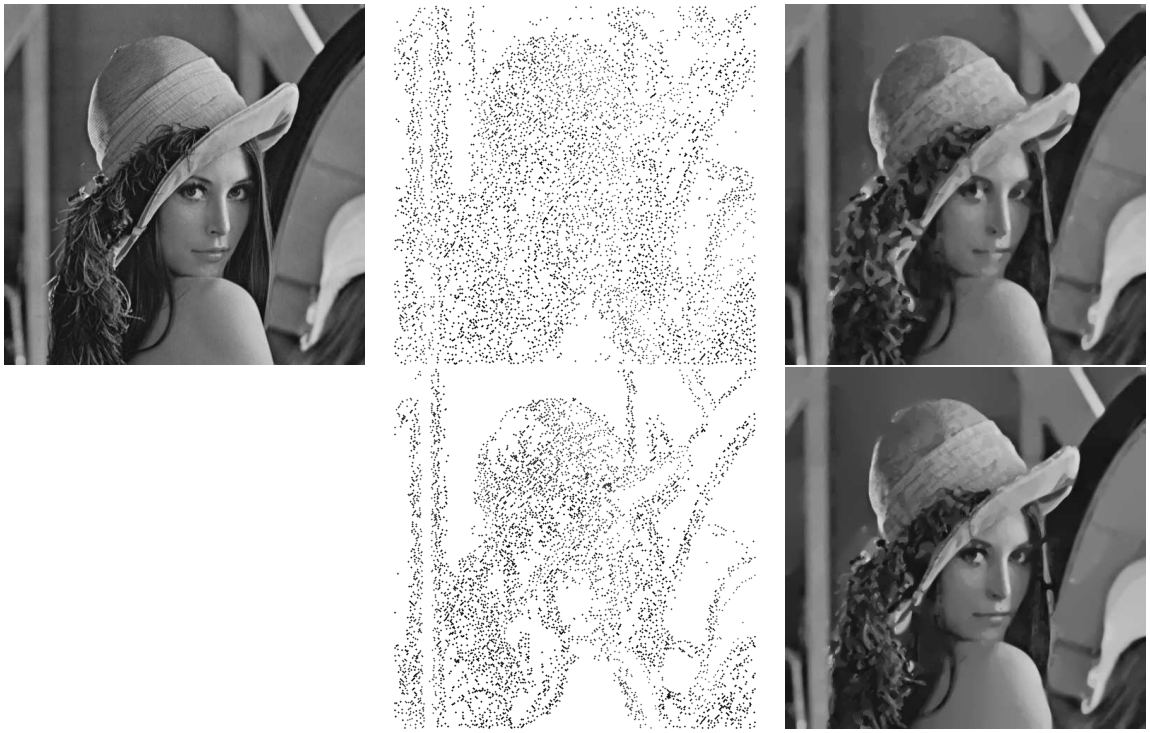

Figure 5.3: Comparison of fixing Θ (top row) with fixing ρ (bottom row). **Top left:** Initial image, *lena* (512×512 pixels). **Top middle:** Sparse image at a compression rate of $1 : 9.6$ (0.83 bpp) using fixed $\sigma = 1, \Theta = 1$ and variable $\rho = 1.25$ (with white background for printing reasons). **Top right:** Corresponding decompression result using interleaved inpainting of EED/MCM with relation $100 : 5$, noise scale $\sigma = 1.0$, contrast parameter $\lambda = 0.1$ and stopping time $T = 82750$. **Bottom middle:** Sparse image at a compression rate of $1 : 9.6$ (0.83 bpp) using fixed $\sigma = 1, \rho = 1$ and variable $\Theta = 3$ (with white background for printing reasons). **Bottom right:** Corresponding decompression result using interleaved inpainting of EED/MCM with same parameters.

looking sparse image, almost like the result of an interrupted edge detector. When fixing

$\Theta = 1$, one gets more corner regions in more or less homogeneous areas, which looks a little bit more unnatural. This impression is further supported by a first glance on the corresponding decompression results. Details like the eye are better restored when fixing $\rho = 1$. Interstingly, an objective comparison using the AAD error measure reveals that our common approach with fixing $\Theta = 1$ yields noticeably better results. Actually we have an AAD of 6.49 here and 6.95 for the new approach fixing $\rho = 1$. This may be explained by the fact that large, almost homogeneous areas are not covered enough with corner regions for the new approach. However, a deeper investigation on the influence of the parameter choice may be fruitful, cf. Section 6.2.1.

5.2 Decompression

Our decompression step relies on PDE-based inpainting of a sparse image, obtained from the stored, compressed data. So, we will present experiments concerning PDE-based inpainting in this section. Most of the time, we will use our interleaved inpainting of EED with MCM in relation 100 : 5 and with the "standard" parameters $\sigma = 1$ and $\lambda = 0.1$.

5.2.1 Choice of Diffusion Filter for Inpainting

One essential decision is which kind of PDE-based filter –in our case mostly a diffusion filter– to use in the inpainting step. From [26], we see that EED seems a good choice, so we started with plain EED inpainting, with the following standard parameter choices:

- Contrast parameter: $\lambda = 0.1$
- Noise scale for pre-smoothing : $\sigma = 1$
- Time step size: $\tau = 1000$

The remaining question is when to stop diffusion, i.e, the selection of the *stopping time* T.

Choice of Stopping Time T

Theoretically, we get our desired decompressed image as the steady state for $t \to \infty$ of our inpainting evolution $u(x, y, t)$ of the sparse image, cf. Section 3.3.4.
This is not implementable, of course, and so we have to find a *finite* stopping time T. The idea we had here is based on the fact that in the steady state we will have $\partial_t u = 0$, i.e., nothing will change anymore if we do another evolution step. So, we decided to use a threshold on the percentage of "stable" pixels in the evolving image u. This means that if the number of pixels in u^k and u^{k+1} with (almost) the same gray value is above

a certain threshold, we can stop diffusion. Formally, we can define stability as

$$stable(u^k, u^{k+1}) = \frac{1}{|\Omega_{disc}|} \sum_{(i,j)\in\Omega_{disc}} \phi(u_{i,j}^{k+1} - u_{i,j}^k), \tag{5.6}$$

with an indicator function

$$\phi(s) := \begin{cases} 1, & \text{if } |s| \geq \varepsilon \\ 0, & \text{else,} \end{cases} \tag{5.7}$$

allowing to count the number of stable pixels with the help of an arbitrary small $\varepsilon > 0$. Note that in the remainder we will for notational convenience just write *stable* instead of $stable(u^k, u^{k+1})$, when the actual time k is not essential.
The next problem is now to find a good threshold for the stability criterion. To reach this, we did some experiments, where we decompressed the same image with different stopping times and at different stability levels.

We see from Figure 5.4, that it (visually) does not make a big difference if we stop at time of approximately 6000, or any larger time, which becomes even clearer if we compare the AAD error measures, as done in Figure 5.5 and visualized in Figure 5.6. Note that for the visualization, we used even more examples than depicted in Figure 5.4 and given in Figure 5.5.
Nevertheless, as a stability criterion we mainly used a larger value of about $stable \geq 0.7$. This decision was justified by the fact we want to solve the steady state of the inpainting equation 3.85, i.e., we actually want to reach the steady state, where we have $\partial_t u = 0$.

Relation between Stopping Time T and Number of Iterations k: The astute reader may have noticed that in Figure 5.5 the stopping time T and the number of iterations k are not proportional. At a first glance, this may be not intuitive, but can be explained as follows:
Recall that we actually used interleaved inpainting of EED with MCM in a relation of $100 : 5$. Theoretically, this yields $\varepsilon = \frac{1}{20}$ and hence $1 - \varepsilon = \frac{20}{21}$ in the sense of Section 4.3.3. There, we have shown that our inpainting is performed by doing 5 iterations MCM with time step size $\tau = 0.2$, followed by 1 iteration EED with $\tau = 20$. This yields a total evolution time of $T = 5 \cdot 0.2 + 1 \cdot 20 = 21$, whereas we have an iteration count of $k = 5 + 1 = 6$.
Even more severe for the non-proportionality of T and k is the fact that we start our evolution with pure EED inpainting until a certain stability in the result is reached. Not till then, we start our interleaved inpainting of EED and MCM. The reasons for that, we will discuss in the following section.

5.2.2 Interleaved Inpainting

Using (pure) EED for inpainting the sparse images works already quite well, as mentioned in Section 3.3.4, but the results can still be improved. One idea was *interleaved*

Figure 5.4: Different decompression results for different stopping times T. Initial image was *lena* (512×512 pixels), compressed at a compression rate of $1 : 9.6$ (0.83 bpp) and decompressed with interleaved inpainting of EED/MCM with relation $100 : 5$, noise scale $\sigma = 1.0$, contrast parameter $\lambda = 0.1$. **Top left:** Stopped after 10 iterations, corresponding to stopping time $T = 2002$. **Top middle:** 100 iterations, corresponding to $T = 2319$. **Top right:** 500 iterations, corresponding time $T = 3996$. **Bottom left:** 1000 iterations, corresponding to $T = 5992$. **Bottom middle:** 5000 iterations, corresponding to $T = 22161$. **Bottom right:** 10000 iterations, corresponding to $T = 42319$.

Iterations k	Stopping time T	Stability *stable*	AAD
10	2002	0.09	28.86
100	2319	0.11	19.61
500	3996	0.11	7.51
1000	5992	0.15	8.74
5000	22161	0.20	6.39
10000	42319	0.22	6.42

Figure 5.5: Error measures in AAD of the decompression results from Figure 5.4.

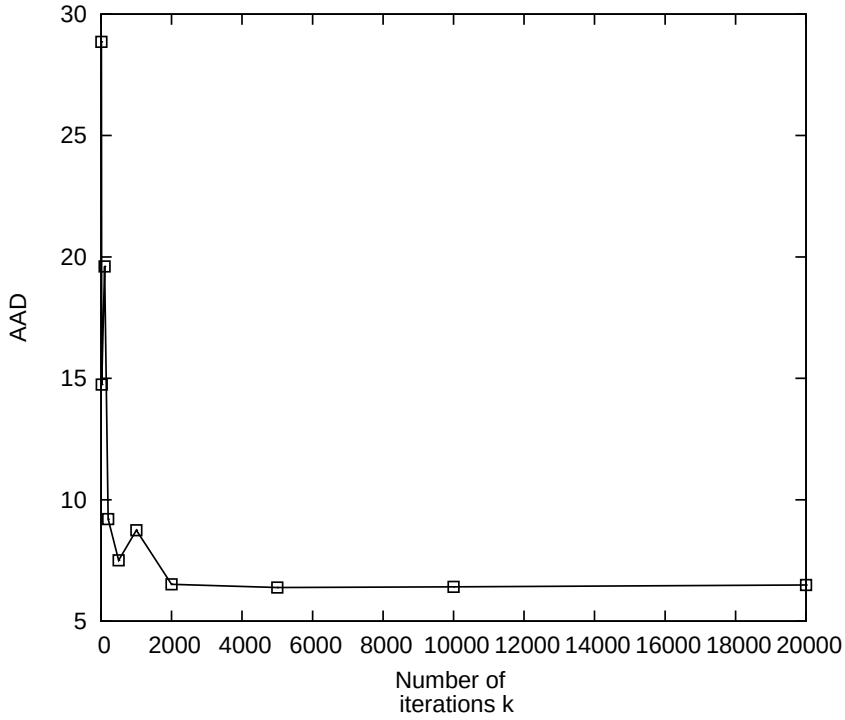

Figure 5.6: Visualization of the relation between stopping time (here given by the number of iterations) and error to the original image, given in AAD.

inpainting, inspired from [8], where the authors interleaved a third order inpainting PDE with a Perona–Malik-like dampened MCM filtering, cf. Section 2.3.
Hence, our approach was to *interleave EED with MCM*, where the choice of MCM as a second PDE-based filter was further supported by the work of Caselles [13], cf. Section 3.3.4. Unfortunately, MCM will only start to efficiently work if the image is not so sparse anymore, so we decided to first do some pure EED inpainting, which we then finish with an interleaving of MCM with EED. Actually, we did EED inpainting with $\tau = 1000$ until $stable \geq 0.05$. Then we interleaved EED inpainting with MCM inpainting in relation $100 : 5$ w.r.t. evolution time t (which means $\varepsilon = \frac{1}{21}$ in the sense of Section 3.3.4) with $\tau = 100$ for EED and $\tau = 0.2$ for MCM. This is iteratively continued until we stop at $stable >= 0.7$.
A comparison of interleaved inpainting (EED/MCM) with pure EED inpainting and the simple homogeneous inpainting can be found in Figure 5.7 and the corresponding error measures are given in Figure 5.8. To be able to actually evaluate the difference between EED and interleaved inpainting, we repeated above experiment with different stopping times T. The results are depicted in Figure 5.9 and visualized in Figure 5.10.

What we see from Figure 5.10 is that interleaved inpainting is everytime better than pure EED inpainting. The difference is more severe for (maybe not so important) small stopping times. Nevertheless, one sees that the steady state will also be better for interleaved inpainting.

Figure 5.7: Different decompression results with different inpainting types at a compression rate of 1 : 9.6 (0.83 bpp) with (approximately) same stopping time $T \approx 22000$. Initial image was *lena* (512×512 pixels). **Left:** Inpainting result with homogeneous diffusion. **Middle:** Inpainting result with EED ($\sigma = 1.0$, $\lambda = 0.1$). **Right:** Inpainting result with interleaving of EED/MCM (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$).

Inpainting Method	AAD
Homogeneous Diffusion	8.97
EED	6.69
Interleaving of EED/MCM	6.39

Figure 5.8: Error measures (AAD) of the decompression results from Figure 5.7.

Figure 5.9: Decompression results of EED- and interleaved inpainting (EED/MCM) for different stopping times T. Initial image was *lena* (512×512 pixels), compressed at a compression rate of 1 : 9.6 (0.83 bpp). **Top row:** Results with EED inpainting ($\sigma = 1.0$, $\lambda = 0.1$). **Bottom row:** Results with interleaved inpainting of EED/MCM (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$). **From left to right:** Stopping times $T \approx 2000, 6000, 22000, 42000$ and $T \approx 83000$.

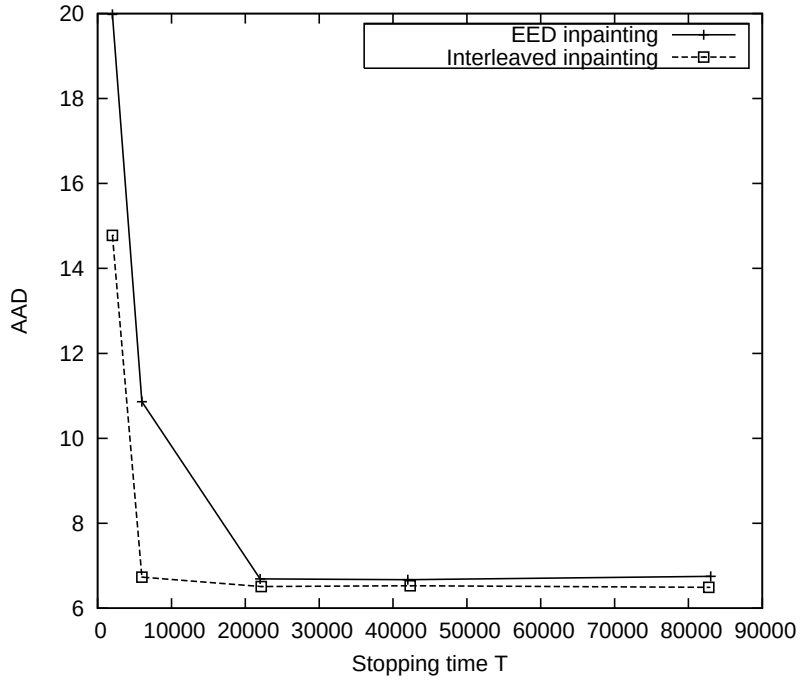

Figure 5.10: Visualization of the comparison between the use of EED and interleaved inpainting in the decompression step of our codec. Depicted is the relation between stopping time T and the error to the initial image, given in AAD.

Parameters for Inpainting

In the following we want to summarize our parameter settings which worked well for our purposes:

- Contrast parameter λ for EED : $\lambda = 0.1$
- Noise scale σ for pre-smoothing : $\sigma = 1$
- Time step size τ :
  - For EED : $\tau = 1000$
  - For MCM : $\tau = 0.2$
- Evolution time T (dependent on *stable*) :

$$T = \min \{t > 0 \mid stable(u^{t-1}, u^t) \geq 0.7\}. \tag{5.8}$$

Practically, this yields stopping times T between 30000 and 80000, dependent on the used compression rate.

Figure 5.11: Comparison of MCM inpainting to inpainting using homogeneous diffusion. **Left:** Inpainting of *lena* (512×512 pixels) at a compression rate of $1 : 9.6$ (0.83 bpp) using homogeneous diffusion and evolution time $T = 30000$. **Right:** Same with MCM inpainting.

5.2.3 Pure MCM Inpainting

When considering Section 3.3, we realize that we actually omitted the (pure) use of mean curvature motion (MCM) as driving PDE for our diffusion equation. The reason for that is quite simple. This method (at least applied alone) does not give reasonable results. It is actually worse than using simple homogeneous diffusion, but computationally much more expensive.
As an example consider Figure 5.11, which compares pure MCM inpainting with inpainting using homogeneous diffusion. The optical impression that MCM inpainting is even worse than homogeneous diffusion inpainting is further supported if we compare the AAD error measure, as done in Figure 5.12.

Method	AAD
MCM-inpainting	11.03
Homogeneous diffusion inpainting	8.97

Figure 5.12: Comparing AAD error measures of MCM and homogeneous diffusion.

5.2.4 How to Fill In the Inpainting Domain

As mentioned in Section 3.3.4, there are several possibilities to fill the inpainting domain $\Omega \setminus \Omega'$ in our sparse image. Hence, we now want to compare results with different fillings

at the same stopping time T. Our options are to fill the inpainting domain with black pixels, the average gray value μ, or with random gray values in $\{0, \ldots, 255\}$. The corresponding decompression results are given in Figure 5.13.

Considering Figure 5.13, we would probably prefer the result at the bottom right, i.e., the result of filling with random pixels. Especially at the eye of the lady, the results are more preferable. Interestingly, an objective comparison using the AAD measure yields that the filling with black pixels gives the best result, cf. the upper right image. The actual AAD measures are given in Figure 5.14. In Section 3.3.4, we stated that the steady state for $t \to \infty$ should be independent from the initial filling of the inpainting domain. This assumption could be approved by some empirical tests, which we visualized in Figure 5.15. So, we finally see that if one spends the time to iterate the evolution until the steady state is (at least approximately) reached, the initial filling is of no importance. However, if one only wants to do some iterations, it can be advisable to use a clever initial filling, i.e., a random guess in our case (cf. Figure 5.15). With this one can get better results with only a small number of iterations.

5.2.5 Robustness with Respect to Noise

Of course, an image codec which is used in practice will have to deal with noisy images. The problem is that noise creates fluctuations which the corner detector in our compression step will consider as corners. Hence, the sparse image will not contain important image points but more or less random ones, which will lead to a quite unsatisfactory compression result, as can be seen in the top row of Figure 5.16.

A remedy can come from preprocessing the noisy image using edge-enhancing diffusion (EED), which gives a lot more reasonable sparse image and a better decompression result, as can be seen in the bottom row of Figure 5.16.
Another, maybe simpler idea to improve robustness w.r.t. noise is to increase the noise scale σ of the used structure tensor $J_\rho(\nabla u_\sigma)$, yielding a larger pre-smoothing. Of course, one than also has to adapt the integration scale ρ and the threshold parameter Θ in order to obtain the same compression rate. Interstingly, this did not give noticeably better results. The reason may be that our test image is too noisy. For images with only a moderate amount of noise, the idea stated above may work well. A further discussion on this will be given in Section 6.2.1.

5.2.6 Computational Complexity

The maybe biggest disadvantage of our approach is the computational quite complex decompression step relying on PDE-based inpainting. To illustrate this, consider the sample images in Figure 5.7 and their (absolute) decompression times on a standard off-the-shelf PC, as shown in Figure 5.17.

Evaluating the results of Figure 5.17, we see that especially our (qualitatively) preferable interleaving approach takes quite long. The main reason is that we used an explicit

Figure 5.13: Decompression results with different initial fillings of the inpainting domain $\Omega \setminus \Omega'$, stopped at time $T = 30244$. **Top left:** Initial image $v(x)$, *lena* (512×512 pixels). **Top right:** Inpainting result using interleaving of EED/MCM (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$) with a compression rate of $1 : 9.6$ (0.83 bpp), sparse image $f(x)$ filled with black pixels. **Bottom left:** Inpainting result with same parameters, but f filled with average gray value μ of initial image $v(x)$. **Bottom right:** Inpainting result with same parameters, but f filled with random pixels in $\{0, \ldots, 255\}$.

Initial Filling	AAD
Black pixels	6.37
Average gray value	6.50
Random pixels	6.49

Figure 5.14: Comparing AAD error measures of different initial fillings of the inpainting domain.

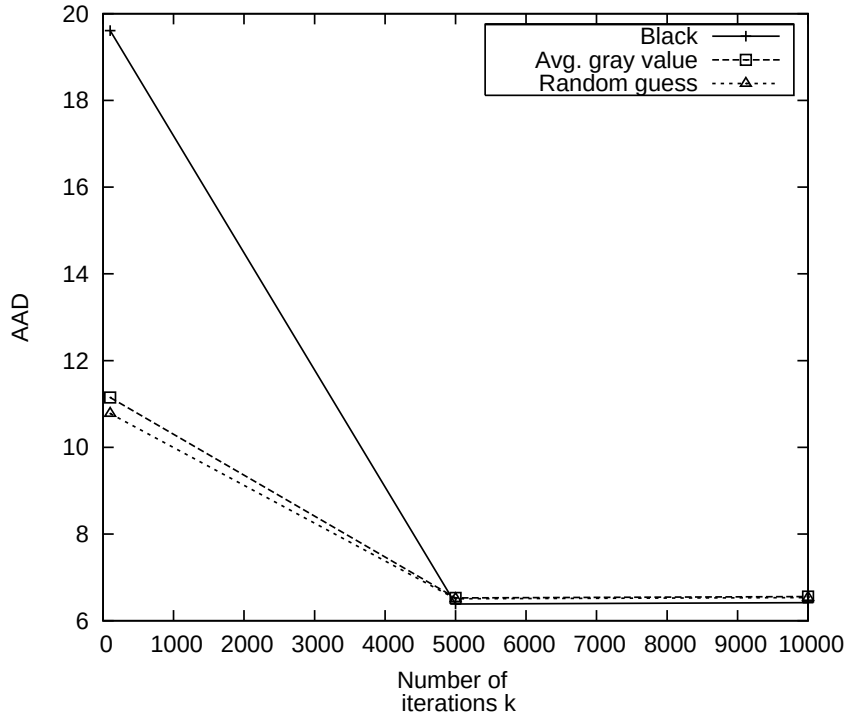

Figure 5.15: Visualization of the relation between different initial fillings of the inpainting domain $\Omega \setminus \Omega'$ and the error to the original image, given in AAD.

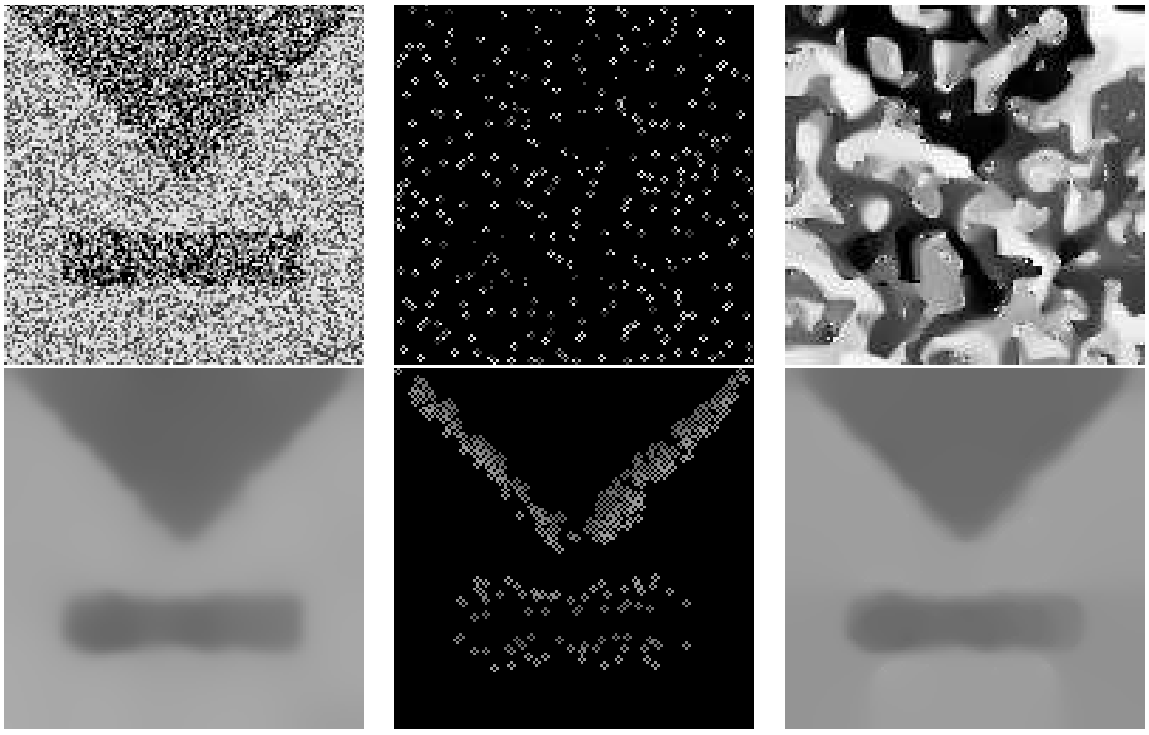

Figure 5.16: Robustness of our codec w.r.t. noise. **Top left:** Initial noisy image (256×256 pixels). **Top middle:** Sparse image for a compression rate of $1 : 10.3$ (0.78 bpp). **Top right:** Decompression result using interleaving of EED/MCM (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$ and $T = 30244$). **Bottom left:** Preprocessed version of the initial image, using EED ($\sigma = 1.0$, $\lambda = 1.0$, $T = 110$). **Bottom middle:** Sparse image of the preprocessed image with a compression rate of $1 : 10$ (0.8 bpp). **Bottom right:** Decompression result of the preprocessed image using interleaving of EED/MCM (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$ and $T = 30244$).

Method	Approximate absolute decompression time
EED inpainting	57s
EED/MCM interleaved inpainting	385s

Figure 5.17: Comparing approximate absolute computation times for different inpainting methods. Used image was *lena* (512×512 pixels). Used hardware configuration: Intel Centrino 1.4 GHz CPU, 768 MB RAM, Debian/GNU Linux.

discretization of MCM, which is only stable for $\tau \leq \frac{1}{4}$, and hence slows down our decompression step. Especially as we want to approximate the steady state for $t \to \infty$, which naturally needs a large stopping time T and hence also a large number of iterations. A logical improvement would be to use an implicit scheme as proposed in [3], and which we will discuss in Section 6.2.

5.2.7 Different Compression Rates

Now we want to compare the decompression results for the same initial image and the same parameter settings, but with different compression rates, which we do in Figure 5.18.

Here, our codec works as suspected, but one has to admit that for high compression rates, i.e, low bit-rates, the results are (visually) not too convincing. This impression is further supported when considering the AAD error measures given in Figure 5.19 and their visualization in Figure 5.20.

5.2.8 Comparison to JPEG

A big challenge for every image compression codec is to compete with JPEG, the de-facto standard codec. Actually, we are not in this situation yet, especially because corners are maybe not the most clever point set to use for sparsification. The main reason for this is probably that corners are a quite seldom image feature. If one is interested, a deeper discussion on the reasons for this will be given in Section 6.1.2 and possible improvements can be found in Section 6.2.
Nevertheless, a comparison of our codec with JPEG at different compression rates w.r.t. the AAD error measure is given in Figure 5.21. These results are visualized in Figure 5.23 and can be inspected in Figure 5.22.

5.2.9 Decompression of Color Images

In Section 3.3.3, we discussed the extension of our presented PDE-based image filters to vector-valued images, i.e., RGB color images in our case. As our CRF format, cf. Section 4.2.3, is also capable to code color images (actually it is capable to code images with an arbitrary number of channels), we can compress and decompress color images. Here, we should note that the computational effort especially for the PDE-based inpainting in the decompression step is about three times larger than in the grayscale case. However, our approaches also work for color images, as can be seen in Figure 5.24.

Why coupling matters

To illustrate the reason why one should take the challenge to deduce coupled approaches for vector valued images, as described in Section 3.3.3, we will compare uncoupled vector-

Figure 5.18: Decompression at different compression rates. **Top left:** Initial Image *lena* (512×512 pixels). **Top right:** Decompression result with a compression rate of $1 : 61.5$ (0.13 bpp), using interleaved EED/MCM inpainting (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$ and stopping time $T = 76041.5$. **Bottom left:** Same with $1 : 36.4$ (0.22 bpp). **Bottom right:** Same with $1 : 9.6$ (0.83 bpp).

Compression rate in bpp	AAD
0.13	15.95
0.22	12.46
0.83	6.37

Figure 5.19: Comparing AAD error measures for different compression rates.

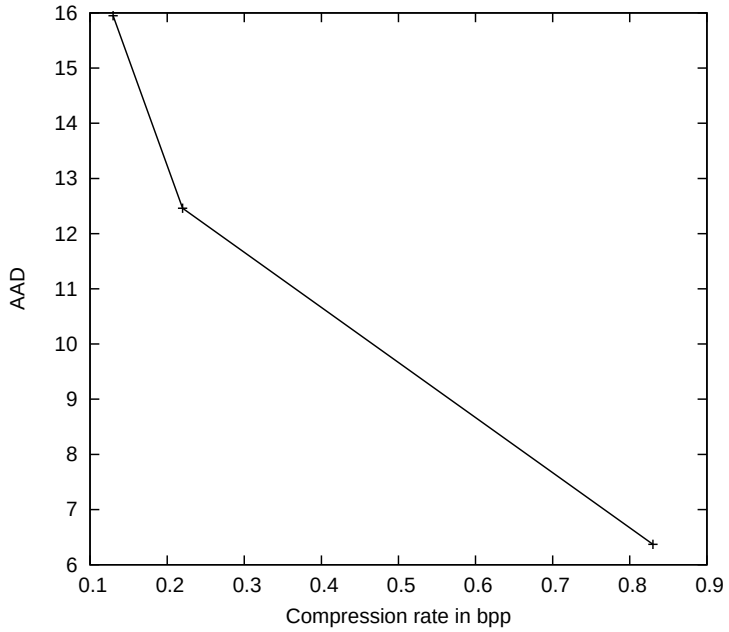

Figure 5.20: Visualization of the relation between the compression rate in bpp and the error given in AAD.

valued MCM with coupled vector-valued MCM. Therefore, let us consider Figure 5.25.

Here, we see that in the uncoupled approach on the left, important edges like the boundary of the hat, are much more blurred than in the coupled approach shown on the right.

5.2.10 Decompression of Artificial Images

Now, we want to see how well our codec works on "artificial images", where we decided to use a black and white image depicting geometric objects, like rectangles or triangles.

As can be seen in Figure 5.26, the corner detector of the compression step gives a good point set for objects with not too smooth boundaries, like the circle or the triangle. Problematic is the rectangle, as only the four corner points can be detected (cf. Figure 5.26, middle), and so its representation is very sparse. This is quite a problem in the decompression step using inpainting (cf. Figure 5.26, right), which is quite good, except for the rectangle. One should note that we had to use a very large stopping time T, because the rectangle is actually very sparse in the sparse image. Furthermore, it is interesting that we could hardly get bit rates above 0.44 bpp, as the corner detector will not classify much more pixels as corners then the ones in the sparse image of Figure 5.26. Of course, this is not a problem, as an image compression codec should usually strive for low bit rates.
However, one thing is still an issue. Consider for example the triangle: Here, our corner

Figure 5.21: Comparison of our codec with JPEG. **Top row:** Our codec : interleaved EED/MCM inpainting (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$, $T = 76041.5$). **Bottom row:** JPEG compression at same compression rates. **From left to right:** Compression rates $1 : 61.5$ (0.13 bpp), $1 : 36.4$ (0.22 bpp) and $1 : 9.6$ (0.83 bpp).

Compression Rate in bpp	AAD of our approach	AAD of JPEG
0.13	15.95	6.27
0.22	12.46	4.57
0.83	6.37	2.29

Figure 5.22: Comparing AAD error measures of our approach (interleaved inpainting) with JPEG at different compression rates.

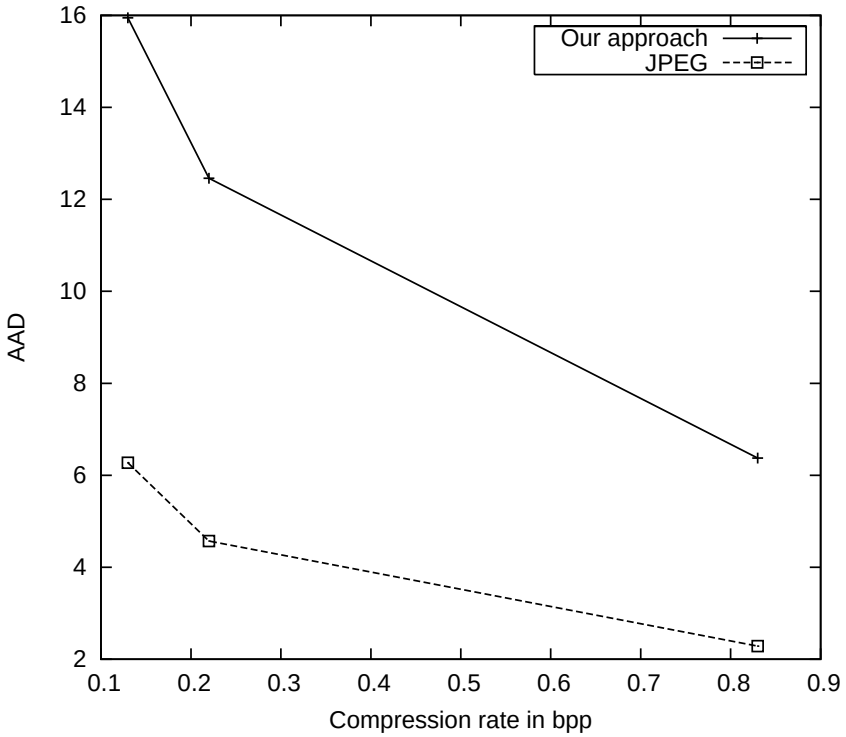

Figure 5.23: Visualization of the comparison of our codec with JPEG w.r.t. the error to the original image, given in AAD.

Figure 5.24: Decompression of color images. **Left:** Initial Image *lena* (512×512 pixels, 3 color channels, RGB). **Middle:** Corresponding sparse image at a compression rate of $1 : 9.9$ (0.81 bpp) (with white background for printing reasons). **Right:** Decompression result with interleaved EED/MCM inpainting (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$, $T = 49137$).

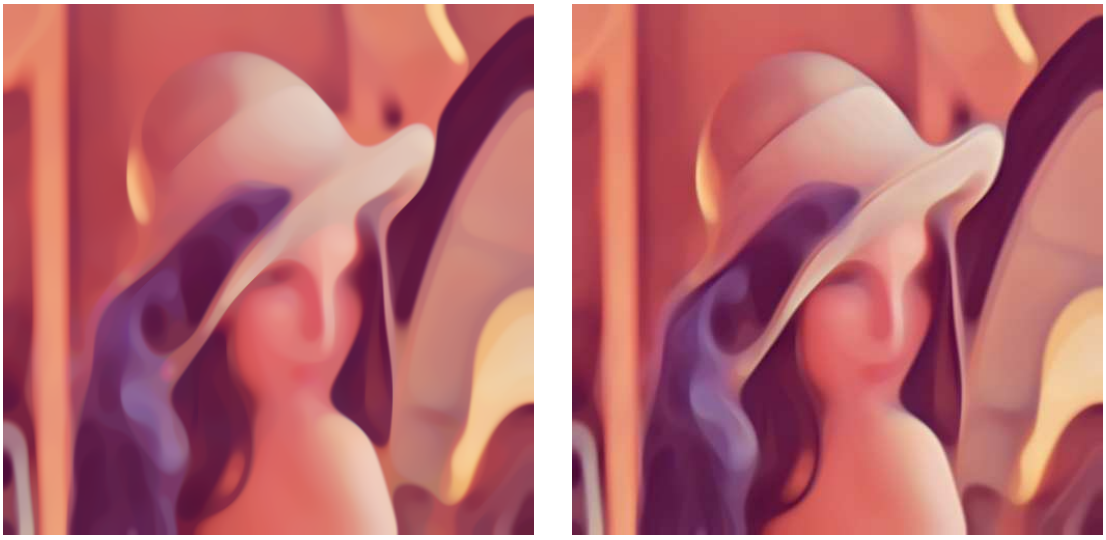

Figure 5.25: Uncoupled versus coupled vector-valued MCM . **Left:** Uncoupled MCM for *lena* (512×512 pixels, 3 color channels, RGB) and stopping time $T = 80$. **Right:** Coupled MCM with same parameters.

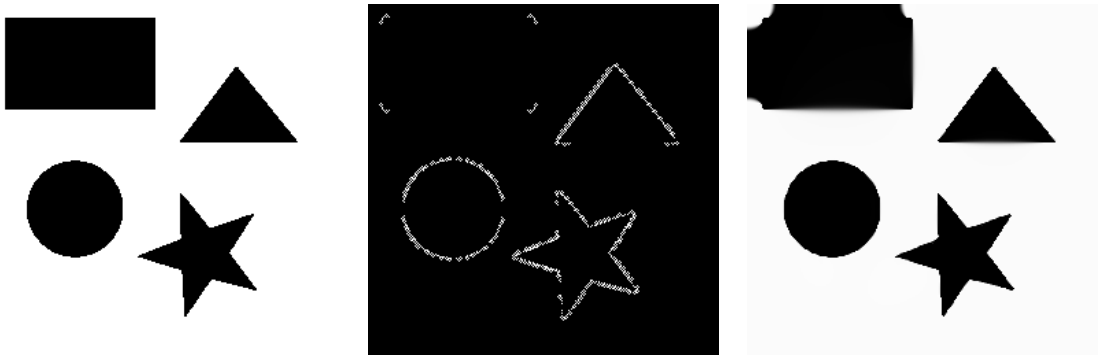

Figure 5.26: Decompression result for an artificial image depicting geometric objects. **Left:** Initial images (256×256 pixels). **Middle:** Sparse image at a compression rate of $1 : 18.2$ (0.44 bpp). **Right:** Decompression result using interleaved EED/MCM inpainting (relation $100 : 5$, $\sigma = 1.0$, $\lambda = 0.1$ and $T = 239833$).

detector works more or less like an edge detector. Theoretically, one would suspect a corner detector to detect just the tree corners of the triangle. The problem is that the discrete pixel structure of the image yields a staircasing of lines not parallel to the x- or y-axis. So, these little steps are detected as corners. To overcome this, one may use a larger threshold parameter $\Theta \approx 10$ and a larger integration scale $\rho \approx 5$. However, this also brings the problem that no points of the rectangle will be detected anymore and hence it will not be visible in the decompression result. A further discussion on improving the parameter selection can be found in Section 6.2.1.

5.2.11 Decompression of Different Images

In this chapter, we mainly used the lena image for experimental purposes to make results better comparable. However, the reader may also be interested in how our codec works on other images, so we depict some results in Figure 5.28 and give the AAD error measures of these decompression results in Figure 5.27.

Initial Image	AAD
camera	12.12
peppers	10.98
head	19.94
trui	8.15
house	41.79

Figure 5.27: Error measures for different initial images, all compressed with a compression rate of approximately 1 : 10 (0.8 bpp).

Considering the results, we see that our codec works especially well for not too textured images with larger smooth regions, as the trui image. For more textured images, like house or head, we have difficulties to restore fine details.

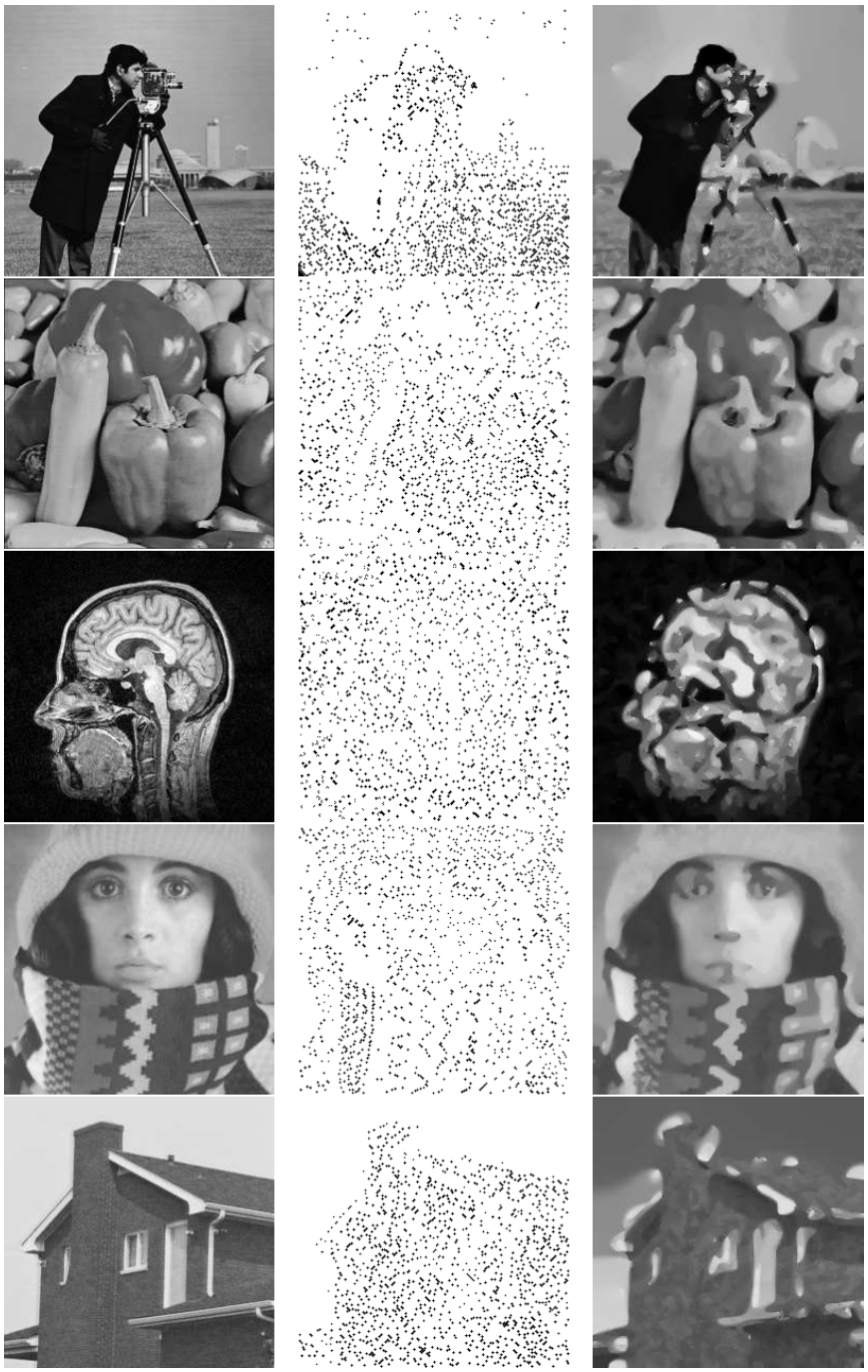

Figure 5.28: Decompression results for different initial images, all compressed with a compression rate of approximately 1 : 10 (0.8 bpp) using interleaved EED/MCM inpainting (relation 100 : 5, $\sigma = 1.0$, $\lambda = 0.1$ and $T = 42319$). **Left row:** Initial images with 256×256 pixels. *From top to bottom:* camera, peppers, head, trui, house. **Middle row:** Sparse images (with white background for printing reasons). **Right row:** Decompression result.

Chapter 6

Conclusions and Future Work

In this final chapter we want to summarize what we have done and learned while developing our image compression codec. We will state advantages and disadvantages of our approach. Considering the latter, we will also discuss ideas on how to overcome some of the disadvantages.

6.1 What have we done

We should at first mention that it is quite hard to compete with well established compression algorithms like JPEG or even with its successor JPEG2000, as they were and are developed from a large and also partially commercial community. In Section 5.2.8, we have seen that our approach is actually *not* capable to outperform JPEG. So, the aim of this thesis can just be to give a proof-of-concept for another kind of compression algorithms, which take into account image features like edges or in our case corners. Furthermore, these approaches use PDE-based inpainting techniques instead of the classical domain transformation techniques of JPEG (DCT) and JPEG2000 (DWT).

6.1.1 Summary of our Codec

To start our conclusions, we should shortly recall and summarize how our codec works. In the compression step, we first apply a corner detector to the given image and store a 4-neighborhood of each corner in an efficient way. Of course, one has to adapt the parameters of the corner detector in order to obtain a desired compression rate. What we practically do is to store the corner coordinate and then the four gray values of the neighborhood in some fixed order, called corner region. This is important to store vital image gradient information. To further improve space efficiency, we just store the offset of successive coordinates, allowing to use less bits per coordinate, and quantize gray values from 8 to 6 bits per value.
From this stored data, we can recover a sparse image and an inpainting mask, which constitute the basis of the decompression step. This step relies on PDE-based inpainting

of the sparse image using the inpainting mask, specifying where we have to inpaint and where not. Here, we used our newly invented interleaved inpainting of EED and MCM to improve decompression results. This is needed as the sparsification using a seldom image feature, like corners, poses quite some challenges, as will be discussed in the next section.

6.1.2 Usefulness of Corners for PDE-based Image Compression

The main point in our work was to show that corners at least give a reasonable point set for the sparsification in the compression step of a PDE-based image compression codec. To be more exact, we did not just use corners, but corner regions.
The big advantages lie on the one hand in the availability of corner detectors, which are theoretically and computationally not very challenging. On the other hand, it also quite easy to efficiently store corner regions, which we do in our CRF format, specified in Section 4.2.3.
The big disadvantages lie in the fact that corners are a quite seldom image feature and so one has to invest in sophisticated PDE-based inpainting techniques (like our interleaved inpainting) to obtain reasonable decompression results. To be more exact, the problem is the following: If one chooses the parameters for the corner detector in a way that it gives a (for a human observer) reasonable corner classification, the sparsification will be too strong. This is due to the fact that corners are seldom. If one then changes the parameters of the corner detector s.t. one gets more "corners", the classification becomes quite fuzzy. This means that also quite homogeneous or noisy parts of the image will be classified as corners then. Interestingly, we could anyway achieve reasonable decompression results, using our interleaved inpainting of EED and MCM.

6.1.3 Interleaved Inpainting

Already in [26], it was shown that PDE-based inpainting, and especially edge-enhancing diffusion (EED), is a good possibility to restore sparse images arising from image compression methods like ours.
The great benefit of these methods is that from their construction the arising compression artifacts consist in a blurring, i.e., *smooth* images are created. These are usually more pleasant for a human observer than the blocky artifacts of the JPEG codec. However, the coding artifacts of JPEG2000, the successor of JPEG, also consist in a blurring, cf. Section 2.1.2.
The big drawback is that inpainting approaches were originally developed to restore *small* defects in images. We in fact put these approaches to another dimension. We used them to restore *large* areas of missing information in our sparse images, which is quite challenging. In addition to that, we saw that corners are quite seldom and so the challenge when using corner regions for sparsification is even larger. So, we developed a

new kind of inpainting, namely interleaved inpainting (inspired from [8]), using two different PDEs for the inpainting. In our case these were edge-enhancing diffusion (EED), a diffusion filter (inspired from [26]), and mean curvature motion (MCM), a morphological filter, which can also be written in PDE form (inspired from [8] and [13]). With this, the inpainting, and hence the decompression results, were actually better than the ones using just EED, but the problem is that already EED inpainting is computational quite expensive, cf. Section 5.2.6. This is due to the fact that the inpainting evolution of the family $\{u(x, y, t) \mid t \geq 0\}$ takes quite some time, because we should recall that we want to approximate the steady state for $t \to \infty$, where we have $\partial_t u = 0$. This means that we have to use very large stopping times T. Nevertheless, one positive aspect may be that one may use smaller stopping times $T' \ll T$ as preview steps of the decompression results. This may make sense, as already at smaller stopping times one may identify important image features.

6.2 What can be Improved

6.2.1 Improving the Compression

This part of our approach is quite hard to improve. Eventually one may think about using other corner detectors, but the research in this area is not very active anymore. More promising would be to invest in maybe totally different approaches for selecting the point set of the sparse images. In this context one should be aware of the usual tradeoff between the quality of the point set compared to the computational complexity of the algorithm. Our corner detection approach is at the one boundary of this tradeoff. It is computationally very cheap, but the quality of the point set is at least not optimal.

More Sophisticated Parameter Selection

Our corner detection algorithms (cf. Section 3.2.1) using the structure tensor J_ρ mainly rely on three parameters: the noise scale σ, the integration scale ρ and a threshold Θ. We decided to fix $\sigma = 1$, as this works out well for most of our images, which are not too noisy. Of course, for noisy images, choosing a larger σ (and adapting ρ and Θ accordingly) may be advisable. The problem here is that the user has to manually inspect the given image and judge how noisy it is, to choose an appropriate σ.
However, we went even further and also fixed $\Theta = 1$ and so we just had to adapt ρ to achieve a desired compression rate. We also experimented with fixing $\rho = 1$ and adapting Θ which also worked, but not as well as the first approach.
So, a logical consequence may be to invest in a compression algorithm which adapts Θ *and* ρ in order to achieve a desired compression rate. The problem is that one first has to deeply understand the influence of Θ and ρ over the resulting sparse image. From this one may develop an approach which adapts both paremeters to obtain a "better" sparse image for a given compression rate.

Other Corner Regions

+ and ×−Corner Regions: We only used 4-neighborhoods which are arranged like a '+', cf. Figure 6.1, left. This choice is in principle only desirable if one faces $n \cdot 90°$ corners for $n \in \mathbb{N}$. For corners which describe angles of $45° + n \cdot 90°$ for $n \in \mathbb{N}_0$ (i.e., $45°, 135°, \ldots$) a ×−corner region as depicted in Figure 6.1, right, may be a better choice. Two questions immediately arise. The first one is "how do we measure angles of corners?". This is quite easy, as we anyway use the structure tensor J_ρ (cf. Section 3.1), which analyzes local structure within an integration scale ρ, and hence may also be used to measure the angles of corners. The second question is "what storage overhead does this imply?". This is also quite simple, we just have to spend 1 bit per stored corner region in addition, which determines if we stored a +− or a ×−corner region.

Figure 6.1: **Left:** +−corner region for $\mathcal{N} = 4$, which we actually used. **Right:** ×−corner region for $\mathcal{N} = 4$, which may be better in some cases.

Not-redundant Edge Tube: The approach of Chan and Shen [19, 18] is to store the edges of an image together with a small 2-pixel neighborhood, which they call edge tube, cf. Section 2.2.2. Already in Section 1.2, we mentioned that this approach introduces quite some redundancy concerning "straight parts" of the edges. The reason is that a sophisticated PDE-based inpainting (like EED) in the decompression step would anyway connect interrupted edges in a smooth, straight manner. We also mentioned that the decompression results with this edge-tube approach are pretty convincing, but the mentioned redundancy makes it unusable for high compression rates.
So, one idea may be to combine this approach with the corner detection approach we used. The basic idea here is that corners are exactly these points of an edge, where it abruptly changes its direction. Hence, these are the important parts of the edge to store. So, we would store the edge tube at, just before and after corner points, which avoids the mentioned redundancy. The favorable result is that with this approach, we store for every corner the optimal neighborhood. The big drawback is that these optimal corner regions cannot be stored as efficiently as fixed +− or ×−corner regions. So, one should invest in this tradeoff between optimal and efficient to store corner regions.

6.2.2 Speeding Up the Inpainting

The computational bottleneck of our approach lies in the PDE-based inpainting of the decompression step. When using pure EED this problem already exists, but is not too severe. The bigger problem is the interleaving of EED with MCM, as we used an explicit discretization of MCM, which is only stable for $\tau \leq \frac{1}{4}$. Practically, this means

that if we have an interleaving relation of EED/MCM of 100 : 5 we will do one iteration of EED with $\tau = 100$, followed by 25 iterations of MCM with $\tau = 0.2 \leq \frac{1}{4}$. So, a logical consequence may be to use an implicit discretization of MCM, as proposed in [3]. This would allow to just do alternating one iteration of EED and one of MCM with corresponding time step sizes τ.

6.2.3 Improving the Inpainting

Actually, we were the first to introduce interleaved inpainting for PDE-based image compression. So, it may also be fruitful to invest in extensions of that.

Other PDE-based Filters

One idea may be to interleave EED with other PDE-based filters. One could think of using coherence-enhancing diffusion (CED), cf. Section 3.3.1. The problem is that CED is computationally even more expensive than EED and MCM.

Interleaving More Filters

For our first approach of interleaved inpainting, we just interleaved *two* different PDE-based filters, namely EED and MCM. A natural extension of this would be to allow an arbitrary number of PDE-based filters to be interleaved. Recall from Section 3.3.4 that every PDE-based image filter can be written in the form

$$\partial_t u = L(u), \tag{6.1}$$

and that interleaving of two different filters was achieved by summing them up and use a relation factor $0 < \varepsilon < 1$, formally

$$\partial_t u = \varepsilon \cdot L_1(u) + (1 - \varepsilon) \cdot L_2(u). \tag{6.2}$$

From this, the generalization to an arbitrary number of $n \in \mathbb{N}$ filters would read as

$$\partial_t u = \sum_{i=1}^{n} \varepsilon_i \cdot L_i(u), \quad \text{where} \quad \sum_{i=1}^{n} \varepsilon_i = 1. \tag{6.3}$$

The resulting inpainting equation is then given by

$$\partial_t u = \big(1 - \mathcal{X}_{\Omega'}\big) \cdot \left(\sum_{i=1}^{n} \varepsilon_i \cdot L_i(u) \right) - \mathcal{X}_{\Omega'} \cdot (u - f), \quad \text{where} \quad \sum_{i=1}^{n} \varepsilon_i = 1. \tag{6.4}$$

One idea to use this may be to interleave EED with MCM and CED, for example.

Bibliography

[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Trans. on Computers*, C-23(1):90–93, 1974.

[2] F. Alter, S. Durand, and J. Froment. Adapted total variation for artifact free decompression of jpeg images. *Journal of Mathematical Imaging and Vision*, 23(2):199–211, September 2005.

[3] L. Alvarez. Images and pde's. *12th International Conference on analysis and optimization of Systems. Images, Wavelets and PDE's, Paris*, pages 3–15, June 1996.

[4] L. Alvarez, F. Guichard, P. L. Lions, and J. M. Morel. Axioms and fundamental equations in image processing. *Archive for Rational Mechanics and Analysis*, 123:199–257, 1993.

[5] H. Anton. *Elementary Linear Algebra*. Wiley, New York, USA, 1991.

[6] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using the wavelet transform. *IEEE Transactions on Image Processing*, 2(2):205–220, April 1992.

[7] E. Bänsch and K. Mikula. A coarsening finite element strategy in image selective smoothing. *Computing and Visualization in Science*, 1(1):53–61, July 1997.

[8] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In K. Akeley, editor, *Siggraph 2000*, pages 417–424. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[9] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM Books, Philadelphia, 2000. Second edition.

[10] R. W. Brockett and P. Maragos. Evolution equations for continuous-scale morphological filtering. *IEEE Transactions on Signal Processing*, 42(12):3377–3386, 1994.

[11] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo. Wavelet transforms that map integers to integers. Technical report, Department of Mathematics, Princeton University, 1996.

[12] J. Canny. A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 8, pages 679–698, Washington, DC, USA, November 1986. IEEE Computer Society.

[13] V. Caselles, B. Coll, and J. Morel. A kanizsa programme. In *Progress in Nonlinear Differential Equs. and their Applications*, volume 25, pages 35–55, 1996.

[14] F. Catte, P. L. Lions, J. M. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *Journal of Numerical Analysis*, 29(1):182–193, 1992.

[15] A. Chambolle. Partial differential equations and image processing. In *International Conference on Image Processing*, pages I: 16–20, 1994.

[16] T. Chan and J. Shen. Mathematical models for local deterministic inpaintings. *UCLA CAM Report*, (00-11), March 2000.

[17] T. Chan and J. Shen. Non-texture inpaintings by curvature-driven diffusions (CCD). *Journal of Visual Communication and Image Representation*, 12(4):436–449, 2001.

[18] T. F. Chan and J. Shen. Mathematical models for local nontexture inpaintings. *Journal of Applied Mathematics*, 62(3):1019–1043, 2001.

[19] T.F. Chan and J. Shen. *Image Processing and Analysis*. SIAM, Philadelphia, 2005.

[20] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. In *IEEE International Conference on Image Processing*, volume 2, pages 168–172, 1994.

[21] R. Distasi and M. Nappi. A B-tree based recursive technique for image coding. In *International Conference on Pattern Recognition*, pages II: 670–674, 1996.

[22] G. Emile-Male. *The Restorer's Handbook of Easel Painting*. Van Nostrand Reinhold, New York, 1976.

[23] E. Feig and S. Winograd. Fast algorithms for the discrete cosine transform. *IEEE Transactions on Signal Processing*, 40(9):2174, 1992.

[24] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. *ISPRS Conference on Fast Processing of Photogrammetric Data*, pages 281–305, 1987.

[25] D.S. Fritsch. A medial description of greyscale image structure by gradient-limited diffusion. *Visualization in Biomedical Computing, SPIE*, 1808:105–117, 1992.

[26] I. Galic, J. Weickert, M. Welk, A. Bruhn, A. G. Belyaev, and H.-P. Seidel. Towards pde-based image compression. In N.Paragios, O. D. Faugeras, T. Chan, and Ch. Schnörr, editors, *Variational, Geometric, and Level Set Methods in Computer Vision, International Workshop, VLSM 2005, Beijing, China, October 16, 2005, Proceedings*, volume 3752 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2005.

[27] G. Gerig, O. Kubler, R. Kikinis, and F. Jolesz. Nonlinear anisotropic filtering of MRI data. *IEEE Transactions on Medical Imaging*, 11 (2):221–232, 1992.

[28] A. Gothandaraman, R. Whitaker, and J. Gregor. Total variation for the removal of blocking effects in DCT based encoding. In *IEEE International Conference on Image Processing*, volume II, pages 455–458, 2001.

[29] C. G. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.

[30] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40:1098–1101, September 1952.

[31] N. N. Janenko. *Die Zwischenschrittmethode zur Lösung mehrdimensionaler Probleme der mathematischen Physik*. Springer, Berlin, 1969.

[32] J. Kacur and K. Mikula. Solution of nonlinear diffusion appearing in image smoothing and edge detection. *Applied Numerical Mathematics*, 50:47–59, 1995.

[33] G. Kanizsa. *Organization in Vision: Essays on Gestalt Perception*. Praeger, 1979.

[34] S. A. Khayam. The discrete cosine transform (DCT): Theory and application. Technical Report WAVES-TR-ECE802.602, Michigan State University, 2003.

[35] P.-G. Lemarié and Y. Meyer. Ondelettes et bases hilbertiennes. *Rev. Mat. Iberoamericana*, 2:1–18, 1986.

[36] S. Mallat. A theory of multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:674–693, 1989.

[37] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M.P. Boliek. An overview of JPEG-2000. In *Data Compression Conference*, pages 523–544, 2000.

[38] D. Marr and E. Hildreth. Theory of edge detection. *Royal Society London*, B207:187–217, 1980.

[39] S. Masnou and J. Morel. Level lines based disocclusion. In *IEEE International Conference on Image Processing*, volume II, pages 259–263, 1998.

[40] Y. Meyer. Principe d'incertitude, bases hilbertiennes et algébres d'opérateurs. *Séminaire Bourbaki*, page 662, 1986.

[41] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, Cambridge, 1995.

[42] S. Osher and J. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[43] W. B. Pennebaker and J. L. Mitchell. *The JPEG Still Image Data Compression Standard*. Van Nostrand-Reinhold, New York, 1992.

[44] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(7):629–639, July 1990.

[45] A. R. Rao and B. G. Schunck. Computing oriented texture fields. *CVGIP: Graphical Models and Image Processing*, 53(2):157–185, March 1991.

[46] K. Rohr. Localization properties of direct corner detectors. *Journal of Mathematical Imaging and Vision*, 4(2):139–150, 1994.

[47] M. Rumpf and T. Preusser. An adaptive finite element method for large scale image processing. In *SCALE-SPACE '99: Proceedings of the Second International Conference on Scale-Space Theories in Computer Vision*, pages 223–234, Berlin, 1999. Springer.

[48] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS, Boston, 1996.

[49] G. Sapiro, R. Kimmel, D. Shaked, B. B. Kimia, and A. M. Bruckstein. Implementing continuous-scale morphology via curve evolution. *Pattern Recognition*, 26(9):1363–1372, 1993.

[50] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, England, 1982.

[51] P. Soille. *Morphological Image Analysis*. Springer-Verlag, Berlin, 1999.

[52] T. Sziranyi, I. Kopilovic, and B. Toth. Anisotropic diffusion as a preprocessing step for efficient image compression. In *International Conference on Pattern Recognition*, volume II, pages 1565–1567, 1998.

[53] D. Taubman. High performance scalable image compression with EBCOT. In *IEEE Transactions on Image Processing (3)*, pages 344–348, 1999.

[54] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[55] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[56] S. Walden. *The Ravished Image*. St. Martin's Press, New York, 1985.

[57] J. Weickert. Scale-space properties of nonlinear diffusion filtering with a diffusion tensor. Preprint, 1994.

[58] J. Weickert. Theoretical foundations of anisotropic diffusion in image processing. In *Theoretical Foundations of Computer Vision*, pages 221–236, 1994.

[59] J. Weickert. Multiscale texture enhancement. In *Computer Analysis of Images and Patterns*, pages 230–237, 1995.

[60] J. Weickert. A review of nonlinear diffusion filtering. In *Scale-Space Theories in Computer Vision*, pages 3–28, 1997.

[61] J. Weickert. *Anisotropic Diffusion in Image Processing*. Teubner-Verlag, 1998.

[62] J. Weickert. Coherence-enhancing diffusion of colour images. *Image and Vision Computing*, 17(3-4):201–212, 1999.

[63] J. Weickert. Nonlinear diffusion filtering. In B. Jähne, H. Haußecker, and P. Geißler, editors, *Handbook on Computer Vision and Applications, Vol. 2: Signal Processing and Pattern Recognition*, pages 423–450. Academic Press, San Diego, 1999.

[64] J. Weickert. Lecture notes of the "Differential Equations in Image Processing and Computer Vision" lecture, Saarland University. http://www.mia.uni-saarland.de/Teaching/dic06.shtml, Summer term 2006. [Online, last accessed 2007-03-02].

[65] J. Weickert, S. Ishikawa, and A. Imiya. Linear scale-space has first been proposed in japan. *Journal of Mathematical Imaging and Vision*, 10(3):237–252, May 1999.

[66] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*, 7(3):398–410, 1998.

[67] Wikipedia. Corner detection. http://en.wikipedia.org/wiki/Corner_detection. [Online, last accessed 2007-03-02].

[68] Wikipedia. Discrete cosine transform DCT. http://en.wikipedia.org/wiki/Discrete_cosine_transform. [Online, last accessed 2007-03-02].

[69] Wikipedia. JPEG. http://en.wikipedia.org/wiki/JPEG. [Online, last accessed 2007-03-02].

[70] Wikipedia. JPEG2000. http://en.wikipedia.org/wiki/JPEG_2000. [Online, last accessed 2007-03-02].

[71] A. P. Witkin. Scale-space filtering. In *International Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

[72] S. Yang and Y.-H. Hu. Coding artifact removal using biased anisotropic diffusion. In *IEEE International Conference on Image Processing*, volume II, pages 346–349, 1997.

[73] S. Yao, W. Lin, Z. Lu, E. P. Ong, and X. Yang. Adaptive nonlinear diffusion processes for ringing artifacts removal on JPEG 2000 images. In *IEEE International Conference on Multimedia and Expo, ICME*, pages 691–694. IEEE, June 2004.

[74] D. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971.

Index