Universität des Saarlandes

Bachelor Thesis

Exploring Corner Regions as Inpainting Domain for PDE-based Image Compression

Author:
Daniel Gusenburger

Supervisor:
Prof. Joachim Weickert

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Mathematical Image Analysis Group
Department of Computer Science

April 16, 2020

UNIVERSITÄT DES SAARLANDES

Abstract

Faculty of Mathematics and Computer Science
Department of Computer Science

Bachelor of Science

Exploring Corner Regions as Inpainting Domain for PDE-based Image Compression

by Daniel Gusenburger

Contents

Chapter 1

Introduction

Before we dive in, let me first explain what this topic is about and try to motivate the thought process behind it as well shortly explain what you are going to read about on the next pages.

1.1 Motivation

As technology evolves, the quality and resolution of digital images improve as well. But as the quality increases so does the memory required to store the image on a hard drive. To counteract this increase in disk space usage, people have tried to reduce the sizes of digital images a lot in the last decades.
One of the most successful and probably most well known *codecs* is **JPEG** and its successor **JPEG 2000**. Both are lossy image compression methods known for fairly high compression rates while still providing a reasonably image quality. For higher compression rates however, the quality deteriorates pretty quickly and the infamous "block artifacts" are being introduced. As a remedy, a new method for image compression has been developed in the last years that aims to create better looking images for higher compression rates than JPEG and even JPEG2000.
This new method roughly works by selecting a small amount of pixels to keep and then filling in the gaps in the reconstruction/decompression step.
As one can imagine, selecting the right data is a fairly minute process and one has to carefully select the pixels to keep. Even though there has been a lot of work done in this area, the selection can still be improved.
In the past, the usefulness of corners for this process was proven in [1] even though the method proposed in this work would not surpass JPEG's abilities. Nonetheless, we want to build on it and explore how keeping larger regions of data around corners plays out in this process.

1.2 Related Work

Next up, we are going to discuss some work related to this thesis to see what advances have been done in the last years and what our work is built upon.

1.2.1 Inpainting

Inpainting as a technique is nothing new, it existed since a long time in the form of e.g. restoration of old images and film. In 2000, Bertalmio et al first proposed an algorithm to digitally inpaint images without user intervention. After consulting actual experts in image restoration they came up with a method imitating human restorators.
The main idea behind their method is to continue the structure surrounding the gap into it and simultaneously fill in the different regions in each gap with the colour at its boundaries[2].
TODO: Explaing inpainting
Although it produces good looking images without obvious artifacts, it lacks the ability to reproduce texture. Furthermore, they proposed to use second order PDEs instead of the high order PDEs they used to solve the inpainting problem. Galić et al touched on this issue in 2008, proposing to use EED because of its inpainting capabilities as a replacement for the higher order PDE[3]. However, they only used EED for inpainting instead of interleaving a PDE based inpainting approach with a mean curvature motion model as proposed in [2]. This was featured in another work that I will cover in the next section.

1.2.2 PDE-based image compression

In 2005, Galić et al. first introduced an alternative image compression method using PDE-based inpainting as a serious alternative to more classical approaches like JPEG and JPEG 2000 [4]. In this work, the authors showed the inpainting capabilities of nonlinear anisotropic diffusion, specifically of a diffusion process called *edge-enhancing diffusion*, or short EED. The specifics of this process will be covered in 2.3.
For data selection they used an *adaptive sparsification scheme relying on B-tree triangular coding (BTTC)*, hence the name *BTTC-EED* [4], as an easy to implement and fast compression method [5]. With this fairly simple approach they were already able to outperform JPEG visually for high compression rates and comic-style images [4].
Improving on this, the authors published a new paper in 2008, adding a number

of additional procedures to the compression phase, with which they were finally able to come close to the quality of JPEG 2000 [3]. Finally, in 2009, Schmaltz et al. optimised the ideas even further, building the so called **R-EED** codec with which they could even beat JPEG 2000 [6]. The main differences between [4] and [6] are the addition of several procedures to optimise the data set that is kept for inpainting in the decompression step. To roughly summarise the whole compression phase [6]:
First, an initial set of points is gathered by using a rectangular subdivision (instead of the previous triangular subdivision) of the image. This works by recursively splitting the image in half whenever the reconstruction using only the boundary points exceeds a certain error threshold. The reconstruction is also done using EED inpainting. After obtaining the initial data set, the brightness values of each of the kept pixels is rescaled to $[0, 255]$ to eliminate possible quantisation artifacts. Due to this brightness rescaling, the optimal contrast parameter for the decompression phase may change and thus has to be adjusted as well. This is generally done alternating between optimising points and the contrast parameter until a certain convergence criterion is met. As a last step, the authors invert the inpainting mask and perform the inpainting process to fill in the kept data as a means to increase the coherence between the optimised pixels and the original image.

All of these measures serve the purpose of decreasing the *mean squared error (MSE)* to a level where the proposed codec is able to outperform JPEG 2000 for compression rates higher than $\mathbf{43 : 1}$.

1.2.3 Image features in Inpainting

Semantically, edges and corners are the most important features of an image. Because of this, there are multiple publications trying to exploit the semantic importance of these features for image inpainting. For example in 2010, Mainberger et al published a paper on the reconstruction of images using only relevant edges and homogeneous diffusion. Building on their work from 2009, the authors were able to successfully reconstruct cartoonish images from only a set of edges they detectedj using the Marr-Hildreth edge detector[7]. In contrast to other inpainting methods that rely on sparse images as their inpainting domain and therefore need to use more sophisticated PDEs in order to successfully reconstruct the image, the algorithm described in this work is built around a simple homogeneous diffusion equation[8]. Their reasoning behind this is that homogeneous diffusion is *one of the analytically best understood inpainting approaches*([8]) as well as, because of its simplicity, computationally the least challenging out of all PDE-based approaches.

Another approach by Zimmer that I already mentioned in 1.1 proved the importance of corners for image inpainting in image compression[1]. It is the Even though they could not beat the quality of JPEG, it still serves as a valuable foundation for future work. Their approach was to create a sparse image from a set of what they called *corner regions* which essentially is the set of pixels directly neighbouring a cornerd detected by the Förstner-Harris corner detector[9]. For the inpainting in the decompression phase, they came up with a more sophisticated version of EED-based inpainting by interleaving it with *Mean Curvature Motion (MCM)* which, in the past, has proven itself valuable especially for inpainting larger regions [2].

1.3 Organisation

The thesis is organised as follows:
First, I will introduce some mathematical concepts such as the structure tensor and diffusion processes in 2 as well as talk about the general theory behind this topic. Afterwards in 3, we will discuss discretisation strategies and how the parameters for corner detection and inpainting were chosen. In 4, I will shortly go over the testing framework I implemented to more efficiently generate test images and simultaneously test the procedure on these images and then show some examples. Last, but not least, we will discuss the shortcomings and future work in 5.

Chapter 2

Theory

2.1 Basics

The concepts used in this thesis require some prior knowledge about basic calculus and linear algebra as well as some more advanced topics that will be introduced in the following sections. But before introducing corner detection and diffusion, we have to first define what an image is mathematically.
A *grey value image* is defined as a function $f : \Omega \to [0, 255]$ where $\Omega \subset \mathbb{R}^2$ is a rectangular subset of $\mathbb{R}^2$ of size $n_x \times n_y$, wheras a *colour image* is defined as a vector-valued function $f : \Omega \to [0, 255]^3$. For the sake of simplicity, we will focus on grey value images as most of the results can easily be transferred to vector-valued images.
Notation: Instead of writing (x, y), I will use $\boldsymbol{x} := (x, y)$ most of the time, as it makes most equations and definitions more readable. Furthermore, lowercase bold letters will denote vectors and uppercase bold letters will denote matrices.

2.1.1 Image gradient

One of the most important operations on functions in image processing is *partial differentiation.* The partial derivative of an image $f : \Omega \to [0, 255]$ in x-direction is herein denoted as f_x or synonymously as $\partial_x f$ and defined as

$$f_x(x, y) = \partial_x f(x, y) = \frac{\partial f}{\partial x}(x, y) := \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h} \tag{2.1}$$

The *gradient* of an image f is the vector containing both partial image derivatives. In multivariable calculus, the gradient of a function is an important tool to find the (both local and global) extrema of a function similar to the first derivative for a function with a single variable.

$$\mathbf{grad}(f) = \boldsymbol{\nabla} f := (f_x, f_y)^\top \tag{2.2}$$

The gradient always points in the direction of the steepest ascent/descent, it is the tangent vector to the surface at the given location[10]. Note that the gradient of a function is a vector-valued function and not a vector.

2.1.2 Convolution

Another operation from calculus that we will need is the *convolution operator*.

$$(f * g)(\boldsymbol{x}) := \int\limits_{\mathbb{R}^2} f(\boldsymbol{x} - \boldsymbol{y}) g(\boldsymbol{y}) d\boldsymbol{y} \tag{2.3}$$

Convolution is especially useful in image and signal processing to design so called linear filters such as a moving average or smoothing operation[11], [12]. As a matter of fact, in a later section we will need the convolution as a tool to smooth our image to reduce noise artifacts. To achieve this, we will use a *Gaussian convolution*, i.e. a convolution with a *Gaussian kernel* which is basically just a two-dimensional Gaussian function with a certain standard deviation[11]:

$$K_\sigma(\boldsymbol{x}) := \frac{1}{2\pi\sigma^2} \exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2\sigma^2}\right) \tag{2.4}$$

where $\|\cdot\|_2$ denotes the *Euclidean norm*. For the rest of this thesis, an image f convolved with a Gaussian with standard deviation σ will be denoted by

$$f_\sigma := K_\sigma * f$$

Note that because of the symmetry of the convolution, it would have been perfectly fine to write it as $f * K_\sigma$.

2.2 The Structure Tensor

For some applications only the gradient of an image does not give us enough information. The gradient on its own is mostly just used as an edge detector, hence we need to come up with something else for e.g. corner detection[13]. One option is the so called *structure tensor*, a matrix that contains information about the surrounding region at a specific position. With the structure tensor, or rather its eigenvalues (cf. 2.2.2), one is able to distinguish between flat regions, edges and corners.

2.2.1 Definition

!! Reconsider the definition, maybe rewrite this paragraph later !!
The structure tensor is defined as a matrix whose eigenvectors tell us the direction of both the largest and smallest grey value change. Mathematically, we can model this as an optimisation problem:
Let u be a grey value image. We want to find a unit vector $\mathbf{n} \in \mathbb{R}^2$ that is 'most parallel' or 'most orthogonal' to the gradient $\boldsymbol{\nabla} u$ within a circle of radius $\rho > 0$, i.e. one wants to optimise the function

$$E(\mathbf{n}) = \int\limits_{B_\rho(\boldsymbol{x})} \left(\mathbf{n}^\top \boldsymbol{\nabla} u\right)^2 d\boldsymbol{x}' \tag{2.5}$$

$$= \mathbf{n}^\top \left(\int\limits_{B_\rho(\boldsymbol{x})} \boldsymbol{\nabla} u \boldsymbol{\nabla} u^\top d\boldsymbol{x}' \right) \mathbf{n} \tag{2.6}$$

This function is also called the *local autocorrelation function/local average contrast*[9], [13]. Since (2.6) is a quadratic form of the matrix

$$M_\rho(\boldsymbol{\nabla} u) := \int\limits_{B_\rho(\boldsymbol{x})} \boldsymbol{\nabla} u \boldsymbol{\nabla} u^\top d\boldsymbol{x}'$$

such an optimal unit vector is by definition also the eigenvector to the smallest/largest eigenvalue of $M_\rho(\boldsymbol{\nabla} u)$[13]. The matrix $M_\rho(\boldsymbol{\nabla} u)$ can also be seen as a component-wise convolution with the indicator function

$$b_\rho(\boldsymbol{x}) = \begin{cases} 1 & \|\boldsymbol{x}\|_2^2 \leq \rho^2 \\ 0 & \text{else} \end{cases}$$

However, as the author stated in [9], using this *binary window function* leads to a noisy response and they therefore suggest using a *Gaussian window function* with standard deviation ρ. This parameter is also called the *integration scale* and determines how localised the structure information is[13]. This ultimately leads to the definition

$$\mathbf{J}_\rho(\boldsymbol{\nabla} u) := K_\rho * \left(\boldsymbol{\nabla} u \boldsymbol{\nabla} u^\top\right) \tag{2.7}$$

It is important to state that almost always, one uses a smoothed or *regularised* image instead of the original unregularised form in order to reduce numeric instabilities caused by differentiation[14]. The definition then becomes

$$\mathbf{J}_\rho(\boldsymbol{\nabla} u_\sigma) := K_\rho * (\boldsymbol{\nabla} u_\sigma \boldsymbol{\nabla} u_\sigma^\top) \tag{2.8}$$

To keep things simpler, I will omit the brackets and just simply use $\mathbf{J}_\rho$ as the structure tensor.

2.2.2 Usage in Corner Detection

The structure tensor is a symmetric matrix and thus possesses orthonormal eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ with real-valued eigenvalues $\lambda_1, \lambda_2 \geq 0$. [13] As mentioned in the preface to this section, we can use these eigenvalues to distinguish between corners, edges and flat regions as seen in figure 2.1. In total, we have to deal with 3 different cases:

1. λ_1, λ_2 are both small $\rightarrow$ flat region
2. one of the eigenvalues is significantly larger than the other one $\rightarrow$ edge
3. both eigenvalues are significantly larger than 0 $\rightarrow$ corner

If one looks at the eigenvalues as indicators of how much the grey value shifts in the corresponding direction, then the classification makes perfect sense. If both eigenvalues are small, then the grey value does not shift much in either direction, thus the area does not contain any features. In the case that one is much larger than the other one, there is an edge in direction of the eigenvector of the larger value since the largest grey value shift is in exactly this direction. For the last case it should be obvious why this refers to a corner region. When both eigenvalues are large, there is a large grey value shift in either direction, therefore there has to be a corner.

There are several approaches to find out which case applies at the current position. The biggest challenge here is to differentiate between edges and corners, i.e. we have to find out whether both eigenvalues are meaningfully larger than 0 and if one is larger than the other.

The most intuitive approach is the one by Tomasi and Kanade, sometimes also called Shi-Tomasi corner detector. It simply compares the smaller eigenvalue against some artificial threshold. The set of local maxima is then the set of corners for the image[15]. However, this approach requires to compute both eigenvalues and can thus be fairly expensive.

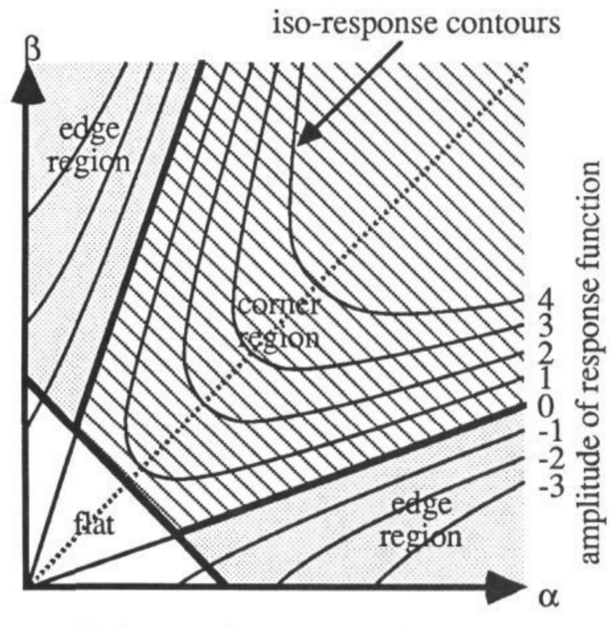

FIGURE 2.1: Visualisation of distinction of image features using the eigenvalues of the structure tensor. α, β are equivalent to the eigenvalues λ_1, λ_2. Source: [9]

!! Find sources for Rohr and Förstner !!

A cheaper approach would be to either threshold the trace

$$tr(\mathbf{J}_\rho) := j_{1,1} + j_{2,2} = \lambda_1 + \lambda_2$$

as proposed by Rohr, 1987 or the determinant

$$det(\mathbf{J}_\rho) := j_{1,1}j_{2,2} - j_{1,2}^2 = \lambda_1\lambda_2$$

as proposed by Harris and Förstner, 1988 and 1986 respectively[9]. Both of these approaches do not need to explicitly compute the eigenvalues of the structure tensor and are thus not as computationally invested. Another difference between both approaches is that the first one requires the trace by itself to be a local maximum whereas in the second approach, $(det(\mathbf{J}_\rho))/(tr(\mathbf{J}_\rho))$ needs to be a local maximum.

For the detection of relevant corners in the data selection phase, I mainly used the approach of Förstner/Harris as well as the approach of Rohr even though the Tomasi-Kanade approach was an option and has also been tested as we will see later in chapter 4. However, it has not proven as successful as the other two methods during the initial testing phase.

2.3 Diffusion

The concept of diffusion is omnipresent in the physical world. It describes, in the broadest sense possible, how particles distribute in a certain medium. This could be anything from heat in air to ink in water. But this is not its only use. It is applicable in many more fields ranging from natural sciences to finance and economics. In this chapter, we will see how diffusion applies to image processing and what benefits we gain from it. Furthermore, I will go over some basic ideas to introduce diffusion mathematically and subsequently explain different types of diffusion commonly found and used in image processing.

2.3.1 A short note on scale spaces

Before we begin to talk about diffusion, I will use this opportunity to shortly introduce scale spaces and explain how they are useful to diffusion processes in image processing and image processing in general.

2.3.2 Mathematical background

To derive the diffusion equation, we first need Fick's law (2.9) that introduces the flux

$$\boldsymbol{j} = -\boldsymbol{D} \cdot \boldsymbol{\nabla} u \tag{2.9}$$

In this equation, $\boldsymbol{D}$ is the *Diffusion tensor*, a 2×2 symmetric and positive definite matrix and $j = (j_1, j_2)$ is the aforementioned flux.[12]
The principle of mass conservation in general is given by the equation

$$\partial_t u = -\text{div}(j) \tag{2.10}$$

where $u : [0, \infty) \times \Omega \to \mathbb{R}$ is a function of time and space, div is the *divergence operator*

$$\text{div}(j) := \partial_x j_1 + \partial_y j_2$$

and j the flux defined in (2.9).
In simple terms, $\text{div}(u)$ tells us how much the value of u at a certain position changes depending on the sign. If we take the example from the introduction, the divergence of a region with high air pressure would be a positive value, since wind flows from high to low pressure regions, thus 'leaving' an area with high pressure (cf. Figure 2.2). Analogously, a region with low air pressure would

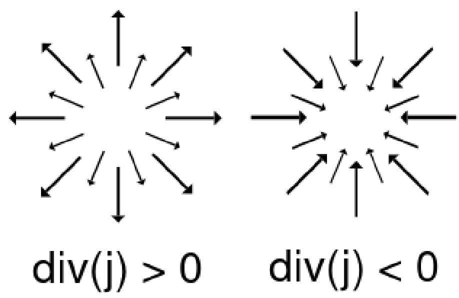

FIGURE 2.2: Visualisation of the divergence operator. Original image [16] edited with GIMP

have a negative divergence value since it 'takes in' wind from high pressure regions.
Together, equations (2.9) and (2.10) then form the general *diffusion equation*[12], [17]

$$\partial_t u = \operatorname{div}(\boldsymbol{D} \cdot \boldsymbol{\nabla} u) \tag{2.11}$$

Depending on the diffusion tensor we can distinguish between different types of diffusion processes:

Linear isotropic diffusion

In this first and simplest case, the diffusion tensor is omitted completely since it is equivalent to the identity matrix, i.e. the diffusivity is the same across the whole image. The consequence for the smooting process is that it is the same everywhere in the image, which is why this type of diffusion is also often referred to as *homogeneous diffusion*. The characteristic diffusion equation in this case boils down to the 2D *heat equation*

$$\partial_t u = \Delta u = \operatorname{div}(\boldsymbol{\nabla} u) \tag{2.12}$$

which can be solved analytically. Its solution is equivalent to a Gaussian convolution hence explaining the homogeneous smoothing. Even though this is a nice property and this case is completely understood in a mathematical sense, it is not very useful if one wants to e.g. enhance edges in an image. To achieve something like this, we have to look into nonlinear methods.

Nonlinear isotropic diffusion

The huge difference to the linear isotropic case is the diffusivity function. Previously, we had a constant diffusivity, which resulted in the same amount of

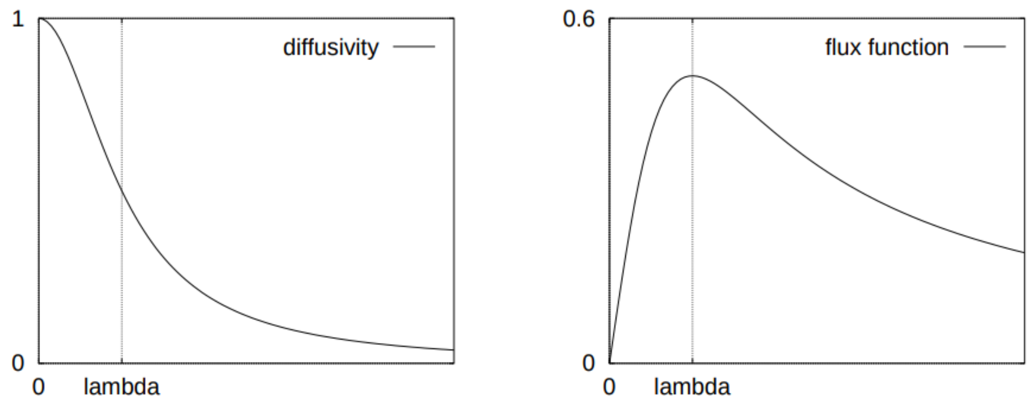

FIGURE 2.3: **Left:** Diffusivity mentioned in (2.14). **Right:** Flux function(2.16). Source: [19]

smoothing at every location. Now we want to smooth some regions more than others, i.e. the diffusivity function depends on the image or rather the gradient magnitude of the image. The most famous approach for this case is the one by Perona and Malik [18].

$$\partial_t u = \operatorname{div}(g(\|\boldsymbol{\nabla} u\|_2^2)\boldsymbol{\nabla} u) \tag{2.13}$$

where

$$g(s^2) = \frac{1}{1 + s^2/\lambda^2} \tag{2.14}$$

The interesting part about this diffusivity function is that it distinguishes between edges and non-edges or flat areas according to the *contrast parameter* λ using the gradient magnitude $\|\boldsymbol{\nabla} u\|_2^2$ as a *fuzzy edge detector*[19]. An important part to understanding why this leads to an edge enhancing effect is the flux function arising from this specific diffusivity. In general, the flux function is simply defined as

$$\Phi(s) = sg(s^2) \tag{2.15}$$

which in this particular case comes down to

$$\Phi(s) = \frac{s}{1 + s^2/\lambda^2} \tag{2.16}$$

As we already know, the flux describes the change in concentration at each position. The edge enhancing effect comes from the so called *backwards diffusion* which happens when the gradient magnitude surpasses the contrast parameter as seen in 2.3. Backwards diffusion is basically the opposite from 'normal' diffusion in a sense that it sharpens image features instead of smoothing them.

If we look at the derivative of the flux function this will become more obvious:

$$\Phi'(s) = \frac{d}{ds}\left(\frac{s}{1+s^2/\lambda^2}\right) = \frac{1-s^2/\lambda^2}{(1+s^2/\lambda^2)^2} \tag{2.17}$$

As we see, we have a maximum in the flux function at $s = \lambda$. Furthermore we can extract from the above equation that $\Phi'(s) < 0$ for $s < \lambda$ and $\Phi'(s) > 0$ for $s > \lambda$. This can also be seen in 2.3. This distinction between an increasing and decreasing flux function for values left and right from the contrast parameter causes the previously mentioned edge enhancing effect. However, this particular process is not *well-posed*[17]. As a remedy, it was generally agreed upon to use a Gaussian smoothed version to *regularise* the process[17].

Nonlinear anisotropic diffusion

But we can still go one step further and even make the direction of the smoothing process dependent on the local structure. For example, one might want to prevent smoothing across edges and rather smooth parallel to them to further embrace the edge-preserving nature of nonlinear diffusion. On the contrary to the isotropic case, in the anisotropic case the diffusivity function is not a scalar-valued function but a matrix-valued one. The diffusion tensor in this case is built in such a way that it encourages smoothing in flat regions as well as smoothing *along* edges but not *across* to them[20]. Its eigenvectors are defined to be parallel and orthogonal to the image gradient, i.e.

$$\boldsymbol{v}_1 \| \boldsymbol{\nabla} u_\sigma \qquad\qquad \boldsymbol{v}_2 \perp \boldsymbol{\nabla} u_\sigma \tag{2.18}$$

As mentioned before, we want to encourage smoothing along edges and in flat regions. Thus, we define the eigenvalue for the eigenvector parallel to the gradient to be 1 and the eigenvalue to the orthogonal one to be a diffusivity function similar to the one in the nonlinear isotropic case.

$$\lambda_1 = g(\|\boldsymbol{\nabla} u_\sigma\|_2^2) \qquad\qquad \lambda_2 = 1 \tag{2.19}$$

The diffusion tensor is then defined as

$$\boldsymbol{D}(\boldsymbol{\nabla} u_\sigma) := (\boldsymbol{v}_1 \| \boldsymbol{v}_2)\ \mathrm{diag}(\lambda_1, \lambda_2) \begin{pmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \end{pmatrix} \tag{2.20}$$

As we see in figure 2.4, EED possesses great restorative and denoising capabili-

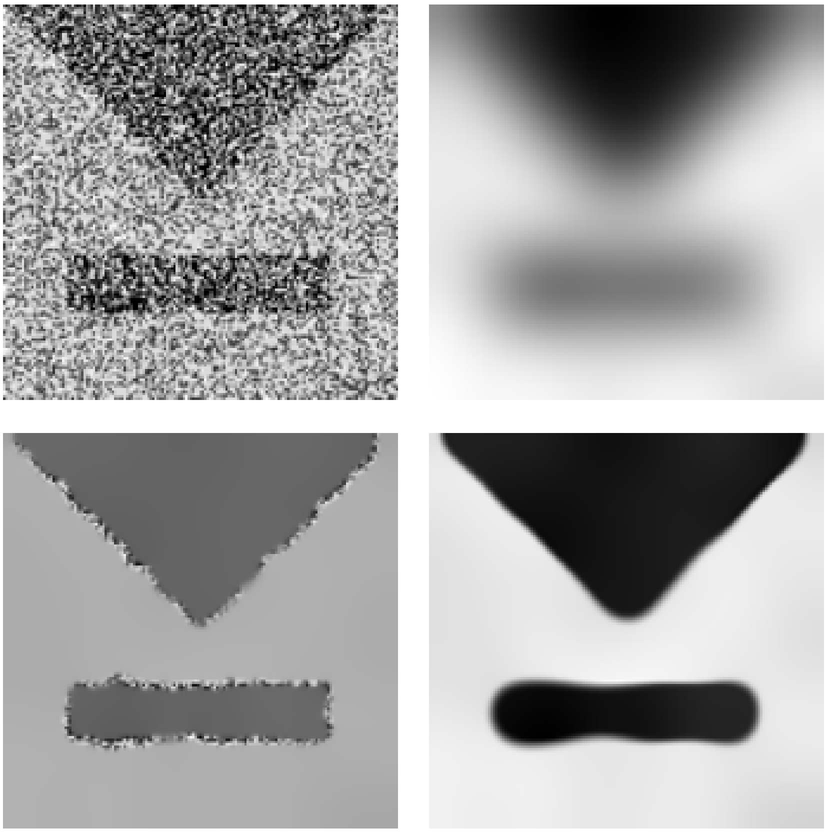

Figure 2.4: Denoising capabilities of the different types of diffusion. **Top Left:** Original image. **Top Right:** Linear isotropic diffusion. **Bottom Left:** Nonlinear isotropic diffusion. **Bottom Right:** Edge-enhancing diffusion. Source: [17]

ties. But as Galić et al. showed, EED is also an excellent choice for PDE-based inpainting[4].

2.4 EED-based inpainting

Inpainting in digital image processing has first been introduced in [2], as mentioned in related work. They used
!! Shortly explain "normal" inpainting !!

!! Explain basics of EED inpainting !!

Chapter 3

Implementation

In this chapter I will go over the technical details of the implementation. We will start with the discretisation of the theory introduced in chapter 2. The topic after this deals with the selection of that various parameters of both the corner detection and inpainting algorithms and what methods were introduced to simplify the selection or at least make it more intuitive.

3.1 Discretisation

Since reality is not infinitely fine, things such as infinitesimal calculations as seen in calculus, e.g. differentation of functions, can not be applied to the real world directly. This is a problem, because digital images are inherently not continuous as they contain only a finite number of pixels. We could have also solved the theoretical problem in a discrete domain but that would have been much more troublesome. That is why we rather develop a continuous theory and then discretise it later to actually implement the ideas as algorithms.

3.1.1 Discrete images

Let $f : \Omega \to \mathbb{R}$ be an image where $\Omega := (0, n_x) \times (0, n_y) \subset \mathbb{R}^2$ as defined in section 2.1. To *sample* the image, i.e. to discretise the image domain, we assume that all pixels lie on a rectangular equidistant grid inside Ω, where each cell in the grid has a size of $h_x \times h_y$. That yields $N_x := n_x/h_x$ pixels in x- and $N_y := n_y/h_y$ pixels in y-direction. That being said, we define the pixel $u_{i,j}$ at grid location $(i, j)^\top$ as

$$u_{i,j} := u(ih_x, jh_y) \qquad \forall (i, j) \in \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \tag{3.1}$$

With that approach, the pixels are defined to lie on the crossing of the grid lines. An alternative idea defines the pixels to lie in the centre of each cell, i.e. at location $((i - \frac{1}{2})h_x,\ (j - \frac{1}{2})h_y)^\top$. As a sidenote, the cell sizes in either direction are pretty much always assumed to be 1 in practice. But to keep the theory as universal as possible, we will use h_x and h_y instead. Sampling of the spatial domain is not the only step necessary to discretise an image. We also have to discretise the *co-domain* or *grey-value-domain*. In theory our grey value domain is just $\mathbb{R}$, but since this is rather unpractical, we quantise the grey value range, i.e. limit it to $[0, 255]$.

3.1.2 Numerical differentiation

Image derivatives are essential to image processing as seen in the previous chapter. Therefore we need a way to compute them even on discrete images. To compute the gradient or in the simpler case just the derivative of a discrete function, one generally uses so called *finite difference schemes*. Such a scheme is normally derived from the *Taylor expansion* of the continuous function. For example, we want to compute the first derivative of a 1D function $f : \mathbb{R} \to \mathbb{R}$. The Taylor expansion of *degree* n of this function around the point $x_0 \in \mathbb{R}$ is given by

$$f(x) = T_n(x, x_0) + \mathcal{O}(h^{n+1}) \tag{3.2}$$

where $\mathcal{O}(h^{n+1})$ describes the magnitude of the leading error term and as such the *approximation quality* of the Taylor series. The actual Taylor series is defined as

$$T_n(x, x_0) = \sum_{k=0}^{n} \frac{(x - x_0)^k}{k!} f^{(k)}(x_0)^{1} \tag{3.3}$$

A finite difference scheme generally uses a weighted sum of neighbouring values to compute the desired derivative expression. In our example, we want to derive a scheme to compute the first derivative of f_i using its neighbours f_{i-1} and f_{i+1}, i.e.

$$f_i' \approx \alpha f_{i-1} + \beta f_i + \gamma f_{i+1} \tag{3.4}$$

[1] $f^{(k)}$ denotes the k-th derivative of the function f

We can now describe f_{i-1} and f_{i+1} in terms of their Taylor expansion around f_i:

$$\begin{aligned} f_{i-1} &= f((i-1)h) \\ &= T_n((i-1)h, ih) + \mathcal{O}(h^{n+1}) \\ &= \sum_{k=0}^{n} \frac{(-h)^k}{k!} f_i^{(k)} + \mathcal{O}(h^{n+1}) \qquad (3.5) \\ f_{i+1} &= \cdots = \sum_{k=0}^{n} \frac{h^k}{k!} f_i^{(k)} + \mathcal{O}(h^{n+1}) \qquad (3.6) \end{aligned}$$

If we now choose a concrete value for n (here $n = 5$) we can actually compute the approximation:

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2} f_i'' - \frac{h^3}{6} f_i''' + \frac{h^4}{24} f_i'''' - \frac{h^5}{120} f_i''''' + \mathcal{O}(h^6) \qquad (3.7)$$

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2} f_i'' + \frac{h^3}{6} f_i''' + \frac{h^4}{24} f_i'''' + \frac{h^5}{120} f_i''''' + \mathcal{O}(h^6) \qquad (3.8)$$

The next step is the *comparison of coefficients*, we insert (3.7) and (3.8) into the equation and solve the arising linear system of equations for α, β, γ.

$$0 \cdot f_i + 1 \cdot f_i' + 0 \cdot f_i'' \overset{!}{=} \alpha f_{i-1} + \beta f_i + \gamma f_{i+1} \qquad (3.9)$$

After the substitution, the right hand side becomes

$$\begin{aligned} &\alpha \left(f_i - hf_i' + \frac{h^2}{2} f_i'' \right) + \beta f_i + \gamma \left(f_i + hf_i' + \frac{h^2}{2} f_i'' \right) \\ &= (\alpha + \beta + \gamma) f_i + h(-\alpha + \gamma) f_i' + \frac{h^2}{2} (\alpha + \gamma) f_i'' \qquad (3.10) \end{aligned}$$

Note that for the comparison of coefficients it suffices to use the first 3 summands of the approximation. The linear system defined by the above equation

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{h} \\ 0 \end{pmatrix} \qquad (3.11)$$

has the solutions $\alpha = -\frac{1}{2h}, \beta = 0, \gamma = \frac{1}{2h}$. This yields the approximation

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2h} \qquad (3.12)$$

To find out how good this scheme is, we re-insert (3.7) and (3.8) to get

$$\begin{aligned}\frac{f_{i+1}-f_{i-1}}{2h} = -\frac{1}{2h}&\left(f_i - hf_i' + \frac{h^2}{2}f_i'' - \frac{h^3}{6}f_i''' + \frac{h^4}{24}f_i'''' - \frac{h^5}{120}f_i''''' + \mathcal{O}(h^6)\right) + \\ \frac{1}{2h}&\left(f_i - hf_i' + \frac{h^2}{2}f_i'' - \frac{h^3}{6}f_i''' + \frac{h^4}{24}f_i'''' - \frac{h^5}{120}f_i''''' + \mathcal{O}(h^6)\right)\end{aligned}$$

Expanding and simplifying yields

$$\begin{aligned}\frac{f_{i+1}-f_{i-1}}{2h} &= f_i' + \underbrace{\frac{h^2}{6}f_i'' + \frac{h^4}{30}f_i'''' + \mathcal{O}(h^5)}_{\text{quadratic leading error term}} \\ \Rightarrow \frac{f_{i+1}-f_{i-1}}{2h} &= f_i' + \mathcal{O}(h^2) \qquad (3.13)\end{aligned}$$

This means that the error of our approximation is quadratic in the grid size. We also say that this approximation has a *consistency order* of 2. Note that for such an approximation to be reasonable, it has to have at least consistency order 1. Otherwise, it is not guaranteed that the error term diminishes if we send the grid size h to 0.

The scheme derived above is also called *central difference scheme*. Not that there are other schemes such as *forward* and *backward* differences, but for the most part, we will only use central differences since it provides us with the highest consistency order out of all three.

$$\begin{aligned}f_i' &= \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h) &&\text{(backward differences)} \\ f_i' &= \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h) &&\text{(forward differences)}\end{aligned}$$

Sometimes however, central differences are not applicable, hence we have to use one of the other schemes. One example is the time discretisation in many diffusion methods as we will see in the next section.

3.1.3 Numerical schemes for diffusion

3.2 Parameter Selection

3.2.1 Corner Detection

3.2.2 Inpainting

Chapter 4

Results

4.1 Testing framework

4.2 Test images

4.3 Experiments

4.4 Results

Chapter 5

Conclusion and Outlook

5.1 What works well

5.2 What does not

5.3 Future Work

Bibliography

[1] Henning Lars Zimmer. "PDE-based Image Compression using Corner Information". MA thesis. 2007.

[2] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, et al. "Image Inpainting". In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 417–424.

[3] Irena Galić, Joachim Weickert, Martin Welk, et al. *Image Compression with Anisotropic Diffusion*. 2008.

[4] Irena Galić, Joachim Weickert, Martin Welk, et al. *Towards PDE-Based Image Compression*. 2005.

[5] Riccardo Distasi, Michele Nappi, and Sergio Vitulano. *Image Compression by-Tree Triangular Coding*. 1997.

[6] Christian Schmaltz, Joachim Weickert, and Andrés Bruhn. "Beating the quality of JPEG 2000 with anisotropic diffusion". In: *Pattern Recognition, volume 5748 of Lecture Notes in Computer Science*. Springer, 2009, pp. 452–461.

[7] Ellen C. Hildreth. "Edge Detection". In: 207 (1985), pp. 187–217.

[8] Markus Mainberger, Andrés Bruhn, Joachim Weickert, et al. *Edge-Based Compression of Cartoon-like Images with Homogeneous Diffusion*. 2010.

[9] Christopher G. Harris and Mike Stephens. "A Combined Corner and Edge Detector". In: *Alvey Vision Conference*. 1988, pp. 147–151.

[10] Joachim Weickert. *Mathematik für Informatiker III*. Lecture Notes. 2016. URL: https://www.mia.uni-saarland.de/Teaching/MFI16/MfI3_Skript.pdf.

[11] Joachim Weickert. "Foundations II: Degradations in Digital Images". In: *Image Processing and Computer Vision*. Lecture note. Saarland University, 2019. URL: https://www.mia.uni-saarland.de/Teaching/IPCV19/ipcv19-02.pdf.

[12] Joachim Weickert. “Linear Diffusion Filtering I: Basic Concepts”. In: *Differential Equations in Image Processing and Computer Vision*. Lecture note. Saarland University, 2018. URL: https://www.mia.uni-saarland.de/Teaching/DIC18/dic18-02.pdf.

[13] Joachim Weickert. “Linear Filters III: Detection of Edges and Corners”. In: *Image Processing and Computer Vision*. Lecture note. Saarland University, 2019. URL: https://www.mia.uni-saarland.de/Teaching/IPCV19/ipcv19-13.pdf.

[14] Joachim Weickert. “Linear Filters II: Derivative Filters”. In: *Image Processing and Computer Vision*. Lecture note. Saarland University, 2019. URL: https://www.mia.uni-saarland.de/Teaching/IPCV19/ipcv19-12.pdf.

[15] Jianbo Shi and Carlo Tomasi. “Good Features to Track”. In: 1994, pp. 593–600.

[16] Wikimedia-User Bfoshizzle1. *Divergence (Captions)*. [Online; Accessed at 15.04.2020]. 2019. URL: https://commons.wikimedia.org/wiki/File:Divergence_(captions).svg.

[17] Joachim Weickert. *Anisotropic Diffusion In Image Processing*. Stuttgart: Teubner, 1996.

[18] P. Perona and J. Malik. “Scale-space and edge detection using anisotropic diffusion”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pp. 629–639.

[19] Joachim Weickert. “Nonlinear Isotropic Diffusion Filtering I: Modelling and Continuous Theory”. In: *Differential Equations in Image Processing and Computer Vision*. Lecture note. Saarland University, 2018. URL: https://www.mia.uni-saarland.de/Teaching/DIC18/dic18-04.pdf.

[20] Joachim Weickert. “Nonlinear Anisotropic Diffusion I: Modelling”. In: *Differential Equations in Image Processing and Computer Vision*. Lecture note. Saarland University, 2018. URL: https://www.mia.uni-saarland.de/Teaching/DIC18/dic18-07.pdf.