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

May 2, 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

List of Figures

Chapter 1

Introduction

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.

Chapter 2

Related Work

2.1 PDE-based image compression

In recent years, a new image codec based on partial differential equations was developed as an alternative to JPEG and JPEG2000. The core idea is to compress the image by selecting a set of pixels that serves as the basis for an inpainting process. This inpainting process is defined by a partial differential equation that is solved iteratively during the reconstruction phase.
In 2005, Galić et al.[2] first introduced an alternative image compression method using nonlinear anisotropic diffusion as a serious alternative to more classical approaches like JPEG. In this work, the authors showed the inpainting capabilities of a diffusion process known as edge-enhancing diffusion, which has since then established itself as the prime choice for PDE-based image compression. The approach they presented relies on efficiently storing a pixel mask that is later on used to reconstruct the image by filling in the missing regions using the aforementioned process of edge-enhancing diffusion and lays the groundwork for later publications building on the codec defined herein.
As already mentioned, the codec relies on computing a pixel mask from the original image. However, this process is a balancing act. On one hand, one wants to get a mask that yields a perfect reconstruction of the original image but on the other hand, one also wants to create a mask that can be encoded and stored efficiently, i.e. using the smallest amount of *bits-per-pixel (bpp)* possible. This step of mask computation is still topic of ongoing research which proves the difficulty of this problem.
In the initial codec, also called the **BTTC-EED** codec, the mask was computed by means of an adaptive sparsification scheme relying on B-tree triangular coding (BTTC) that was already introduced by Distasi et al[3] back in 1997. This fairly simple approach iteratively subdivides the image diagonally if the error of the reconstruction of the image using only the corner points of the subdivision exceeds an a priori defined threshold. In this version, the reconstruction

is approximated by a simple linear interpolation inside the triangle. The efficiency of constructing a mask using BTTC lies in the binary tree structure of the subdivision that can be encoded extremely efficiently by a simple binary string. Furthermore, grey values are encoded using a straight forward entropy coding method like Huffman coding[4]. With this fairly simple approach they were already able to outperform JPEG visually for high compression rates and comic-style images [2].
In 2008, the same authors improved this version by adding a requantisation step to the encoding phase, in which they quantised the grey values form 256 values to 64 in order to further shorten the sequences obtained by Huffman coding.

2.2 Image features in image compression

Features such as edges and corners are very important in the field of image processing as they provide almost all of the semantics of an image[5]. Therefore, feature detection has been a staple in this field for a long time. Actually, corner and edge detection algorithms such as the ones by John Canny and Chris Harris[6], [7] are still used today. Despite their semantical importance, edges and corners have not had that much impact on image compression so far.

In 2007, Zimmer introduced a new approach using corners to compute inpainting masks for PDE-based image compression[1]. In their approach, they used a simple corner detection method to sample a set of the most important corners. The inpainting mask was then obtained by storing the 8-neighbourhood around each corner. In the reconstruction/inpainting phase they used a method following the idea proposed by Bertalmio et al[8]. Instead of just using pure EED to inpaint the image, they interleaved edge-enhancing diffusion with mean curvature motion in order to improve the inpainting in regions with sparser inpainting domains.
Even though their codec was not able to compete with widely used codecs like JPEG and JPEG2000 they proved that corners are indeed useful for PDE based inpainting. They found the largest disadvantages to be the sparsity of corners in images. The reasoning behind this is that because of the sparsity of corners in an image, one has to either invest in a very sophisticated and complex inpainting mechanism to fill in the large areas between the few detected corners *or* adjust the corner detector to be more 'fuzzy'. On one hand, this leads to a denser mask and hence better results reconstruction-wise but on the other also creates suboptimal inpainting masks. Due to the fuzzy nature of the corner

detector, flat or homogeneous areas are classified as corners and therefore kept in the mask, even though these regions could have easily been filled in by even a simple inpainting process like linear diffusion.
To improve on their work, they suggested to touch on the parameter selection for the corner detector, especially the selection of both the cornerness threshold and integration scale used in the computation of the structure tensor since these heavily influence the amount and quality of corners that are detected. As another improvement, they suggested different shapes of corner regions. My thesis is largely built upon the results from this work and tries to implement the improvement ideas proposed by Zimmer to see how well they work.

In [9], [10], the authors implemented a diffusion based reconstruction using mainly edges as the inpainting mask. For the mask construction they used the Marr-Hildreth edge detector combined with *hysteresis thresholding* as proposed by Canny[7]. They stated however, that they were not locked in on the Marr-Hildreth edge detector and that others such as the classical Canny edge detector could also be used, especially for images that contain a larger amount of blurry edges. The addition of hysteresis thresholding yielded closed and well localised contours which they encoded using the lossless *JBIG* encoding[11]. To reconstruct the image from the stored contours, they used a simple linear diffusion approach which, in this case, was good enough. For cartoon-like images, they could beat the quality of JPEG and even the more sophisticated JPEG2000 in terms of the PSNR (peak signal to noise ratio) of the reconstructed image to the original one. This was all done in the earlier work[9]. In the follow-up publication from the same authors in 2010 they introduced some optimisations such as different entropy coders for the edge locations as well as an optimised method of storing the grey/colour values of the mask pixels. Another improvement was the use of an advanced method for solving the linear systems arising from the inpainting problem. In contrast to the earlier publications a fast *full multigrid scheme* was implemented to speed up the decoding phase significantly to a point where the codec is now realtime capable.

In [12], region of interest(ROI) coding was first introduced for the R-EED codec. Previously, it was not possible to specify certain regions of an image to be reconstructed with higher detail. With this new contribution, they allowed the user to specify a weighting mask that is used during the mask construction phase to adapt the mask in such a way that the specified regions would have a denser mask than other regions. This is expecially useful in medical imaging,

where some regions of an image, e.g. in a CT scan of a brain region, need to be reconstructed exactly in order to prevent false diagnoses. In this work however, the weight masks had to be chosen manually, which is not always very practical (cf. section 6.1: Discussion).
ROI coding was not the only contribution in this publication. The authors also proposed *progressive modes* for the R-EED codec to be able to compete better with widely available codecs. Progressive modes allow an image to be displayed in a coarse manner, e.g. when shown on a website that has not fully loaded the image data yet, and gradually refine the representation as more data becomes available. Last but certainly not least, they showed that their PDE-based codec is also capable of realtime video decoding and playback which sets a milestone on the way to prove the *suitability of R-EED for real-world applications*[12]. In their words this publication, or rather the extensions presented therein, mark the first step of an evolution of PDE-based compression codecs from the proof-of-concept stage to fully grown codecs with relevance for practical applications.[12]

2.3 Outline

So far, we have reviewed some of the work that is closely related and motivated the goal of our work. To explain what exactly was done in order to get to this goal, we first have to go over some of the theoretical background. We will introduce basic concepts from calculus and linear algebra (3.1) as well as some more advanced topics like the structure tensor (3.2) and inpainting methods (3.4).
In chapter 4 we will then talk about the technical details of our implementation, for example the discretisation used to implement the theoretical ideas from the previous chapter (4.1). Furthermore, we will present some additions we made and the reasoning behind them (4.2).
Afterwards, we discuss the experiments that were concluded in order to evaluate the initial idea as well as show the results these experiments yielded.
Last but not least in chapter 6, we conclude our work, discuss the strengths and shortcomings of our approach, h ow it could see some improvements and where it could be applied to.

Chapter 3

Theoretical background

3.1 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 $f : \Omega \to \mathbb{R}$ 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 $f : \Omega \to \mathbb{R}^3$. 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.

3.1.1 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

$$f_x(x, y) = \partial_x f(x, y) = \frac{\partial f}{\partial x}(x, y) := \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h} \tag{3.1}$$

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.

$$\mathbf{grad}(f) = \boldsymbol{\nabla} f := (f_x, f_y)^\top \tag{3.2}$$

The gradient always points in the direction of the steepest ascent/descent, it is the tangent vector to the surface at the given location[13]. Note that the gradient of a function is a vector-valued function and not a vector.

3.1.2 Convolution

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

$$(f * g)(\boldsymbol{x}) := \int_{\mathbb{R}^2} f(\boldsymbol{x} - \boldsymbol{y}) g(\boldsymbol{y}) d\boldsymbol{y} \tag{3.3}$$

Convolution is especially useful in image and signal processing to design so called linear filters such as a moving average or smoothing operation[14], [15]. 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[14]:

$$K_\sigma(\boldsymbol{x}) := \frac{1}{2\pi\sigma^2} \exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2\sigma^2}\right) \tag{3.4}$$

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 $f * K_\sigma$.

3.2 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[16]. 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. 3.2.2), one is able to distinguish between flat regions, edges and corners.

3.2.1 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

$$E(\mathbf{n}) = \int_{B_\rho(\boldsymbol{x})} \left(\mathbf{n}^\top \boldsymbol{\nabla} u\right)^2 d\boldsymbol{x}' \tag{3.5}$$

$$= \mathbf{n}^\top \left(\int_{B_\rho(\boldsymbol{x})} \boldsymbol{\nabla} u \boldsymbol{\nabla} u^\top d\boldsymbol{x}' \right) \mathbf{n} \tag{3.6}$$

This function is also called the *local autocorrelation function/local average contrast*[6], [16]. Since (3.6) is a quadratic form of the matrix

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

such an optimal unit vector is by definition also the eigenvector to the smallest/largest eigenvalue of $M_\rho(\boldsymbol{\nabla} u)$[16]. 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 [6], 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[16]. This ultimately leads to the definition

$$\mathbf{J}_\rho(\boldsymbol{\nabla} u) := K_\rho * (\boldsymbol{\nabla} u \boldsymbol{\nabla} u^\top) \tag{3.7}$$

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[17]. The definition then becomes

$$\mathbf{J}_\rho(\boldsymbol{\nabla} u_\sigma) := K_\rho * (\boldsymbol{\nabla} u_\sigma \boldsymbol{\nabla} u_\sigma^\top) \tag{3.8}$$

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

3.2.2 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 $\lambda_1, \lambda_2 \geq 0$. [16] 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 3.1. 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[18]. However, this approach requires to compute both eigenvalues and can thus be fairly expensive.

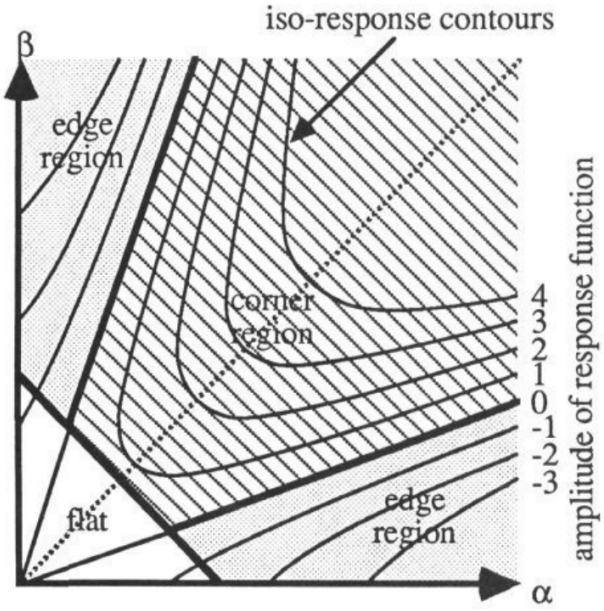

FIGURE 3.1: Visualisation of distinction of image features using the eigenvalues of the structure tensor. α, β are equivalent to the eigenvalues λ_1, λ_2. Source: [6]

!! 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[6]. 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 5. However, it has not proven as successful as the other two methods during the initial testing phase.

3.3 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.

3.3.1 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.
A scale space is generally defined as a family of images with a time parameter t that become increasingly 'simpler' as $t \to \infty$. The point of a structure like this is, that certain image features do only exist at a specific scale or a range of scales and that it is therefore beneficial to the understanding of an image to basically have a hierarchy of features.
Mathematically, a scale space needs to meet certain requirements like the semi-group property, maximum-minimum principles and others. A full explanation of all the scale space requirements would be beyond the scope of this thesis and is frankly not needed to understand the following sections.
To embed an image $f : \mathbb{R}^2 \to \mathbb{R}$ in a scale space, we need a smoothing operator T that is applied iteratively to the image. One such an operation is the Gaussian convolution:

$$T_t f := K_{\sqrt{2t}} * f \tag{3.9}$$

This operation spans the *Gaussian scale space*, one of if not the the oldest and best understood scale space. In the western world, it was first mentioned by Witkin et al.[19] 1984, but as was later found out, researchers in Japan already proposed it in 1963[20].
Scale spaces as a tool are very important to diffusion filtering, since diffusion is an iterative process and as such benefits from the notion of a scale spaces. Embedding an image in such a scale space allows us to develop a theory of evolving the image over time as we will see in the next sections, where I will first show the basic idea behind the general diffusion equation and subsequently

introduce different types of diffusion important to image processing. To avoid confusion we define the gradient for scale spaces as the *spatial gradient*

$$\boldsymbol{\nabla} u(x, y, t) = \begin{pmatrix} \partial_x u(x, y, t) \\ \partial_y u(x, y, t) \end{pmatrix} \tag{3.10}$$

instead of the spatiotemporal gradient.

3.3.2 Mathematical background

To get a glimpse of the basic idea of diffusion, we have to take a small dive into the world of physics. As mentioned in the introduction to this section, diffusion is used to describe processes of particle or concentration distribution. The differences in concentration are evened out by a flow or *flux* that is aimed from high concentration areas to areas with low concentration. This principle is stated by *Fick's law*(3.11) (hence this type of diffusion is called *Fickian diffusion*):

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

Here, the flow from high to low concentration areas is modelled by a flow whose direction is proportional to the inverse direction of the largest change in concentration, i.e. the gradient. While most of the time, the gradient determines the direction of the flux, the so called *diffusion tensor* $\boldsymbol{D}$ determines its strength. In general, this diffusion tensor is defined as a 2×2 positive definite matrix, but as we will see later, it can be reduced to a scalar valued function in more simple cases. In more complex cases, also called *anisotropic*, the diffusion tensor can also adjust the direction of the flow. We will see how this works in the section about *nonlinear anisotropic diffusion*.
Another important principle for diffusion is the conservation of mass, since mass cannot be created out of thin air and similarly cannot simply vanish. This principle is given by the equation

$$\partial_t u = -\operatorname{div}(\boldsymbol{j}) \tag{3.12}$$

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

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

In simple terms, the divergence of the flux measures whether the concentration at the current location diverges, i.e. diffuses away from the current location, or converges, i.e. moves in towards the current location. Together, equations (3.11)

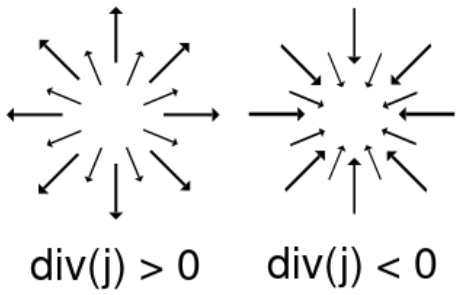

FIGURE 3.2: Visualisation of the divergence operator. Original image [21] edited with GIMP

and (3.12) then form the general *diffusion equation*[15], [22]

$$\partial_t u = \mathrm{div}(\boldsymbol{D} \cdot \boldsymbol{\nabla} u) \tag{3.13}$$

The solution to this equation can be interpreted as an image embedded in a scale space with the evolution time as the scale parameter. To get different scale spaces ergo different diffusion processes, one can alter the diffusion tensor in certain ways. In the following, we will see how different diffusion processes are characterized and what they are generally most useful for in image processing.

Linear isotropic diffusion

In this first and simplest case, the diffusion tensor is replaced by a constant diffusivity resulting in equal smoothing all across the image. The constant diffusivity can also be expressed as the identity matrix $\boldsymbol{I}$ which simplifies the diffusion equation to the 2D *heat equation*

$$\partial_t u = \Delta u = \mathrm{div}(\boldsymbol{\nabla} u) \tag{3.14}$$

which can be solved analytically. The solution to this equation has proven to be equivalent to a Gaussian convolution, thus this type of diffusion also produces a Gaussian scale space.
Obviously, homogeneous smoothing is not very useful if one wants to create a filter that is able to enhance edges. Still, it is the best understood and oldest scale space and used for many applications in image processing. One example is optical character recognition. However, to achieve edge enhancing properties, we need to look into nonlinear methods.

Nonlinear isotropic diffusion

Previously, we replaced the diffusion tensor by the identity matrix, which resulted in the same amount of smoothing at every location. To get spatially varying smoothing, we have to introduce a *diffusivity* function that depends continuously on the input image.

$$\boldsymbol{D} = g(\|\boldsymbol{\nabla} u\|_2^2)\boldsymbol{I} \tag{3.15}$$

$$\Rightarrow \partial_t u = \operatorname{div}(g(\|\boldsymbol{\nabla} u\|_2^2)\boldsymbol{\nabla} u) \tag{3.16}$$

The diffusivity function determines how much the image should be smoothed at the current location. The most well known choice for a diffusivity function was proposed by Perona and Malik [23]

$$g(s^2) = \frac{1}{1 + s^2/\lambda^2} \tag{3.17}$$

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*[24]. 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

$$\Phi(s) = sg(s^2) \tag{3.18}$$

which in this particular case comes down to

$$\Phi(s) = \frac{s}{1 + s^2/\lambda^2} \tag{3.19}$$

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 3.3. 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:

$$\Phi'(s) = \frac{d}{ds}\left(\frac{s}{1 + s^2/\lambda^2}\right) = \frac{1 - s^2/\lambda^2}{(1 + s^2/\lambda^2)^2} \tag{3.20}$$

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 3.3. This distinction between an increasing and

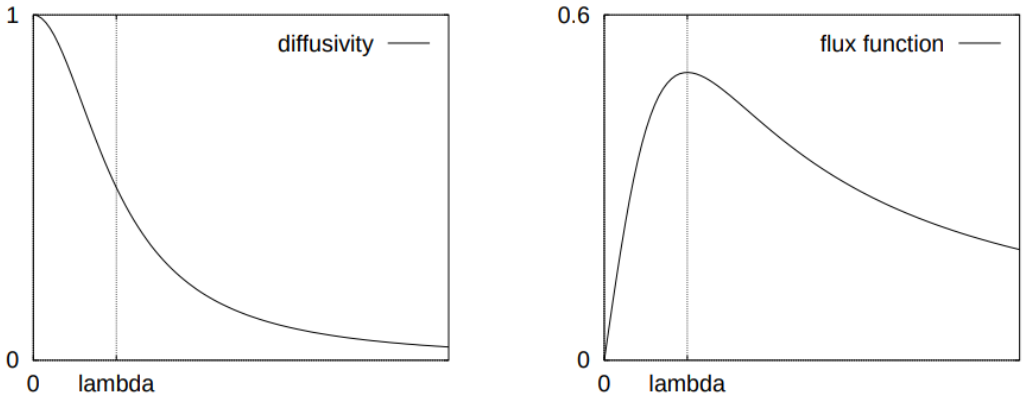

FIGURE 3.3: **Left:** Diffusivity mentioned in (3.17). **Right:** Flux function(3.19). Source: [24]

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*[22], i.e. it is very sensitive to high frequent noise. To solve this, it was proposed to use Gaussian smoothing for *regularisation*, i.e. to convolve the original image with a Gaussian kernel to damp the high frequencies that cause numerical instabilities[25].

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 further embracing the edge-preserving nature of nonlinear diffusion.

The diffusion tensor in this case is constructed from its eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ and eigenvalues λ_1, λ_2 using the theorem of matrix diagonalisation

$$\boldsymbol{D} = (\boldsymbol{v}_1 | \boldsymbol{v}_2) \operatorname{diag}(\lambda_1, \lambda_2) \begin{pmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \end{pmatrix} \tag{3.21}$$

We define the the eigenvectors to be

$$\boldsymbol{v}_1 \| \boldsymbol{\nabla} u_\sigma \qquad\qquad \boldsymbol{v}_2 \perp \boldsymbol{\nabla} u_\sigma \tag{3.22}$$

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.

$$\lambda_1 = g(\|\nabla u_\sigma\|_2^2) \qquad \lambda_2 = 1 \tag{3.23}$$

There are different viable choices for a diffusivity function that supports backwards diffusion. First, there is the already mentioned Perona-Malik diffusivity (3.17). Another option is the diffusivity function introduced by Weickert[22]

$$g(s^2) = \begin{cases} 1 & s^2 = 0 \\ 1 - \exp\left(\frac{-3.31488}{(s/\lambda)^8}\right) & s^2 > 0 \end{cases} \tag{3.24}$$

According to [24], diffusivities of this type, i.e. rapidly decreasing diffusivities, lead to more segmentation-like results. Finally, the diffusivity that was mainly used for inpainting in this work is the *Charbonnier diffusivity* [26]

$$g(s^2) = \frac{1}{\sqrt{1 + s^2/\lambda^2}} \tag{3.25}$$

3.4 EED-based inpainting

Inpainting in digital image processing has first been introduced in [8], as mentioned in related work. They used

!! Shortly explain "normal" inpainting !!

!! Explain basics of EED inpainting !!

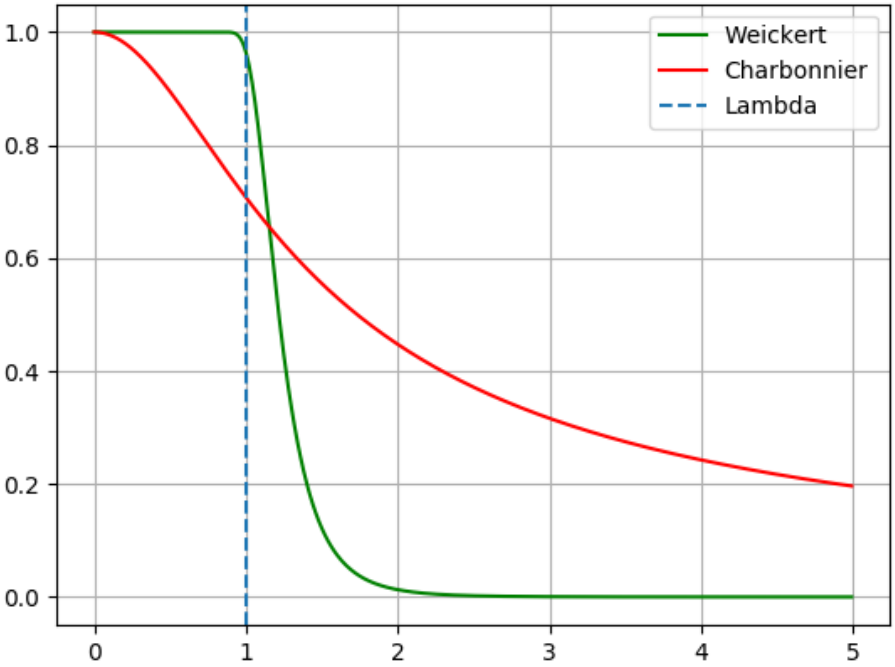

FIGURE 3.4: Weickert (3.24) and Charbonnier (3.25) diffusivities for $\lambda = 1$. Graph created with Matplotlib

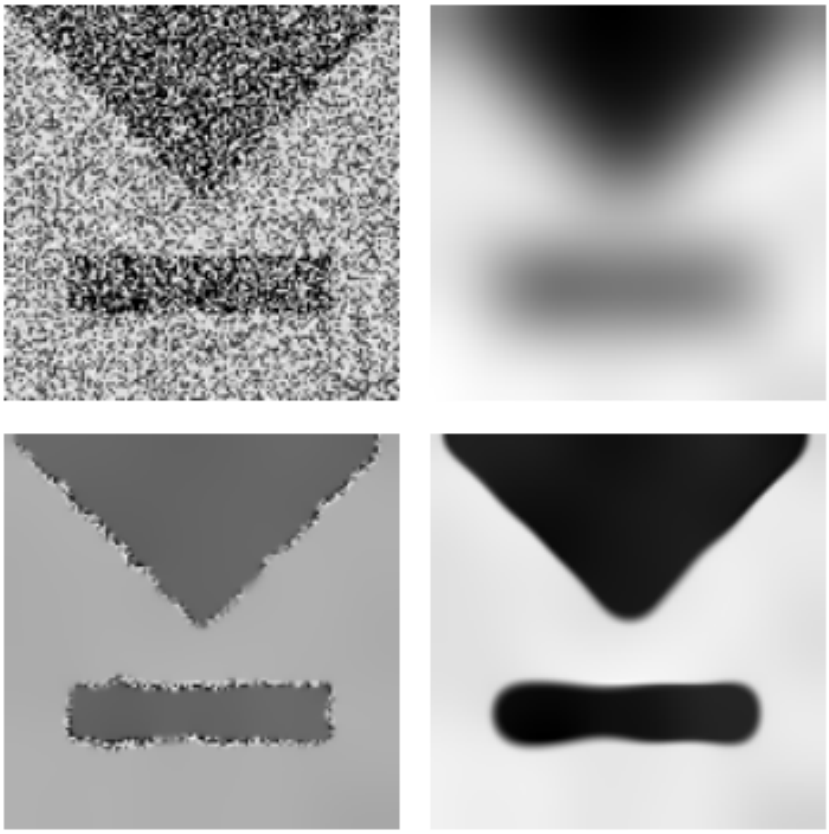

FIGURE 3.5: Denoising capabilities of the different types of diffusion. **Top Left:** Original image. **Top Right:** Linear isotropic diffusion. **Bottom Left:** Nonlinear isotropic diffusion. **Bottom Right:** Edge-enhancing diffusion. Source: [22]

Chapter 4

Implementation

In this chapter, I will present my contributions to this work and go over the technical side of things. First off, we need to discretise the theory introduced in chapter 3 in order to implement the corner detection and inpainting algorithms. To do that, we need to talk about discrete images and the method of finite differences to approximate image derivatives. After that we shortly talk about discretisation of diffusion processes which we need to implement the inpainting/restoration part to test the mask we chose.
In the second chapter I will introduce corner regions as an essential concept to this work and talk about the additions we made to the corner detection algorithm in order to select the most valuable corners for our inpainting mask.
All the source code is written in pure C and can be found in the appendix. The initial corner detection algorithm as well as the code for the inpainting alogorithm was given to me courtesy of Joachim Weickert.

4.1 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.

4.1.1 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 3.1. 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

$$u_{i,j} := u(ih_x, jh_y) \qquad \forall (i,j) \in \{1,\dots,N_x\} \times \{1,\dots,N_y\} \tag{4.1}$$

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 fully 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 limit it to $[0, 255]$. This step of the discretisation process is also called *quantisation*.

4.1.2 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

$$f(x) = T_n(x, x_0) + \mathcal{O}(h^{n+1}) \tag{4.2}$$

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

$$T_n(x, x_0) = \sum_{k=0}^{n} \frac{(x-x_0)^k}{k!} f^{(k)}(x_0)^{1} \tag{4.3}$$

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},

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

i.e.

$$f_i' \approx \alpha f_{i-1} + \beta f_i + \gamma f_{i+1} \tag{4.4}$$

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

$$\begin{aligned} f_{i-1} &= f((i-1)h) \\ &= T_n((i-1)h, ih) + \mathcal{O}(h^{n+1}) \\ &= \sum_{k=0}^{n} \frac{(-h)^k}{k!} f_i^{(k)} + \mathcal{O}(h^{n+1}) \end{aligned} \tag{4.5}$$

$$f_{i+1} = \cdots = \sum_{k=0}^{n} \frac{h^k}{k!} f_i^{(k)} + \mathcal{O}(h^{n+1}) \tag{4.6}$$

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

$$f_{i-1} = f_i - h f_i' + \frac{h^2}{2} f_i'' - \frac{h^3}{6} f_i''' + \frac{h^4}{24} f_i'''' - \frac{h^5}{120} f_i''''' + \mathcal{O}(h^6) \tag{4.7}$$

$$f_{i+1} = f_i + h f_i' + \frac{h^2}{2} f_i'' + \frac{h^3}{6} f_i''' + \frac{h^4}{24} f_i'''' + \frac{h^5}{120} f_i''''' + \mathcal{O}(h^6) \tag{4.8}$$

The next step is the *comparison of coefficients*, we insert (4.7) and (4.8) into the equation and solve the arising linear system of equations for α, β, γ.

$$0 \cdot f_i + 1 \cdot f_i' + 0 \cdot f_i'' \stackrel{!}{=} \alpha f_{i-1} + \beta f_i + \gamma f_{i+1} \tag{4.9}$$

After the substitution, the right hand side becomes

$$\begin{aligned} &\alpha \left(f_i - h f_i' + \frac{h^2}{2} f_i'' \right) + \beta f_i + \gamma \left(f_i + h f_i' + \frac{h^2}{2} f_i'' \right) \\ &= (\alpha + \beta + \gamma) f_i + h (-\alpha + \gamma) f_i' + \frac{h^2}{2} (\alpha + \gamma) f_i'' \end{aligned} \tag{4.10}$$

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

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{h} \\ 0 \end{pmatrix} \tag{4.11}$$

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

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2h} \tag{4.12}$$

To find out how good this scheme is, we re-insert (4.7) and (4.8) 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

$$\begin{aligned}\frac{f_{i+1} - f_{i-1}}{2h} &= f_i' + \underbrace{\frac{h^2}{6}f_i'' + \frac{h^4}{30}f_i'''' + \mathcal{O}(h^5)}_{\text{quadratic leading error term}} \\ \Rightarrow \frac{f_{i+1} - f_{i-1}}{2h} &= f_i' + \mathcal{O}(h^2)\end{aligned} \tag{4.13}$$

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.

$$f_i' = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h) \qquad \text{(backward differences)}$$

$$f_i' = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h) \qquad \text{(forward differences)}$$

4.1.3 Numerical schemes for diffusion

4.2 Corner regions

Our starting point for the detection of the most relevant corners was the classic Foerstner-Harris corner detector based on the structure tensor (cf. section 3.2.2).

$$\frac{\det(\boldsymbol{J}_\rho)}{\operatorname{tr}(\boldsymbol{J}_\rho)} = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2} \tag{4.14}$$

With that we build on the method that Zimmer [1] examined in 2007. They chose the Foerstner-Harris detector because it is very accurate and simultaneously not as computationally invested as the Tomasi-Kanade detector that requires to compute both eigenvalues of the structure tensor using a principal axis transformation.
But unlike in [1], we aim to to not just keep a small outline around the most important corners but increase the radius and keep a disc of pixels around each of them.

$$\texttt{corner\_region}_R(\boldsymbol{x}) := \{\boldsymbol{y} \in \Omega \ : \ \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 \leq R^2\} \tag{4.15}$$

One problem of this approach that was already apparent in [1] is the sparsity of corners or similar features in images. Therefore, we had to come up with a solution to somehow solve this issue in order to create masks with which we can reconstruct the image reasonably well. Another issue is that we have to limit the number of corners depending on the size of the corner regions since too many corners would result in a very dense mask not very suitable and/or useful for image compression. To solve these issues reasonably well, we introduced two additional steps to the corner detection process.

4.2.1 Circular non-maximum suppression

In the original version of the Foerstner-Harris corner detector, corners are detected as the local maxima of the cornerness measure (4.14). However, in this method, the local maxima were only determined in a 4- or 8-neigbourhood. This works fine in a setting where corner regions are very small as they were in [1]. When using larger regions however, this might create a very unique problem.
What we noticed when using the usual 8-neighbourhood non-maximum suppression together with larger circular corner regions is that in some images where multiple corners were located right next to each other this approach would create masks with many overlapping corner regions in a single spot and no regions in other places. To counteract this, we introduced *circular non-maximum suppression* (**CNMS**).
The idea behind this addition is that in larger corner regions, one might have multiple important corners in a single corner region. Now, instead of creating a disc around every single one of these corners, we checked if there would be another maximum in an imaginative circle around this corner. If this is the case, we disregard this corner region since the current corner is already contained in another region. Using this approach, we were able to create masks with little to no overlap between the corner regions. However, this introduced

```
def circular_suppression(harris, (x, y), r, out):
    ''' Circular non-maximum suppression at location (x,y) with
    radius r.
    Parameters:
        -harris: Harris measure map
        -(x, y): current location
        -r: radius of corner regions
        -out: map of accepted corners after suppression
    '''
    for x', y' in circle(x, y, r):
        if harris[x',y'] > harris[x,y]:
            # The current location is not a maximum
            out[x, y] = 0
            return
    # Current location is a maximum, keep the region around it
    out[x, y] = harris[x, y]
    return

```

FIGURE 4.1: Pseudo-Code for circular non-maximum suppression

one problem that we were not able to fix properly as discussed in section 6.1. After circular non-maximum suppression, we applied *percentile thresholding* to make sure, that, out of the set of corners we just detected, we only keep the most important corner regions.

!! Redo this section for the new thresholding !!

4.2.2 Percentile thresholding

In the classic version of Harris corner detection, an artificial threshold parameter T is introduced to weed out 'bad' corners. This parameter however is fairly sensible to the input image. As a workaround to make this parameter a bit more robust, we introduced a percentile parameter that is in turn used to compute a more robust threshold.

In statistics, the *n-th percentile* of a set is the value that is larger than n percent of all values in this set. We computed the percentile using the *nearest-rank method* since this was the easiest approach and worked good enough already. In this method, the n-th percentile is just simply computed as the value at position $\lceil \frac{n}{100} \cdot N_x N_y \rceil$ of the ordered set of values.

Since in an image, there are more non-corners than actual corners, we have to deal with many zero or close-to-zero cornerness values. This is a problem because the percentile computation would be skewed heavily if e.g. more than 80

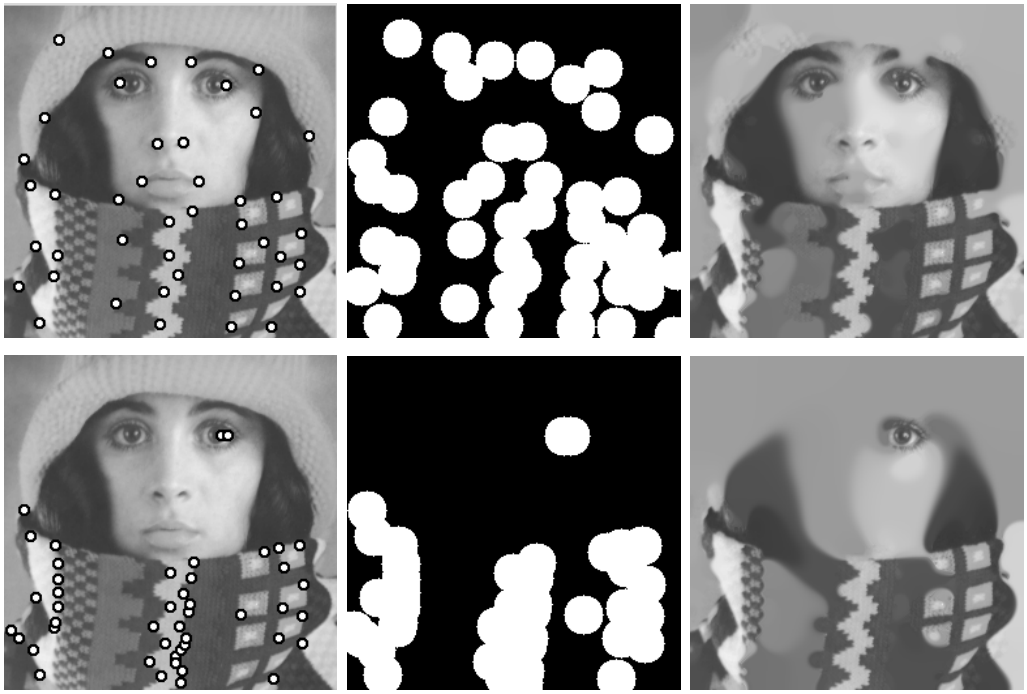

FIGURE 4.2: Effect of circular non maximum suppression on spread of corners across the image. **Top row:** Position of corners, inpainting mask and inpainting results **with** CNMS. **Bottom row:** Position of corners, inpainting mask and inpainting results **without** CNMS. Corner detection with Foerstner-Harris corner detector, $\sigma = 1, \rho = 2.5, R = 15$ and a percentile of 0.5

percent of all values are zero already. To solve this, we first filtered out all values below a threshold close to zero and then accumulated the remaining values into an array of appropriate size. This array is then sorted using an already implemented Quicksort algorithm from the C standard library. Subsequently, the new threshold is given as the value at the index calculated above.

```
1  /* We want to make sure that we keep a maximum of p percent of
     all pixels */
2  long n_total = nx * ny;
3  float max_area_p_corner = pi * radius * radius;
4  float perc_keep = (1 - perc) * n_total;
5  long n_corners = ceil(perc_keep / max_area_p_corner);
6  float vals[n_total];
7
8  /* Flatten cornerness map and prepare for sorting */
9  for (i = 1; i <= nx; ++i) {
10   for (j = 1; j <= ny; ++j) {
11     vals[nx * (i-1) + (j-1)] = v[i][j];
12   }
13 }
14
15 qsort(vals, n_total, sizeof(float), float_cmp);
16
17 float T;
18 if (n_total - n_corners < 0)
19   T = vals[0];
20 else
21   T = vals[n_total - n_corners];
22
23 printf("Optimal number of corners: %ld\n", n_corners);
24
25 n_corners = 0;
26 for (i = 1; i <= nx; i++) {
27   for (j = 1; j <= ny; j++) {
28     if (v[i][j] <= T) {
29     } else {
30       v[i][j] = 255.0;
31       ++n_corners;
32     }
33   }
34 }
35
36 printf("Actual number of corners: %ld\n", n_corners);
37
38 /* free storage */
39 disalloc_matrix(u, nx + 2, ny + 2);
```

FIGURE 4.3: Percentile thresholding

With these two additions we were able to make sure that the resulting masks are not too dense and the data is reasonably distributed across the image as

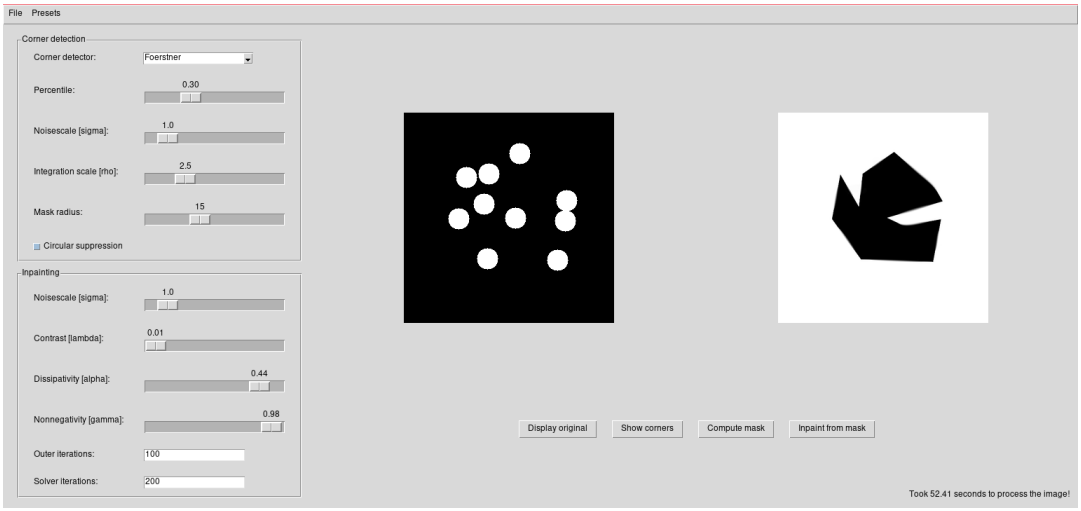

FIGURE 4.4: GUI for easier testing

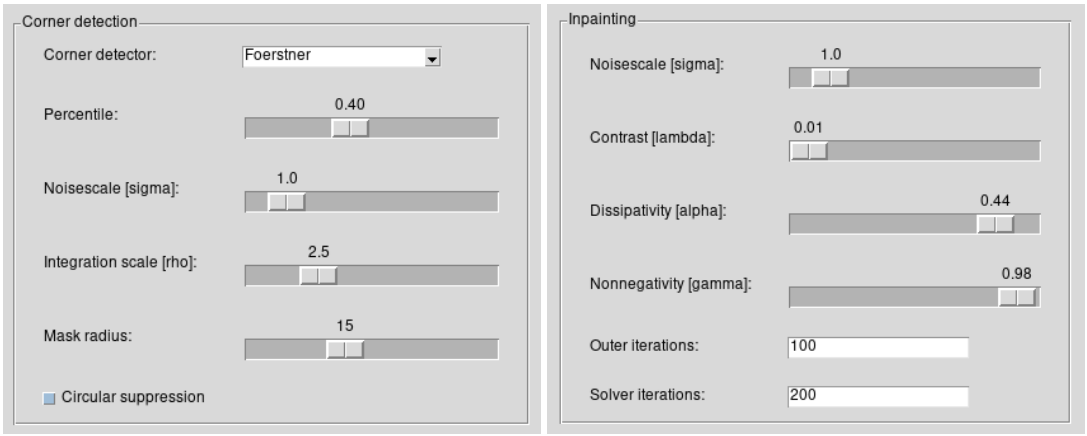

FIGURE 4.5: Panels to control parameters

seen in the examples in 5.3. Another interesting part is that thanks to CNMS, the number of corners detected is now somewhat dependent on the size of the corner regions which is a very useful result as it takes care of the second problem mentioned in the preface to this chapter on the size of the corner regions which is a very useful result as it takes care of the second problem mentioned in the preface to this chapter (cf. section 5.3).

4.3 Graphical User Interface (GUI)

To make experimenting with different parameters and images much simpler, we implemented a Graphical User Interface using Python's `Tkinter` module. We wrote wrapper functions that call the compiled C programme from the command line with the given parameters.

Chapter 5

Experiments

In this chapter, we will focus on some experiments that we performed and their results. In the first part, we show the images that were used for the experiments. Afterwards we are going to tackle the point of parameter selection since it is not trivial to find the optimal set of parameters. Therein, we explain the meaning of each parameter in both the corner detection and inpainting process as well as present a reasonable choice for them. Lastly, we present some results from the experiments and whether or how this is useful for PDE-based image compression methods.

5.1 Test images

We divide the test images in 2 different groups because as we will see later in this chapter, the method works better on a certain category of images. First, we have the 'normal' grey value images as seen in figure 5.1. All these images were provided by Joachim Weickert and count to the standard test images in the image processing community. The second groud is the group of binary grey value images, i.e. images that contain only black and white pixels instead of 256 different grey values as is the case in the first category. Of this second category (cf. figure 5.2), only the picture `cat` was provided by Joachim Weickert, the other ones were generated by me using the image manipulation programme `GIMP`. Each of these generated pictures was created to test or highlight different aspects of the corner detection or inpainting method as seen in section 5.3.

5.2 Parameter Selection

As mentioned earlier, the optimal set of parameters is difficult to choose, that is why we have to experiment a lot to find a decent approximation to this optimal set. In the following we will therefore explain what the different parameters do, how that influences the image or mask quality and ultimately, how we chose

FIGURE 5.1: Grey value images used for testing. From left to right and from top to bottom: `trui`, `house`, `lena512`, `bank`

the parameters that we used to get our results. First off, we talk about corner detection as this was the primary focus of our work. The parameter choice was mainly fixed from the beginning as we used sets of parameters proven to be fairly optimal in previous work by <insert reference here>. Nonetheless, we shortly go over the different parameters and explain their influence.

5.2.1 Corner Detection

For corner detection there are not that many different parameters to play around with. In the classical approach using the structure tensor (cf section 3.2.2) we have only 3 parameters:

- the noise scale σ

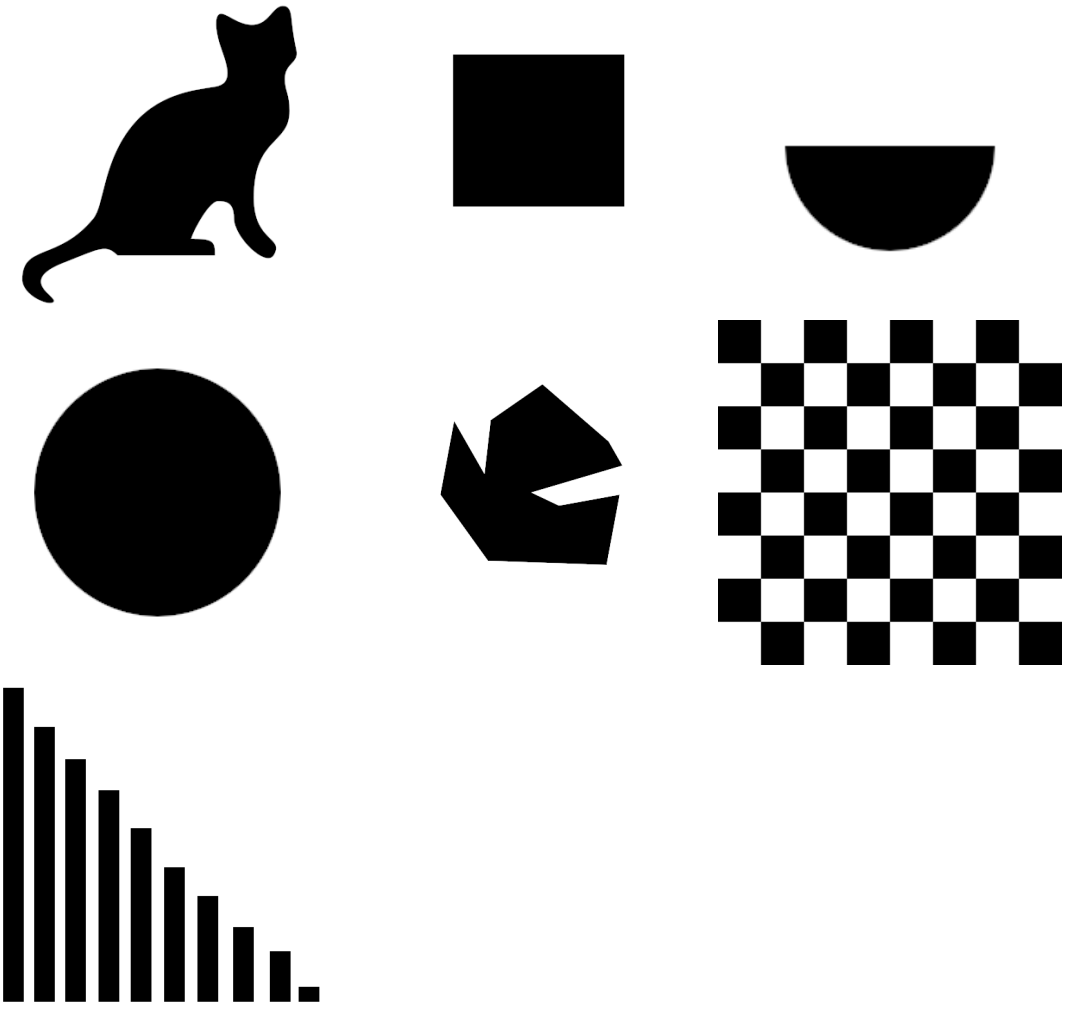

FIGURE 5.2: Binary images used for testing. From left to right and from top to bottom: `cat`, `rect`, `semicircle`, `ellipse`, `abstract1`, `checkerboard32`, `testlength`

- the integration scale ρ and
- a threshold parameter T to filter out non-important corners

but as mentioned in section 4.2, we replaced the threshold by a percentile parameter to make it more robust. However, we introduced a new parameter R, that determines the size of the corner regions and subsequently also the size of the region where the non-maximum suppression is performed in.

Regarding the noise scale, we generally want to choose it as large as necessary but keep it as small as possible, meaning that we want to choose the smallest noise scale that gets rid of most of the noise in the image since with a larger σ one often faces the problem that the detected corners can not be located as accurately anymore, since more and more relevant features are smoothed away (cf. scale space section). Another problem is that the gaussian scale space

(iterated gaussian smoothing) may even introduce new corners.[22] Most of the time however, a σ of 1 is sufficient enough to remove most of the noise and unnecessary details and still provide an accurate result.
Twe integration scale ρ basically determines how 'local' the corner detection is as it influences the size of the region structural information is averaged in during the computation of the structure tensor. ρ should always be chosen larger than the noise scale σ, which leads us to a 'standard' value of 2-2.5. In our experiments, these values usually yielded the best results.
The newly introduced percentile parameter regulates the amount of corners that are ultimately being detected since the percentile thresholding is applied *after* the CNMS.

5.2.2 Inpainting

The main parameters required by the inpainting process are

- the noise scale σ,
- the integration scale ρ,
- the contrast parameter λ,
- the dissipativity parameter α and
- a non-negativity parameter γ.

The implementation of the algorithm that was provided by Joachim Weickert also requires some technical parameters such as the choice of diffusivity function, time discretisation scheme, time step size and iterative solver used for the semi-implicit time discretisation scheme. All these paramters were fixed beforehand. We used the Charbonnier diffusivity function mentioned in <theory>. As the other parameters are concerned, a semi-implicit scheme with a time step size of 1000 has been used. As an iterative solver for the system of equations that arises from the semi-implicit scheme a conjugate gradient solver with 200 iterations was used.
If we recall from the theory section, when using EED, we do not need the integation scale, as the integration scale only influences the radius of the structure tensor which is actually not needed for this method of inpainting. Thus, this parameter will be fixed to 0.
The parameters α and γ are purely numerical parameters that are used to stabilise the algorithm or rather help to ensure that the stencil weights of the

discretisation of the diffusion process meet certain requirements. A fairly optimal choice for these parameters has been proposed in [27], namely $\alpha = 0.44, \gamma = 0.98$. In general, one could image the parameter α as a sharpness parameter: the larger the α (but not larger than 0.5), the sharper the image. More on the nature of these two parameters can be read about in [22], [27].
Next up is the contrast parameter γ that is required in the diffusivity function (3.17). As already explained in the 3.2, this parameter helps to distinguish between edges and non-edges. For the EED inpainting this is especially important since it basically determines how strongly edges will be continued into inpainting regions. We experimented with different choices for this parameter but came to the conclusion that a fairly small value yields the best results. In general, we used a value of 0.03 for most of the images, but found that for a certain set of images, an even smaller value of 0.01 yielded better results.
Last but not least, the noise scale σ determines how much the initial image is smoothed before the computation of the image derivatives. In general, this parameter is always fixed at 1.0 as it only serves to regularise the differentiation process.

5.3 Results

Chapter 6

Conclusion and Outlook

6.1 Discussion

6.1.1 What works well...

6.1.2 ...and what does not

6.2 Future Work

6.2.1 Improvements that can be done

6.2.2 Implications for image compression

Bibliography

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

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

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

[4] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101.

[5] D Marr and A Vision. *A computational investigation into the human representation and processing of visual information*. 1982.

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

[7] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698.

[8] 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.

[9] Markus Mainberger and Joachim Weickert. "Edge-based image compression with homogeneous diffusion". In: *Computer Analysis of Images and Patterns*. Springer, 2009, pp. 476–483.

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

[11] Joint Bi-level Image Experts Group. "Information technology – progressive lossy/lossless coding of bi-level images". In: *ISO/IEC JTC1 11544, ITU-T Rec. T.82* (1993).

[12] Pascal Peter, Christian Schmaltz, Nicolas Mach, et al. "Beyond Pure Quality: Progressive Modes, Region of Interest Coding and Real Time Video decoding for PDE-based Image Compression". In: *Journal of Visual Communication and Image Representation*. Vol. 31. 2015, pp. 253–265.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] A. Witkin. "Scale-space filtering: A new approach to multi-scale description". In: *ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 9. 1984, pp. 150–153.

[20] Joachim Weickert, Seiji Ishikawa, and Atsushi Imiya. "Linear Scale-Space has First been Proposed in Japan". In: *Journal of Mathematical Imaging and Vision* 10 (1999), pp. 237–252.

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

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

[23] 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.

[24] 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.

[25] Francine Catté, Pierre-Louis Lions, Jean-Michel Morel, et al. "Image Selective Smoothing and Edge Detection by Nonlinear Diffusion". In: *Siam Journal on Numerical Analysis* 29 (1992), pp. 182–193.

[26] P. Charbonnier, L. Blanc-Feraud, G. Aubert, et al. "Two deterministic half-quadratic regularization algorithms for computed imaging". In: *Proceedings of 1st International Conference on Image Processing*. Vol. 2. 1994, pp. 168–172.

[27] Joachim Weickert, Martin Welk, and Marco Wickert. "L^2-Stable, Nonstandard Finite Differences For Anisotropic Diffusion". In: *A. Kujiper, K. Bredies, T. Pock, H. Bischof (eds) Scale-Space and Variational Methods in Computer Vision. SSVM 2013. Lecture Notes in Computer Science*. Vol. 7893. Springer, Berlin, Heidelberg, 2013, pp. 380–391.