Universität des Saarlandes

Bachelor Thesis

Exploring Circular Corner Regions as Seed Points for PDE-based Inpainting

Author:
Daniel Gusenburger

Supervisor/Advisor/Reviewer:
Prof. Dr. Joachim Weickert

Second reviewer:
Dr. Pascal Peter

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

June 23, 2020

UNIVERSITÄT DES SAARLANDES

Abstract

Faculty of Mathematics and Computer Science
Department of Computer Science

Bachelor of Science

Exploring Circular Corner Regions as Seed Points for PDE-based Inpainting

by Daniel Gusenburger

Over the last years, a new class of image compression codecs has been developed making use of novel inpainting techniques. These newly developed methods have been shown to have the potential to surpass common codecs like JPEG and JPEG2000. The idea behind them is to store a sparse subset of pixels and fill in the missing information with a digital image inpainting algorithm. The challenge with such an approach is the choice of pixels that serve as the basis to reconstruct the image. While the most successful ideas are based on stochastic or subdivision based methods, semantic approaches have barely been explored. In this thesis I build on results from a previous publication and further explore the potential of the reconstruction of images by only using a set of corners from the image. I examine how the accuracy of the corner detection influences the reconstruction quality and lastly introduce additional procedures that aim to make the mask calculation a bit more robust with respect to the pixel density of the inpainting mask.

Contents

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**[1] and its lesser known but more advanced successor **JPEG 2000**[2]. Both are lossy image compression methods known for fairly high compression rates while still providing a reasonable 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 class of image compression methods based on *digital image inpainting* have been developed over the last years [3]–[11] that aim to create better looking images for higher compression rates than JPEG and even JPEG2000.
Normally, inpainting methods are used to restore damaged paintings, remove objects from images, etc., but by pushing these methods to their limits, they can also be used for image compression. This can be accomplished by carefully selecting a very small subset of all pixels in the image, and then use such an inpainting method to restore the image from only the given data. 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[12]–[14], the selection can still be improved.
In the past, few methods explored how well corners/corner-like features do as candidates for the inpainting process. In [7], a method for image compression using corners was proposed, but was not able to come close to the performance of JPEG. However, this method did not explore its full potential. One of the main points of criticism is, that the corner localisation was not accurate enough, or rather the inaccuracies were not accounted for in the mask selection[15]. This is why I want to improve on this approach and try to tap into its unexplored potential.

Goal

The goal of this thesis is two-fold: First, I want to explore how the inaccuracies in the corner localisation can be properly accounted for in the mask creation. This includes finding out how large the corner region disks have to be to counteract the inaccuracy introduced during the detection phase.
Secondly, I want to introduce some additional procedures to improve the selection of corner regions and make it a bit more robust with respect to the pixel density in the final mask, as it is fairly important to reliably produce masks containing only a certain percentage of pixels if one wants to use this approach for image compression. One of the biggest factors that I had to consider was, that corners and similar features can be quite rare in images, severely limiting the amount of candidates one can choose from, especially compared to approaches like [11], [14]. I will cover these procedures in 4.2 while the first goal will be covered in 5.1.

2 Related Work

In this chapter I will discuss some publications that are closely related to my thesis. In the first part I will give a short recap of the evolution of PDE-based image compression from the first mention to today to give more context about the overall field of research this thesis resides in. Afterwards I am going to adress some publications in the more narrow field of image compression using semantic features like edges and corners, one of which is subject to improvement in that is discussed in this thesis.

2.1 PDE-based image compression

In recent years, a new image codec relying on an inpainting process based on solving 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 the 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.[3] first introduced an 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 (or short EED), 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 therein.
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 or at least close to 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 gives a good idea of 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[16] 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[17]. With this approach they were already able to outperform JPEG visually for high compression rates and comic-style images [3].

In [11], this approach was further improved by the addition of several optimisation steps. They found that by replacing the triangular subdivision by an adaptive rectangular subdivision procedure, the quality of the reconstructed images could be improved. Furthermore instead of the simple Huffman encoding, they switched to a more advanced entropy coding standard called *PAQ*[18]. Additional optimisations they introduced include a brightness rescaling step, a reconstruction parameter optimisation step and finally a step they called 'interpolation swapping'. In the brightness rescaling step, the grey value range of mask pixels was adjusted to use up the full range in order to reduce quantisation artifacts. The parameter optimisation aims to find an optimal diffusivity parameter for the diffusion step in the reconstruction phase. This optimised parameter is then also stored in the compressed image. By including this step, they were able to cut the reconstruction error in half.
Last but not least, they introduced the process called 'interpolation swapping' that is performed after the decompression phase. In this procedure, in the reconstructed image, the roles of mask pixels and inpainted pixels are reversed for a final diffusion step to smooth out any artifacts that may occur on the boundary between known and unknown pixels during the decompression phase. With all these additions, this new codec called **R-EED** can outperform the successor to the JPEG codec, namely JPEG2000 even for high compression rates. In [14], an alternative approach to the rectangular subdivision used in [11] was proposed. They considered two different methods. First, they used the results from [12] to derive an *analytical approach* by calculating the *Laplacian of a Gaussian* $|\Delta f|^s$ for some exponent $s > 0$ and then apply electrostatic halftoning[19] in order to get a binary point mask.

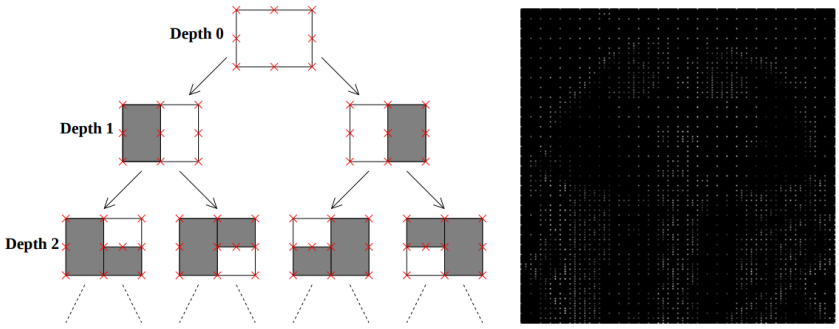

FIGURE 2.1: Examples for rectangular subdivision in the **R-EED** codec. Source:[11]

The second method they proposed is called *probabilistic sparsification*. It works by iteratively selecting a set percentage of pixels from the original image into the mask, reconstructing the image using the same inpainting operator as in the decompression step, then calculating the reconstruction error for each pixel in the inpainting mask and then throwing out the pixels with the largest reconstruction error. This is repeated until a predefined pixel density in the mask is reached. However, the authors stated that this method is not guaranteed to give an optimal solution because of its greedy nature. To improve the mask they derived in the earlier step, they introduced the so called *nonlocal pixel exchange*. It aims to solve the problem that the greediness of the probabilistic sparsification poses, more specifically that pixels that were removed once are not considered again at a later point in time even though they could possibly positively influence the quality of the final mask.
The nonlocal pixel exchange randomly selects a subset of non-mask pixels, from which the pixel with the largest reconstruction error is put in the mask in exchange for a random mask pixel. If the overall reconstruction quality improves with the new mask, the mask is updated to the new mask and serves as the initial mask for the next iteration step. If the quality deteriorates with the new mask, the change is undone and the procedure continued. The mask resulting from this process can only be better than the previous one by construction of the algorithm [14]. Finally, they proposed several algorithms for *tonal optimisation*, i.e. grey value optimisation by means of a linear least squares optimisation problem. They showed some important mathematical properties like the uniqueness and existence of a solution for this problem and came up with a numerical algorithm to solve the linear system of equations arising for this optimisation problem. Using all these optimisations, they were able to cut down the reconstruction error in terms of the mean squared error by a significant portion and achieve a quality that has not been reached before with similar PDE based

inpainting approaches for the same mask density.

2.2 Image features in image reconstruction

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[20]. 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[21], [22] 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[7]. 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[23]. 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 masks with respect to compression. 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.
The main point of criticism for this approach is that the inaccuracy of the corner detector was not accounted for in the selection of the mask pixels.[15] Their

method of limiting the amount of mask pixels by adjusting the integration scale leads to a fairly inaccurate localisation of the corners. The cause for this will be discussed in a later section. Since only a small neighbourhood of pixels around each estimated corner location was kept, the likelihood that the actual corner is not included in the final mask increases the larger the integration scale is chosen, as we will see in chapter 4, where I will also adress how to approach this problem and provide a possible solution.

In [8], [9], 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[24] combined with *hysteresis thresholding* as proposed by Canny[22]. 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[25]. 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[8]. 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 [26], 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.
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*[26]. 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.[26]

In [27], a method is presented that uses extrema and saddle points in the Gaussian scale space of an image as a basis for reconstruction using a linear reconstruction scheme proposed in [28].
In [29], the authors explore the reconstruction of an image using only its local descriptors. They use a classical image indexing software to retrieve descriptors like SIFT[30], then query an image database to find patches that are similar to these descriptors retrieved from the image. These patches are then transformed to fit the geometry in the original image and stitched together afterwards in a seamless fashion.

2.3 Outline

So far, we have reviewed some of the work that is closely related to my work and motivated the goal that I am tring to achieve. To explain what exactly was done in order to achieve this goal, we first have to go over some of the theoretical background. I will introduce basic concepts from calculus and linear algebra (3.1) as well as some more advanced topics like the structure tensor (3.2), diffusion processes (3.3) and digital image inpainting (3.4).
In chapter 4 I will then talk about the technical details of the implementation, for example the discretisation used to implement the theoretical ideas from the previous chapter (4.1). Furthermore, I will describe the shortcomings of the approach described in [7] in more detail and discuss a method to address these issues. In this chapter I will also present some additional procedures that I implemented in order to improve the selection of mask pixels.
Afterwards, I am going to review the experiments that were performed in order

to evaluate the approach and show some examples.
Last but not least in chapter 6, I conclude the work, discuss the strengths and shortcomings of my approach and how it could be improved.

3 Prerequisites

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 explaining 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 Partial derivatives

One of the most important operations on functions in image processing is *partial differentiation*. The partial derivative of an image $f : \Omega \to \mathbb{R}$ 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 at position (x, y) is the vector containing both partial image derivatives at this position. 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.

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

The gradient always points in the direction of the steepest ascent/descent, it is the tangent vector to the surface at the given location[31]. Furthermore, a vector that is *orthogonal* to the gradient always points into the direction of the

smallest absolute change at the given position.
Two vectors $x, y \in \mathbb{R}^n$ are called *orthogonal* to eachother if the euclidean dot product defined by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i \tag{3.3}$$

vanishes or in other words becomes zero. The geometric interpretation for this property is that the angle between both vectors is 90°.

3.1.2 Convolution

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

$$(f * g)(\boldsymbol{x}) := \int_{\mathbb{R}^2} f(\boldsymbol{x} - \boldsymbol{y}) g(\boldsymbol{y}) d\boldsymbol{y} \tag{3.4}$$

The *discrete* convolution of two vectors $(f_i)_{i \in \mathbb{Z}}, (g_i)_{i \in \mathbb{Z}}$ is usually defined as

$$(f * g)_i := \sum_{j \in \mathbb{Z}} f_{i-j} g_i \tag{3.5}$$

In both definitions, the function/vector g is normally called the *convolution kernel*. In praxis though, discrete convolution has to be used as dealing with continuous functions in a computational context is strictly not possible. This is why continuous convolution is only used to develop a theory, that is later on *discretised*, i.e. transformed to the digital domain.
Discrete convolution kernels are usually described in the so called *stencil notation*, for example, a convolution filter that computes the average of 3 pixels is defined by the stencil

$$\frac{1}{3} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline \end{array} \tag{3.6}$$

The cells in the stencil define the weights for the corresponding pixel relative to the center pixel. To visualise the correspondence, we can have a look at the following 2D stencil:

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

(3.7)

Convolution is useful for all kinds of operations on images as almost all operations can be expressed in terms of a stencil.[32] One example that I want to

present here, as it is important for the rest of the theory in my thesis, is *Gaussian convolution*, i.e. a convolution with a *Gaussian kernel*. It is also known as *Gaussian blur* or *Gaussian smoothing* and is widely used as a preprocessing step in the computation of image derivatives in order to reduce noise. The continuous Gaussian convolution kernel is basically just a Gaussian function, also known as *probability density function of a normal distribution*. In the 2-dimensional case it is defined as

$$K_\sigma(\boldsymbol{x}) := \frac{1}{2\pi\sigma^2} \exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2\sigma^2}\right) \tag{3.8}$$

where $\|\cdot\|_2$ denotes the *Euclidean norm*.

$$\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \tag{3.9}$$

For the rest of this thesis, an image f convolved with a Gaussian with standard deviation σ will be denoted by

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

Convolution inhibits some very important properties that I will not cover here, since they are not inherently necessary to understand the rest of the thesis. The interested reader can find more information about convolution in [33][32]

3.1.3 Error measures

The difference between two images is often measured as the sum of the individual pixel errors. One of the most widely used error measures of this kind is the *mean squared error(MSE)*. It is defined as

$$\mathrm{MSE}(f, g) = \frac{1}{nm} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} (f_{i,j} - g_{i,j})^2 \tag{3.10}$$

Obviously, two images are more similar to each other, the lower the mean squared error is. Another measure that is widely used and that will also be used in this thesis to evaluate the quality of an inpainted image, is the so called *peak signal to noise ratio (PSNR)*. It is usually defined in terms of the MSE:

$$\mathrm{PSNR}(f, g) = 10 \cdot \log_{10}\left(\frac{255^2}{\mathrm{MSE}(f, g)}\right) \tag{3.11}$$

Since the PSNR is defined in terms of the inverse of the MSE, we observe that

$$\lim_{\|f-g\|^2 \to 0} \mathrm{PSNR}(f,g) = \infty \tag{3.12}$$

So for the PSNR, a larger value is a better one unlike the MSE, where 0 is the best value one can possibly reach.

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[32]. 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 (see Section 3.2.2), one is able to distinguish between flat regions, edges and corners.

3.2.1 Definition

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 ∇u within a circle of radius $\rho > 0$, i.e. one wants to optimise the function

$$E(\mathbf{n}) = \int_{B_\rho(\boldsymbol{x})} \left(\mathbf{n}^\top \nabla u\right)^2 d\boldsymbol{x}' \tag{3.13}$$

$$= \mathbf{n}^\top \left(\int_{B_\rho(\boldsymbol{x})} \nabla u \nabla u^\top d\boldsymbol{x}' \right) \mathbf{n} \tag{3.14}$$

where

$$B_\rho(\mathbf{x}) = \left\{ \mathbf{y} \in \mathbb{R}^2 \mid \|\mathbf{x}-\mathbf{y}\|_2^2 \leq \rho^2 \right\} \tag{3.15}$$

is a *disk with radius ρ and origin* $\mathbf{x}$. This function is also called the *local autocorrelation function/local average contrast*[21], [32]. Since (3.14) is a quadratic form of the matrix

$$M_\rho(\nabla u) := \int_{B_\rho(\boldsymbol{x})} \nabla u \nabla u^\top d\boldsymbol{x}' \tag{3.16}$$

such an optimal unit vector is by definition also the eigenvector to the smallest/largest eigenvalue of $M_\rho(\nabla u)$[32]. The matrix $M_\rho(\nabla u)$ can also be seen as a component-wise convolution with the indicator function

$$b_\rho(\boldsymbol{x}) = \begin{cases} 1 & \|\boldsymbol{x}\|_2^2 \leq \rho^2 \\ 0 & \text{else} \end{cases} \tag{3.17}$$

However, as the author stated in [21], 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[32]. This ultimately leads to the definition

$$\mathbf{J}_\rho(\nabla u) := K_\rho * (\nabla u \nabla u^\top) \tag{3.18}$$

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[32]. The definition then becomes

$$\mathbf{J}_\rho(\nabla u_\sigma) := K_\rho * (\nabla u_\sigma \nabla u_\sigma^\top) \tag{3.19}$$

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 $\lambda_1, \lambda_2 \geq 0$. [32] 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 po-

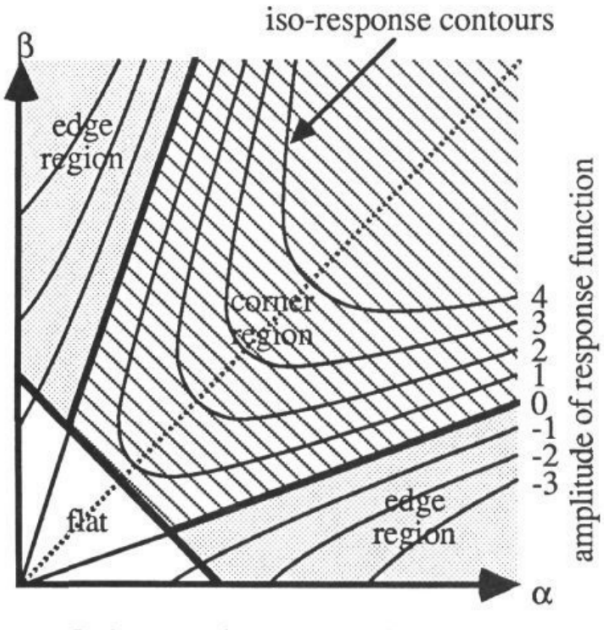

FIGURE 3.1: Visualisation of distinction of image features using the eigenvalues of the structure tensor. α, β are equivalent to the eigenvalues λ_1, λ_2. Source: [21]

sition. 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[34], 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[35]. However, this approach requires to compute both eigenvalues and can thus be fairly expensive.

A cheaper approach would be to either threshold the trace

$$\mathrm{tr}(\mathbf{J}_\rho) := j_{1,1} + j_{2,2} = \lambda_1 + \lambda_2 \tag{3.20}$$

as proposed by Rohr, 1991[36] or the determinant

$$\det(\mathbf{J}_\rho) := j_{1,1}j_{2,2} - j_{1,2}^2 = \lambda_1\lambda_2 \tag{3.21}$$

as proposed by Harris[21] and Förstner[37], 1988 and 1987 respectively. 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 (3.20) requires

$$\mathrm{tr}(\mathbf{J}_\rho) \tag{3.22}$$

by itself to be a local maximum whereas in (3.21)

$$\frac{\det(\mathbf{J}_\rho)}{\mathrm{tr}(\mathbf{J}_\rho)} \tag{3.23}$$

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:

$$T_t f := K_{\sqrt{2t}} * f \tag{3.24}$$

This operation spans the *Gaussian scale space*, one of if not the oldest and best understood scale space. In the western world, it was first mentioned by Witkin et al.[38] 1984, but as was later found out, researchers in Japan already proposed it in 1963[39].
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*

$$\nabla u(x, y, t) = \begin{pmatrix} \partial_x u(x, y, t) \\ \partial_y u(x, y, t) \end{pmatrix} \tag{3.25}$$

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*(3.26) (hence this type of diffusion is called *Fickian*

diffusion):

$$\boldsymbol{j} = -\boldsymbol{D} \cdot \nabla u \tag{3.26}$$

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

$$\partial_t u = -\text{div}(\boldsymbol{j}) \tag{3.27}$$

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

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

In simple terms, the divergence of the flux measures whether the concentration

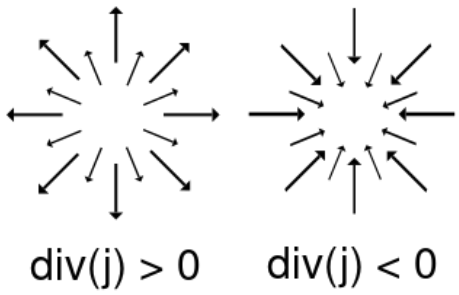

FIGURE 3.2: Visualisation of the divergence operator. Original image [40] edited with GIMP

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 (3.26) and (3.27) then form the general *diffusion equation*[41], [42]

$$\partial_t u = \text{div}(\boldsymbol{D} \cdot \nabla u) \tag{3.28}$$

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.

3.3.3 Isotropic diffusion

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*

$$\partial_t u = \Delta u = \operatorname{div}(\nabla u) \tag{3.29}$$

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.

$$\boldsymbol{D} = g(\|\nabla u\|_2^2)\boldsymbol{I} \tag{3.30}$$

$$\Rightarrow \partial_t u = \operatorname{div}(g(\|\nabla u\|_2^2)\nabla u) \tag{3.31}$$

The diffusivity function determines how much the image should be smoothed at the current location. The most well known choice for a diffusivity function

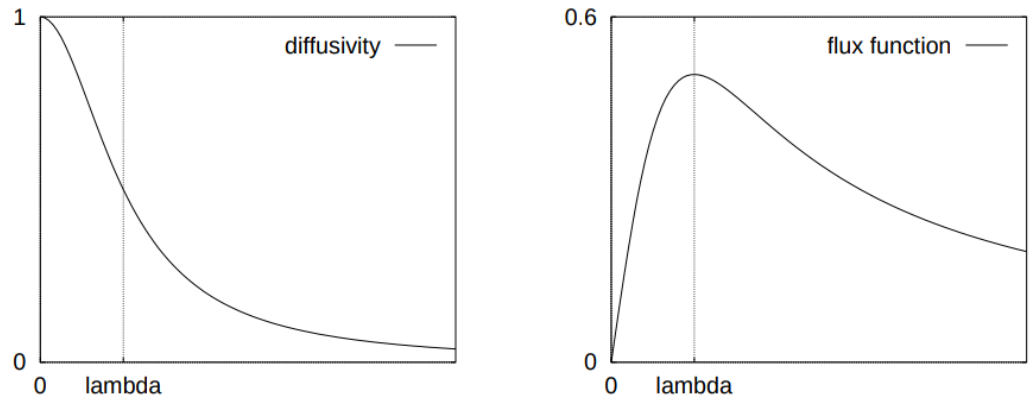

FIGURE 3.3: **Left:** Diffusivity mentioned in (3.32). **Right:** Flux function(3.34). Source: [41]

was proposed by Perona and Malik [43]

$$g(s^2) = \frac{1}{1 + s^2/\lambda^2} \tag{3.32}$$

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 $\|\nabla u\|_2^2$ as a *fuzzy edge detector*[41]. 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

$$\Phi(s) = sg(s^2) \tag{3.33}$$

which in this particular case comes down to

$$\Phi(s) = \frac{s}{1 + s^2/\lambda^2} \tag{3.34}$$

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:

$$\Phi'(s) = \frac{d}{ds}\left(\frac{s}{1 + s^2/\lambda^2}\right) = \frac{1 - s^2/\lambda^2}{(1 + s^2/\lambda^2)^2} \tag{3.35}$$

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 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*[42], 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[44].

3.3.4 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

$$\boldsymbol{D} = (\boldsymbol{v}_1 | \boldsymbol{v}_2) \operatorname{diag}(\lambda_1, \lambda_2) \begin{pmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \end{pmatrix} \tag{3.36}$$

We define the eigenvectors to be

$$\boldsymbol{v}_1 = \nabla u_\sigma \qquad \boldsymbol{v}_2 = \nabla u_\sigma^\perp \tag{3.37}$$

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.

$$\lambda_1 = g(\|\nabla u_\sigma\|_2^2) \qquad \lambda_2 = 1 \tag{3.38}$$

There are different viable choices for a diffusivity function that supports backwards diffusion. First, there is the already mentioned Perona-Malik diffusivity (3.32). Another option is the diffusivity function introduced by Weickert[42]

$$g(s^2) = \begin{cases} 1 & s^2 = 0 \\ 1 - \exp\left(\frac{-3.31488}{(s/\lambda)^8}\right) & s^2 > 0 \end{cases} \tag{3.39}$$

According to [41], 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* [45]

$$g(s^2) = \frac{1}{\sqrt{1 + s^2/\lambda^2}} \tag{3.40}$$

Overall, we can write the EED equation as

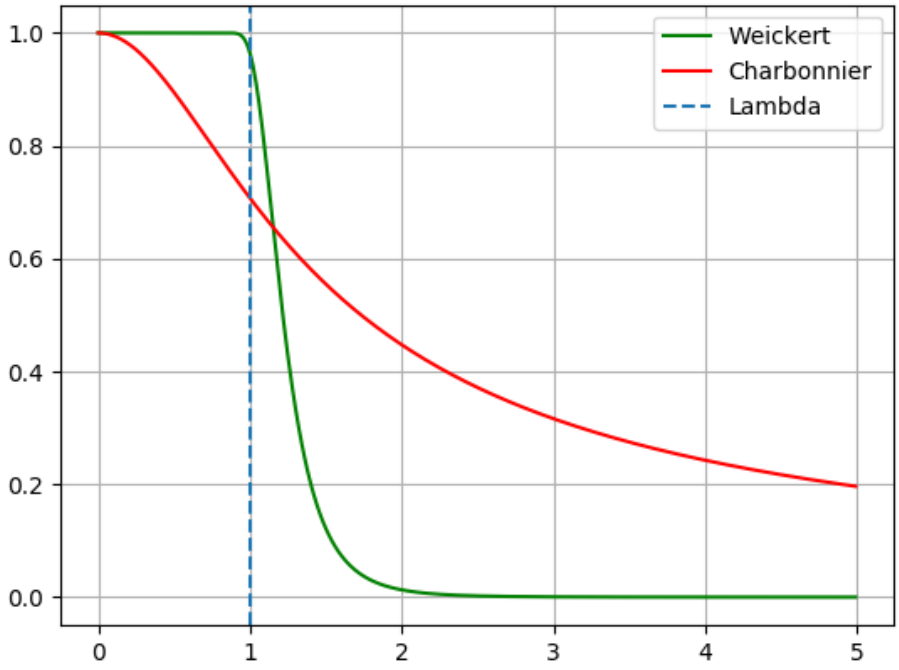

FIGURE 3.4: Weickert (3.39) and Charbonnier (3.40) diffusivities for $\lambda = 1$. Graph created with Matplotlib

$$\partial_t u = \operatorname{div}(g(\nabla u_\sigma \nabla u_\sigma^\perp)\nabla u) \tag{3.41}$$

As we see in figure 3.5, EED possesses great restorative and denoising capabilities. But as already mentioned in Related Work (2), edge-enhancing diffusion is not only suited for image denoising[3], [42], but also one of the best methods for inpainting based on partial differential equations[4], [11], [13].

3.4 Inpainting

Inpainting itself originated as a technique used by image restorators and conservators and has been in use for many decades, centuries even. The main objective of inpainting is to restore damaged regions in an image or even filling

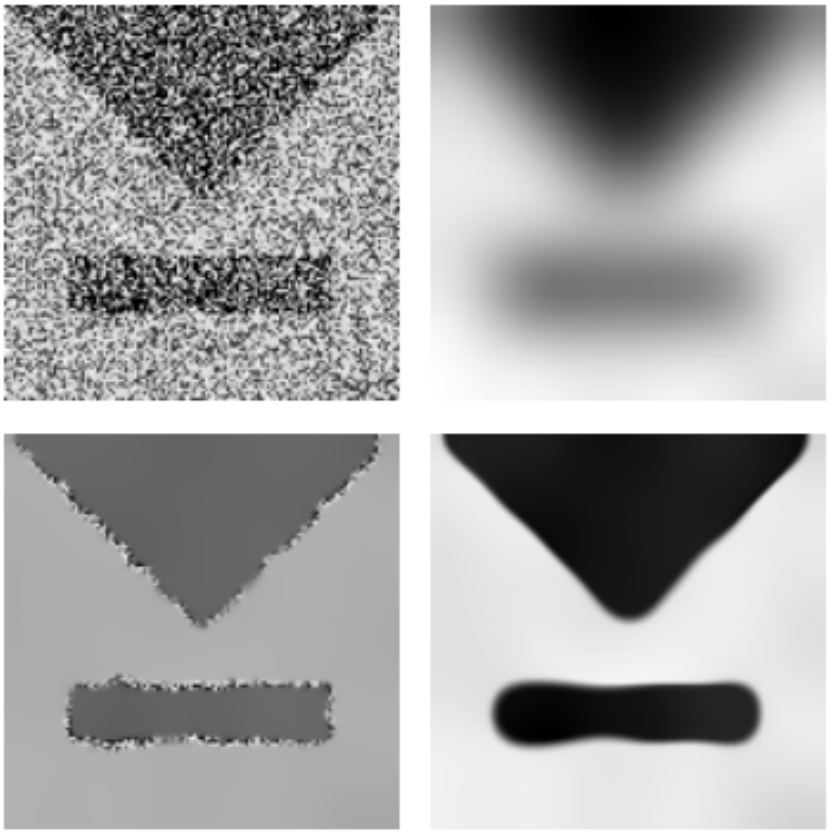

FIGURE 3.5: Denoising capabilities of the different types of diffusion. **Top Left:** Original image. **Top Right:** Linear isotropic diffusion. **Bottom Left:** Nonlinear isotropic diffusion. **Bottom Right:** Edge-enhancing diffusion. Source: [42]

in areas that are missing completely. In the analog or physical world, inpainting is often done in multiple steps as described in [23]:
First, the lines defining the structure surrounding the blank area are continued inside the area. The smaller areas that result by drawing in the structure lines are then filled in with colours that match the colours outside the blank area. Lastly, texture and small details are added.

Since this obviously cannot be applied to digital images, people had to come up with a way to translate this process to the digital domain.

3.4.1 The origin of digital inpainting

In 2000, a first method to digitize this technique was proposed by Bertalmio et al. [23]. The idea behind their algorithm was to iteratively continue so called *isophote lines*, i.e. lines of equal brightness, inside the missing areas. This was

implemented by an image evolution described by

$$\partial_t u = \nabla_{\mathbf{n}} L(u) \tag{3.42}$$

where $L(u) = \text{div}(\nabla u) = u_{xx} + u_{yy}$ and ∇_n is the directional derivative in the direction n. Note that in this particular case, the evolution is only performed in the inpainting regions. The regions of the image that are already known, are left unprocessed.
The direction n is given by the vector

$$n = (\nabla u)^{\perp} = \begin{pmatrix} u_y \\ -u_x \end{pmatrix} \tag{3.43}$$

which is orthogonal to the image gradient ∇u as defined in 3.2. The formula can also be written as

$$\partial_t u = \langle \nabla L(u), \nabla u^{\perp} \rangle \tag{3.44}$$

where $\langle \cdot, \cdot \rangle$ is the euclidean inner product.
The evolution given by this formula basically propagates information from outside of the blank area along the direction of isophote lines arriving at the boundary of this area, conserving the angle of arrival between the boundary and the isophote line.[23] This direction is given by a vector orthogonal to the gradient, i.e. the direction of the largest grey value change. Why this is the case becomes more obvious if we restate this to say that a vector orthogonal the gradient determines the direction of the *smallest* grey value change, which ultimately is the direction of a line of equal brightness.
This propagation is then complemented by a (*anisotropic*) diffusion process that is performed regularly to *periodically curve the lines to keep them from crossing each other*[23]. By alternatingly applying the prolongation and diffusion process, the inpainting regions are shrinked progressively until they vanish as can be seen in 3.6.

FIGURE 3.6: Results of the inpainting process at different stages showing the progressive shrinking of the inpainting regions. Source: [23]

3.4.2 PDE-based inpainting

The general idea behind inpainting methods based on partial differential equations is to retrieve an image in the form of an unknown function $v : \Omega \to \mathbb{R}$ as described in [3]. Let $\Omega_1 \subset \Omega$ be the subset of locations where the values of v are known from the original image. The *inpainting domain*, that is, the locations where we want to reconstruct the image, is then given as the set $\Omega \backslash \Omega_1$. The goal is now to find a sufficiently often continuously differentiable function $u : \Omega \to \mathbb{R}$ that satisfies

$$u(x) = v(x) \qquad \forall x \in \Omega_1 \tag{3.45}$$

$$|u(x) - v(x)| < \epsilon \qquad \forall x \in \Omega \backslash \Omega_1 \tag{3.46}$$

for some $\epsilon > 0$.
The idea to find such a function is to specify an image evolution which yields the desired function as its *steady state*, i.e. a state where $\partial_t u = 0$. More intuitively steady state means that the function does not evolve any further with increasing t once it has reached this state.
Generally, the evolution for a PDE-based inpainting problem is given by

$$\partial_t u(x,t) = (1 - c(x))Lu(x,t) - c(x)(u(x,t) - f(x)) \tag{3.47}$$

where

$$c(x) := \begin{cases} 1 & x \in \Omega_1 \\ 0 & \text{elsewhere} \end{cases} \tag{3.48}$$

is the characteristic function of the subset Ω_1 and $f : \Omega \to \mathbb{R}$ is the initial state of the evolution given by

$$f(x) := \begin{cases} v(x) & x \in \Omega_1 \\ 0 & \text{elsewhere} \end{cases} \tag{3.49}$$

On the boundary $\partial\Omega$ of the image domain, homogeneous Neumann boundary conditions are applied, which can be expressed mathematically as

$$\partial_{\mathbf{n}} u = \langle \mathbf{n}, \nabla u \rangle = 0$$

where $\mathbf{n}$ is the normal vector to the image boundary. The so called *steady state equation*

$$(1 - c(x))Lu(x,t) - c(x)(u(x,t) - f(x)) = 0 \tag{3.50}$$

determines the behaviour of the evolution as $t \to \infty$ and therefore defines the desired function u. Namely, for $x \in \Omega_1$, (3.50) implies that

$$u(x,t) = f(x)$$

by using $c(x) = 1$ and similarly we get for $x \in \Omega \backslash \Omega_1$, that $c(x) = 0$ and thus

$$Lu(x,t) = 0$$

Using this, one can rewrite the original equation (3.47) in a simpler way by introducing *Dirichlet boundary conditions* given by the subset Ω_1 of known values on the inpainting domain and solving the evolution equation

$$\partial_t u(x,t) = Lu(x,t) \tag{3.51}$$

The choice of the differential operator L is still an active topic of research, but it was shown in [13], that edge-enhancing diffusion (4.14) is the best choice among those tested. The authors' explanation of the 'favourable' performance of this operator in the context of inpainting is, that the inherent anisotropic behaviour of the operator allows it to create sharp edges while still being able to propagate directional information to outside of the known area. They explain this ability to propagate information by the Gaussian smoothing that is found in the definition of the diffusion tensor (3.36). Other candidates that were being tested in [13] include the *biharmonic and triharmonic operator* respectively defined as

$$Lu = -\Delta^2 u$$
$$Lu = \Delta^3 u$$

a nonlinear isotropic diffusion operator as described in (3.31) and simple homogeneous diffusion (3.29). In figure 3.7, (f) and (e) refer to the nonlinear isotropic diffusion defined in (3.31) and the regularised version also mentioned in 3.3.3.

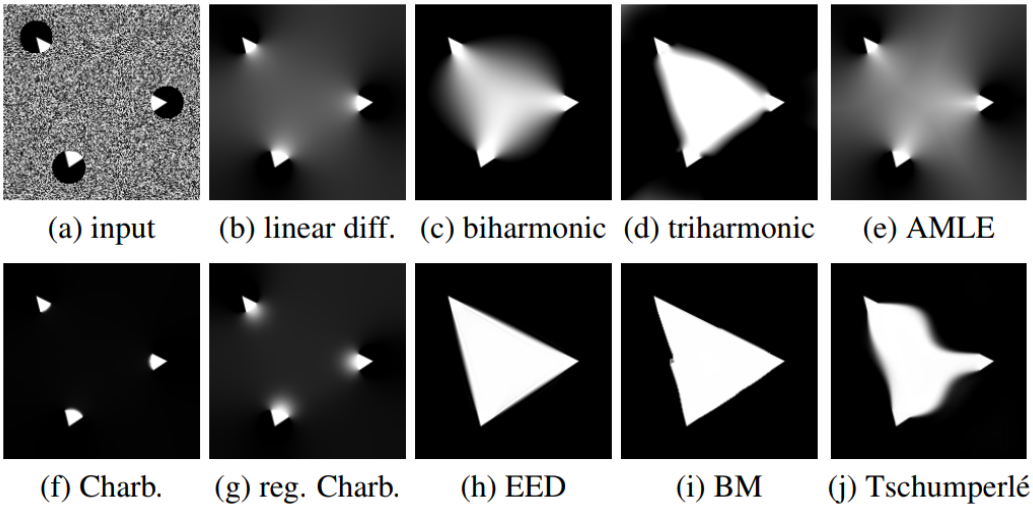

FIGURE 3.7: Comparison of different differential operators for PDE-based inpainting. Source:[13]

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}, i.e.

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

[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:

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

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

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

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

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

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

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

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

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

Expanding and simplifying yields

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

$$\Rightarrow \frac{f_{i+1}-f_{i-1}}{2h} = f_i' + \mathcal{O}(h^2) \tag{4.13}$$

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*. Note 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

As discretisation for the inpainting process, the non-standard discretisation scheme that was derived in [46] was used. The actual derivation of the scheme is fairly complex so I will just explain the basics and then refer the interested reader to [46] and [42] for more information on numerical schemes for diffusion processes in general.

We remember the defining equation of EED:

$$\partial_t u = \operatorname{div}(\underbrace{g(\nabla u_\sigma \nabla u_\sigma^\top)}_{=:D}\nabla u) \tag{4.14}$$

To discretise this equation, we first write the diffusion tensor D as a symmetric matrix

$$D = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \tag{4.15}$$

Resubstituting this into (4.14), we get

$$\partial_t u = \operatorname{div} \begin{pmatrix} a \cdot \partial_x u + b \cdot \partial_y u \\ b \cdot \partial_x u + c \cdot \partial_y u \end{pmatrix} = a \cdot \partial_{xx} u + 2b \cdot \partial_{xy} u + c \cdot \partial_{yy} u \tag{4.16}$$

These second order derivatives can easily be discretised using central differences. This leads to a fairly complex stencil that the interested reader can look up in [42]. The problem with this discretisation is that it is not guaranteed to satisfy the *nonnegativity requirement* or *stability in the maximum norm*[42]. This requirement is an important theoretical requirement as it is needed to guarantee the well-posedness of the diffusion process, which in turn guarantees the numerical stability of the algorithm. [42] To enable this discretisation to fulfill this requirement, one has to impose the restriction that the *spectral condition number* κ, which is the ratio between the smallest and largest eigenvalue of the matrix D, is bounded by

$$\kappa \leq 3 + 2\sqrt{2} \tag{4.17}$$

This restriction might seem arbitrary, but the proof of this upper bound can be found in Weickert's book about anisotropic diffusion[42]. However, this requirement also puts a restriction on the anisotropy of the diffusion process as the ratio between the largest and smallest eigenvalue of the diffusion tensor is now bounded from above, which is why one might consider another discretisation, that is not necessarily stable in the maximum norm. In [46], a whole framework for discretisations stable in the *euclidean or 2-norm* was derived. This framework embeds a series of 8 different stencils that are dependent on 2 parameters α, β. Let us now denote such a spatial discretisation of (4.14) by

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}(\mathbf{u})\mathbf{u} \tag{4.18}$$

It can be shown that the matrix $\mathbf{A}$ is negative semidefinite, i.e.

$$x^\top \mathbf{A} x \leq 0$$

for all vectors x, if the parameters α, β satisfy

$$0 \leq \alpha \leq \frac{1}{2} \tag{4.19}$$

$$|\beta| \leq 1 - 2\alpha \tag{4.20}$$

The negative semidefiniteness of $\mathbf{A}$ is useful to show that the semi-implicit and explicit time discretisations both are stable in the 2-norm. The difference between the explicit and the implicit time discretisation is fairly small, but makes a huge difference in efficiency **!!!!sources!!!!**. Both time discretisations work by applying a forward difference on the temporal derivative on the left side with a time step size τ (this corresponds to the grid size in the spatial derivative). The difference is now, that in the explicit scheme, the right hand side does not involve any terms 'from the future', while in the semi-implicit scheme it does. We can now write the explicit scheme given by

$$\frac{u^{k+1} - u^k}{\tau} = \mathbf{A}(u^k)u^k \tag{4.21}$$

as

$$u^{k+1} = (I + \tau\mathbf{A}(u^k))u^k \tag{4.22}$$

The upper index k denotes the 'time level' of u. Stability for this scheme can be guaranteed, if

$$\rho(I + \tau\mathbf{A}(u^k)) \leq 1 \;\Leftrightarrow\; \tau \leq \frac{2}{\rho(\mathbf{A}(u^k)))} \tag{4.23}$$

where ρ denotes the *spectral norm* or *modulus of the largest eigenvalue.*[46] The semi-implicit scheme however is given by

$$\frac{u^{k+1} - u^k}{\tau} = \mathbf{A}(u^k)u^{k+1} \tag{4.24}$$

which boils down to solving the linear system of equations

$$(I - \mathbf{A}(u^k))u^{k+1} = u^k \tag{4.25}$$

Note that the matrix on the left hand side is invertible thanks to the negative semidefiniteness of $\mathbf{A}(u^k)$ since one can show, that because of the negative semidefiniteness, $(I - \mathbf{A}(u^k))$ only has eigenvalues larger than 1. This all shows that the semi-implicit scheme given by

$$u^{k+1} = (I - \mathbf{A}(u^k))^{-1}u^k \tag{4.26}$$

is stable in the 2-norm for every time step size τ.

Furthemore, the semi-implicit scheme the next iteration can not be computed explicitly (as the name suggests) as is the case with the explicit scheme. Instead one has to solve the linear system of equations given by (4.25). This is usually done with an iterative solver like Gauss-Seidel or conjugate gradients(CG) [47]. I will not explain these solvers as this is way beyond the scope of this thesis. The only thing important to know, is that for the evaluation of the inpainting masks in my thesis, I used a semi-implicit scheme with a conjugate gradients solver and a timestep size of 1000.

4.2 Circular corner regions

As mentioned in the beginning, the main concern for the approach of Zimmer[7] was that the corner localisation is not accurate enough that we can just keep a 4 or 8-neighbourhood of pixels. The implication is that the regions they inserted into the mask were were too small to cover the potential uncertainty in the corner localisation process stemming from the Gaussian smoothing in the computation of the structure tensor (see 3.2). Specifically because in order to limit the amount of corners detected, they chose to vary the integration scale parameter increasing the uncertainty in the localisation, this poses a problem. That is why I want to explore increasing the neighbourhood of pixels that is kept around each corner. A disk-shaped neighbourhood with radius r and origin point (x, y) (also called a *(circular) corner region* in the following) is defined as

$$B_r(x, y) = \{(x', y') \in \Omega \mid (x' - x)^2 + (y' - y)^2 \leq r^2\} \tag{4.27}$$

The optimal choice for the radius of the corner disks is discussed in section **??**. Increasing the radius of the corner regions obviously leads to denser masks if one does not adjust the limit of corner regions that are put in the mask. Since doing this manually for every picture would not be practical at all in the context of a codec, we have to come up with a way to make sure the masks that are produced are always similar in pixel density.
One way to achieve this would be to introduce a percentile parameter instead of using the constant threshold introduced in the corner detection method described in 3.2.2.

4.2.1 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, a percentile parameter is introduced 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. I 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[48].
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 percent of all values are zero already. To solve this, I 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.
While this approach works a bit better than the original version using the fixed threshold, it is still not perfect. For example, if one uses a percentile of 0.5, meaning that we want to throw out 50% of all corners, we could still get wildly different results for two different input images as we can see in 4.2.
To tackle this issue, I propose a new method I call *Total Pixel Percentage Thresholding*. The idea behind this method is, that one makes the amount of corners that are kept in the masks dependent on the mask radius, i.e. the radius of a corner region in the inpainting mask. In other words, one specifies the desired mask pixel density and depending on the mask radius, the maximum number of corners that can be inserted into the mask is calculated. The problem of calculating the number of pixels inside a disc of radius r is also known as *Gauss' circle problem*[49]. The exact solution to this problem is given by[50]

$$N(r) = 1 + 4 \sum_{i=0}^{\infty} \left(\left\lfloor \frac{r^2}{4i+1} \right\rfloor - \left\lfloor \frac{r^2}{4i+3} \right\rfloor \right) \tag{4.28}$$

but since this is an infinite sum, the exact solution can not be computed this

way. This is why I chose the naive way by just iterating through a loop and count the numbers inside the disk as seen in 4.3. I could have also chosen to use an upper bound and estimate the number of pixels this way, but this would not have been as accurate.

After computing the area per corner region we can simple calculate how many regions can be inserted into the mask without exceeding the pixel density threshold.

$$N_{corners} = \left\lceil \frac{q \cdot N_x N_y}{N_{disc}} \right\rceil, \qquad 0 \leq q \leq 1 \tag{4.29}$$

where q is the percentile parameter, N_x, N_y the number of pixels in the respective direction and N_{disc} the amount of pixels per corner region. $N_{corners}$ is then an upper bound on the amount of corners. Using this method, the pixel density of inpainting masks can be consistently recreated, as we will see in the next chapter where I will present some examples of this method in action.
Note that I did not consider the potential overlap of corner regions inside the computation of this upper bound, since there would have been no easy way to determine the amount of overlapping pixels a priori. Instead, I introduced another method tackling the issue of potentially overlapping corner regions called *Circular non-maximum suppression*. We will see how this method works in the next section 4.2.2. Another thing to mention is that the amount of pixels in the mask using this approach is limited by the number of corners detected since I do not even consider pixels that have a sufficiently small cornerness value as mask candidates.

4.2.2 Circular non-maximum suppression

As mentioned previously, overlapping corner regions tend to worsen the estimation of the mask pixel density. Furthermore, as Zimmer already stated in [7] the sparsity of corners in images poses a problem for the reconstruction quality. This method is supposed to tackle both of these issues by increasing the radius of the local maximum search during the corner detection. In the original corner detection method, corners are determined by the local maxima inside an 8-neighbourhood of the cornerness measure. The idea is to adapt the radius of the neighbourhood for the local maximum search to the mask radius. As experiments showed (see section 5.3), this minimises the overlap in corner regions in the final mask and enables a fairly accurate estimation of the mask pixel density allowing us to consistently produce masks with similar density.
The other issue that this method tries to solve is the uneven distribution of corners across the whole image.

When limiting the amount of corners that are selected into the mask, one should not only care about the 'cornerness' of a corner, but also about its position. In other words, if a lot of good corners are in close proximity to each other, it would be better to choose the ones that encase the most good corners around them. That way, we get multiple corners 'for the price of one' and can still introduce some of the 'worse' corners to the mask. By minimising the overlap between corner regions through this circular non-maximum suppression, we can actually eliminate corners that are already covered by another disk from 'a better' corner and thus free up some slots for corners that may otherwise not have been selected for being sub-par.
One should note that this is not a perfect solution as certain constellations of corners create a corner case in which a corner that is assumed to cover another corner is removed for being in the disk of a better corner. This leads to the case where the corner that was formerly assumed to be covered is suddenly not covered anymore and disappears from the mask without replacement, so future work might be invested into finding a better solution/implementation for this approach.

```
1
2   /* Flatten cornerness map and prepare for sorting */
3   long idx = 0;
4   for (i = 1; i <= nx; ++i) {
5     for (j = 1; j <= ny; ++j) {
6       if (u[i][j] > 0.01) {
7         vals[idx++] = (pixel_t){i, j, u[i][j]};
8   qsort(vals, idx, sizeof(pixel_t), pixel_cmp);
9
10  /* Initialize */
11  for (i = 1; i <= nx; i++) {
12    for (j = 1; j <= ny; j++) {
13      u[i][j] = 0;
14    }
15  }
16
17  /* Keep best corners */
18  long nums = 0;
19  for (i = idx - 1; i >= MAX(idx - n_corners, 0); --i) {
20    long x = vals[i].i;
21    long y = vals[i].j;
22    u[x][y] = 255.0f;
23    nums++;
24  }
25
26  printf("Inserted %ld corners\n", nums);
27 }
28
29 /**
30  * \brief Computes the n-th percentile given by the parameter
      perc on a
31  * cornerness map
32  *
33  * Compute the n-th percentile on the given cornerness map and
      threshold it
34  * against the threshold computed this way. Afterwards, u is 0
      at locations
35  * that were below the threshold and 255 for locations above the
       threshold.
36  *
```

FIGURE 4.1: Total pixel percentage thresholding

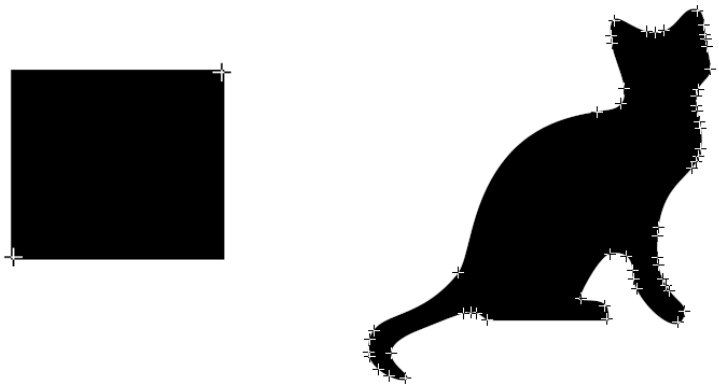

FIGURE 4.2: Corner detection with the same parameters

```
1
2 /**
3  * \brief Check if (x,y) is in a circle of a given radius with
     centre
4  * (cx, cy)
5  *
6  * Returns 1 if (x-cx)^2 + (y-cy)^2 <= radius^2, 0 else
7  *
8  * @param x: x coordinate
9  * @param y: y coordinate
10 * @param cx: x coordinate of the centre of the circle
11 * @param cy: y coordinate of the centre of the circle
12 * @param radius: radius of the circle
```

FIGURE 4.3: Computation of number of pixels in disk

```
1 # Circular non-maximum suppression at x with radius r.
2 # Parameters:
3 #     -harris: Harris measure map
4 #     -x: current location as tuple
5 #     -r: radius of corner regions
6 #     -out: map of accepted corners after suppression
7 def circular_suppression(harris, x, r, out):
8     for y in circle(x, r):
9         if harris[y] > harris[x]:
10             # The current location is not a maximum
11             out[x] = 0
12             return
13     # Current location is a maximum, keep the region around it
14     out[x] = harris[x]
15
```

FIGURE 4.4: Pseudo-Code for circular non-maximum suppression

5 Experiments

In this chapter, I will present the experiments conducted to test and verify the methods I introduced as well as figure out what the best parameters for both corner detection and inpainting are.

5.1 Parameter Selection

Both the corner detection process and the inpainting process introduce many parameters that need to be chosen carefully to get the best results. The focus in this thesis lies mainly on examining the choice of parameters for the corner localisation. While we will still explore how the parameters in the reconstruction step influence the quality of the end result, the interesting thing will be to see what role the mask radius plays in this context.

5.1.1 Corner Detection

In the corner detection phase, we have to deal with 4 parameters:

- the noise scale σ
- the integration scale ρ
- the percentile parameter p and
- the mask radius (and also radius for circular non-maximum suppression) R

Depending on whether one chooses to use the circular non-maximum suppression (CNMS) and the total pixel percentage thresholding (TPPT), the mask radius and percentile parameter will have different or additional meaning. The percentile parameter determines either the percentage of corners (without TPPT) to keep or the maximum bound on the pixel density in the inpainting mask (with TPPT). The mask radius also serves as the radius for the neighbourhood in the CNMS procedure.

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 (see 3.3.1). Another problem is that the gaussian scale space (iterated gaussian smoothing) may even introduce new corners.[42] 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.
The integration scale ρ is an interesting parameter as it basically determines how well the corner can be localised. Here, one has to be careful as the choice of this parameter is somewhat dependent on the noise scale in the sense that it (the integration scale) always should be at least as large as the noise scale. On the other hand, by increasing the integration scale, we also increase the inaccuracy in the corner localisation as we see in figure 5.1. This demonstrates

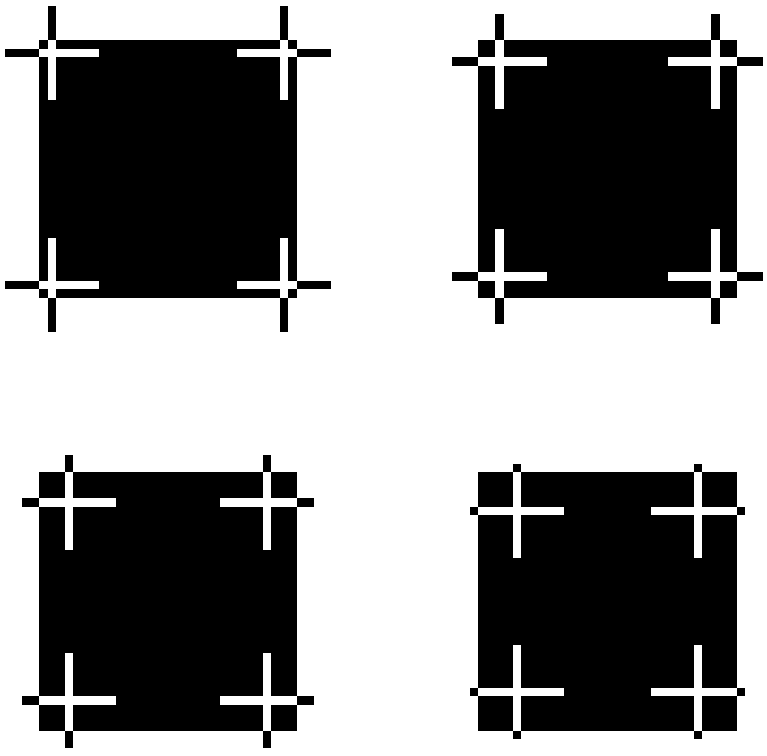

FIGURE 5.1: Location of corners with different values for ρ.
Top left: $\rho = 2$, **Top right:** $\rho = 4$, **Bottom left:** $\rho = 6$,
Bottom right $\rho = 8$
. Original image: rect_tiny.pgm (50x50), created with GIMP

the issue mentioned in chapter 2 with Zimmer's approach [7] quite nicely. We can see that with increasing integration scale, the inaccuracy increases steadily with it. The problem with Zimmer's approach is that they did not adress these inaccuracies in their mask generation. They fixed the mask radius to be just a small neighbourhood of 4-8 pixels. To adjust the amount of corners introduced into the mask, they adapted the integration scale. Obviously this introduces the aforementioned inaccuracies and combined with the fixed neighbourhood size, can lead to the actual corner not even being included in the final mask. (see figure 5.2)

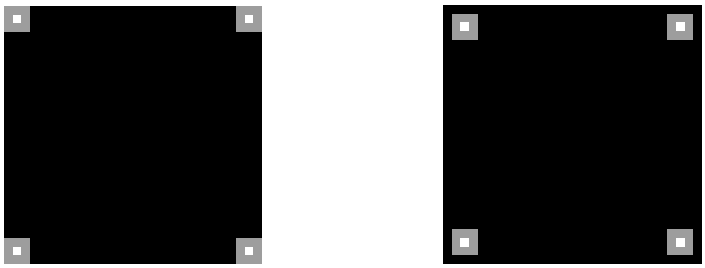

FIGURE 5.2: Position of corner regions for examples from figure 4.3. Position of the corner in white. Size of the corner region: 9px (8-neighbourhood) **Left:** $\rho = 2$, **Right:** $\rho = 4$ (Corner regions in grey for visualisation purposes, not the actual mask)

We will discuss the optimal choice for the mask radius in the next section (5.2), but first I will show some examples demonstrating the ideas behind the methods introduced in Section 4.2.1 and 4.2.2. As we can see in figure 5.3, CNMS allows us to distribute the corner regions in the mask more evenly across the whole image. While in the example without CNMS, most of the corners are overlapping and grouped up in the lower half of the image, using CNMS, the overlapping corners are removed and free up slots for corners with a lower cornerness measure, i.e. less sharp corners in the upper half. Later in Figure **??**, we can also observe that thanks to CNMS a few extra slots are freed up to be used in other regions.

In Figure 5.5, we can also see that TPPT actually produces masks of a similar size, bound by the percentage given by the parameter p. What I also noticed is that the percentage is not as accurate when using a normal non-maximum suppression approach. This is due to the potential overlap that is not accounted for in the initial estimation of the amount of corner regions that

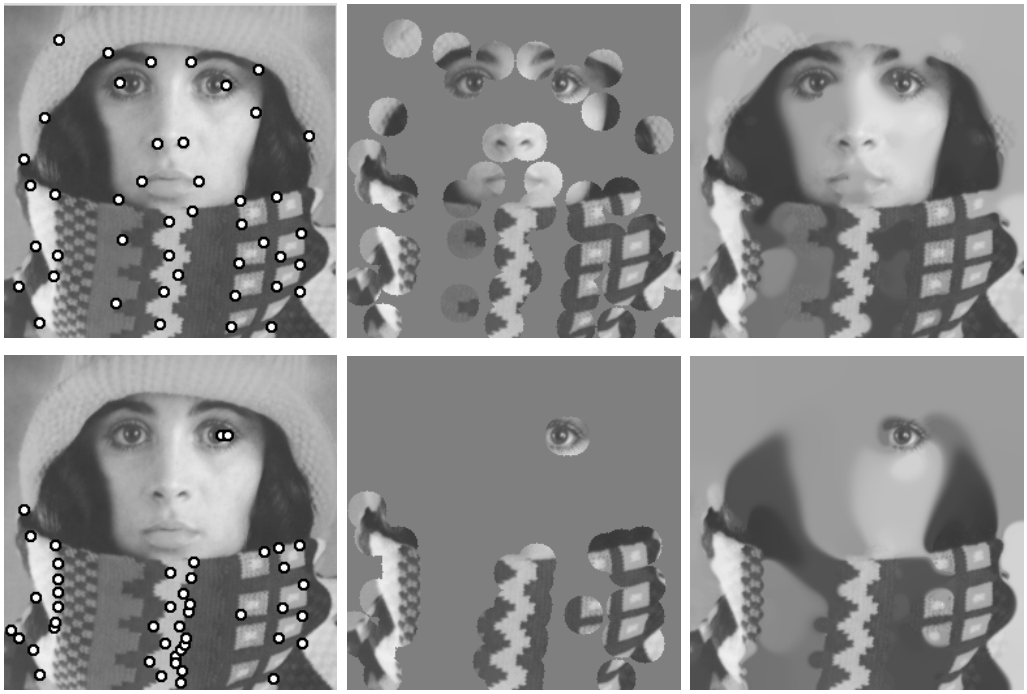

FIGURE 5.3: Effect of circular non maximum suppression on spread of corners across the image. **Top row:** Position of corners, inpainting mask and inpainting results **with** CNMS. **Bottom row:** Position of corners, inpainting mask and inpainting results **without** CNMS. Corner detection with Foerstner-Harris corner detector, $\sigma = 1, \rho = 2.5, R = 15$ and a percentile of 0.5

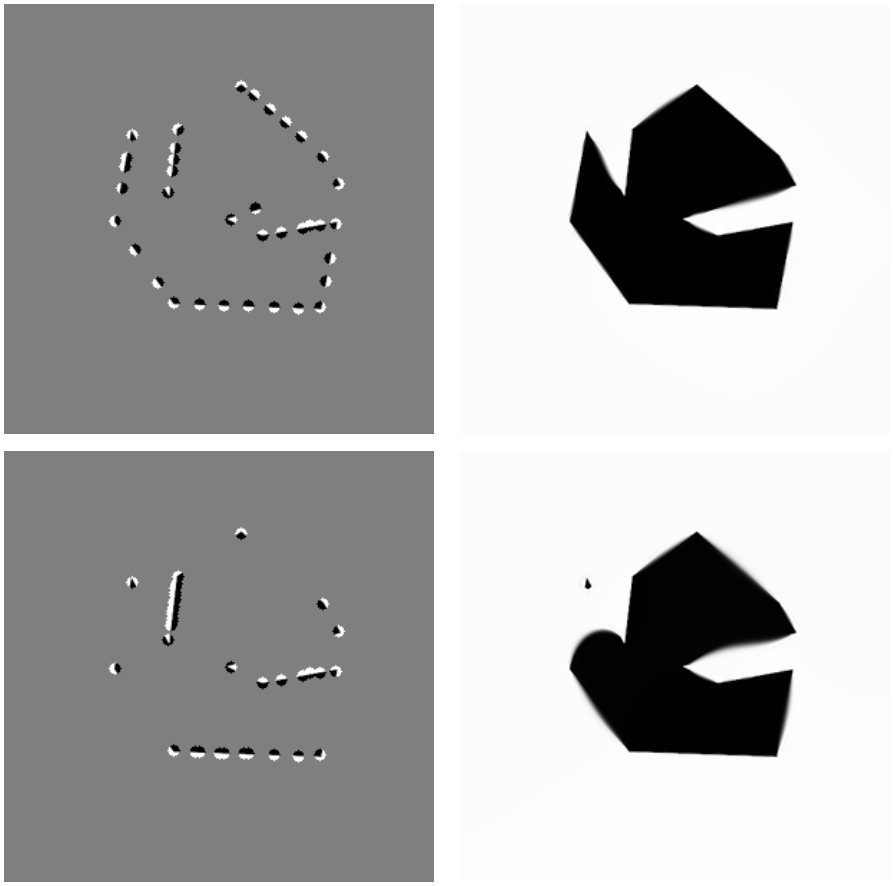

FIGURE 5.4: Demonstration of the effects of CNMS on the spread of corner regions. **Top row:** $\sigma = 1, \rho = 1, R = 4, q = 0.02$, with CNMS, Pixel density: 1.96%, PSNR: 31.45 **Bottom row:** $\sigma = 1, \rho = 1, R = 4, q = 0.02$, without CNMS, Pixel density: 1.51%, PSNR: 21.12 Inpainting parameters: $\sigma = 2, \lambda = 0.1, \alpha = 0.49, \gamma = 1, N = 1000$ Original image: abstract1_small.pgm

can be introduced into the mask. The TPPT approach is not perfect however. It does not always yield accurate masks since for one reason, the maximum possible pixel density depends on the amount of corners detected. On the other hand with increasing mask radius it is also harder to get an accurate pixel density. Some possible remedies would include having variable mask radii, inserting random pixels after to fill up the remaining percentage or just, instead of using the parameter as an upper bound, minimising the squared error (allow overshoots).

5.1.2 Inpainting

As already mentioned, we are using an EED-based inpainting algorithm with a Charbonnier diffusivity as explained in section 3.4. The main parameters

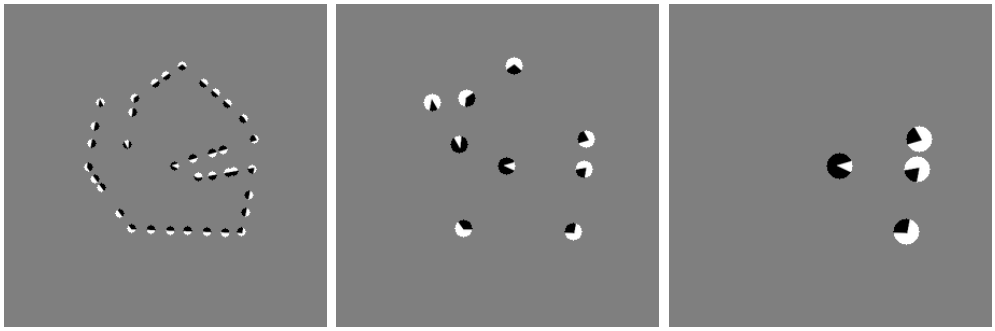

FIGURE 5.5: Effect of TPPT. Upper bound on the pixel density: 2%. Actual values: 1.95%, 1.97%, 1.96%

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.

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. In this work I fixed these parameters to $\alpha = 0.49$, $\gamma = 1.0$[15]. 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 [42], [46].
More interesting is the contrast parameter γ that is required in the diffusivity function (3.32). As already explained in section 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. The caveat with this parameter is, that even though a smaller λ (≤ 0.1) yields very sharp results, especially for binary images, it increases the chance of the algorithm becoming unstable. Normally a $\lambda < 0.1$ should not be used[15]. Since the optimal choice for this parameter

is still a topic of current research, I had to experiment a little to find the best choice. Furthermore it was suggested that the optimal λ depends on the input image [13]. The noise scale σ was always chosen to be somewhere between 1 and 4 as this seemed to give the best results.

5.2 Mask radius

By introducing the mask radius as an additional parameter, we now have to pose the question of what radius would yield the best result. One hypothesis[15] was that it would be best to compute the the mask radius in dependence of the integration scale ρ in order to account for the inaccuracies mentioned in Section 5.1.1.

The hypothesis was tested by switching out the Gaussian convolution in the computation of the structure tensor in the corner detection process for a regular convolution with a so called *normalised pillbox kernel*, which is basically just averaging inside a disk-shaped neighbourhood, to get a better representation of the uncertainty in the corner localisation[15]. After that, it was tested whether it would be sensible in terms of reconstruction quality to choose a mask radius equal to the radius of the pillbox kernel[15]. In Figure 5.6 we can see the motivation behind the idea. There we can see how the actual corner slowly moves out of the corner region for increasing ρ. I should not that the corner locations and thus the masks and inpainting results for $\rho = 1, 2, 3$ are the same. The interesting thing however is that the quality suddenly drops when we have

$$\frac{R}{\rho} \leq 1 \tag{5.1}$$

This observation motivated the next experiment in which I tested different integration scales paired with multiple mask radii to create the matrix we can see in Figure 5.7. Looking at the resulting inpainted images (see Figure 5.8) and their respective PSNR values (see Figure 5.9) we observe a similar effect, namely that the PSNR jumps by a large margin every time the ratio between the mask radius and the integration scale goes beyond 1. Another interesting observation is that the biggest jump is almost always at a transition from a ratio smaller than 1 to a ratio of 1 (cells in grey), suggesting that as soon as the inaccuracy is matched by the mask radius, the inpainting quality increases. The only exception here is $\rho = 3$. I can only explain this by stating, that in this case, the corner location is the same for both $\rho = 2$ and $\rho = 3$, which is why all

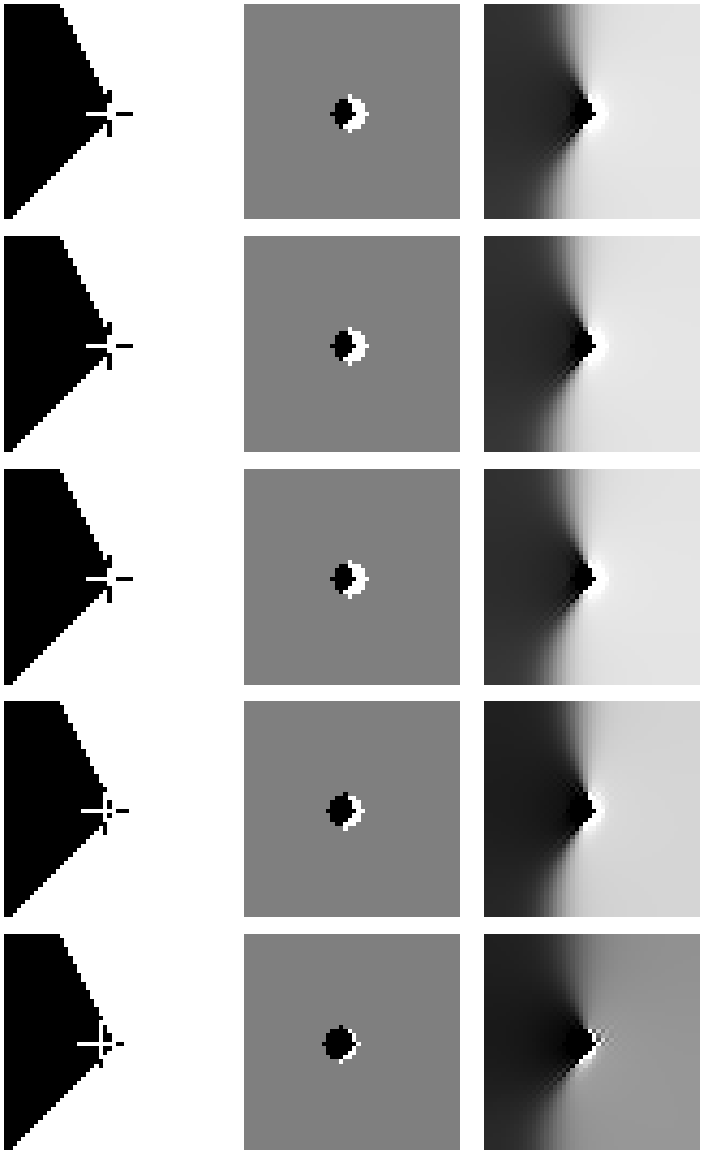

FIGURE 5.6: Experiment with identical mask radius $R = 4$ and varying integration scale. ρ from top to bottom: 1, 2, 3, 4, 5. PSNR from top to bottom: 12.49, 12.49, 12.49, 11.31, 8.09

the inpainted images and masks are identical. Similar results can be observed in Figure 5.12.

All in all we can say that the experiments provided reasonable proof that the hypothesis stated above is true. As a corollary from this we can deduce that a more accurate corner detection method could result in smaller mask sizes and ultimately better compression rates.

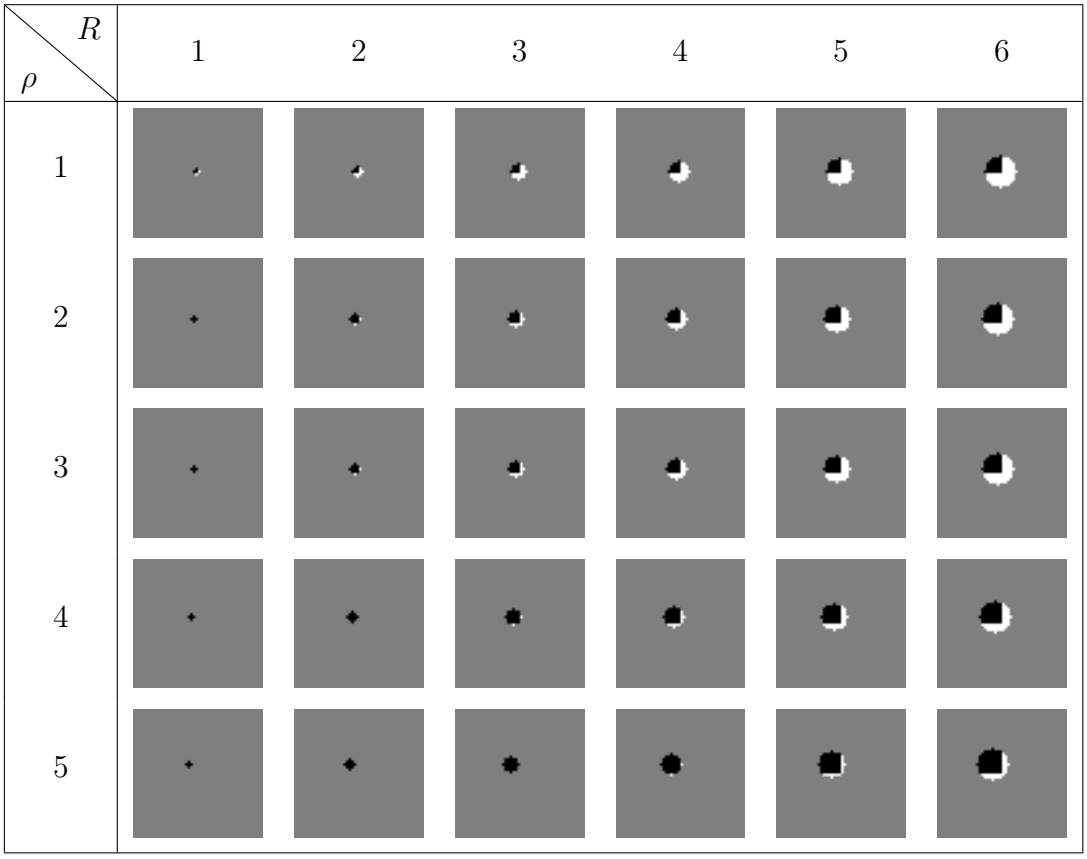

FIGURE 5.7: Inpainting masks for image corner.pgm for varying integration scale and mask radius.

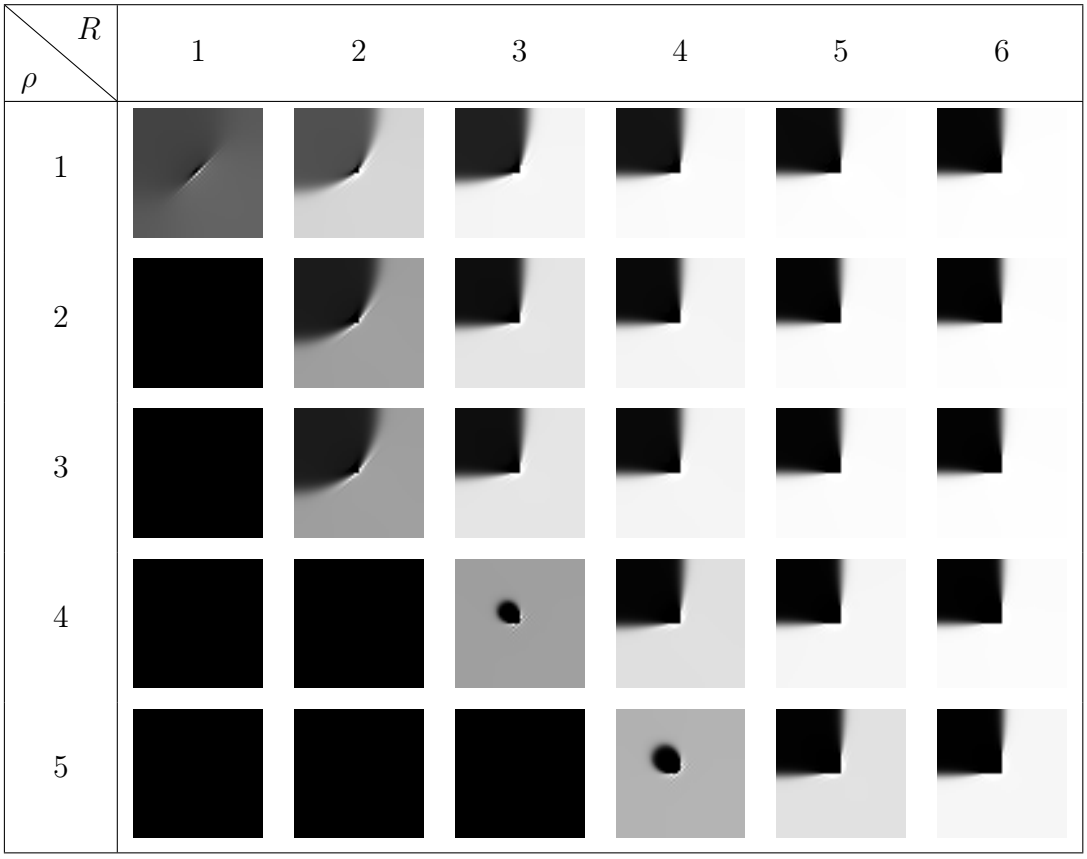

FIGURE 5.8: Inpainting results for mask shown in Figure 5.7. Inpainting parameters: $\sigma = 2, \lambda = 0.4, \alpha = 0.49, \gamma = 1$. (Original image see Figure 5.17)

ρ \ R	1	2	3	4	5	6
1	**4.83**	**10.97**	15.70	20.39	22.71	23.58
2	1.25	7.89	**17.31**	**21.48**	**23.45**	24.13
3	1.25	7.89	**17.31**	**21.48**	**23.45**	24.13
4	1.25	1.25	7.21	16.91	22.92	**24.34**
5	1.25	1.25	1.25	7.84	17.96	22.82

FIGURE 5.9: PSNR values of the reconstructed images in Figure 5.8. Highlighted values are the best in the respective column. Gray cells are cells where the R and ρ are equal.

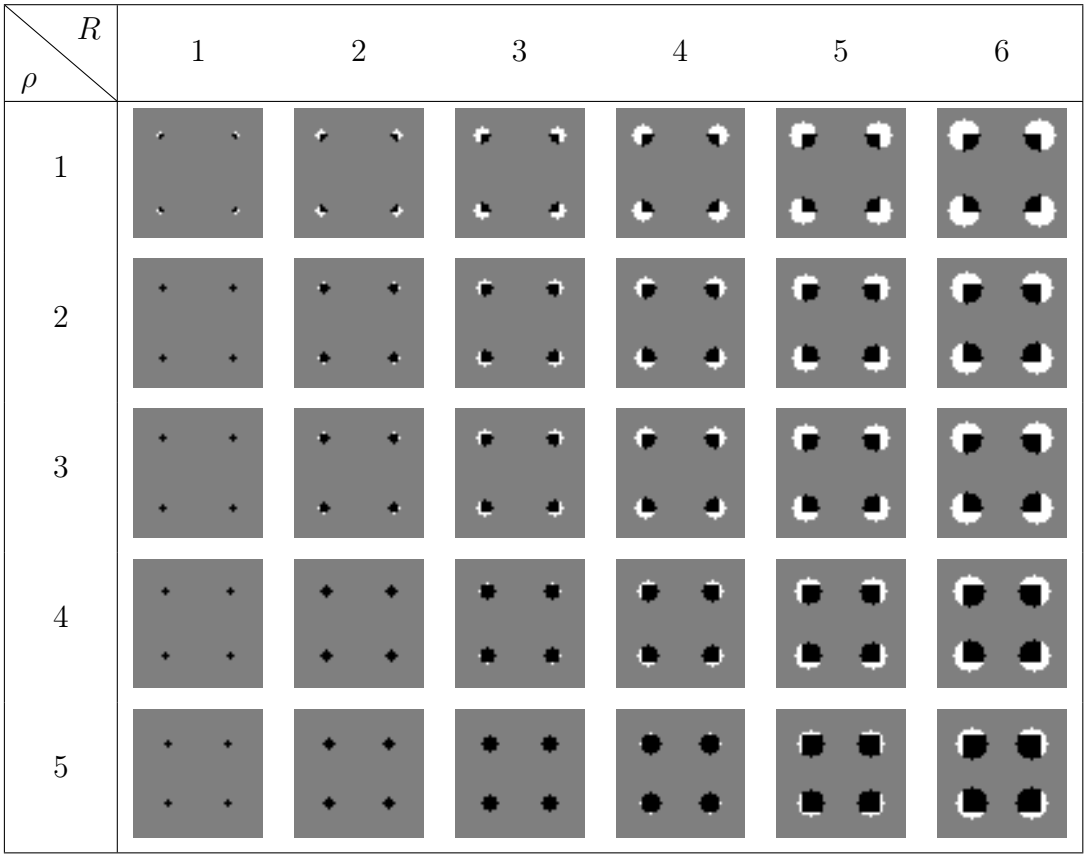

FIGURE 5.10: Inpainting masks for image rect_tiny.pgm for varying integration scale and mask radius.

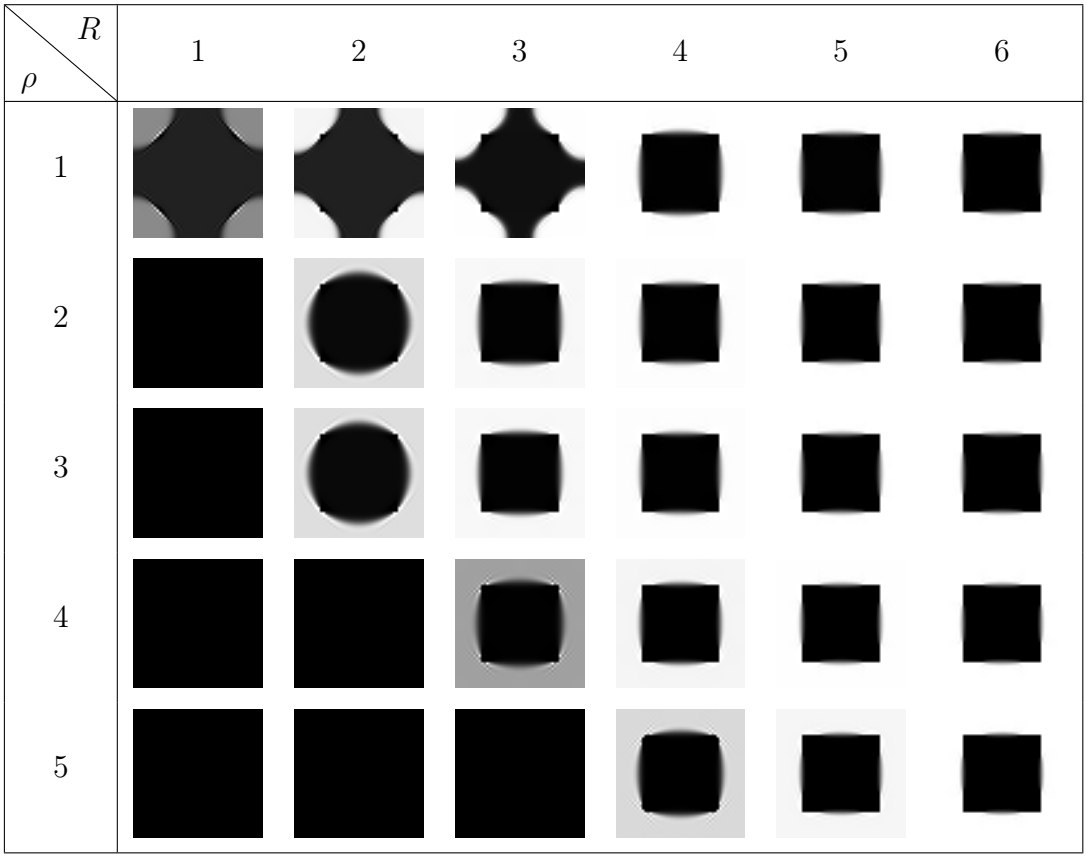

FIGURE 5.11: Inpainting results for mask shown in Figure 5.10. Inpainting parameters: $\sigma = 2, \lambda = 0.4, \alpha = 0.49, \gamma = 1$. Original image see Figure 5.17

ρ \ R	1	2	3	4	5	6
1	**4.76**	6.26	7.22	19.83	23.08	24.64
2	1.93	**9.71**	**19.41**	**22.86**	24.54	25.63
3	1.93	**9.71**	**19.41**	**22.86**	24.54	25.63
4	1.93	1.93	9.51	21.29	**25.35**	26.59
5	1.93	1.93	1.93	14.39	24.03	**26.81**

FIGURE 5.12: PSNR values of the reconstructed images in Figure 5.11. Highlighted values are the best in the respective column. Gray cells are cells where the R and ρ are equal.

5.3 Examples

Finally, I will present some results of the reconstruction process using only corner regions as seed points. First up, I want to demonstrate the approach with multiple binary images shown in Figure 5.17. As we can see in Figure 5.13,

Figure 5.11 and Figure 5.8, the algorithm works fairly well especially for binary images with clearly defined corners even though the edges are not as sharp as one might have hoped. However, the inpainting parameters were not highly optimised in these examples. The purpose of these experiments was to show the influence of the ratio between mask radius and integration scale on the final result. As a more practical example we see in Figure 5.19 that piecewise straight edges can be reconstructed fairly well whereas longer curves can be a challenge for the algorithm. On the other hand, for example in the image of the cat, curves pose a problem since the algorithm can not really detect any corners there because of the small curvature. Thus, we have too little information to reconstruct this area properly. Finally, we can see that the method still works well, even with a bit of noise applied to it (see Figure 5.14) However, if one increases the noise level we obsere that the quality of the reconstruction deteriorates further which is to be expected.

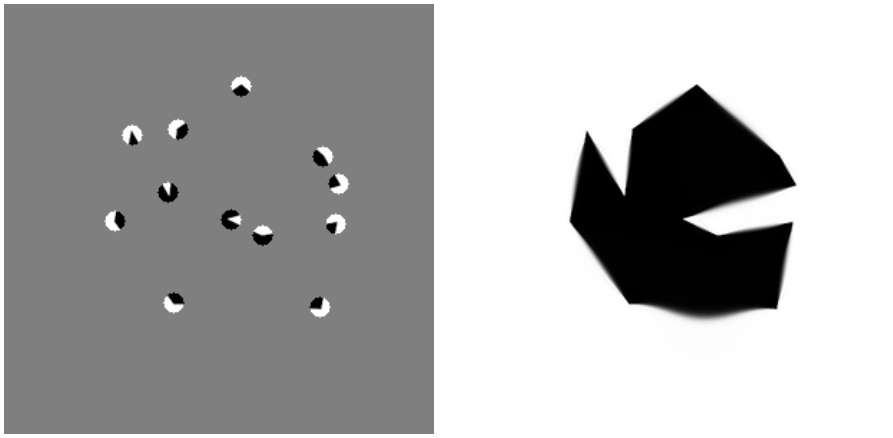

FIGURE 5.13: Inpainting of image `abstract1_small`. $\sigma = 1, \rho = 1, R = 7, q = 0.02$, Pixel density: 1.99%, PSNR: 24.57, Inpainting parameters: $\sigma = 2, \lambda = 0.1, \alpha = 0.49, \gamma = 1, N = 1000$, Original image: abstract1_small.pgm

If we now look to more textured 8-bit grey value images like `house` and `bank`, we see that the inpainting results become a bit worse compared to the binary images. This is primarily due to the lacking availability of corners in natural images.

In Figure 5.21, we see that reconstruction using only the corners one would naturally classify as corners (bottom row) is worse than the reconstruction where we added some corners that would not qualify as a corner directly. For example the addition of the single corner region in the top right corner improves the reconstruction in this area by a large margin, despite being more of a part of an edge.

FIGURE 5.14: Example of inpainting with noise degradation. Corner detection parameters: $\sigma = 1.5, \rho = 2.0, R = 6, q = 0.02$, Pixel density: 1.88%, Inpainting parameters: $\sigma = 2, \lambda = 0.2, \alpha = 0.49, \gamma = 1$, PSNR: 20.74

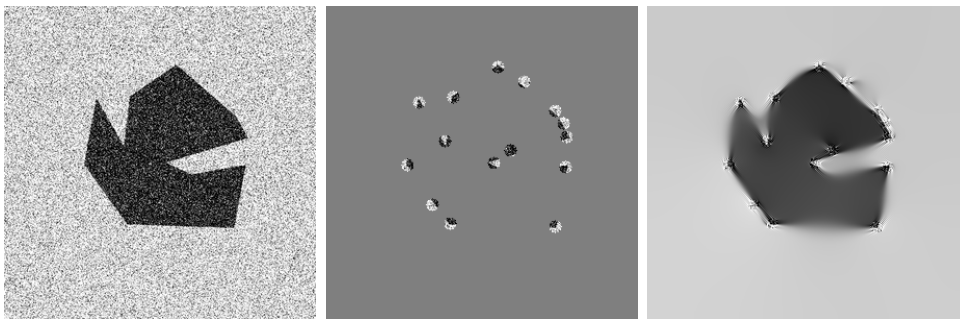

FIGURE 5.15: Example of inpainting with noise degradation. Corner detection parameters: $\sigma = 1.5, \rho = 2.0, R = 6, q = 0.02$, Pixel density: 1.88%, Inpainting parameters: $\sigma = 2, \lambda = 0.2, \alpha = 0.49, \gamma = 1$, PSNR: 16.25

This same figure is also a prime example for the removal of textures if the textured regions are underrepresented.

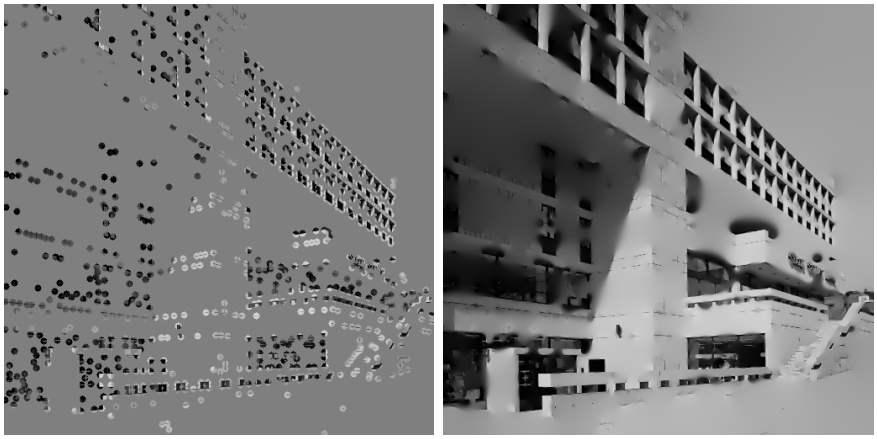

FIGURE 5.16: Inpaining of `bank`. Corner detection parameters: $\sigma = 1, \rho = 2, R = 4, q = 0.11$,TPPT+CNMS, Pixel density: 10.76%, Inpainting parameters: $\sigma = 2, \lambda = 0.5, \alpha = 0.49, \gamma = 1$ PSNR: 18.37

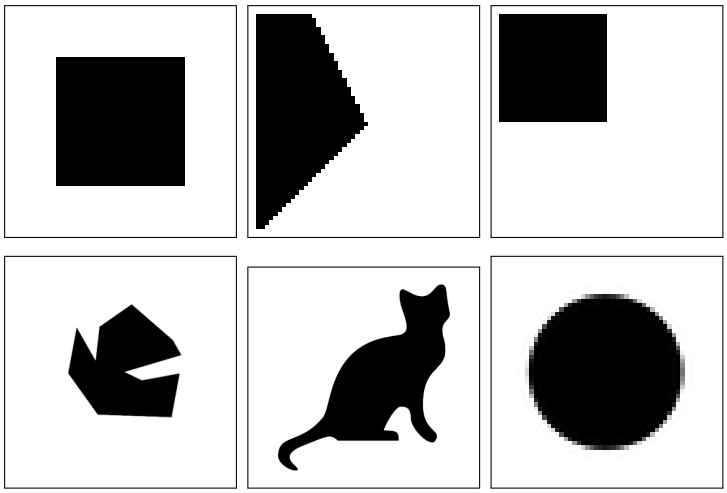

FIGURE 5.17: Binary test images. From top left to bottom right: `rect_tiny`, `flatcorner1`, `corner`, `abstract1_small`, `cat`, `ellipse_small`

FIGURE 5.18: (8-bit) Grayscale test images. From left to right: `house`, `bank`, `trui`. Images provided by Joachim Weickert.

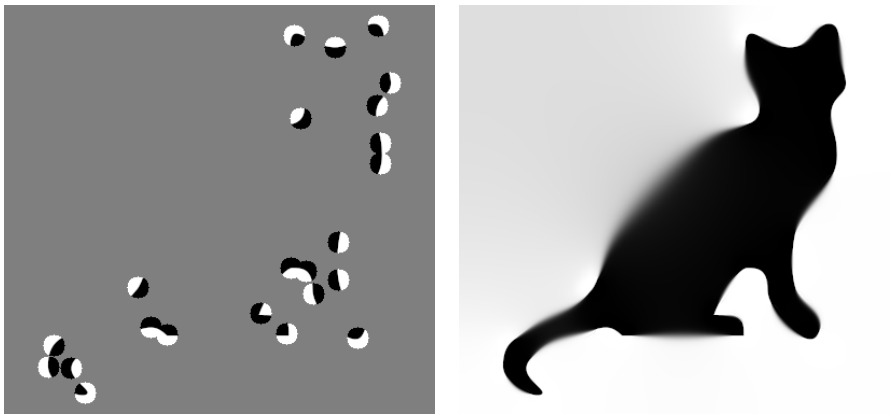

FIGURE 5.19: Mask and inpainted result for cat.pgm. Corner detection parameters: $\sigma = 1, \rho = 1, R = 10$, pixel density: 4.74%, Inpainting parameters: $\sigma = 2, \lambda = 0.2$, PSNR: 18.35

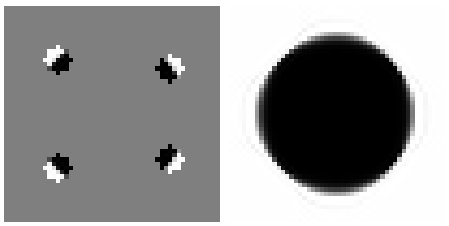

FIGURE 5.20: Example for the image ellipse_small (see Figure 5.17). Corner detection parameters: $\sigma = 1, \rho =, R = 4$. Inpainting parameters: $\sigma = 2, \lambda = 0.1, \alpha = 0.49, \gamma = 1$. PSNR: 26.64

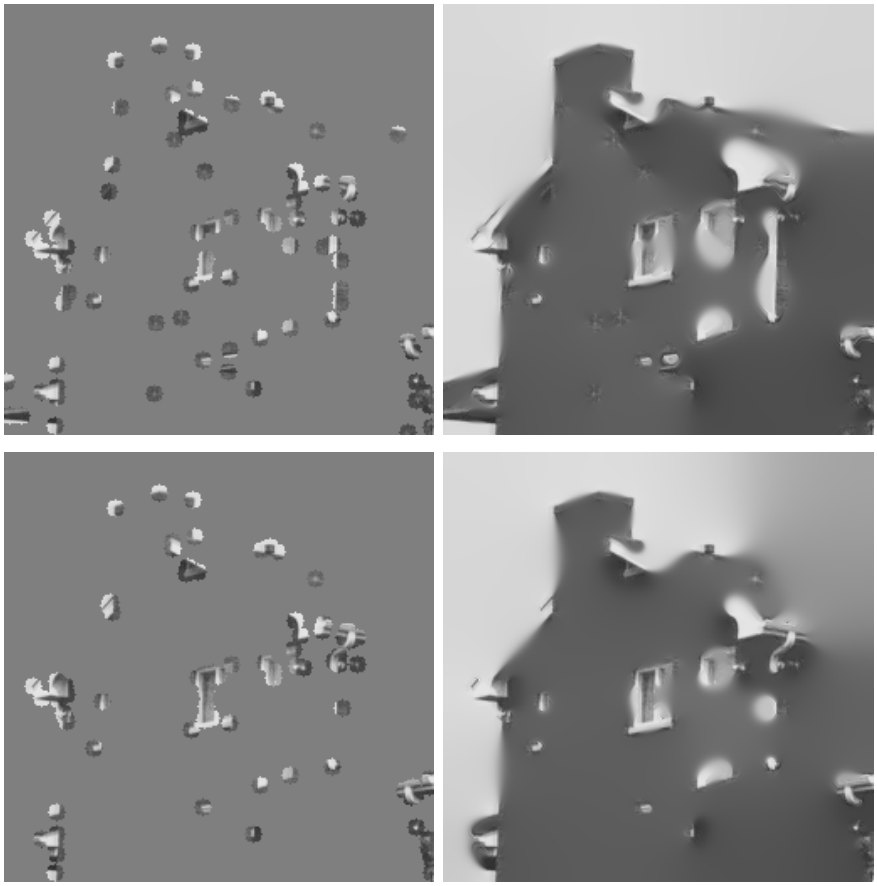

FIGURE 5.21: Mask and inpainting results of the image `house`. **Top row:** Corner detection parameters: $\sigma = 1, \rho = 1, R = 5, q = 0.1,$ CNMS, Pixel density: 9.23%, Inpainting parameters: $\sigma = 3, \lambda = 0.2, \alpha = 0.49, \gamma = 1$, PSNR: 21.11
Bottom row: Corner detection parameters: $\sigma = 1, \rho = 1.5, R = 5, q = 0.1,$ no CNMS, Pixel density: 7.71%, Inpainting parameters: $\sigma = 2, \lambda = 0.4, \alpha = 0.49, \gamma = 1$, PSNR: 18.68

6 Conclusion and Outlook

6.1 What has been done?

To summarise, I have adapted the method proposed in [7] by introducing circular corner regions and explored their potential for PDE based inpainting and compression approaches.
I used a simple structure tensor based approach for corner detection, namely the Förstner-Harris corner detector to find corner candidates. Using the new additional procedures called *circular non-maximum suppression* and *total pixel percentile thresholding*, these candidates were filtered to get a mask that satisfies a pre-defined pixel density criterion. With these additions, aside from special cases, it was possible to construct masks of similar pixel densities quite consistently. However, one should mention that both methods are pretty naive approaches far from perfect.
I tested the hypothesis that the mask radius should be chosen in dependence of the integration scale during the computation of the structure tensor. Moreover, the hypothesis stated that the mask radius should always be larger than the integration scale in order to cover the possible inaccuracy in the localisation of the corner. The experiments yielded reasonable evidence that this is the case. All in all it was shown that by increasing the size of corner regions one can cover the uncertainty introduced during the corner detection process and improve the reconstruction quality compared to the method proposed in [7].

6.2 Pros and Cons

The method examined herein is pretty well suited for reconstruction/compression of binary images with little to no texture. The problem in the compression of highly textured grey value images lies in the fact that due to the large corner regions, if one wants to cover highly textured regions, either one chooses the corner regions fairly small to cover as much of the texture as possible, similar to a subdivision based or stochastic approach. The downside here is that for small mask radii the inaccuracy in the corner detection might influence the outcome.

Another possibility is to choose the corner regions larger such that more corners are covered by a single region. By doing this however, the mask will become so dense that it will be suboptimal in a compression context. Another point to mention, that also Zimmer's method[7] struggled with, is the sparsity of corners in images. While corners are semantically important features, they are pretty seldom in organic images. (see the image `lena512` or `trui`).
Something to consider as well is the compression of noisy images. Here one has to pose the question whether the decompressed version of these images should contain noise or not. On one hand, noise in images is mostly a degradation that one wants to remove. On the other hand, image compression mostly aims at reconstructing the images that were compressed. By applying PDE based inpainting approaches in this context, most noise is removed due to the nature of these algorithms as we saw in Section 5.3, so the image that is decompressed is not the same as the one that was compressed. While in most cases this could be considered a benefit of the procedure, one could think of examples where such an inherent denoising would be considered unwanted behaviour (images of white noise?).

6.3 Future Work

Lastly, we will discuss some possible improvements to corner region based inpainting/compression approaches.

Improvement of the corner detection

As we saw in Section 5.1.1, the accuracy of the corner localisation is key to a good inpainting result. While the Förstner-Harris method is pretty accurate, one could think of other methods that suggest higher, even sub-pixel accuracy. A possible method could be one proposed by Alvarez et al.[51] in 1997. The method is based on a scale-space approach that is supposed to be able to trace back the origin of a corner to its actual location with subpixel accuracy.

Optimising corner region radius

Something that one could investigate further would be the optimal corner region radius and whether it would be sensible to make it variable in space. The idea behind that is, that in regions where corners are sparse, one could increase the radius to include more context from around the corner. Similarly, in textured

regions the radius could be decreased in order to get a mask that looks more like a mask from a probabilistic sparsification or subdivision based approach.

Combination with ideas from other approaches

Since it is pretty difficult to produce good inpainting masks, i.e. masks that result in reconstructions with a high PSNR, only using corners and corner regions, one could consider applying methods from other approaches to corner based mask creation. One method that comes into mind would be probabilistic sparsification[13], [14] applied to corner regions.
Seeing what impact tonal optimisation had[13], [14], one could also consider applying this to a corner region based approach.

Possible application to region of interest(ROI) coding

On a more unrelated note, corner regions could see application in region of interest coding as proposed in [26]. Inpainting masks based on corner regions could be used as weighting masks for the subdivision approach to be able to better reconstruct regions of higher semantic importance as indicated by a higher density of corners.

Bibliography

[1] Gregory K. Wallace. "The JPEG still picture compression standard". In: *Communications of the ACM* (1991), pp. 30–44.

[2] David Taubman and Michael Marcellin. "JPEG2000: standard for interactive imaging". In: vol. 90. 2002, pp. 1336–1357.

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

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

[5] Markus Mainberger, Sebastian Hoffmann, Joachim Weickert, et al. "Optimising Spatial and Tonal Data for Homogeneous Diffusion Inpainting". In: *Scale Space and Variational Methods in Computer Vision*. Ed. by Alfred M. Bruckstein, Bart M. ter Haar Romeny, Alexander M. Bronstein, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 26–37. ISBN: 978-3-642-24785-9.

[6] Markus Peloquin. *PDE-based Image Compression*. 2009.

[7] Henning Lars Zimmer. "PDE-based Image Compression using Corner Information". MA thesis. Saarland University, 2007.

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

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

[10] Dong Liu, Xiaoyan Sun, Shipeng Li, et al. "Image Compression With Edge-Based Inpainting". In: *Circuits and Systems for Video Technology, IEEE Transactions on* 17 (Nov. 2007), pp. 1273–1287. DOI: 10.1109/TCSVT.2007.903663.

[11] 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.

[12] Zakaria Belhachmi, Dorin Bucur, Bernhard Burgeth, et al. "How to Choose Interpolation Data in Images". In: *SIAM Journal of Applied Mathematics* 70 (Jan. 2009), pp. 333–352. DOI: 10.1137/080716396.

[13] Christian Schmaltz, Pascal Peter, Markus Mainberger, et al. "Understanding, Optimising, and Extending Data Compression with Anisotropic Diffusion". In: *International Journal of Computer Vision* 108 (July 2014). DOI: 10.1007/s11263-014-0702-z.

[14] Laurent Hoeltgen, Markus Mainberger, Sebastian Hoffmann, et al. "Optimising Spatial and Tonal Data for PDE-based Inpainting". In: *CoRR* abs/1506.04566 (2015). arXiv: 1506.04566. URL: http://arxiv.org/abs/1506.04566.

[15] Joachim Weickert. *Personal conversation.*

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

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

[18] M. Mahoney. "Adaptive Weighing of Context Models for Lossless Data Compression". In: *Technical Report CS-2005-16.* Florida Institute of Technology, Melbourne, Florida, Dec. 2005.

[19] Christian Schmaltz, Pascal Gwosdek, Andres Bruhn, et al. "Electrostatic Halftoning". In: *Comput. Graph. Forum* 29 (Dec. 2010), pp. 2313–2327. DOI: 10.1111/j.1467-8659.2010.01716.x.

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

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

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

[23] 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.

[24] D. Marr and Ellen Hildreth. "Theory of Edge Detection". In: *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)* 207 (Feb. 1980), pp. 187–217. DOI: 10.1098/rspb.1980.0020.

[25] 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).

[26] 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.

[27] Frans Kanters, Martin Lillholm, Remco Duits, et al. "On Image Reconstruction from Multiscale Top Points". In: vol. 3459. Apr. 2005, pp. 431–442. DOI: 10.1007/11408031_37.

[28] Bart Janssen, Frans Kanters, Remco Duits, et al. "A Linear Image Reconstruction Framework Based on Sobolev Type Inner Products". In: vol. 70. Mar. 2005, pp. 85–96. DOI: 10.1007/11408031_8.

[29] Philippe Weinzaepfel, Hervé Jégou, and Patrick Pérez. "Reconstructing an image from its local descriptors". In: *Computer Vision and Pattern Recognition* (June 2011). DOI: 10.1109/CVPR.2011.5995616.

[30] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: vol. 60. 2004, pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94.

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

[32] Joachim Weickert. *Lecture Notes for "Image Processing and Computer Vision"*. Lecture notes. 2019. URL: https://www.mia.uni-saarland.de/Teaching/ipcv19.shtml.

[33] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 1997. URL: http://www.dspguide.com/pdfbook.htm.

[34] Carlo Tomasi and Takeo Kanade. *Detection and Tracking of Point Features*. Tech. rep. International Journal of Computer Vision, 1991.

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

[36] Karl Rohr. "Localization properties of direct corner detectors". In: *Journal of Mathematical Imaging and Vision* 4 (May 1991), pp. 139–150. DOI: 10.1007/BF01249893.

[37] W. Förstner and Gülch E. "A fast operator for detection and precise location of distinct points, corners and centres of circular features". In: *ISPRS Conference on Fast Processing of Photogrammetric Data* (1987), pp. 281–305.

[38] 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.

[39] 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.

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

[41] Joachim Weickert. *Lecture Notes for "Differential Equations in Image Processing and Computer Vision"*. Lecture notes. 2019. URL: https://www.mia.uni-saarland.de/Teaching/dic19.shtml.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] Magnus R. Hestenes and Stiefel Eduard. "Methods of Conjugate Gradients for Solving Linear Systems". In: *Journal of Research of the National Bureau of Standards* 49(6) (Dec. 1952), pp. 409–436. DOI: 10.6028/jres.049.044.

[48] *Percentile*. https://en.wikipedia.org/wiki/Percentile. Last accessed: June 17, 2020.

[49] *Gauss's Circle Problem*. https://mathworld.wolfram.com/GausssCircleProblem.html. Last accessed: June 16, 2020.

[50] David Hilbert and Stephan Cohn-Vossen. *Anschauliche Geometrie*. Berlin, Heidelberg: Springer-Verlag, 1996. DOI: 10.1007/978-3-642-19948-6.

[51] Luis Alvarez and Freya Morales. "Affine Morphological Multiscale Analysis of Corners and Multiple Junctions". In: *International Journal of Computer Vision* 25 (Nov. 1997), pp. 95–107. DOI: 10.1023/A:1007959616598.