Universität Regensburg
Fakultät für Mathematik

Solving Convex and Non-Convex Functionals in Image Processing using a Primal-Dual Algorithm

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science
im Studiengang Computational Science

Eingereicht bei Prof. Dr. Harald Garcke
am 17.03.2016

Vorgelegt von:
Michael Bauer
Geboren am 23. August 1987 in Gräfelfing
Matrikelnummer 152 8558

Acknowledgements

This thesis which I could never have done without the support of several people, is the last part of my degree course Computational Science. Now, I get the chance to thank everybody who contributed directly or indirectly to this thesis and gave me assistance, whenever I needed it.

For being patient, supportive, motivating and always positive I have to thank you so much Sylvia Wallner. Also lot of thanks to my mother for your patience and support. For fruitful discussions, listening to all my presentations at least once and correcting silly mistakes I want to thank Leonie Jennert and Bernhard Zimmermann. Thank you Michael Huttner for the great discussions about programming and your help.

The basis of this thesis are works published by Prof. Dr. Daniel Cremers (TUM). Thanks to you and your group, especially Thomas Moellenhoff, for the chance to join the CUDA course and for your external contribution to this work.

The thesis itself was written at the chair of Prof. Dr. Harald Garcke, who also supervised me. So, last but not least I want to thank you so much for your input, your support and your constructive criticism. Our discussions always brought me one step further and let me see each problem from another point of view.

Abstract

In this thesis we consider three different functionals in image processing. Our goal is to minimze the energy of each functional to approximate an input image optimally. By solving these optimization problems we derive applications to imaging like image cartooning, denoising, inpainting and segmentation.

The algorithm, which computes the minimal energy of each functional, is a primal-dual algorithm. This framework solves saddle-point problems and is highly parallelizable. In order to be able to apply the algorithm we turn all our minimization into saddle-point problems. One convex functional we consider is the so called ROF functional, first proposed by Rudin, Osher and Fatemi, hence the name. It is based on a least squares and a total variation term. Using the total variation together with a term based on the L1 norm we derive the convex TVL1 functional which we also consider in this work.

Even though the third functional, namely the Mumford-Shah functional, is non-convex, we use convex analysis to turn the minimization of the functional into a saddle-point problem. This strategy does not lead to a provable minimal value of the Mumford-Shah functional, but we get high-quality approximations of images. Additionally, we consider a second approach for minimizing the non-convex Mumford-Shah functional. We make use of convex relaxation techniques to turn the minimization into a convex saddle-point problem. Then, we apply again the primal-dual algorithm to solve this optimization problem.

In terms of the convex relaxed functional the primal-dual algorithm has a bad performance. This is related to Dykstra's projection algorithm which is used in each iteration step of the primal-dual algorithm. To resolve the run-time issue we present a method for solving the convex relaxed Mumford-Shah model using Lagrange multipliers. We show that this approach creates a tremendous performance increase.

Finally, we present image cartooning and a comparison for image denoising with the proposed models. A modified version of the ROF functional is taken into account when we present image inpainting. And at last we show that the Mumford-Shah functional is a good model for image segmentation.

Contents

1 Introduction

In the second decade of the twenty-first century autonomous driving seems to be an important thing for the car, computer and software industry. The expectation is nothing less than having fewer, or even no, accidents. With these cars, industry is trying to change the world for a better. As this may comfort many people, it poses a lot of technical and computational issues, for instance a stable hardware or even more important viable software. One field of research - of many others - is called Machine Learning, where the computer is trained to make the right decisions in the right situations. The car should be able to accommodate to all possible circumstances which could appear during a drive. To make it even more complicated, the onboard computer needs to decide in real-time. As one can imagine, it is extremely difficult to develop fast, stable and tractable computer programs.
In order to learn from a certain situation the computer needs data from the environment. This can be the shape of another car, traffic lights, differences in the lighting conditions or the distance to other objects. What all these information have in common: They can be collected via images. Small cameras are tracking the traffic and surrounding and providing the necessary data for the software. But it is not only the autonomously driving car which makes use of images. Doctors use X-ray view or MRI scans in patient treatment and semiconductor companies use pictures of wafers looking for damages on it, for example scratches or dark spots. For this reason, image processing has become more important during the last decades. Researchers dealt with many problems like edge detection, deblurring, denoising, inpainting or image segmentation. In the field of edge detection the probably most famous researcher is John F. Canny. He presented a stable edge detection algorithm in 1986 (cf. [10]), also known as Canny-Edge-Detector. Only three years after Canny published his work two researchers, namely David Mumford and Jayant Shah, published a paper ([17]), whose impact is still present. According to Google Scholar their publication was cited over 4.703 times, as of March 12th 2016. They suggested to minimize the energy of the so called Mumford-Shah functional in order to approximate an input image optimally. Solving this optimization problem leads to applications like image denoising, inpainting and segmentation. Unfortunately, this functional is by definition non-convex. For this reason finding the minimal energy of it is a hard task.
One approach for minimizing the Mumford-Shah functional was published in 2009 by Pock, Cremers, Bischof and Chambolle in [21]. We consider this approach in this thesis and present an alternative approach to their proposed method using a Lagrange multiplier formalism. This will lead us to a tremendous performance increase compared to the original suggested framework. Pock et al. also proposed a novel primal-dual algorithm in [21] in order to compute the minimizer of the functional. The algorithm

itself is a solver for saddle-point problems and highly parallelizable. In 2011 Pock and Chambolle published a work ([3]), cited 1.234 times (Google Scholar) as of March 12th 2016, in which they generalized the primal-dual algorithm and showed applications to imaging with it. So, this framework has grown to a powerful tool in the last years to solve saddle-point problems efficiently.
One functional Pock and Chambolle consider in their work of 2011 is the ROF functional. It was proposed in 1992 by the researchers Rudin, Osher and Fatemi ([23]). They suggested to minimize the energy of this convex functional. The functional itself involves a least squares data term and uses the total variation as regularizer term. By replacing the norm of the least squares data term in the ROF functional with the L1 norm one derives the convex TVL1 functional. Minimizing these two convex functionals can efficiently be done by using techniques of convex analysis. By applying convex analysis we turn the minimization of the functionals into saddle-point problems, for which we can use the primal-dual algorithm as a solver. Further, we discuss an approach to the Mumford-Shah functional which proceeds in the same fashion as for minimizing the ROF and TVL1 functional. In 2014 Strekalovskiy and Cremers used convex analysis in order to minimize the non-convex Mumford-Shah functional. Even though, this strategy does not lead to a provable minimal value of the optimization problem, we show that we derive high-quality image approximations with this formulation.
In this work we introduce the three functionals mentioned, the total variation and concepts of convex analysis. After providing the frameworks for minimizing the functionals we estimate applicable parameters used in the models and present applications to imaging. Whereas cartooning and denoising are impressive applications, image inpainting and image segmentation are really imposing.

2 Basic Concepts

In this chapter we give an introduction to the most important concepts we meet in this thesis. We start by defining images in a mathematical manner, then introduce basic concepts of convex analysis and the total variation.

2.1 Images in Mathematics

Images can be viewed as mathematical objects. We can distinguish between discrete and continuous images. Let us first introduce images in the mathematical sense and then give some examples. The definition and examples can be found in the work of Kristian Bredies and Dirk Lorenz [9].

Definition 2.1 (Image) *Let $\Omega \subseteq \mathbb{R}^n$ be an image domain and C be a color space. An n-dimensional image u is a mapping $u : \Omega \longrightarrow C$, where each point in Ω corresponds to a color value in C.*

Remark 2.2 (Continuous and discrete images) *Images can be discrete or continuous. The difference between these two classes is determined in the image domain Ω.*

- *Let $\Omega = \{1, ..., N\}^n$, then u is a discrete image, since Ω is a discrete set. All images on computers are discrete, because each single pixel lies at a fixed, discrete position (i, j).*
- *Let $\Omega \subseteq \mathbb{R}^n$, then u is a continuous image. For instance, we could set $\Omega = [a, b]^n$ with $a, b \in \mathbb{R}$ and $a < b$.*

There are several image types, like binary or colored images. The property, to which type an image u belongs, is determined by the color space C:

- *Let $C = \{0, 1\}$, then we call u binary or black-and-white image.*
- *In the case of grayscaled images we set $C = \{0, ..., 2^k - 1\}$, where the value k determines the bit depth. Usually, we find $k = 8$ in computers, i.e. grayscaled images take on values in between 0 and 255, where $0 =$ black and $255 =$ white.*
- *A n-dimensional color image has $C = \{0, ..., 2^k - 1\}^n$, for instance $n = 3$ for RGB (red-green-blue) images.*
- *An image u could also map to continuous color spaces, e.g. $C = [a, b]^n$ or $C = \mathbb{R}^n$. Most common are RGB images with $n = 3$, $a = 0$ and $b = 1$.*

In this thesis we consider two discrete settings for the domain Ω. One where Ω is two-dimensional, and the other where Ω is a three-dimensional space. In the first case we call a point (i, j) in Ω pixel, in the second a point (i, j, k) is called voxel.

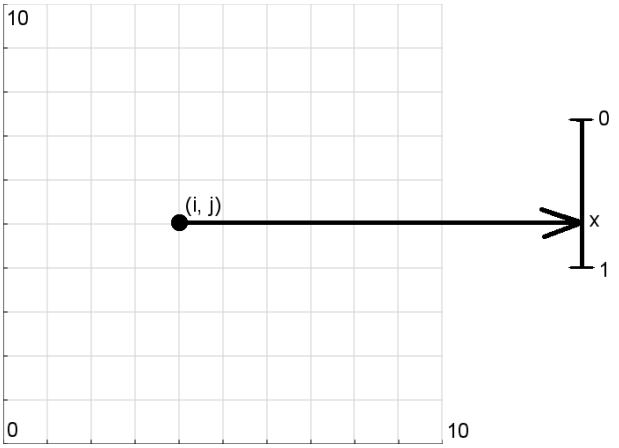


Figure 2.1: An image function $u : \{0, 1, ..., 10\}^2 \to [0, 1]$, mapping a pixel (i, j) to the value $x \in [0, 1]$.

2.2 Convex Optimization and Convex Analysis

In this section we cover some topics which are related to convex analysis. Since we are facing (convex) optimization problems, we first define this class of programs. We follow with our notations, definitions and theorems the book of Nocedal and Wright ([19]). An optimization problem is given by

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0 & i \in \mathcal{E}, \\ c_i(x) \geq 0 & i \in \mathcal{I}, \end{cases} \tag{2.1}$$

where f and for all $i \in \mathcal{E} \cup \mathcal{I}$ the functions c_i are smooth, real-valued functions on a subset of $\mathbb{R}^n$. Further, $\mathcal{E}$ and $\mathcal{I}$ denote two finite sets of indices. Here, f is the objective function, whereas the c_i for $i \in \mathcal{E}$ are the equality constraints and c_i with $i \in \mathcal{I}$ the inequality constraints. If we have no constraints, meaning $\mathcal{E} = \mathcal{I} = \emptyset$, we call the System (2.1) unconstrained. It is a convex optimization problem if all, the objectiv, equality and inequality functions, are convex. We further want to define the feasible set Ω. It is the set of all points, which satisfy the constraints c_i for all i. Then we have

$$\Omega = \{x \mid c_i(x) = 0,\ i \in \mathcal{E};\ c_i \geq 0,\ i \in \mathcal{I}\}.$$

We can rewrite our optimization problem in terms of the set Ω to $\min_{x \in \Omega} f(x)$. Additionally, we want to define the so called active set for the constraint functions.

Definition 2.3 (Active Set) *The active set $\mathcal{A}(x)$ at any feasible x consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints i, for which $c_i(x) = 0$ that is*

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(x) = 0\}.$$

We call a inequality constraint active if $c_i = 0$ and inactive if $c_i > 0$, for an $i \in \mathcal{I}$. The definition of the active set is an important basis for the following definition.

Definition 2.4 (LICQ) *Given the point x and the active set $\mathcal{A}(x)$, we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients*

$$\{\nabla c_i(x),\, i \in \mathcal{A}(x)\}$$

is linearly independent.

With these two definitions we can propose the first-order necessary conditions which are also well known as the Karush-Kuhn-Tucker (KKT) optimality conditions.

Theorem 2.5 (Karush-Kuhn-Tucker Optimality Conditions) *Suppose that x^* is a local solution of Equation (2.1) that the functions f and c_i are continuously differentiable, and that the LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^*, with components $\lambda_i^*, i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*).*

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{2.2a}$$
$$c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E}, \tag{2.2b}$$
$$c_i(x^*) \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{2.2c}$$
$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{2.2d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E} \cup \mathcal{I}. \tag{2.2e}$$

Remark 2.6 *A proof of this theorem can be found in [19]. Further, we state that the last condition (2.2e) in this theorem is also known as the complementary slackness condition. It implies that either the i-th constraint is active or $\lambda_i^* = 0$. It is also possible that both are zero.*

We need this theorem in Subsection 4.3.5, in which we compute a minimal value to a strictly convex optimization problem with one constraint function.
Since convex optimization is deeply related to convex analysis we define the class of convex functions and its properties. We set $X \subseteq \mathbb{R}^n$ and its dual space $Y = X^*$. The following definitions, propositions and theorems can be found in the work of Chambolle et al. ([11]). They are all influenced by and related to the book of Ralph Tyrrell Rockafellar ([22]). If other results are taken into account we point this out. Further note that we use the notation $\mathbb{R}_\infty$ for the extended real-values, namely $\mathbb{R} \cup \{+\infty\}$, in this thesis.

Definition 2.7 (Convex Set) *A subset $C \subseteq \mathbb{R}^n$ is said to be convex if and only if for any $u_1, u_2 \in C$, the line segment $[u_1, u_2] \subseteq C$ that is, for any $t \in [0, 1]$,*

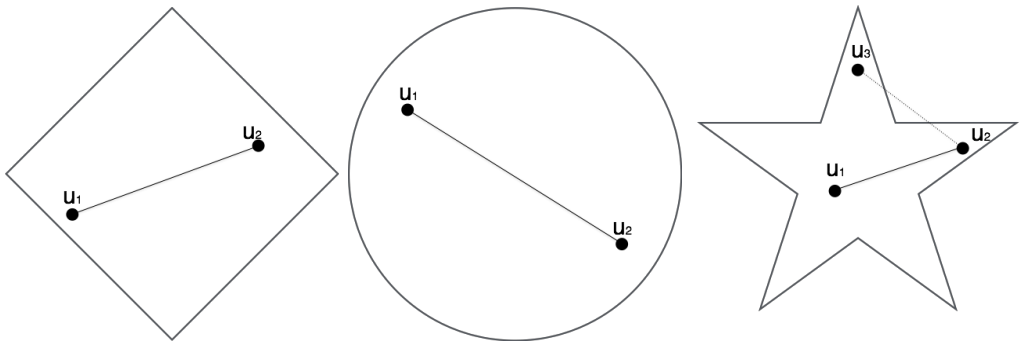
$$u_1 t + u_2(1 - t) \in C.$$

This definition ensures that if C is convex, then for any two arbitrary points u_1, u_2 in C, also the line segment $[u_1, u_2]$ with end points u_1, u_2 lies fully in C (see Figure 2.2).

Example 2.8 *The upper half-plane of $\mathbb{R}^2$ denoted by $\mathbb{H}$ is a convex set. We can express the set by $\mathbb{H} = \{u \in \mathbb{R}^2 \mid u_2 \geq 0\}$. To prove this we let $u_1, u_2 \in \mathbb{H}$ and $t \in [0, 1]$. Then we compute $u_1^1 t + u_2^1(1 - t) \in \mathbb{R}$ and further*

$$\underbrace{u_1^2 t}_{\geq 0} + \underbrace{u_2^2(1 - t)}_{\geq 0} \geq 0,$$

for which $u_1 t + u_2(1 - t) \in \mathbb{H}$.



(a) $l1$ unit sphere. (b) $l2$ unit sphere. (c) Star: a non-convex set.

Figure 2.2: Three different sets, where (a) and (b) are convex and (c) is non-convex. (a) is the unit sphere of the $l1$ norm, where (b) is the unit sphere of the $l2$ norm. In (c) we see that the line segment of $[u_2, u_3]$ does not fully lie in the set.

Definition 2.9 (Indicator Function, Support Function) *For any convex subset C of a vector space X, the indicator function $\delta_C : X \longrightarrow \mathbb{R}_\infty$ is defined as*

$$\delta_C(u) = \begin{cases} 0 & \text{if } u \in C \\ \infty & \text{if } u \notin C \end{cases}.$$

The support function of the set C is defined as

$$S_C(u) = \sup_{p \in C} \langle p, u \rangle,$$

where we allow $S_C(u)$ to be $+\infty$.

Definition 2.10 (Convex Function, Proper, Lower-Semicontinuous) *Let $C \subseteq \mathbb{R}^n$ be a convex set. A function $F : C \longrightarrow [-\infty, \infty]$ is said to be*

- *convex if and only if for all $u_1, u_2 \in C$ and any $t \in [0,1]$ the inequality*

$$F(u_1 t + u_2(1-t)) \leq F(u_1)t + F(u_2)(1-t) \tag{2.3}$$

  is satisfied. F is called strictly convex if Inequality (2.3) holds strictly whenever u_1, u_2 are distinct points and $t \in (0,1)$.

- *proper if and only if F is not identically $-\infty$ or $+\infty$.*

- *lower-semicontinuous (l.s.c) if and only if for any $u \in C$ and a sequence (u_n) converging to u,*

$$F(u) \leq \liminf_{n\to\infty} F(u_n).$$

We define $\Gamma_0(C)$ as the set of all convex, proper, l.s.c. functions on C.

A common example for a strictly convex function would be a quadratic function, cf. Example 2.13 and Figure 2.3 (a). It is illustrated together with the plot in (b) of the function

$$F(u) = \begin{cases} u^2 & x \in [-1,1] \\ -\frac{1}{2}u(u-4) & x \in (1,3] \end{cases}, \tag{2.4}$$

which is a lower-semicontinuous function. The following theorem can for instance be found in [22].

Theorem 2.11 *If F, G are proper convex functions on $\mathbb{R}^n$, then $F + G$ is convex.*

Proof Define $H(u) = F(u) + G(u)$ with F, G being proper and convex and choose $u_1, u_2 \in X$ and $t \in [0,1]$. Then we obtain

$$\begin{aligned} H(u_1 t + u_2(1-t)) &= F(u_1 t + u_2(1-t)) + G(u_1 t + u_2(1-t)) \\ &\leq F(u_1)t + F(u_2)(1-t) + G(u_1)t + G(u_2)(1-t) \\ &= \underbrace{F(u_1)t + G(u_1)t}_{=H(u_1)t} + \underbrace{F(u_2)(1-t) + G(u_2)(1-t)}_{H(u_2)(1-t)} \\ &= H(u_1)t + H(u_2)(1-t). \end{aligned}$$

■

Remark 2.12 (Concave Function) *If $-F$ is (strictly) convex, then we say that F is (strictly) concave. If F is both convex and concave we say that F is affine, i.e. equality holds in Equation (2.3).*

Example 2.13 *1. Let $A \in \mathbb{R}^{n\times n}$ be a real, symmetric, positiv definite matrix and $F : \mathbb{R}^n \longrightarrow \mathbb{R}$ a linear function with $F(u) = \frac{1}{2}u^T Au + b^T u = \frac{1}{2}||u||_A^2 + b^T u$, where*

$||\cdot||_A^2$ denotes the quadratic A-norm. Then F is strictly convex. We can verify this with $t \in (0,1)$ and

$$\begin{aligned}
F(u_1t + u_2(1-t)) &= \frac{1}{2}||u_1t + u_2(1-t)||_A^2 + b^T(u_1t + u_2(1-t)) \\
&\overbrace{\leq}^{(*)} \frac{1}{2}\left(||u_1t||_A^2 + ||u_2(1-t)||_A^2\right) + (b^Tu_1)t + (b^Tu_2)(1-t) \\
&= \frac{1}{2}\left(\underbrace{t^2||u_1||_A^2}_{<t||u_1||_A^2} + \underbrace{(1-t)^2||u_2||_A^2}_{<(1-t)||u_2||_A^2}\right) + (b^Tu_1)t + (b^Tu_2)(1-t) \\
&< \underbrace{\frac{1}{2}t||u_1||_A^2 + (b^Tu_1)t}_{=F(u_1)t} + \underbrace{\frac{1}{2}(1-t)||u_2||_A^2 + (b^Tu_2)(1-t)}_{=F(u_2)(1-t)} \\
&= F(u_1)t + F(u_2)(1-t).
\end{aligned}$$

In $(*)$ we used the triangle inequality.

2. *Each norm $||\cdot|| : X \longrightarrow \mathbb{R}$ on a normed vector space X is convex (see also Figure 2.10 for the $l1$ and $l2$ norm).*

Proof *Choose $u, v \in X, t \in [0,1]$, then*

$$||ut + v(1-t)|| \overbrace{\leq}^{(*)} ||ut|| + ||v(1-t)|| \overbrace{=}^{(**)} ||u||t + ||v||(1-t).$$

In $(*)$ we used the triangle inequality and in $(**)$ absolute homogeneity with the fact that $t \in [0,1]$ and for that $(1-t) \in [0,1]$. ■

There is another definition of convex functions. Therefore, let us first introduce the epigraph and the domain of a function.

Definition 2.14 (Domain, Epigraph) *For any function $F : X \longrightarrow \mathbb{R}_\infty$, we define the domain*

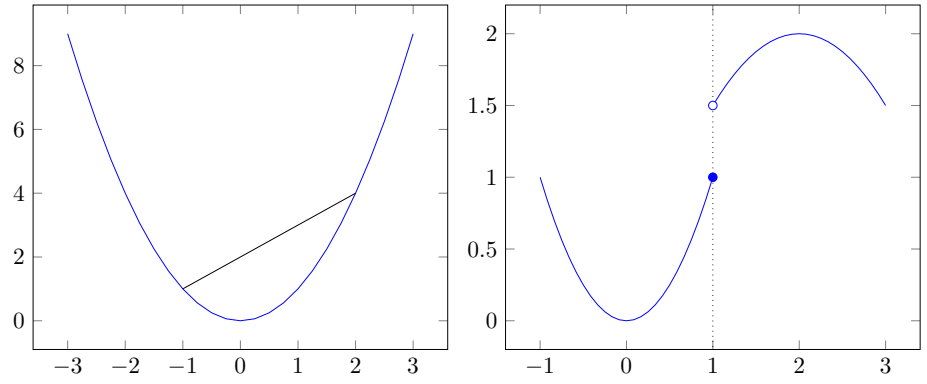
$$\mathrm{dom}(F) = \{u \in X : F(u) < +\infty\}$$

and the epigraph

$$\mathrm{epi}(F) = \{(u,t) \in X \times \mathbb{R} : t \geq F(u)\} \in \mathbb{R}^{n+1}.$$

An illustration of the domain and epigraph can be seen in Figure 2.4. In the previous definition we used that a function F is convex if and only if its domain is convex. Now, we have an equivalent definition of convex functions by using the definition of the epigraph.

Definition 2.15 (Convex Function) *A function $F : X \longrightarrow \mathbb{R}$ is convex if and only if* $\mathrm{epi}(F)$ *is a convex set.*



(a) A quadratic function. (b) A l.s.c. function.

Figure 2.3: (a) The plot of one line segment above a quadratic function, which lies completely in epi(F). (b) The lower term u^2 accesses the value $(1, 1)$, but the upper term of the function $-\frac{1}{2}u(u-4)$ does not.

Example 2.16 *The indicator function $\delta_C : X \to \mathbb{R}_\infty$ of a set C is convex if and only if C is a convex set. This can be seen by:*
"$\Longrightarrow$": If $\delta_C(u)$ is a convex function, we know that the domain of $\delta_C(u)$ is convex by Definition 2.15. So, $C = \mathrm{dom}(\delta_C(u))$ is a convex set.
"$\Longleftarrow$": Reversely, if C is a convex set, the domain of $\delta_C(u)$ denoted by $\mathrm{dom}(\delta_C(u)) = C$ is convex. Then $\delta_C(u)$ is a convex function.

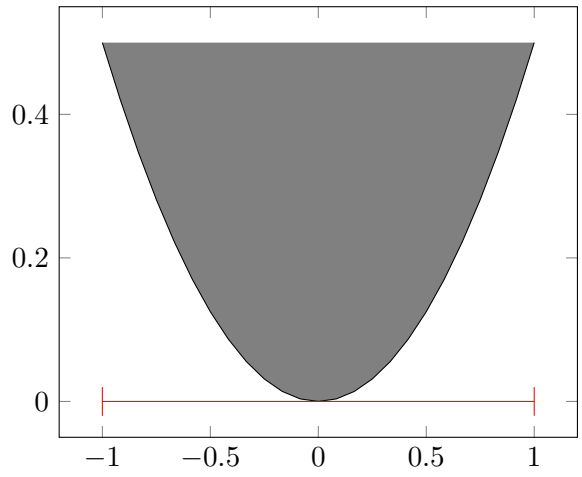


Figure 2.4: The domain (red) denoted as $\mathrm{dom}(F) = [-1, 1]$ and epigraph $\mathrm{epi}(F) = \{(u, t) \in [-1, 1] \times \mathbb{R} : t \geq \frac{1}{2}u^2\}$ (gray) of a function $F : [-1, 1] \longrightarrow \mathbb{R}$ with $F(u) = \frac{1}{2}u^2$.

Now, that we are set up with convexity, we want to introduce an important concept which is used in this thesis.

Definition 2.17 (Legendre-Fenchel conjugate) *Let $F : X \longrightarrow \mathbb{R}$. We define the Legendre-Fenchel conjugate F^* of F for any $p \in Y$ by*

$$F^*(p) = \sup_{u \in X} \left(\langle p, u \rangle - F(u) \right). \tag{2.5}$$

Remark 2.18

- *Without a proof we state that F^* - as the supremum of linear, continuous functions - is convex and lower-semicontinous, even F is not convex. In addition, F^* is proper if F is convex and proper.*
- *In some literature the Legendre-Fenchel conjugate is also denoted as convex conjugate. We use these two expressions equivalently.*

Theorem 2.19 *Let $F \in \Gamma_0(X)$, then $F^{**} = F$.*

The theorem assures that for a convex function F we can rewrite Equation (2.5) into

$$F(u) = \left(F^*(p)\right)^* (u) = \sup_{p \in Y} \left(\langle u, p \rangle - F^*(p) \right).$$

A proof of this theorem can be found in [22].

Example 2.20 *Let us view some examples on the Legendre-Fenchel conjugate.*

1. *The Legendre-Fenchel conjugate of the indicator function of a set $C \subseteq X$ is given by*

   $$\delta_C^*(p) = \sup_{u \in X} \left(\langle p, u \rangle - \delta_C(u) \right) = \sup_{u \in C} \langle p, u \rangle,$$

   which is the support function of the set C.

2. *Let $F : X \to \mathbb{R}$ be a convex function and $\alpha > 0$. Then the convex conjugate of $\alpha F(u)$ is given by*

   $$\alpha F^*(\frac{p}{\alpha}).$$

   To verify this equality, we compute

   $$F^*(p) = \sup_{u \in X} \left(\langle p, u \rangle - \alpha F(u) \right) = \alpha \underbrace{\sup_{u \in X} \left(\langle \frac{p}{\alpha}, u \rangle - F(u) \right)}_{= F^*(\frac{p}{\alpha})} = \alpha F^*(\frac{p}{\alpha}).$$

3. *Now, let $|| \cdot ||$ be a norm on X with dual norm $|| \cdot ||_*$ on Y. We show that*

   $$F^*(p) = \begin{cases} 0 & \text{if } ||p||_* \leq 1 \\ \infty & \text{else} \end{cases} = \delta_{||p||_* \leq 1}(p),$$

   i.e. the convex conjugate of a norm is the indicator function of the unit ball of its dual norm. In order to show this, we compute

   $$F^*(p) = \sup_{u \in X} \left(\langle p, u \rangle - ||u|| \right).$$

First assume that $||p||_* > 1$. *By definition of the dual norm* $||p||_* = \sup_{||x||\leq 1} |\langle p, x\rangle|$ *there is a* $x \in \mathbb{R}^n$ *for which we observe that* $\langle p, x\rangle > 1$. *We set* $u = tx$ *and let* $t \longrightarrow \infty$, *then we get*

$$\langle p, u\rangle - ||u|| = t(\langle p, x\rangle - ||x||) \longrightarrow \infty.$$

This shows $F^*(p) = \infty$. *On the other hand if we assume that* $||p||_* \leq 1$, *Cauchy-Schwarz-inequality assures*

$$\langle p, u\rangle \leq ||u||\,||p||_*$$

for all $u \in X$. *This implies*

$$\langle p, u\rangle - ||u|| \leq ||u||\,||p||_* - ||u|| = ||u||\,(||p||_* - 1) \leq 0.$$

Since $||p||_* \leq 1$ *holds, we need to choose* $u = 0$ *to attain the supremum in* $F^*(p)$. *This shows that* $F^* = \sup_{u\in X}(\langle p, 0\rangle - ||0||) = 0$.

4. *As a last example we show that for a function* $F(u) = \frac{1}{2}||u||^2$ *its convex conjugate is given by* $F^*(p) = \frac{1}{2}||p||_*^2$. *With Cauchy-Schwarz-inequality we have*

$$\langle p, u\rangle - \frac{||u||^2}{2} \leq ||p||_*\,||u|| - \frac{||u||^2}{2},$$

where the righthand side is a quadratic function of $||u||$. *Define* $H(x) := -\frac{x^2}{2} + y\,x$, *then the maximal value of* H *is* $x = y$, *since* $H^{'}(x) = -x + y$ *and* $H^{''}(x) = -1 < 0$, *then* $H^{'}(x) = 0$ *implies* $x = y$. *Setting* $x = ||u||$ *and* $y = ||p||_*$ *we have* $||u|| = ||p||_*$. *It follows*

$$\langle p, u\rangle - \frac{||u||^2}{2} \leq ||p||_*||u|| - \frac{||u||^2}{2} = ||p||_*||p||_* - \frac{||p||_*}{2} = \frac{||p||_*^2}{2}.$$

This shows that $F^*(p) = \sup_{u\in X}\left(\langle u, p\rangle - \frac{||u||^2}{2}\right) \leq \frac{||p||_*^2}{2}$. *Let us now proof the inequality in the other direction. We assume that we find an arbitrary* u *which satisfies* $\langle u, p\rangle = ||u||\,||p||_*$. *Further, we assume that* $||u|| = ||p||_*$. *Then we have*

$$\langle p, u\rangle - \frac{||u||^2}{2} = ||u||\,||p||_* - \frac{||u||^2}{2} = \frac{||p||_*^2}{2},$$

and conclude

$$\frac{||p|_*^2}{2} = \langle p, u\rangle - \frac{||u||^2}{2} \leq \sup_{u\in X}\left(\langle p, u\rangle - \frac{||u||^2}{2}\right) = F^*(p).$$

From these two inequalities it follows that $F^*(p) = \frac{||p||_*^2}{2}$.

Another important property of functions is differentiability. Unfortunately, in some cases

a function F is not differentiable everywhere. For instance the absolute value function is not differentiable in 0. If a non-differentiable function is convex, we have a generalization of the derivative, called subderivative or subdifferential. For this, we want to define the subdifferential and corresponding to it, the subgradient.

Definition 2.21 (Subgradient, Subdifferential) *Let $X \subseteq \mathbb{R}^n$ be an open, convex set and $F : X \longrightarrow \mathbb{R}$ a convex function. A vector y is called subgradient of F in $u_0 \in X$, if*

$$F(u) \geq F(u_0) + \langle y, u - u_0 \rangle \ \forall u \in X. \tag{2.6}$$

The set

$$\partial F(u_0) = \{y \in X : \ F(u) \geq F(u_0) + \langle y, u - u_0 \rangle \ \forall u \in \mathrm{dom}(F)\}$$

is called subdifferential of F in $u_0 \in X$ and $\mathrm{dom}(\partial F) = \{u : \partial F(u) \neq \emptyset\} \subseteq F$.

If F and G are differentiable, we have $\partial(F+G) = \partial F + \partial G$. For the subdifferential this holds under some conditions. We want to give the corresponding proposition without a proof, for which we refer to [22]. But, we state that in our computations this equality can always be used.

Proposition 2.22 *Let F, G be convex and assume $\mathrm{int}(\mathrm{dom}G) \cap \mathrm{dom}F \neq \emptyset$: then*

$$\partial(F + G) = \partial F + \partial G.$$

To illustrate the subgradient and subdifferential, respectively, we want to give some examples.

Example 2.23 *1. Take the absolute value function $F(u) = |u|$ in $\mathbb{R}$, defined by*

$$F(u) = \begin{cases} u & \text{if } u \geq 0, \\ -u & \text{else.} \end{cases}$$

Since $F(u)$ is not differentiable in 0, but on $\mathbb{R} \setminus \{0\}$, we compute the subgradient y at a point u_0 by

- *$\partial F(u_0) = 1$ if $u_0 > 0$,*
- *$\partial F(u_0) = -1$ if $u_0 < 0$,*
- *and finally if $u_0 = 0$*

$$F(0) + y(u - 0) \leq F(u) \iff yu \leq |u|.$$

If $u \geq 0$ this is equivalent to $y \leq 1$. If $u < 0$ we get

$$yu \leq |u| \iff yu \leq -u \iff y \geq -1.$$

We get the subgradient in u_0 with

$$y = \begin{cases} 1 & \text{if } u_0 > 0 \\ -1 & \text{if } u_0 < 0 \\ \text{[-1, 1]} & \text{if } u_0 = 0 \end{cases}$$

and in addition the subdifferential for all $u \in \mathrm{dom}(F(u))$

$$\partial F(u) = \begin{cases} 1 & \text{if } u > 0 \\ -1 & \text{if } u < 0 \\ \text{[-1,1]} & \text{if } u = 0 \end{cases}.$$

2. *Let $F(u) = ||u||_1$ be the $l1$ norm, which is defined by*

$$||u||_1 := \sum_{i=1}^{n} |u_i|,$$

for $u \in \mathbb{R}^n$. As a non-differentiable function in $u_0 = 0$, we seek for the subdifferential. Looking at the i-th component of the subgradient, we evaluate the subgradient of the absolute value function. With the first example, we know that $y_i = -1$, if $u_{0_i} < 0$ and $y_i = 1$, if $u_{0_i} > 0$. In the case, where $u_{0_i} = 0$, we have $y_i \in [-1, 1]$. Overall, we set

$$y_i = \begin{cases} 1 & \text{if } u_{0_i} > 0 \\ -1 & \text{if } u_{0_i} < 0 \\ \text{[-1, 1]} & \text{if } u_{0_i} = 0 \end{cases}$$

for all $i = 1, ..., n$. Note, that for the whole vector y the inequality $||y||_\infty \leq 1$ is satisfied. Then, the subdifferential is given by

$$\partial F(u) = \{y : ||y||_\infty \leq 1, \langle y, u \rangle = ||x||_1\}.$$

The following proposition can be found in the work of Pock and Chambolle ([11]). We provide it without a proof.

Proposition 2.24 *Let F be convex, then $\hat{u} \in \arg\min_{u \in X} F(u)$ if and only if $0 \in \partial F(\hat{u})$.*

Another important property of convex minimization problems is summarized in the next proposition. It can be found in [22], along with a proof.

Proposition 2.25 *Let $F \in \Gamma_0(X)$. Then if $\hat{u}$ is a local minimum of F, then $\hat{u}$ is in fact a global minimum, i.e.*

$$\hat{u} \in \arg\min_{u \in X} F(u).$$

These two propositions give us some important statements. If we find a (local) minimizer of a convex minimization problem, we already know that this minimizer is indeed the global minimal value. Further, we can verify that a solution of a minimization problem

is valid, if zero is an element of the subdifferential of the cost function at this specific minimal point. With this, we later compute the proximity operator of the functions in our underlying models. This operator is defined in the next definition and is also used in Moreau's theorem (cf. Theorem 2.27).

Definition 2.26 (Projection Operator, Proximity Operator, Moreau Envelop) *For any non-empty closed set C the projection operator for an arbitrary point $z \notin C$ onto the set C is defined by*

$$\Pi_C(z) = \arg\min_{u \in C} \frac{1}{2}||z-u||_2^2, \tag{2.7}$$

meaning the orthogonal projection onto C. We define the proximity operator in a similar fashion due to

$$\text{prox}_F^\alpha(z) = \arg\min_{u \in X} \frac{1}{2}||u-z||_2^2 + \alpha\, F(u), \tag{2.8}$$

and the Moreau envelop as

$$M_F^\alpha(z) = \min_{u \in X} \frac{1}{2}||u-z||_2^2 + \alpha\, F(u),$$

for any $\alpha > 0$.

The next theorem, namely Moreau's theorem, was first proven in 1965 by Jean J. Moreau and can be found in [22], along with a proof.

Theorem 2.27 (Moreau's Theorem) *Let F be a closed proper convex function on $\mathbb{R}^n$ and $F^*(p)$ its convex conjugate. Then*

$$M_F^1(z) + M_{F^*}^1(z) = \frac{1}{2}||z||_2^2,$$

i.e. for each $z \in \mathbb{R}^n$ one has

$$\min_{u \in X}\left(\frac{1}{2}||u-z||_2^2 + F(u)\right) + \min_{p \in Y}\left(\frac{1}{2}||p-z||_2^2 + F^*(p)\right) = \frac{1}{2}||z||_2^2,$$

where both minima are uniquely attained. The unique vectors u and p for which the respective minima are attained for a given z are the unique vectors u and p such that

$$z = u + p, \quad p \in \partial F(u), \tag{2.9}$$

and they are given by

$$u = \text{prox}_F^1(z), \quad p = \text{prox}_{F^*}^1(z). \tag{2.10}$$

Remark 2.28 *Another way to denote the proximity operator is the notation*

$$\text{prox}_F^\alpha(z) = (\text{Id} + \alpha\partial F)^{-1}(z). \tag{2.11}$$

It can be derived by Proposition 2.24. So, we assume that $\hat{u} \in \operatorname{prox}_F^\alpha$. *Then*

$$0 \in \partial \left(\alpha F(\hat{u}) + \frac{1}{2}||\hat{u} - z||_2^2 \right) = \alpha\, \partial F(\hat{u}) + \hat{u} - z.$$

But this can be rewritten to

$$z \in \alpha\, \partial F(\hat{u}) + \hat{u} \iff \hat{u} = (\mathrm{Id} + \alpha\, \partial F)^{-1}(z).$$

According to Chambolle et al. in [11], we get due to Moreau the following identity:

$$z = (\mathrm{Id} + \alpha\, \partial F)^{-1}(z) + \alpha \left(\mathrm{Id} + \frac{1}{\alpha} \partial F^* \right)^{-1} \left(\frac{z}{\alpha} \right). \tag{2.12}$$

This equation is best known as Moreau's identity. For $\alpha = 1$ *we observe*

$$z = (\mathrm{Id} + \partial F)^{-1}(z) + (\mathrm{Id} + \partial F^*)^{-1}(z).$$

To this end, let us discuss some examples on the projection and proximity operator, respectively.

Example 2.29 • *The lp projection of a point* $\tilde{u} \in \mathbb{R}^n$ *onto the* l^p *unit sphere with* $\tilde{u} \notin C = \{u : ||u||_p \leq 1\}$ *is given by the following minimization problem:*

$$\begin{array}{ll} \min\limits_{u \in \mathbb{R}^n} & \frac{1}{2}||u - \tilde{u}||_2^2 \\ \text{subject to} & ||u||_p \leq 1. \end{array}$$

*According to [24] the l*2 *and the* l^∞ *projection are given by*

1. $\Pi_{||\cdot||_2 \leq 1}(\tilde{u}) = \frac{\tilde{u}}{\max(1, |\tilde{u}|)}$, *or equivalently*

$$u^* = \begin{cases} \tilde{u} & \text{if } ||\tilde{u}||_2 \leq 1 \\ \frac{\tilde{u}}{||\tilde{u}||_2} & \text{otherwise} \end{cases}$$

2. *and* $\Pi_{||\cdot||_\infty \leq 1}(\tilde{u}) = \min(1, \max(-1, \tilde{u}))$, *or for all* $k = 1, ..., n$

$$u_k^* = \begin{cases} \tilde{u}_k & \text{if } |\tilde{u}_k| \leq 1 \\ 1 & \text{otherwise} \end{cases}.$$

- *If* $F(u) = \delta_C(u)$, *we compute the proximity operator by*

$$\operatorname{prox}_F^\alpha(\tilde{u}) = \min_{u \in X} \frac{||u - \tilde{u}||_2^2}{2} + \alpha\, \delta_C(u) = \min_{u \in C} \frac{||u - \tilde{u}||_2^2}{2} = \Pi_C(\tilde{u}).$$

We see a connection between the two operators. Proximation of the indicator function is the same as doing a euclidean projection onto the corresponding set C.

2.3 Total Variation

The definition of the total variation, the variation of functions and functions of bounded variation can be found in [13], together with the first example in this section. The total variation is the basis when talking about variational methods in image processing. We cover this topic superficial, but this is enough in our context.

Definition 2.30 (Total Variation) *Let Ω be an open subset of $\mathbb{R}^n$. For a function $u \in L^1(\Omega)$, the* total variation *of u in Ω is defined as*

$$\mathrm{TV}(u) = \sup\left\{ -\int_\Omega u \operatorname{div}(\varphi)\, dx : \varphi \in C_0^1(\Omega, \mathbb{R}^n), |\varphi(x)| \leq 1, \forall x \in \Omega \right\}.$$

Example 2.31 *Let $u \in W_1^1(\Omega)$, then integration by parts and φ having compact support leads to*

$$\begin{aligned} -\langle u, \operatorname{div}(\varphi)\rangle &= -\int_\Omega u \operatorname{div}(\varphi)\, dx \\ &= -\left(\underbrace{\int_{\partial\Omega} u\, \varphi\, d\mathcal{H}^{n-1}}_{=0} - \int_\Omega \nabla u\, \varphi\, dx \right) \\ &= \int_\Omega \nabla u\, \varphi\, dx \\ &= \langle \nabla u, \varphi \rangle, \end{aligned} \tag{2.13}$$

for every $\varphi \in C_0^1(\Omega, \mathbb{R}^n)$ so that

$$\mathrm{TV}(u) = \int_\Omega |\nabla f| dx. \tag{2.14}$$

The idea to derive Equation (2.14) is that we first evaluate the case where

$$\mathrm{TV}(u) \leq \int_\Omega |\nabla f| dx.$$

This can be seen by using Cauchy-Schwarz-inequality

$$\begin{aligned} \int_\Omega \nabla u\, \varphi\, dx &\leq \left| \int_\Omega \nabla u\, \varphi dx \right| dx \\ &\leq \int_\Omega |\nabla u| \underbrace{|\varphi|}_{\leq 1}\, dx \leq \int_\Omega |\nabla u|\, dx. \end{aligned}$$

For the inequality in the other direction, the key idea would be to set $\varphi = \frac{\nabla u}{|\nabla u|}$. Then, $|\varphi| = 1$. Since $\varphi \notin C_0^1$ we would approximate φ in L1. This space lies close in the space of smooth functions with compact support and for that satisfies the assumptions.

Remark 2.32 *Note that $W_1^1(\Omega)$ denotes the L1-Sobolev-Space of Ω, which means that $u \in L^1(\Omega)$ and $Du \in L^1(\Omega)$. Here, Du is meant in the sense of distributional derivatives. Further note, that the total variation is convex and lower-semicontinous. See [11] for details.*

To get a better intuition of the total variation we want to propose the defintion of the variation of functions $u \in [a, b] \subseteq \mathbb{R}$.

Definition 2.33 (Variation of a function) *Let $u : \mathbb{R} \longrightarrow \mathbb{R}$ and $a < b$ be real numbers. Then define the variation of u on $[a, b]$ as*

$$V_a^b(u) = \sup \left\{ \sum_{i=1}^{n} |u(x_i) - u(x_{i-1})| : n \in \mathbb{N} \,\text{and}\, a = x_0 < x_1 < ... < x_n = b \right\}.$$

This definition is well known and suited for functions from $\mathbb{R}$ to $\mathbb{R}$. According to [13] this definition is ill-posed in the sense of measures and integrals.

Example 2.34 *Let $a, b \in \mathbb{R}$ with $a < b$. If $u : [a, b] \longrightarrow \mathbb{R}$ is monotonically increasing, then for any $a = x_0 < x_1 < ... < x_n = b$ we observe*

$$\begin{aligned} & \sum_{i=1}^{n} |u(x_i) - u(x_{i-1})| = \sum_{i=1}^{n} u(x_i) - u(x_{i-1}) \\ = \;& u(x_1) - u(x_0) + u(x_2) - u(x_1) + ... + u(x_{n-1}) - u(x_{n-2}) + u(x_n) - u(x_{n-1}) \\ = \;& u(x_n) - u(x_0) = u(b) - u(a). \end{aligned}$$

It follows that $V_a^b(u) = \sup\{u(b) - u(a)\} = u(b) - u(a)$.

Another class of functions, which is considered in Chapter 4 are (special) functions of bounded variation. Even though we do not explicitly make use of this class of functions, we want to give the definitions for the sake of completeness. Further, (special) functions of bounded variation are strongly related to total variation as the following definition ensures (see for instance [13]).

Definition 2.35 (Functions of bounded variation) *A function $u \in L^1(\Omega)$ is said to have bounded variation in Ω if $\mathrm{TV}(u) < \infty$. We define $BV(\Omega)$ as the space of all functions in $L^1(\Omega)$ with bounded variation. This space is equipped with the norm*

$$||u||_{BV} = ||u||_{L^1} + \int_\Omega |Du|,$$

and for that it is a Banach space.

Relating to Example 2.34 we find that $u \in BV(\mathbb{R})$, since $V_a^b(u) = u(b) - u(a) < \infty$ for $a, b \in \mathbb{R}$, $a < b$. Additionally, BV-functions are proper and we have if $u \in BV(\Omega)$ then also $u \in \Gamma_0(\Omega)$, because the total varition is convex and l.s.c. (see again [11]).

Definition 2.36 (Special functions of bounded variation [21]) *SBV denotes the special functions of bounded variation, i.e. functions u of bounded variation for which the derivative Du is the sum of an absolutely continuous part $\nabla u \cdot dx$ and a discontinuous singular part S_u.*

This class of functions plays a role, when applying convex relaxation techniques to the Mumford-Shah functional. But, as mentioned, the definition is only given for the sake of completeness in this thesis and we will not explicitly make use of it.

3 The ROF, TVL1 and Mumford-Shah Functionals

In this chapter we consider convex optimization problems, namely minimization of the ROF and TVL1 functional. Furthermore, we seek for a minimizer of the non-convex Mumford-Shah functional. We apply techniques of convex analysis to turn these minimization into saddle-point problems. In the case of the Mumford-Shah functional we also use these techniques, even if the functional itself is non-convex. There is no proof that this strategy yields the minimal value of the Mumford-Shah functional. Nonetheless, this approach leads to high-quality image approximations.
Our goal is to approximate an input image g by a function u. The underlying functionals are modeled with two terms and we try to minimize the energy of each functional to find a good approximation. We consider problems of the form

$$\textit{minimize} \ \ (\text{Functional}) = \textit{minimize} \ \ (\text{Data Fidelity Term} + \text{Regularizer Term})\,.$$

The data fidelity term assures that an approximation u is as close to the input image g as possible. The regularizer adds further assumptions to the model, for example seeks for smoothness of the approximation u in specific regions. Before we consider the three models of this chapter we need some definitions and notations.

3.1 The General Saddle-Point Problem

Let us begin by introducing the general class of problems we consider. For this, we follow Chambolle et al. ([11] Section 3.5 and [3] Chapter 2). We define the map $K : X \longrightarrow Y$ as a continuous linear operator with induced norm

$$||K|| = \max\{||Kx||_Y : x \in X \text{ with } ||x||_X \leq 1\}\,.$$

Furthermore, we define the map $K^* : Y \longrightarrow X$ as the adjoint operator of K.
Let $u \in X$, $p \in Y$ and define $G : X \longrightarrow [0, +\infty)$ and $F^* : Y \longrightarrow [0, +\infty)$, where $G, F^* \in \Gamma_0$ and F^* being the Legendre-Fenchel conjugate of a convex, l.s.c. function F. We are trying to find a saddle point (u, p) of the following problem:

$$\min_{u \in X} \max_{p \in Y} \left(\langle Ku, p\rangle + G(u) - F^*(p)\right). \tag{3.1}$$

We also call this saddle-point problem the primal-dual problem, where u is the primal and p the dual variable. We define the corresponding primal problem by

$$\min_{u \in X} \left(G(u) + F(Ku) \right) \tag{3.2}$$

and the dual problem by

$$\max_{p \in Y} -(G^*(-K^*p) + F^*(p)). \tag{3.3}$$

We can show that these three different classes of problems are equivalent if F itself is convex. Then, the convex conjugate assures that $F(Ku) = \sup_{p \in Y} \left(\langle p, Ku \rangle - F^*(p) \right)$. Further, the so called *Von Neumann's minimax theorem* must hold, cf. [29]. It states that the objective function needs to be convex in the primal and concave in the dual variable and the feasible set for u or the feasible set for p must be compact. With this we get

$$\begin{aligned}
\min_{u \in X} \left(F(Ku) + G(u) \right) &= \min_{u \in X} \max_{p \in Y} \left(\langle p, Ku \rangle - F^*(p) + G(u) \right) \\
&= \max_{p \in Y} \min_{u \in X} \left(\langle K^*p, u \rangle + G(u) - F^*(p) \right) \\
&= \max_{p \in Y} \left(- \max_{u \in X} - \left(\langle K^*p, u \rangle + G(u) \right) - F^*(p) \right) \\
&= \max_{p \in Y} \left(\underbrace{- \max_{u \in X} \left(\langle -K^*p, u \rangle - G(u) \right)}_{= -G^*(-K^*p)} - F^*(p) \right) \\
&= \max_{p \in Y} -(G^*(-K^*p) + F^*(p)).
\end{aligned}$$

In the case of the ROF and TVL1 functional all conditions are satisfied. Finding the optimal value $\hat{u}$ for the primal problem can for instance be done by a gradient descent in u. On the other hand, if $\hat{p}$ is an optimal value, we can find it by a gradient ascent in p of the dual problem. The basic idea of the primal-dual algorithm, presented in the next section, is then to do both, a gradient descent in u and a gradient ascent in p, simultaneously, to solve the saddle-point problem.

3.2 A Primal-Dual Algorithm

According to Chambolle et al. in [11], the idea of the primal-dual algorithm goes back to Arrow and Hurwicz, see [5], and the first time this algorithm was stated in such a framework was probably in [4]. As mentioned in the last section, we simultaneously compute a gradient descent in u and a gradient ascent in p. After each ascent and descent, respectively, we apply the proximity operator for F^* and G, respectively, to the

estimated values. Choosing time-steps $\sigma, \tau > 0$ we get

$$\begin{cases} p^{n+1} = (\mathrm{Id} + \sigma\,\partial\, F^*)^{-1}(p^n + \sigma\, K u^n) \\ u^{n+1} = (\mathrm{Id} + \tau\,\partial\, G)^{-1}(u^n - \tau\, K^* p^{n+1}) \end{cases}.$$

The scheme was first proposed in a paper of Zhu and Chan in [30]. Unfortunately, there is no proof of convergence for it. But in 2009 Pock, Cremers, Bischof and Chambolle published a paper ([21]) that we also consider in Chapter 4, whose contribution was an extension of the above scheme for which convergence can be guaranteed under certain conditions. The idea is to attach an additional extrapolation step to the algorithm, as seen in line three of the Primal-Dual Algorithm 3.1. In [3], Pock and Chambolle generalized this algorithm. They also proposed some variations of the algorithm itself. Depending on the properties of the corresponding functions F^* and G one can derive a better convergence rate. We will not show details of this and only make use of the first algorithm proposed in [3]. Additionally, we consider a variant of this algorithm, proposed in [26]. Furthermore, we will not provide a proof of convergence for these algorithms. For details, we refer again to [3]. The general primal-dual algorithm can be written as follows:

Algorithm 3.1 (Primal-Dual Algorithm) *Choose $(u^0, p^0) \in X \times Y$ and let $\bar{u}^0 = u^0$. Further let $\tau, \sigma > 0$ and $\theta \in [0, 1]$. Then, we let for each $n \geq 0$*

$$\begin{cases} p^{n+1} = (\mathrm{Id} + \sigma\,\partial\, F^*)^{-1}(p^n + \sigma\, K \bar{u}^n) \\ u^{n+1} = (\mathrm{Id} + \tau\,\partial\, G)^{-1}(u^n - \tau\, K^* p^{n+1}) \\ \bar{u}^{n+1} = u^{n+1} + \theta(u^{n+1} - u^n) \end{cases}.$$

Computing $p^n + \sigma\, K\bar{u}^n$ and $u^n - \tau\, K^* p^{n+1}$ is easy, since these two are sums of vectors. The last line of this scheme is again the summation of vectors. In the next section, we introduce the notation for K and K^*. We will see that computing the linear operators applied to u and p is quite simple. Computing the proximity operators in line one and two of the algorithm needs further work and varies within each model. We will compute the proximity operators for each model explicitly in the corresponding sections. If all updates for u, p and $\bar{u}$ are computed, this iteration scheme solves saddle-point problems of the form (3.1).

Remark 3.2 *In the theorem of convergence, found in [3], we have that convergence is guaranteed if we set the chosen parameters τ and σ appropriately. Additionally, the algorithm is independent of initialization. As long as the inequality $\tau\sigma L^2 < 1$ is satisfied, where $L = ||K||$ is the bound on the norm of the linear operator K, convergence is guaranteed. The better the choice for these parameters beforehand, the faster the algorithm converges. Estimating the best time-steps is ongoing research. A first approach was a preconditioning scheme, published in [28] by Pock and Chambolle. Of course, it is not the best way having an algorithm which is dependent on manually chosen parameters. But, on the other hand, the parameter τ is controllable, with respect to the fact that σ can be computed from τ by $\sigma = \frac{1}{\tau * L^2}$. Despite the fact that we then have $\tau\sigma L^2 = 1$, convergence is still given, according to the first remark in [3].*

Note that in our computations we always set $\theta = 1$. So, the third line of the algorithm becomes

$$\bar{u}^{n+1} = 2u^{n+1} - u^n.$$

The convergence rate for our primal-dual algorithm is $\mathcal{O}\left(\frac{1}{N}\right)$. A variant of this proposed algorithm can be found in the work of Strekalovskiy and Cremers ([26]) and will be used for the real-time minimization framework of the Mumford-Shah functional.

Algorithm 3.3 (Real-Time Primal-Dual Algorithm) *Choose $(u^0, p^0) \in X \times Y$ and let $\bar{u}^0 = u^0 = f$. Further let $\tau_0 = \frac{1}{2d}, \sigma_0 = \frac{1}{2}$. Then, we let for each $n \geq 0$ as long as $||u^{n+1} - u^n|| < \varepsilon$*

$$\begin{cases} p^{n+1} = (\mathrm{Id} + \sigma_n \, \partial \, F^*)^{-1}(p^n + \sigma_n \, K\bar{u}^n) \\ u^{n+1} = (\mathrm{Id} + \tau_n \, \partial \, G)^{-1}(u^n - \tau_n \, K^* p^{n+1}) \\ \theta_n = \frac{1}{\sqrt{1+4\tau_n}}, \; \tau_{n+1} = \theta_n \tau_n, \; \sigma_{n+1} = \frac{\sigma_n}{\theta_n} \\ \bar{u}^{n+1} = u^{n+1} + \theta_n(u^{n+1} - u^n) \end{cases}.$$

The difference of this scheme to Algorithm 3.1 is that τ and σ are determined by initialization. Further, they are updated, together with θ, in each iteration step. These updates lead to a better convergence rate, namely $\mathcal{O}\left(\frac{1}{N^2}\right)$. But, this approach is only applicable if G or F^* is uniformly convex (see also [3], Algorithm 2).

3.3 Discrete Setting

On computers images are discrete objects with discrete pixel locations. For that, we need a proper notation for our framework. In our discrete setting we consider a regular pixel grid of size $N \times M$ and set

$$\mathcal{G} = \Big\{(i,j) : i = 1, ..., N \text{ and } j = 1, ..., M\Big\}, \tag{3.4}$$

where the indices (i, j) denote the discrete locations in the pixel grid. We define an image $u \in X : \mathcal{G} \longrightarrow \mathbb{R}$ where $X \in \mathbb{R}^{N \times M}$ is a finite dimensional vector space equipped with the standard inner product

$$\langle u, v \rangle_X = u^T v = \sum_{i=1}^{N} \sum_{j=1}^{M} u_{i,j} v_{i,j}, \;\; u, v \in X \tag{3.5}$$

and the standard euclidean norm

$$||u||_2 = \langle u, u \rangle^{\frac{1}{2}}, \;\; u \in X.$$

Furthermore, let $Y = X \times X$ be the dual space of X, equipped with the inner product defined by

$$\langle p, q \rangle_Y = p^T q = \sum_{i=1}^{N} \sum_{j=1}^{M} p_{i,j}^1 q_{i,j}^1 + p_{i,j}^2 q_{i,j}^2, \tag{3.6}$$

with $p_{i,j} = \left(p_{i,j}^1, p_{i,j}^2\right)^T, q_{i,j} = \left(q_{i,j}^1, q_{i,j}^2\right)^T \in Y$, also equipped with the euclidean norm. Additionally, we define two important norms. The discrete isotropic total variation norm is given by

$$||\nabla u||_1 = \sum_{i=1}^{N} \sum_{j=1}^{M} |(\nabla u)_{i,j}|, \text{ where } |(\nabla u)_{i,j}| = \sqrt{((\nabla u)_{i,j}^1)^2 + ((\nabla u)_{i,j}^2)^2}$$

and the discrete maximum norm by

$$||p||_\infty = \max_{i,j} |p_{i,j}|, \text{ where } |p_{i,j}| = \sqrt{(p_{i,j}^1)^2 + (p_{i,j}^2)^2}.$$

It is left to present the representation of the linear operator K. Relating to the definition of the total variation in Equation (2.13) we have $\langle \nabla u, p \rangle = -\langle u, \operatorname{div}(p) \rangle$. In a finite dimensional vector space we have $\langle \nabla u, p \rangle = \langle u, \nabla^T p \rangle$, if ∇ is a matrix. For this, we set $K = \nabla$ and $K^* = \nabla^T = -\operatorname{div}$.

Definition 3.4 (Discrete gradient operator) *We define the discrete gradient of a vector $u \in X$ by $\nabla u = ((\partial_i u)_{i,j}, (\partial_j u)_{i,j})^T$ using forward differences with Neumann boundary conditions, i.e.*

$$(\partial_i u)_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < N \\ 0 & \text{if } i = N \end{cases},$$
$$(\partial_j u)_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < M \\ 0 & \text{if } j = M \end{cases}.$$

Definition 3.5 (Discrete divergence operator) *We define the discrete divergence of a vector $p \in Y$ by $\nabla^T p = (\partial_i p^1)_{i,j} + (\partial_j p^2)_{i,j}$ using backward differences with Dirichlet boundary conditions, i.e.*

$$(\partial_i p^1)_{i,j} = \begin{cases} p_{i,j}^1 - p_{i-1,j}^1 & \text{if } 1 < i < N \\ p_{i,j}^1 & \text{if } i = 1 \\ -p_{i-1,j}^1 & \text{if } i = N \end{cases},$$
$$(\partial_j p^2)_{i,j} = \begin{cases} p_{i,j}^2 - p_{i,j-1}^2 & \text{if } 1 < j < M \\ p_{i,j}^2 & \text{if } j = 1 \\ -p_{i,j-1}^2 & \text{if } j = M \end{cases}.$$

We show that these two definitions resemble $K = \nabla$ and $K^* = \nabla^T = -\text{div}$. For that, we first take a look at the matrices $(\nabla)^1$ and $(\nabla)^2$ of the gradient operator in Definition 3.4. We have

$$(\nabla)^1 = \begin{pmatrix} -1 & & & \\ 1 & -1 & & \\ & \ddots & \ddots & \\ & & 1 & -1 \\ & & & 1 \end{pmatrix} \text{ and } (\nabla)^2 = \begin{pmatrix} -1 & 1 & & \\ & -1 & 1 & \\ & & \ddots & \ddots \\ & & & -1 \\ & & & 0 \end{pmatrix},$$

as the two components of ∇. On the other hand, if we analyse the two components of ∇^T, we observe

$$(\nabla^T)^1 = \begin{pmatrix} 1 & -1 & & \\ & 1 & -1 & \\ & & \ddots & \ddots \\ & & & 1 \\ & & & 0 \end{pmatrix} \text{ and } (\nabla^T)^2 = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ & & & -1 \end{pmatrix}.$$

With this representation, we see that

$$-((\nabla)^1)^T = -(\text{div})^1 = (\nabla^T)^1 \quad \text{and} \quad -((\nabla)^2)^T = -(\text{div})^2 = (\nabla^T)^2.$$

We additionally compute the bound on the norm of these two operators, where we follow the notation of Pock and Chambolle in [3].

Proposition 3.6 (Bound on the norm of ∇) *The bound on the norm of the proposed discrete linear operators is given by*

$$L^2 = ||K|| = ||\nabla|| = ||\nabla^T||^2 \leq 8.$$

Proof We set $p^1_{0,j} = 0$ and $p^2_{i,0} = 0$. Then we compute

$$\begin{aligned} L^2 &= ||\nabla||^2 = ||\nabla^T||^2 = \max_{||p||_Y \leq 1} ||\nabla^T p||_X^2 \\ &= \max_{||p||_Y \leq 1} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(p^1_{i,j} - p^1_{i-1,j} + p^2_{i,j} - p^2_{i,j-1}\right)^2 \\ &\overbrace{\leq}^{(*)} 4 \max_{||p||_Y \leq 1} \sum_{i=1}^{N} \sum_{j=1}^{M} (p^1_{i,j})^2 + (p^1_{i-1,j})^2 + (p^2_{i,j})^2 + (p^2_{i,j-1})^2 \\ &\leq 8 \max_{||p||_Y \leq 1} ||p||_Y^2 = 8, \end{aligned}$$

in $(*)$ we used Young's inequality $ab \leq \frac{a^p}{p} + \frac{b^q}{q}$ twice, with $p = q = 2$. ■

3.4 The ROF Model

Now, that everything is defined properly we consider the minimization of the ROF functional, named after Leonid I. Rudin, Stanley Osher and Emad Fatemi. This functional was first proposed in 1992 in [23]. It is the prototype when talking about variational methods in image processing.

Definition 3.7 (ROF Functional) *Let $\Omega \in \mathbb{R}^n$ be the n-dimensional image domain, $u \in W_1^1(\Omega)$ and $g \in L^1(\Omega)$ an input image. Then the ROF functional is defined as*

$$E_{ROF}(u) = \mathrm{TV}(u) + \frac{\lambda}{2}\int_\Omega |u-g|^2\,dx = \int_\Omega |\nabla u|\,dx + \frac{\lambda}{2}\int_\Omega |u-g|^2\,dx \tag{3.7}$$

and we seek to compute the minimizer over all u.

The appearing parameter λ is used to handle the tradeoff between the regularizer, the total variation, and the data fidelity term which is a least squares term. Using a larger value for λ we get an approximation u closer to the input image g. If we choose λ to be small the weighting of the regularizer is higher. For that, the approximation u is smoother, because the gradient in the total variation penalizes jumps in the approximation u. Reformulation of Equation (3.7) into a discrete minimization problem gives us:

$$\min_{u\in X} E_{ROF}(u) = \min_{u\in X} ||\nabla u||_1 + \frac{\lambda}{2}||u-g||_2^2, \tag{3.8}$$

with $u \in X$ being the unknown approximation and $g \in X$ the given data. This minimization problem is convex, because the cost function as a sum of norms is convex. Let us now rewrite the minimization of the ROF functional into a saddle-point formulation.

3.4.1 ROF Model as Saddle-Point Problem

In order to obtain a saddle-point formulation of Equation (3.8), we identify the functions $F(\nabla u) = ||\nabla u||_1$ and $G(u) = \frac{\lambda}{2}||u-g||_2^2$, respectively. Hence, we face

$$\min_{u\in X}\ F(\nabla u) + G(u) = \min_{u\in X}\ ||\nabla u||_1 + \frac{\lambda}{2}||u-g||_2^2. \tag{3.9}$$

Since F is a convex function we can express it in terms of its convex conjugate by

$$F(\nabla u) = F^{**}(\nabla u) = \sup_{p\in Y}\left(\langle p, \nabla u\rangle - F^*(p)\right).$$

We observe, by inserting the expression of $F(\nabla u)$ into Equation (3.9), the saddle-point problem

$$\min_{u\in X}\max_{p\in Y}\ \langle p, \nabla u\rangle - F^*(p) + G(u).$$

It remains to show how $F^*(p)$ is represented. From Example 2.20 2. and the fact that the conjugate norm of the $l1$ norm is the l^∞ norm, we have

$$F^*(p) = \begin{cases} 0 & \text{if } ||p||_\infty \leq 1 \\ \infty & \text{else} \end{cases},$$

or equivalently $F^*(p) = \delta_P(p)$ for a set P given by

$$P = \{p \in Y : ||p||_\infty \leq 1\}. \tag{3.10}$$

Using this information we seek to solve the saddle-point problem

$$\min_{u \in X} \max_{p \in Y} \langle p, \nabla u \rangle + \frac{\lambda}{2}||u - g||_2^2 - \delta_P(p). \tag{3.11}$$

We also want to propose the dual formluation of Equation (3.11). For that, we need to compute G^*. Set $v = u - g$ which is equivalent to $u = v + g$ and compute

$$\begin{aligned} G^*(p) &= \sup_{u \in X} \left(\langle u, p \rangle - \frac{\lambda}{2}||u - g||_2^2 \right) \\ &= \sup_{v \in X} \left(\langle v + g, p \rangle - \frac{\lambda}{2}||v||_2^2 \right) \\ &= \underbrace{\left(\sup_{v \in X} \langle v, p \rangle - \frac{\lambda}{2}||v||_2^2 \right)}_{= \frac{1}{2\lambda}||p||_2^2 \ (*)} + \langle g, p \rangle \\ &= \frac{1}{2\lambda}||p||_2^2 + \langle g, p \rangle. \end{aligned}$$

In $(*)$ we used Examples 2.20 2. and 4. Inserting F^* and G^*, respectively, into Equation (3.3) and setting $K^* = \nabla^T = -\text{div}$ we get the dual formulation by

$$\begin{aligned} \max_{p \in Y} -(G^*(-K^*p) + F^*(p)) &= \max_{p \in Y} -\left(\frac{1}{2\lambda}|| - \nabla^T p||_2^2 + \langle g, -\nabla^T p \rangle_X + \delta_P(p) \right) \\ &= \max_{p \in Y} -\left(\frac{1}{2\lambda}||\text{div}\,(p)||_2^2 + \langle g, \text{div}\,(p) \rangle_X + \delta_P(p). \right) \end{aligned}$$

Let us now compute the corresponding proximity operators of Problem (3.11) to be able to solve this saddle-point problem using the primal-dual algorithm.

3.4.2 The Proximity Operators of the ROF Model

In Algorithm 3.1 we presented how saddle-point problems can be solved efficiently. In two of the three steps of the proposed primal-dual algorithm we needed the proximity operators

$$(\text{Id} + \sigma\, \partial\, F^*)^{-1} \text{ and } (\text{Id} + \tau\, \partial\, G)^{-1}.$$

Within the saddle-point problem in (3.11) we identify $G(u) = \frac{\lambda}{2}||u-g||_2^2$ and therefore $F^*(p) = \delta_P(p)$. With the identity of Equation (2.8) we compute the proximity operator of the indicator function $\delta_P(p)$ by

$$(\mathrm{Id} + \sigma\,\partial\, F^*)^{-1}(\tilde{p}) = \arg\min_{p\in Y} \frac{||p-\tilde{p}||_2^2}{2} + \sigma\,\delta_P(p) = \arg\min_{p\in P} \frac{||p-\tilde{p}||_2^2}{2}.$$

This is nothing but the euclidean projection of a vector $\tilde{p} \notin P$ onto the convex set P. As we follow Pock and Chambolle in [3], we state that P is the cross product of pointwise $L2$ balls, see also Figure 3.1. So, we compute pointwise euclidean (l2) projections onto P. We get with Example 2.29 1.

$$p = (\mathrm{Id} + \sigma\,\partial\, F^*)^{-1}(\tilde{p}) = \Pi_P(\tilde{p}) \iff p_{i,j} = \frac{\tilde{p}_{i,j}}{\max(1, |\tilde{p}_{i,j}|)},$$

for all $i = 1, ..., N, j = 1, ..., M$.

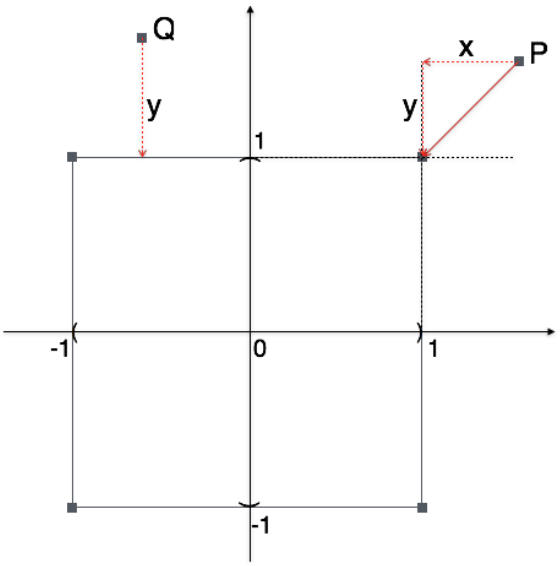


Figure 3.1: The point $P \in \mathbb{R}^2$ needs two pointwise projections. First in the direction of x and afterwards in y direction. The point $Q \in \mathbb{R}^2$ only needs one pointwise projection in the direction of y.

It remains to compute the proximity operator for the function G. Here, we have

$$(\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u}) = \arg\min_{u\in X} \frac{||u-\tilde{u}||_2^2}{2} + \frac{\tau\lambda}{2}||u-g||_2^2 = \arg\min_{u\in X} \mathcal{L}(u).$$

Since $\mathcal{L}(u)$, as a sum of quadratic norms, is convex, a local minimizer is also a global minimizer. Let $\hat{u} \in \arg\min_{u\in X} \mathcal{L}(u)$. With Proposition 2.24 we observe

$$0 \in \partial\mathcal{L}(\hat{u}) \iff 0 \in (\hat{u}-\tilde{u}) + \tau\lambda(\hat{u}-g) \iff (1+\tau\lambda)\hat{u} = \tilde{u} + \tau\lambda g$$

and this implies $\hat{u} = \frac{\tilde{u}+\tau\lambda g}{1+\tau\lambda}$. Overall, the equivalence

$$u = (\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u}) \iff u_{i,j} = \frac{\tilde{u}_{i,j} + \tau\lambda g}{1+\tau\sigma} \tag{3.12}$$

holds pointwise for all $i = 1, ..., N$ and $j = 1, ..., M$. As a last step we show an appropriate stopping criterion to determine convergence of the primal-dual algorithm.

3.4.3 Stopping Criterion

We have several possibilities to determine convergence. As we seek to find the minimal energy of the ROF functional we could stop the algorithm as soon as this energy does not change too much. Hence, we compute

$$|E_{ROF}(u^{n+1}) - E_{ROF}(u^n)| \leq \varepsilon,$$

for a ε small enough. Estimating the whole energy once every iteration step in the primal-dual algorithm would generate computational overhead and for that slow down the code. We partially follow the idea proposed in [26] by Strekalovskiy and Cremers, but use another threshold for ε and evaluate once every iteration

$$||u^{n+1} - u^n|| \leq \varepsilon,$$

with $\varepsilon = \cdot 10^{-6}$. Furthermore, the underlying norm is defined by

$$||\tilde{u}|| = \frac{1}{N \cdot M} \sum_{k=1}^{B} \sum_{i=1}^{N} \sum_{j=1}^{M} |\tilde{u}_{i,j}|.$$

Here, B is the number of color channels. We set $B = 1$ for grayscaled and $B = 3$ for colored images. This computation is not as costly as the calculation of the whole energy function and as Chapter 5 shows, a good criterion to determine convergence.

3.5 The TVL1 Model

The idea of the TVL1 functional is to replace the squared $L2$ norm in the data fidelity term $G(u)$ with the $L1$ norm. This norm is more robust in removing the so called salt and pepper noise, meaning noisy extrem values white or black in an input image g. The name of this functional rises from the fact that it also uses the total variation as a regularization term. The functional is defined as follows:

Definition 3.8 (TVL1 Functional) *Let $\Omega \in \mathbb{R}^n$ be the n-dimensional image domain, $u \in W_1^1(\Omega)$ and $g \in L^1(\Omega)$ an input image. Then the TVL1 functional is defined as*

$$E_{TVL1}(u) = \mathrm{TV}(u) + \lambda \int_\Omega |u - g|\, dx = \int_\Omega |\nabla\, u|\, dx + \lambda \int_\Omega |u - g|\, dx \tag{3.13}$$

and we seek to compute the minimizer of this functional over all u.

The parameter λ handles the tradeoff between the data fidelity and the regularizer term, as seen in the ROF model. In this functional λ controls the total variation term. This means a higher value on λ amounts more smoothness in the approximation u. A lower value assures that u is closer to the input image g. The discrete primal formulation is represented by

$$\min_{u \in X} E_{TVL1}(u) = \min_{u \in X} ||\nabla u||_1 + \lambda ||u - g||_1. \tag{3.14}$$

As done before, we want to turn this minimization problem into a saddle-point problem. Then, we are able to apply the primal-dual algorithm and for that solve this optimization problem.

3.5.1 TVL1 as Saddle-Point Problem

Let us first note that the function $F(\nabla u)$, namely the total variation, remains the same as in the ROF model. Expressing F in terms of its convex conjugate, as in Subsection 3.4.1, we obtain

$$\min_{u \in X} \max_{p \in Y} \langle p, \nabla u \rangle - F^*(p) + G(u).$$

Adapting $F^* = \delta_P(p)$ and the set P as in Equation (3.10), we observe the primal-dual formulation

$$\min_{u \in X} \max_{p \in Y} \langle p, \nabla u \rangle + \lambda ||u - g||_1 - \delta_P(p).$$

For the representation of the dual problem of the TVL1 saddle-point problem we need to compute G^*. We find that the convex conjugate of the function $G(u) = \lambda ||u - g||_1$ is given by

$$G^*(p) = \begin{cases} \langle p, g \rangle & \text{if } ||p||_\infty \leq \lambda \\ \infty & \text{else} \end{cases},$$

or equivalently $G^*(p) = \delta_Q(p) + \langle p, g \rangle$ for a set

$$Q = \{p \in Y : ||p||_\infty \leq \lambda\}.$$

In order to derive this representation of the conjugate function we set $z = u - g$ which is equivalent to $u = z + g$. Then with the definition of the convex conjugate we get

$$\begin{aligned} G^*(p) = \sup_{u \in X} (\langle p, u \rangle - G(u)) \iff G^*(p) &= \sup_{z \in X} (\langle p, z + g \rangle - G(z + g)) \\ &= \sup_{z \in X} (\langle p, z + g \rangle - \lambda ||z||_1) \\ &= \sup_{z \in X} \left(\langle \frac{1}{\lambda} p, z \rangle - ||z||_1 \right) + \langle p, g \rangle. \end{aligned}$$

Using Example 2.20 2. and the fact that the conjugate norm of the $l1$ norm is the l^∞ norm, leads us to

$$G^*(p) = \begin{cases} \langle p, g\rangle & \text{if } ||\frac{1}{\lambda}p||_\infty \leq 1 \\ \infty & \text{else} \end{cases} \iff G^*(p) = \begin{cases} \langle p, g\rangle & \text{if } ||p||_\infty \leq \lambda \\ \infty & \text{else} \end{cases}.$$

Or equivalently $G^*(p) = \delta_Q(p) + \langle p, g\rangle$. The dual formulation of the TVL1 saddle-point problem is then given by

$$\begin{aligned} \max_{p\in Y} -(G^*(-K^*p) + F^*(p)) &= \max_{p\in Y} -\left(\delta_Q(-\nabla^T p) + \langle -\nabla^T p, g\rangle + \delta_P(p)\right) \\ &= \max_{p\in Y} -\left(\delta_Q(\operatorname{div}(p)) + \langle \operatorname{div}(p), g\rangle + \delta_P(p)\right). \end{aligned}$$

It remains to compute the proximity operators for the functions G and F^*.

The Proximity Operators of the TVL1 Model

The proximity operator of F^* remains the same as in Subsection 3.4.2. We have

$$p = (\mathrm{Id} + \sigma\,\partial\, F^*)^{-1}(\tilde{p}) \iff p_{i,j} = \frac{\tilde{p}_{i,j}}{\max(1, |\tilde{p}_{i,j}|)},$$

for all $i = 1, ..., N, j = 1, ..., M$. To compute the proximity operator of the function G we get

$$(\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u}) = \arg\min_{u\in X} \frac{||u - \tilde{u}||_2^2}{2} + \lambda\tau||u - g||_1.$$

Therefore, we define $\mathcal{L}(u) = \frac{||u-\tilde{u}||_2^2}{2} + \lambda\tau||u - g||_1$ which is - as the sum of convex terms - a convex function. Let $\hat{u} \in \arg\min_{u\in X} \mathcal{L}(u)$. Then with Proposition 2.24 we have that

$$0 \in \partial\mathcal{L}(\hat{u}) \iff 0 \in \hat{u} - \tilde{u} + \tau\lambda\partial(||\hat{u} - g||_1).$$

By evaluating the (i, j)-th component of $\partial\mathcal{L}(\hat{u})$ we have

$$\begin{aligned} 0 \in \partial_{i,j}\mathcal{L}(\hat{u}) &\iff 0 \in \hat{u}_{i,j} - \tilde{u}_{i,j} + \tau\lambda\partial_{i,j}(||\hat{u} - g||_1) \\ &\iff 0 \in \hat{u}_{i,j} - \tilde{u}_{i,j} + \tau\lambda\partial_{i,j}(|\hat{u}_{i,j} - g_{i,j}|). \end{aligned}$$

This means we need to compute the subgradient of the absolute value function for all partial subderivatives (i, j) with $i = 1, ..., N$, $j = 1, ..., M$. In Example 2.23 1. we saw how the subgradient for the absolute-value function is computed. We have

$$y_{i,j} = \begin{cases} 1 & \text{if } \hat{u}_{i,j} - g_{i,j} > 0 \\ -1 & \text{if } \hat{u}_{i,j} - g_{i,j} < 0 \\ [\text{-}1,\ 1] & \text{if } \hat{u}_{i,j} - g_{i,j} = 0 \end{cases},$$

where $y_{i,j}$ is the subgradient of the (i,j)-th component of $|\hat{u}_{i,j} - g_{i,j}|$. We check all three cases:

1. Let $y_{i,j} = 1$. Then we obtain

$$0 \in \hat{u}_{i,j} - \tilde{u}_{i,j} + \tau\lambda \iff \hat{u}_{i,j} = \tilde{u}_{i,j} - \tau\lambda.$$

   With $\hat{u}_{i,j} - g_{i,j} > 0$ and $\hat{u}_{i,j} = \tilde{u}_{i,j} - \tau\lambda$ this equation holds if

$$\tilde{u}_{i,j} - \tau\lambda - g_{i,j} > 0 \iff \tilde{u}_{i,j} - g_{i,j} > \tau\lambda.$$

2. Now, let $y_{i,j} = -1$. We get

$$0 \in \hat{u}_{i,j} - \tilde{u}_{i,j} - \tau\lambda \iff \hat{u}_{i,j} = \tilde{u}_{i,j} + \tau\lambda.$$

   Using this equality in the constraint, where $y_{i,j} = -1$, leads us to

$$\tilde{u}_{i,j} + \tau\lambda - g_{i,j} > 0 \iff \tilde{u} - g_{i,j} > \tau\lambda.$$

3. Finally, let $y_{i,j} \in [-1,1]$ with $\hat{u}_{i,j} = g_{i,j}$. We get

$$0 \in g_{i,j} - \tilde{u}_{i,j} + \tau\lambda y_{i,j} \iff \tilde{u}_{i,j} - g_{i,j} = \tau\lambda y_{i,j}.$$

   We apply the absolute value function to each side of the last equation and note that $|y_{i,j}| \leq 1$. Then

$$|\tilde{u}_{i,j} - g_{i,j}| = |\tau\lambda y_{i,j}| \leq \tau\lambda.$$

Overall, we set for all $i = 1, ..., N$, $j = 1, ..., M$

$$u = (\mathrm{Id} + \tau\,\partial G)^{-1}(\tilde{u}) \iff u_{i,j} = \begin{cases} \tilde{u}_{i,j} - \tau\lambda & \text{if } \tilde{u}_{i,j} - g_{i,j} > \tau\lambda \\ \tilde{u}_{i,j} + \tau\lambda & \text{if } \tilde{u}_{i,j} - g_{i,j} < -\tau\lambda \\ g_{i,j} & \text{if } |\tilde{u}_{i,j} - g_{i,j}| \leq \tau\lambda \end{cases}. \tag{3.15}$$

As we computed all necessary operators for the primal-dual algorithm, we need an appropriate criterion for convergence. We completely adapt the idea of Subsection 3.4.3. As shown in Chapter 5, it is also a well-suited criterion to determine convergence for minimizing the TVL1 functional.

3.6 The Mumford-Shah Model

In the previous sections we showed that we can rewrite convex minimization into saddle-point problems. Further, we computed the proximity operators for the underlying functions in these models. With these, we were able to apply the primal-dual algorithm to solve each saddle-point problem. In this section we introduce a non-convex functional, namely the Mumford-Shah functional, we seek to minimize. The idea is, despite the non-convexity, to apply convex analysis to the minimization problem and use

Moreau's identity to compute one of the proximity operators. Let us start by defining the Mumford-Shah functional.

Definition 3.9 (The Mumford-Shah Functional) *Let $\Omega \subseteq \mathbb{R}^2$ be a rectangular image domain. In order to approximate an input image $g : \Omega \longrightarrow \mathbb{R}^k$ in terms of a piecewise smooth function $u : \Omega \longrightarrow \mathbb{R}^k$, the Mumford-Shah Functional is given by*

$$E_{MS}(u) = \int_{\Omega} (u-g)^2 dx + \lambda \int_{\Omega \setminus K} |\nabla u|^2 dx + \nu |K|, \tag{3.16}$$

where $\nu, \lambda > 0$ are weighting parameters and k is the number of color channels. Further, $K = K_1 \cup ... \cup K_N$ and $|K|$ denotes the length of the curves K.

Remark 3.10 *In this section we follow the representation of the Mumford-Shah functional of [26].*

This functional differs from the ones in the previous sections. The first term, the data fidelity term, remains the same as in the ROF model, except a scaling factor $\frac{1}{2}$ and the parameter λ now controls the first term of the regularizer. This regularizer consists of two terms in this functional. The first term of the Mumford-Shah regularizer uses again the gradient, but not in the set Ω itself. Instead it states that the approximation u should be smooth in sets of Ω separated by curves K, namely in $\Omega \setminus K$. We call K the discontinuity set (or jump set). The length of the discontinuity set K is also measured and taken into account in the energy $E_{MS}(u)$. We find two weighting parameters ν and λ in this model. Where λ handles the tradeoff between the first term of the regularizer and the data fidelity term, ν controls the length of the discontinuity set K. A smaller ν yields a smoother image, where a higher ν leads to sharper edges in the approximation. Yet, the parameter λ plays another important role. If we choose λ small enough then the model is also called piecewise-smooth Mumford-Shah model. On the other hand, in the limiting case $\lambda \to \infty$, we can only attain a minimum over all u if we set $\nabla u = 0$ in $\Omega \setminus K$. Then the model is known as the piecewise-constant Mumford-Shah model. In [26] Strekalovskiy and Cremers suggested to rewrite the regularizer in the discrete setting by using the following function:

$$R_{MS}(f) = \min(\lambda |f|^2, \nu). \tag{3.17}$$

Then minimizing the discrete Mumford-Shah energy is expressed by

$$\begin{aligned} \min_{u \in X} E_{MS}(u) &= \min_{u \in X} F(\nabla u) + G(u) \\ &= \min_{u \in X} \left(\left(\sum_{i=1}^{N} \sum_{j=1}^{M} R_{MS}((\nabla u)_{i,j}) \right) + ||u-g||_2^2 \right) \\ &= \min_{u \in X} \left(\sum_{i=1}^{N} \sum_{j=1}^{M} R_{MS}((\nabla u)_{i,j}) + |u_{i,j} - g_{i,j}|^2 \right). \end{aligned} \tag{3.18}$$

According to [26], the idea behind this formulation is to model the discontinuity set K explicitly in terms of the function u and measure the length of K in terms of the parameter ν. This means that K is the set of all points where the minimum in Equation (3.17) attains ν. In other words, if the gradient $(\nabla u)_{i,j}$ is large enough we have for the explicit set K_{MS}:

$$K_{MS} = \left\{ (i,j) \in \Omega : |(\nabla u)_{i,j}|^2 \geq \sqrt{\frac{\nu}{\lambda}} \right\}. \tag{3.19}$$

We observe for a point $(i,j) \in K_{MS}$

$$R_{MS}(\nabla u_{i,j}) = \min(\underbrace{\lambda |(\nabla u)_{i,j}|^2}_{\geq \lambda \sqrt{\frac{\nu}{\lambda}}^2 = \nu}, \nu) = \nu$$

and if $(i,j) \notin K_{MS}$ we have

$$R_{MS}(\nabla u_{i,j}) = \min(\underbrace{\lambda |(\nabla u)_{i,j}|^2}_{< \lambda \sqrt{\frac{\nu}{\lambda}}^2 = \nu}, \nu) = \lambda |(\nabla u)_{i,j}|^2.$$

Remark 3.11 *In Chapter 5 we will explicitly make use of the set K_{MS} for edge highlighting. Further, note that in the piecewise-constant case, where $\lambda \to \infty$, Equation (3.17) changes to*

$$R_{MS}(f) = \begin{cases} \nu & \text{if } f \neq 0 \\ 0 & \text{else} \end{cases},$$

since $\min(\infty, \nu) = \nu$ *and* $\min(0, \nu) = 0$ *for* $\nu \geq 0$.

3.6.1 Mumford-Shah as Saddle-Point Problem

We formulate this model as a saddle-point problem to be able to apply the real-time Primal-Dual Algorithm 3.3. We identify

$$F(\nabla u) = \sum_{i=1}^{N} \sum_{j=1}^{M} R_{MS}((\nabla u)_{i,j}) \text{ and } G(u) = \sum_{i=1}^{N} \sum_{j=1}^{M} (u_{i,j} - g_{i,j})^2.$$

In the earlier sections we could easily compute the Legendre-Fenchel conjugate of the function F, since the total variation is convex and for that $F^{**} = F$ holds. Unfortunately, the function F, containing R_{MS}, is non-convex in the considered minimization problem. We could still compute the convex conjugate, but would not be able to compute an appropriate proximity operator for R^*_{MS}. Because of this, we assume this model to have an arbitrary regularizer R which we further assume to be convex. Then, we consider the convex minimization problem

$$\min_{u \in X} \left(\sum_{i=1}^{N} \sum_{j=1}^{M} R((\nabla u)_{i,j}) + |u_{i,j} - g_{i,j}|^2 \right).$$

Expressing R in terms of its Legendre-Fenchel conjugate, namely

$$\sum_{i=1}^{N}\sum_{j=1}^{M} R((\nabla u)_{i,j}) = \sup_{p\in Y}\left(\langle p, \nabla u\rangle - \sum_{i=1}^{N}\sum_{j=1}^{M} R^*(p_{i,j})\right),$$

we observe the primal-dual formulation by

$$\min_{u\in X}\max_{p\in Y}\left(\sum_{i=1}^{N}\sum_{j=1}^{M}\langle p_{i,j}, (\nabla u)_{i,j}\rangle - R^*(p_{i,j}) + |u_{i,j} - g_{i,j}|^2\right).$$

We do not compute R^* directly, and for that will not provide the dual formulation of this model. We go on by evaluating the proximity operators.

3.6.2 The Proximity Operators of the Mumford-Shah Model

Let us first evaluate the proximity operator for the function $G = ||u-g||_2^2$. We can partially adapt it from Equation (3.12), because the underlying data fidelity term is the same as in the ROF model, excluding the factor $\frac{\lambda}{2}$. Define

$$\mathcal{L}(u) = \frac{||u-\tilde{u}||_2^2}{2} + \tau\frac{||u-g||_2^2}{2},$$

which is a convex function. We observe for the proximity operator

$$(\mathrm{Id} + \tau\partial G)^{-1}(\tilde{u}) = \arg\min_{u\in X}\mathcal{L}(u) \iff 0 \in \partial\mathcal{L}(\hat{u})$$

if $\hat{u} \in \arg\min_{u\in X}\mathcal{L}(u)$. From this characterization it follows

$$0 \in \hat{u} - \tilde{u} + 2\tau(\hat{u} - g) \iff (1+2\tau)\hat{u} = \tilde{u} + 2\tau g \iff \hat{u} = \frac{\tilde{u} + 2\tau g}{1+2\tau}.$$

Then, we get pointwise for all $i = 1, ..., N$ and $j = 1, ..., M$

$$u = (\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u}) \iff u_{i,j} = \frac{\tilde{u}_{i,j} + 2\tau g_{i,j}}{1+2\tau}.$$

As we originally considered the regularizer R_{MS} and want to take it into account in the proximity operator, we compute the proximity operator for $R^*(p)$ by using Moreau's identity (2.12). This means, we compute

$$p = (\mathrm{Id} + \sigma\partial R^*)^{-1}(\tilde{p}) = \tilde{p} - \sigma\left(\mathrm{Id} + \frac{1}{\sigma}\partial R\right)^{-1}\left(\frac{\tilde{p}}{\sigma}\right). \tag{3.20}$$

Then, we can insert the original function R_{MS} into this formula to observe the proximity operator for R^*_{MS}. We start with computing

$$(\mathrm{Id} + \gamma \partial R_{MS})^{-1}(\tilde{x}) = \arg\min_{x \in X} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) \right)$$

for some parameter $\gamma > 0$. We need to do a case to case study, since the minimum function $R_{MS}(x_{i,j})$ can have two different outcomes: $\lambda |x_{i,j}|^2$ and ν. We want to compute the critical points at which the gradient vanishes. For that we compute the subderivatives in the (i,j)-th direction. We observe

$$\partial_{i,j} \left(\sum_{i=1}^{N} \sum_{j=1}^{M} \frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) \right) = x_{i,j} - \tilde{x}_{i,j} + \gamma \partial_{i,j}(R_{MS}(x_{i,j})) = 0. \quad (3.21)$$

The partial derivative of R_{MS} depends on which value the minimum function attains. There are three cases which can appear: $\lambda |x_{i,j}|^2 < \nu$, $\lambda |x_{i,j}|^2 > \nu$ and $\lambda |x_{i,j}|^2 = \nu$. Let us take a closer look at each of these cases:

1. Assume that $R_{MS}(x_{i,j}) = \nu$, then $\partial_{i,j}(R_{MS}(x_{i,j})) = 0$ and for that we observe with Equation (3.21) $x_{i,j} = \tilde{x}_{i,j}$. If this $x_{i,j}$ is the minimal value in the (i,j)-th component we get

$$\min_{x_{i,j}} \frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) = \frac{|\tilde{x}_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma \nu = \gamma \nu.$$

2. Now, assume that $R_{MS}(x_{i,j}) = \lambda |x_{i,j}|^2$. We get $\partial_{i,j}(R_{MS}(x_{i,j})) = 2\lambda x_{i,j}$ and with Equation (3.21)

$$x_{i,j} - \tilde{x}_{i,j} + 2\gamma\lambda x_{i,j} = 0 \iff x_{i,j} = \frac{\tilde{x}_{i,j}}{1 + 2\gamma\lambda}.$$

Then, we attain the minimum at

$$\begin{aligned}
\min_{x_{i,j}} \frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) &= \frac{|\frac{\tilde{x}_{i,j}}{1+2\gamma\lambda} - \tilde{x}_{i,j}|^2}{2} + \gamma\lambda \left| \frac{\tilde{x}_{i,j}}{1 + 2\gamma\lambda} \right|^2 \\
&= \frac{4\gamma^2\lambda^2}{2(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 + \frac{\gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{2\gamma^2\lambda^2 + \gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{(1+2\gamma\lambda)\gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{\gamma\lambda}{1+2\gamma\lambda} |\tilde{x}_{i,j}|^2. \quad (3.22)
\end{aligned}$$

3. In the last case, where $\lambda|x_{i,j}|^2 = \nu$, we need to decide which of the previously computed minima is the global minimal value. In Figure 3.2 we see that the objective function can attain several minima. As we consider the boundary of two convex functions, namely $\frac{|x_{i,j}-\tilde{x}_{i,j}|^2}{2} + \gamma\lambda|x_{i,j}|^2$ and $\frac{|x_{i,j}-\tilde{x}_{i,j}|^2}{2} + \gamma\nu$, we know that one of the previously computed minimal values is the global minimum and that the minimum can not be attained on the bound where $\lambda|x_{i,j}|^2 = \nu$. By comparing both minima we need to check

$$\begin{aligned} \frac{\gamma\lambda}{1+2\gamma\lambda}|\tilde{x}_{i,j}|^2 \leq \gamma\nu \quad &\Longleftrightarrow \quad \frac{\lambda}{1+2\gamma\lambda}|\tilde{x}_{i,j}|^2 \leq \nu \\ &\Longleftrightarrow \quad |\tilde{x}_{i,j}|^2 \leq \frac{\nu}{\lambda}(1+2\gamma\lambda) \\ &\Longleftrightarrow \quad |\tilde{x}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}. \end{aligned} \tag{3.23}$$

   As long as this inequality holds, we can choose $x_{i,j} = \frac{\tilde{x}_{i,j}}{1+2\gamma\lambda}$.

The proximity operator of the function R_{MS} is therefore given pointwise by

$$x = (\mathrm{Id} + \gamma R_{MS})^{-1}(\tilde{x}) \iff x_{i,j} = \begin{cases} \dfrac{1}{1+2\gamma\lambda}\tilde{x}_{i,j} & \text{if } |\tilde{x}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)} \\ \tilde{x}_{i,j} & \text{else} \end{cases}, \tag{3.24}$$

for all $i = 1, ..., N$ and $j = 1, ..., M$. In order to derive the representation of the proximity operator for R^*_{MS} we insert the operator of Equation (3.24) into Moreau's identity formula of Equation (3.20). We need to do a case by case study to take the constraints in (3.24) into account:

1. Assume that $|\tilde{p}_{i,j}| > \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}$ at a position (i,j). Then, the proximity operator, due to Moreau's identity, is given by

$$\begin{aligned} p &= (\mathrm{Id} + \sigma\partial R^*_{MS})^{-1}(\tilde{p}) = \tilde{p} - \sigma\left(\mathrm{Id} + \frac{1}{\sigma}\partial R_{MS}\right)^{-1}\left(\frac{\tilde{p}}{\sigma}\right) \\ \Longleftrightarrow\ p_{i,j} &= \tilde{p}_{i,j} - \sigma\frac{\tilde{p}_{i,j}}{\sigma} = 0. \end{aligned}$$

2. Now, assume that $|\tilde{p}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}$ at (i,j). This leads to

$$\begin{aligned} p &= \left(\mathrm{Id} + \sigma\partial R_{MS}^*\right)^{-1}(\tilde{p}) = \tilde{p} - \sigma\left(\mathrm{Id} + \frac{1}{\sigma}\partial R_{MS}\right)^{-1}\left(\frac{\tilde{p}}{\sigma}\right) \\ \iff p_{i,j} &= \tilde{p}_{i,j} - \sigma\frac{\frac{\tilde{p}_{i,j}}{\sigma}}{1+2\frac{1}{\sigma}\lambda} \\ &= \tilde{p}_{i,j} - \frac{\sigma}{\sigma+2\lambda}\tilde{p}_{i,j} \\ &= \left(1 - \frac{\sigma}{\sigma+2\lambda}\right)\tilde{p}_{i,j} \\ &= \left(\frac{\sigma+2\lambda-\sigma}{\sigma+2\lambda}\right)\tilde{p}_{i,j} \\ &= \frac{2\lambda}{\sigma+2\lambda}\tilde{p}_{i,j}. \end{aligned}$$

As we did for the proximity operator for R_{MS}, we also need to show for which condition these two cases hold. Using Inequality (3.23) and setting $\gamma = \frac{1}{\sigma}$ gives us

$$\begin{aligned} \frac{\frac{1}{\sigma}\lambda}{1+2\frac{1}{\sigma}\lambda}|\tilde{p}_{i,j}|^2 &\leq \frac{1}{\sigma}\nu \\ \iff \frac{\lambda}{\sigma+2\lambda}|\tilde{p}_{i,j}|^2 &\leq \sigma\nu \\ \iff |\tilde{p}_{i,j}|^2 &\leq \frac{\nu}{\lambda}\sigma(\sigma+2\lambda) \\ \iff |\tilde{p}_{i,j}| &\leq \sqrt{\frac{\nu}{\lambda}\sigma(\sigma+2\lambda)}. \end{aligned}$$

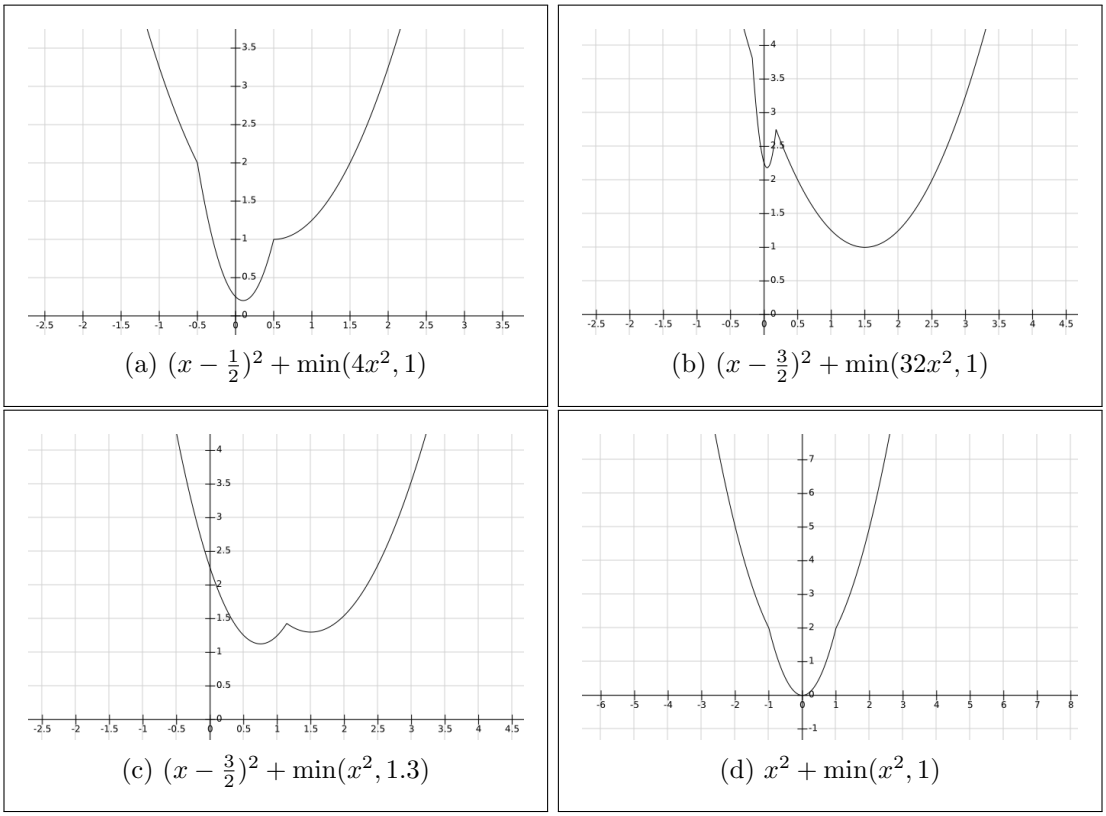
Then, the proximity operator for R_{MS}^* is defined by

$$p = \left(\mathrm{Id} + \sigma\partial R_{MS}^*\right)^{-1}(\tilde{p}) \iff p_{i,j} = \begin{cases} \dfrac{\lambda}{\lambda+\sigma}\tilde{p}_{i,j}, & \text{if } |\tilde{p}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}\sigma(\sigma+2\lambda)} \\ 0 & \text{else} \end{cases}, \quad (3.25)$$

for all $i = 1, ..., N$, $j = 1, ..., M$. According to Strekalovskiy and Cremers in [26], there is no proof of convergence to the minimal value of the Mumford-Shah functional for this framework. Despite that fact, they show the boundedness of u^n in the piecewise smooth case. This is summarized in the following proposition.

Proposition 3.12 *The sequence (u^n, p^n) generated by Algorithm 3.3 is bounded and thus compact for $\lambda < \infty$, for instance it has a convergent subsequence.*

The proof can be found in the supplementary material of [26]. To determine convergence we also partially follow [26] as described in Subsection 3.4.3, but set $\varepsilon = 5 \cdot 10^{-5}$. The difference in our framework is that we evaluate the norm in each iteration step. In [26]



(a) $(x - \frac{1}{2})^2 + \min(4x^2, 1)$

(b) $(x - \frac{3}{2})^2 + \min(32x^2, 1)$

(c) $(x - \frac{3}{2})^2 + \min(x^2, 1.3)$

(d) $x^2 + \min(x^2, 1)$

Figure 3.2: In (a) and (d) the objective function of the minimization problem only attains one minimum. In (c) we have two minima, but the global minimum is attained for the condition $|\tilde{x}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1 + 2\gamma\lambda)}$. In (b) the global minimum is attained in the region in which $|\tilde{x}_{i,j}| > \sqrt{\frac{\nu}{\lambda}(1 + 2\gamma\lambda)}$.

they compute this norm every ten iteration steps.

In this chapter we have provided a framework for minimizing the ROF and TVL1 functionals, as well as a possibility to minimize the Mumford-Shah functional. In the next chapter we present a second approach to minimize the Mumford-Shah functional.

4 Minimizing the Mumford-Shah Functional

As we revisit the Mumford-Shah functional, we also rewrite Definition 3.9 to be consistent with the publication of Pock, Cremers, Bischof and Chambolle ([21]) we mainly follow in this chapter. If other results are taken into account we point it out.

Definition 4.1 (Mumford-Shah Functional) *Let $\Omega \subseteq \mathbb{R}^2$ be a rectangular image domain. In order to approximate an input image $f : \Omega \longrightarrow \mathbb{R}$ in terms of a piecewise smooth function $u : \Omega \longrightarrow \mathbb{R}$, the Mumford-Shah functional is defined by*

$$E_{MS}(u) = \lambda \int_{\Omega} (f-u)^2 dx + \int_{\Omega \setminus S_u} |\nabla u|^2 dx + \nu \mathcal{H}^1(S_u), \tag{4.1}$$

where $\lambda, \nu > 0$ are weighting parameters, $S_u = S_u^1 \cup ... \cup S_u^N$ and $\mathcal{H}^1(S_u)$ denotes the one-dimensional Hausdorff-measure of the curves in S_u. And we seek to minimize $E_{MS}(u)$ over all u.

The difference to Chapter 3 is that we interchanged the set K with $S_u = S_u^1 \cup ... \cup S_u^N$ and instead of computing $|K|$, we use the more general notation of measure theory, namely $\mathcal{H}^1(S_u)$. In the case $u \in \Omega \subseteq \mathbb{R}^2$, the one-dimensional Hausdorff-measure of S_u is nothing but $|S_u|$. Another difference to Definition 3.9 is the parameter λ which handles the tradeoff between the data fidelity term and the first term in the regularizer. It is swapped and controls the term where the image f enters the functional. This swapping of the parameter also rescales ν in this formulation. The rescaling does not change the energy, but we need other pairs (λ, ν) for a good image approximation as in the real-time minimzer framework of Subsection 3.6. We discuss this issue in Chapter 5.

As mentioned in the previous chapter this functional is non-convex. The idea to derive a convex saddle-point representation of the Mumford-Shah functional is to make use of some results presented by Alberti, Bouchitte and Dal Maso.

4.1 Convex Relaxation

In order to state a convex representation of the Mumford-Shah functional we introduce the characteristic or level set function.

Definition 4.2 *Let $\Omega \subseteq \mathbb{R}^2$ denote the image plane and let $u \in SBV(\Omega)$. The upper level sets of u are denoted by the characteristic function $\mathbb{1}_u : \Omega \times \mathbb{R} \longrightarrow \{0,1\}$ of the subgraph of u:*

$$\mathbb{1}_u(x,t) = \begin{cases} 1 & \text{if } t < u(x) \\ 0 & \text{else} \end{cases} . \tag{4.2}$$

In Figure 4.1 one can see an example with a characteristic function in $\mathbb{R}^2$ for a function $u \in SBV(\Omega)$. So, for a one-dimensional signal $u(x)$ the characteristic function becomes two-dimensional. In our case, as we consider two-dimensional images, the characteristic function ensures that we will act in a three-dimensional space. In Figure 4.1 the gray

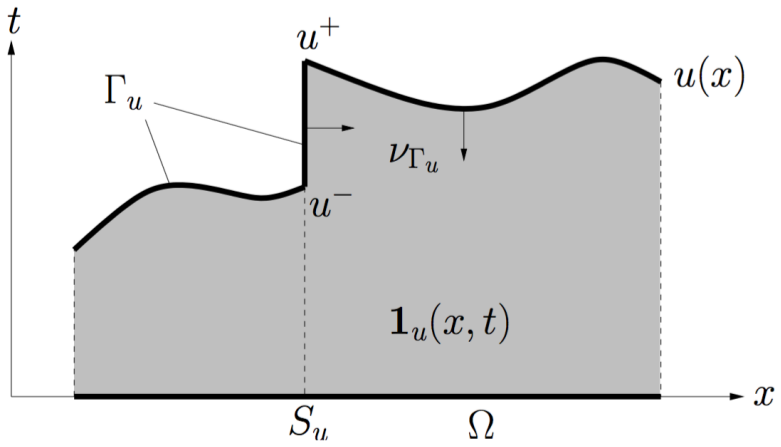


Figure 4.1: This picture (taken from [21]) shows the characteristic function $\mathbb{1}_u(x,t)$ of a function $u(x) \in SBV(\Omega)$.

shaded area is the part where $\mathbb{1}_u(x,t) = 1$. Otherwise, the characteristic function is set to zero. The upper interface of the gray domain is denoted by Γ_u. This set is the set which holds all parts of the function $u(x)$ and each curve S_u connecting the points u^- and u^+. Additionally, the notation ν_{Γ_u} denotes the normals on Γ_u.

Using $\mathbb{1}_u(x,t)$, one can find in [1] and [2] the proposed method of Alberti, Bouchitte and Dal Maso to approximate the Mumford-Shah energy by a convex maximization problem. For a proof of this concept we refer to these two publications. We state the theorem in the fashion of [21] which summarizes the results of [1] and [2].

Theorem 4.3 (Convex Relaxation of the Mumford-Shah Functional) *For a function $u \in SBV(\Omega)$ the Mumford Shah functional can be written as*

$$E_{MS}(u) = \sup_{\varphi \in K} \int_{\Omega \times \mathbb{R}} \varphi D\mathbb{1}_u \tag{4.3}$$

with a convex set

$$K = \Big\{ \varphi \in C^0(\Omega \times \mathbb{R}, \mathbb{R}^3) \quad : \quad \varphi^t(x,t) \geq \frac{\varphi^x(x,t)^2}{4} - \lambda(t - f(x))^2, \tag{4.4}$$

$$\left| \int_{t_1}^{t_2} \varphi^x(x,s) ds \right| \leq \nu \Big\}, \tag{4.5}$$

where the inequalities in the definition of K hold for all $x \in \Omega$ and for all $t, t_1, t_2 \in \mathbb{R}$.

Remark 4.4

- *We used in the theorem that we regard the function φ as the representation $\varphi(x,t) = \left(\varphi^x(x,t), \varphi^t(x,t)\right)^T$ where $\varphi(x,t) \in \mathbb{R}^3$, $\varphi^x(x,t) \in \mathbb{R}^2$ and $\varphi^t(x,t) \in \mathbb{R}$.*
- *The idea to approximate the Mumford-Shah functional is to maximize the flux of a vector field φ through the interface Γ_u. The advantage of this technique is to get a convex representation of the Mumford-Shah functional and for this we derive in the following a convex saddle-point formulation. Then we can - again - make use of the primal-dual algorithm to solve the optimization problem.*
- *Note that $\varphi \in C^0(\Omega \times \mathbb{R}, \mathbb{R}^3)$. In [21], Pock et al. propose $\varphi \in C^0(\Omega \times \mathbb{R}, \mathbb{R}^2)$ which is a typo.*

Our goal is to minimize (4.3). We seek to solve

$$\min_u E(u) = \min_u \left(\sup_{\varphi \in K} \int_{\Omega \times \mathbb{R}} \varphi D\mathbb{1}_u \right).$$

Following [21] we additionally substitute $\mathbb{1}_u$ by a function

$$v(x,t) : \Omega \times \mathbb{R} \longrightarrow [0,1] \text{ and } \lim_{t \to -\infty} v(x,t) = 1, \quad \lim_{t \to +\infty} v(x,t) = 0 \tag{4.6}$$

to compute the minimal energy of the Mumford-Shah functional. Overall, we are going to face the following convex optimization problem:

$$\min_{v \in [0,1]} \sup_{\varphi \in K} \langle v, D\varphi \rangle = \min_{v \in [0,1]} \sup_{\varphi \in K} \int_{\Omega \times \mathbb{R}} \varphi Dv. \tag{4.7}$$

According to [21] there is no proof that finding an optimal pair (v^*, φ^*) for the optimization problem of (4.7) leads to the global minimizer of Equation (4.1). Further, they state that only if v^* is binary one gets indeed the global minimum of the Mumford-Shah functional. Despite that fact, solving Equation (4.7) leads to high-quality approximations u of an input image f.

4.2 Discrete Setting

In this section we consider the System (4.7) in its discrete version. Using the characteristic function and substitute it by the function v means that we are adding an additional space to our two dimensional image domain. This extra label space needs to be considered in the discrete setting. In [21] Pock et al. consider $\Omega = [0,1]^2$ and for that the subgraph of u to be in $[0,1]^3$. This would imply that we discretize these two spaces by adding a step-size h. Further, they consider all their operators without having this additional step-size. We propose all our spaces and operators without a step size. Then the image domain is defined as $\Omega = \{1,2,...,N\} \times \{1,2,...,M\}$ and the subgraph of the

function $u : \mathbb{R}^2 \longrightarrow [0,1]$ is defined in the cube $\Omega \times \{1,2,...,S\}$. In this discrete setting we define the pixel grid $\mathcal{G}$ with size $N \times M \times S$ and the following notation

$$\mathcal{G} = \Big\{(i,j,k) : i = 1,2,...,N, j = 1,...,M, k = 1,2,...,S\Big\},$$

where i, j, k are the discrete locations of each voxel. For a reformulation of (4.7) we also need to define the corresponding discrete versions of v, φ. So, let $u \in X : \mathcal{G} \longrightarrow [0,1]$ and $p \in Y : \mathcal{G} \longrightarrow \mathbb{R}^3$ be the discrete versions of the continuous functions in Equation (4.7), where u corresponds to v and p to φ. If we replace the inner-product for infinite dimensions in Equation (4.7) by the inner-product for finite dimensions, we are going to face the saddle-point problem

$$\min_{u \in C} \max_{p \in K} \langle Au, p \rangle. \tag{4.8}$$

Here, A is a continuous linear operator, earlier denoted with K. Then, this notation looks familiar to us. And we define the constraint set C by

$$C = \{u \in X : u(i,j,k) \in [0,1], u(i,j,1) = 1, u(i,j,S) = 0\}. \tag{4.9}$$

In order to take the limits of the function v into account in its discrete version u, we set the values in the first label space to 1 and those in the S-th label space to 0. According to [21] the discrete version of the convex set K from Equation (4.5) has the following representation:

$$K = \{p = (p^x, p^t)^T \in Y \quad : \quad p^t(i,j,k) \geq \frac{||p^x(i,j,k)||_2^2}{4} - \lambda(\frac{k}{S} - f(i,j))^2, \tag{4.10}$$

$$\left| \sum_{k_1 \leq k \leq k_2} p^x(i,j,k) \right| \leq \nu\}, \tag{4.11}$$

whereas we define $p^x(i,j,k) := (p^1(i,j,k), p^2(i,j,k))^T$ and $p^t(i,j,k) := p^3(i,j,k)$ for a $p(i,j,k) \in \mathbb{R}^3$. For this $p = (p^x, p^t) \in Y \subseteq \mathbb{R}^{N \cdot M \cdot S \cdot 2} \times \mathbb{R}^{N \cdot M \cdot S}$. The constraint in Equation (4.10) goes pointwise for all $(i,j,k) \in \mathcal{G}$. The second constraint is more involved, since the constraint in (4.11) holds for all $i = 1,...,N$, $j = 1,...,M$ and all possible combinations (k_1, k_2) with $1 \leq k_1 \leq k \leq k_2 \leq S$. What looks like having a set K with two constraints, turns out that the set K is an intersection of a couple of convex sets. One has that for a fixed voxel (i,j,k) one can compute $\frac{S^2+S}{2} + 1$ many convex sets. This number rises from the fact that taking into account all possible combinations (k_1, k_2) over all levels S and adding the local-constraint yields

$$\left(\binom{S}{2} + S\right) + 1 = \left(\frac{n!}{k! \cdot (n-k)!} + S\right) + 1 = \left(\frac{S(S-1)}{2} + S\right) + 1 = \frac{S^2 + S}{2} + 1.$$

The large amount of several convex sets will lead us to a long run-time in the suggested framework of [21]. Before we discuss this in detail, we first continue with the definitions of the discrete setting.

Remark 4.5 *In addition, to discretize the variable t in (4.5) one gets $\frac{k}{S}$ in the discrete version of (4.10). Note that t is a value in the continuous setting, which determines at which point the characteristic function vanishes. The bound on the norm L depends on the discrete gradient operator, like in Propositions 3.6 and 4.8. Pock et al. proposed in [21] to set the discrete version of t to $\frac{k}{L}$ which is a mistake.*

Furthermore, we define the linear operator A in the same fashion as in Section 3.3, but extended to the additional label space. We have $A = \nabla$ and for that $A^* = \nabla^T = -\text{div}$.

Definition 4.6 (Discrete gradient operator) *We define the discrete gradient operator of $u \in X$ by $\nabla u = ((\partial_i u)_{i,j,k}, (\partial_j u)_{i,j,k}, (\partial_k u)_{i,j,k})^T$ using forward differences with Neumann boundary conditions, i.e.*

$$(\partial_i u)_{i,j,k} = \begin{cases} u_{i+1,j,k} - u_{i,j,k} & \text{if } i < N \\ 0 & \text{if } i = N \end{cases}, (\partial_j u)_{i,j,k} = \begin{cases} u_{i,j+1,k} - u_{i,j,k} & \text{if } j < M \\ 0 & \text{if } j = M \end{cases},$$
$$(\partial_k u)_{i,j,k} = \begin{cases} u_{i,j,k+1} - u_{i,j,k} & \text{if } k < S \\ 0 & \text{if } k = S \end{cases}.$$

Definition 4.7 (Discrete divergence operator) *We define the discrete divergence operator of $p \in Y$ by $\nabla^T p = (\partial_i p^1)_{i,j,k} + (\partial_j p^2)_{i,j,k} + (\partial_k p^3)_{i,j,k}$ using backward differences with Dirichlet boundary conditions, i.e.*

$$(\partial_i p^1)_{i,j,k} = \begin{cases} p^1_{i,j,k} - p^1_{i-1,j,k} & \text{if } 1 < i < N \\ p^1_{i,j,k} & \text{if } i = 1 \\ -p^1_{i-1,j,k} & \text{if } i = N \end{cases},$$
$$(\partial_j p^2)_{i,j,k} = \begin{cases} p^2_{i,j,k} - p^2_{i,j-1,k} & \text{if } 1 < j < M \\ p^2_{i,j,k} & \text{if } j = 1 \\ -p^2_{i,j-1,k} & \text{if } j = M \end{cases},$$
$$(\partial_k p^3)_{i,j,k} = \begin{cases} p^3_{i,j,k} - p^3_{i,j,k-1} & \text{if } 1 < k < S \\ p^3_{i,j,k} & \text{if } k = 1 \\ -p^3_{i,j,k-1} & \text{if } k = S \end{cases}.$$

And we can compute the bound on the norm of A, as in Proposition 3.6.

Proposition 4.8 (Bound on the norm of ∇) *The bound on the norm of the proposed discrete linear operator is given by*

$$L^2 = ||\nabla||^2 = ||\nabla^T||^2 \leq 12.$$

Proof The proof remains the same as for Proposition 3.6 by adding the additional discretization variable $p^3_{i,j,k}$ and applying Young's inequality thrice with $p = q = 2$. ■

4.3 Projection onto the sets C and K

In order to be able to apply the primal-dual algorithm we rewrite Equation 4.8 into

$$\min_{u\in X}\max_{p\in Y}\langle Au,p\rangle - \delta_K(p) + \delta_C(u).$$

For this we want to evaluate the proximity operators for this framework. As we identify the functions $F^*(p) = \delta_K(p)$ and $G(u) = \delta_C(u)$ we know from Example 2.29 that the proximity operator of the indicator function is a euclidean projection onto the corresponding set. Before we propose a method to project onto C and K, respectively, we rewrite the Primal-Dual Algorithm 3.1 to be consistent with the work of Pock et al. in [21].

Algorithm 4.9 *Choose $(u^0,p^0)\in C\times K$ and let $\bar{u}^0 = u^0$. We choose $\tau,\sigma>0$. Then, we let for each $n\geq 0$*

$$\begin{cases} p^{n+1} = \Pi_K(p^n + \sigma K\bar{u}^n) \\ u^{n+1} = \Pi_C(u^n - \tau K^* p^{n+1}) \\ \bar{u}^{n+1} = 2u^{n+1} - u^n \end{cases}. \tag{4.12}$$

We start with the projection onto the set C and go on by providing an algorithm to project onto the set K.

4.3.1 Projection onto C

The projection onto C can efficiently be computed. By definition of the proximity operator we have

$$u = \arg\min_{u\in X}\frac{||u-\tilde{u}||_2^2}{2} + \tau\delta_C(u) = \arg\min_{u\in C}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{S}\frac{|u_{i,j,k}-\tilde{u}_{i,j,k}|^2}{2} = \Pi_C(\tilde{u}).$$

Assume that $\tilde{u}_{i,j,k}\in C$ for a voxel (i,j,k). Then the best choice for $u_{i,j,k}$ is to set $u_{i,j,k} = \tilde{u}_{i,j,k}$, since the term $|u_{i,j,k}-\tilde{u}_{i,j,k}|^2$ is then equal to zero for which the quadratic function is minimal. On the other hand if $\tilde{u}_{i,j,k}\notin C$, the euclidean distance to $C=[0,1]$ is the shortest distance $\tilde{u}_{i,j,k}$ onto the bound of C. This means if $\tilde{u}_{i,j,k}<0$ then the shortest distance from $\tilde{u}_{i,j,k}$ to C is to set $u_{i,j,k}=0$. Reversely, if $\tilde{u}_{i,j,k}>1$ then the euclidean distance to C is given by setting $u_{i,j,k}=1$. This idea is also illustrated in Figure 4.2. For an arbitrary intervall $[a,b]$ we have the following algorithm:

Algorithm 4.10 (Clipping) *Let $u^n\notin[a,b]$ and $a,b\in\mathbb{R}_{\geq 0}$ with $a<b$. The projection of u^n onto the interval is given by*

$$u^{n+1} = \min\{b,\max\{a,u^n\}\} \tag{4.13}$$

Remark 4.11

- *By projecting onto C, we also need to take care of the limits in (4.6). For that we set $u(i,j,1)=1$ and $u(i,j,S)=0$ in each projection.*

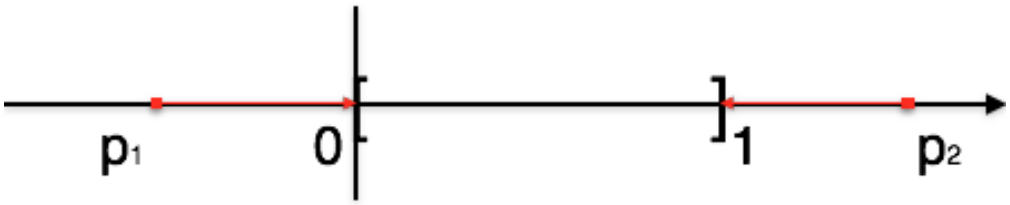


Figure 4.2: A point $p_1 < 0$ is clipped to 0, whereas a point $p_2 > 1$ is clipped to 1.

- *In the case of our framework - acting in a discrete cube $\mathcal{G}$ - Equation (4.13) can be regarded as*
$$u_{i,j,k}^{n+1} = \min\{1, \max\{0, u_{i,j,k}^{n}\}\},$$
for all $i = 1, ..., N$, $j = 1, ..., M$ and $k = 1, ..., S$.

- *We can easily verify that Equation (4.13) is the clipping on the bound of the intervall $[a, b]$. Let $u^n < a$, then*
$$u^{n+1} = \min\{b, \max\{a, u^n\}\} = \min\{b, a\} = a.$$
Reversely, we observe for a $u^n > b$ that
$$u^{n+1} = \min\{b, \max\{a, u^n\}\} = \min\{b, u^n\} = b.$$

4.3.2 The projection onto K

The projection onto K is more involved since K takes into account local and non-local constraints. In other words, the set K is an intersection of several convex sets. The projection onto the intersection of convex sets can be done by Dykstra's projection algorithm. The idea behind this algorithm is to project a point x onto each set seperately in an iteration step n. Then, as $n \to \infty$ the algorithm finds a point $\hat{x}$ in K for which the distance $||x - \hat{x}||_2$ is minimal, see also Figure 4.3. The full scheme was first proposed by Boyle and Dykstra in [8], where one can also find a proof of convergence for the proposed algorithm. For more information about Dykstra's projection algorithm, we also refer to [8]. We follow the notation of Cremers and Kolev in [12] in our proposed algorithm.

Algorithm 4.12 *Consider P convex sets X_i with $\mathbb{R}^n \ni X = X_1 \cap X_2 \cap ... \cap X_P$. Let Π_i denote the projection onto the i-th set for $i = 1, ..., P$. And let $u_c \in \mathbb{R}^n$ be the current estimate with $u_c \notin X$, $u_i^k \in \mathbb{R}^n$ for $i = 0, ..., P$ and $v_i^k \in \mathbb{R}^n$ for $i = 1, ..., P$. For $k = 1, 2, ...$ set $u_P^0 = u_c$ and $v_i^0 = 0$ for all $i = 1, ..., P$. Then iterate until convergence*

(e.g. $||u_0^k - u_P^k||_2 \leq \varepsilon$ *with* ε *small):*

$$\begin{aligned} u_0^k = \quad & u_P^{k-1}, \\ \text{for} \quad & i = 1, 2, ..., P: \\ & u_i^k = \Pi_i(u_{i-1}^k - v_i^{k-1}), \\ & v_i^k = u_i^k - (u_{i-1}^k - v_i^{k-1}). \end{aligned}$$

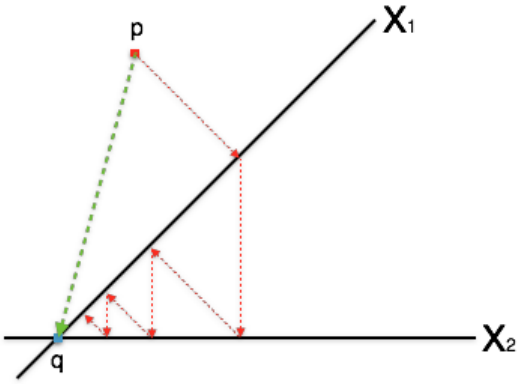


Figure 4.3: The point p should be projected onto q which is the point of the intersection of the curves (sets) X_1 and X_2. In Dykstra's projection algorithm we alternatingly do euclidean projections onto the corresponding sets X_1 and X_2. This is illustrated with the red doted lines.

With this algorithm the projection onto the entire set K can be established. Since the algorithm itself needs the projections onto the constraint sets in each iteration we present how we project onto the subsets of K.

4.3.3 Decomposition of K

We will decompose the set K into sets K_p and K_{nl}, where the first resembles the local, or more precisely, a parabola constraint and the second corresponds to the non-local constraint. Overall, we have $K = K_p \cap K_{nl}$ with

$$K_p = \left\{ p^t(i,j,k) \geq \frac{||p^x(i,j,k)||_2^2}{4} - \lambda(\frac{k}{S} - f(i,j))^2 \right\},$$

for all $i = 1, ..., N$, $j = 1, ..., M$ and $k = 1, ..., S$. And for the non-local constraint we have

$$K_{nl} = \left\{ \left| \sum_{k_1 \leq k \leq k_2} p^x(i,j,k) \right| \leq \nu \right\},$$

again for all $i = 1, ..., N$ and $j = 1, ..., M$, and in this case for all combinations (k_1, k_2) with $1 \leq k_1 \leq k_2 \leq S$. In the following we deduce the projection onto these two sets separately. Let us start with the projection onto the parabola constraint.

4.3.4 Projection onto K_p

Since the projection onto the set K_p goes pointwise we drop the indices (i, j, k). Note that we do not necessarily need that p^x is an element of $\mathbb{R}^2$. The following derivation holds for a larger class of problems, namely having $p^x \in \mathbb{R}^n$. Let $\alpha > 0$, $p^x \in \mathbb{R}^n$, $p^t \in \mathbb{R}$ and $p = (p^x, p^t)^T \in \mathbb{R}^n \times \mathbb{R}$. Assume that $p_0^t < \alpha ||p_0^x||_2^2$ holds for a point $p_0 \in \mathbb{R}^n \times \mathbb{R}$. Then the projection of p_0 onto the parabola $\alpha ||p_0^x||_2^2$ can be written as the following minimization problem:

$$\begin{aligned} \min_{p \in \mathbb{R}^n \times \mathbb{R}} \quad & \frac{1}{2} ||p - p_0||_2^2 \\ \text{subject to} \quad & p^t \geq \alpha ||p^x||_2^2. \end{aligned}$$

To find the solution of this optimization problem we introduce a Lagrange multiplier $\mu \in \mathbb{R}$ and define the Lagrangian by

$$\mathcal{L}(p, \mu) = \frac{||p - p_0||_2^2}{2} - \mu \left(p^t - \alpha ||p^x||_2^2 \right).$$

The minimization problem we consider is convex, because the cost function is convex, the inequality constraint is convex and the feasible set, which is th epigraph of a parabola, is also convex. With this we know that computing a critical point of the Lagrangian function $\mathcal{L}$ also leads to a global optimum of the optimization problem itself. We compute a critical point by

$$\nabla \mathcal{L}(p, \mu) = \begin{pmatrix} \partial_{p^x} \mathcal{L}(p, \mu) \\ \partial_{p^t} \mathcal{L}(p, \mu) \\ \partial_{\mu} \mathcal{L}(p, \mu) \end{pmatrix} = \begin{pmatrix} p^x - p_0^x + \mu 2 \alpha p^x \\ p^t - p_0^t - \mu \\ \alpha ||p^x||_2^2 - p^t \end{pmatrix} = 0. \tag{4.14}$$

This means, we need to solve a linear system. In the first equation of (4.14) we get

$$p_0^x = (\mu 2\alpha + 1) p^x \iff p^x = \frac{p_0^x}{\mu 2\alpha + 1}, \tag{4.15}$$

and the second equation leads us to

$$p^t = p_0^t + \mu. \tag{4.16}$$

In the following we discuss two different possibilities how the System (4.14) can be solved.

1. Inserting the Equalities (4.15) and (4.16) into the third line of Equation (4.14)

leads us to

$$\begin{aligned}
& \alpha||p^x||_2^2 - p^t = 0 \\
\iff \quad & p^t - \alpha||p^x||_2^2 = 0 \\
\iff \quad & p_0^t + \mu - \alpha\left|\left|\frac{p_0^x}{\mu 2\alpha + 1}\right|\right|_2^2 = 0 \\
\iff \quad & p_0^t + \mu - \frac{\alpha}{(\mu 2\alpha + 1)^2}||p_0^x||_2^2 = 0 \\
\overbrace{\iff}^{\cdot(\mu 2\alpha+1)^2} \quad & (\mu 2\alpha + 1)^2 p_0^t + (\mu 2\alpha + 1)^2\mu - \alpha||p_0^x||_2^2 = 0 \\
\iff \quad & (4\mu^2\alpha^2 + 4\mu\alpha + 1)p_0^t + 4\mu^3\alpha^2 + 4\mu^2\alpha + \mu - \alpha||p_0^x||_2^2 = 0 \\
\iff \quad & 4\alpha^2\mu^3 + \mu^2(4\alpha^2 p_0^t + 4\alpha) + \mu(4\alpha p_0^t + 1) + p_0^t - \alpha||p_0^x||_2^2 = 0.
\end{aligned}$$

In order to solve this equation for μ, we define a function

$$h(\mu) = 4\alpha^2\mu^3 + \mu^2(4\alpha^2 p_0^t + 4\alpha) + \mu(4\alpha p_0^t + 1) + p_0^t - \alpha||p_0^x||_2^2,$$

for which we are seeking for the zeroes. Computing the zeroes can efficiently be established by using Newton's algorithm. So, for $k = 1, 2, ...$ compute

$$\mu^{k+1} = \mu^k - \frac{h(\mu)}{h'(\mu)}.$$

The first derivative of h is given by

$$h'(\mu) = 12\alpha^2\mu^2 + 2\mu(4\alpha^2 p_0^t + 4\alpha) + (4\alpha p_0^t + 1).$$

Overall, we get the update equation for a μ^{k+1} with

$$\mu^{k+1} = \mu^k - \frac{4\alpha^2\mu^3 + \mu^2(4\alpha^2 p_0^t + 4\alpha) + \mu(4\alpha p_0^t + 1) + p_0^t - \alpha||p_0^x||_2^2}{12\alpha^2\mu^2 + 2\mu(4\alpha^2 p_0^t + 4\alpha) + (4\alpha p_0^t + 1)}.$$

In [11], Chambolle et al. suggested to set $\mu^0 = \max\{0, -\frac{2p_0^t}{3}\}$, where they state that Newton's method converges within 7-10 iterations to a quite accurate solution. With Equations (4.15) and (4.16) and the computed value for μ we are able to evaluate p by

$$p = \left(\frac{p_0^x}{\mu 2\alpha + 1}, p_0^t + \mu\right).$$

We do not suggest using this method, since the primal-dual algorithm will be extremely slow in the case of this framework. Having a few iterations of Newton's algorithm in each primal-dual iteration, additionally generates more overhead and for that would decelerate the program. Further, Newton's method is inexact and as it turns out, the second approach to this problem will lead to an exact solution which can be computed within one loop of straightforward computations.

2. For the second approach we note that (4.15) and (4.16) hold and plug these equalities into the third equation in (4.14) which can then be expressed by

$$p^t = \alpha||p^x||_2^2 \iff p_0^t + \mu = \alpha||p^x||_2^2. \tag{4.17}$$

With Equation (4.15) we can compute the solution of μ by

$$\begin{aligned}
p^x = \frac{p_0^x}{\mu 2\alpha + 1} \quad &\iff \quad ||p^x||_2 = \left|\left|\frac{p_0^x}{1+2\alpha\mu}\right|\right|_2 \\
&\iff \quad ||p^x||_2 = \frac{1}{1+2\alpha\mu}||p_0^x||_2 \\
&\iff \quad \frac{1}{1+2\alpha\mu} = \frac{||p^x||_2}{||p_0^x||_2} \\
&\iff \quad 1 + 2\alpha\mu = \frac{||p_0||_2}{||p_x||_2} \\
&\iff \quad 2\alpha\mu = \frac{||p_0^x||_2}{||p^x||_2} - 1 \\
&\iff \quad \mu = \frac{1}{2\alpha}\left(\frac{||p_0^x||_2}{||p^x||_2}\right) - \frac{1}{2\alpha}.
\end{aligned}$$

We insert the solution of μ into (4.17) and observe

$$\begin{aligned}
&\alpha||p^x||_2^2 = p_0^t + \frac{1}{2\alpha}\left(\frac{||p_0^x||_2}{||p^x||_2}\right) - \frac{1}{2\alpha} \\
\overset{\cdot 2\alpha||p^x||_2}{\iff} \quad &2\alpha^2||p^x||_2^3 = 2\alpha p_0^t||p^x||_2 + ||p_0^x||_2 - ||p^x||_2 \\
\iff \quad &2\alpha^2||p^x||_2^3 + ||p^x||_2 - 2\alpha p_0^t||p^x||_2 - ||p_0^x||_2 = 0. \\
\iff \quad &2\alpha^2||p^x||_2^3 + (1 - 2\alpha p_0^t)||p^x||_2 - ||p_0^x||_2 = 0. \\
\overset{\cdot 4\alpha}{\iff} \quad &8\alpha^3||p^x||_2^3 + 4\alpha(1 - 2\alpha p_0^t)||p^x||_2 - 4\alpha||p_0^x||_2 = 0. \\
\iff \quad &(2\alpha||p^x||_2)^3 + 2(1 - 2\alpha p_0^t)2\alpha||p^x||_2 - 4\alpha||p_0^x||_2 = 0. \\
\iff \quad &t^3 + 3bt - 2a = 0,
\end{aligned} \tag{4.18}$$

with $a = 2\alpha||p_0^x||_2$, $b = \frac{2}{3}(1 - 2\alpha p_0^t)$ and $t = 2\alpha||p^x||_2$.
The cubic Equation (4.18) in t can efficiently be solved using the analytical formula for solving cubic equations published by J. P. McKelvey in 1984 in [16].

The result of McKelvey's publication is summarized in the following algorithm. Note that we already computed the factors a and b. The other factors follow with [16]. We propose our algorithm in the fashion of [25].

Algorithm 4.13 *Let $\alpha > 0, p_0^t \in \mathbb{R}, p_0^x \in \mathbb{R}^n$. If $p_0^t \geq \alpha||p_0^x||_2^2$ is already satisfied set*

$(p^x, p^t) = (p_0^x, p_0^t)$. *Otherwise, with* $a = 2\alpha||p_0^x||_2$, $b = \frac{2}{3}(1 - 2\alpha p_0^t)$, *and*

$$d = \begin{cases} a^2 + b^3 & \text{if } b \geq 0 \\ (a - \sqrt{-b}^3)(a + \sqrt{-b}^3) & \text{else} \end{cases},$$

set

$$v = \begin{cases} c - \dfrac{b}{c} \text{ with } c = \sqrt[3]{a + \sqrt{d}} & \text{if } d \geq 0 \\ 2\sqrt{-b}\cos\left(\dfrac{1}{3}\arccos\dfrac{a}{\sqrt{-b}^3}\right) & \text{else} \end{cases}.$$

If $c = 0$ *in the first case, set* $v = 0$. *The solution is then given by*

$$p^x = \begin{cases} \dfrac{v}{2\alpha}\dfrac{p_0^x}{||p_0^x||_2} & \text{if } p_0^x \neq 0 \\ 0 & \text{else} \end{cases},$$

and $p^t = \alpha||p^x||_2^2$.

The above method states that the projection onto the parabola can be done by one cycle of straightforward computations. The implementation of it is simple and the algorithm has a fast run-time. It is only left to show, how we project onto K_{nl}.

4.3.5 Projection onto K_{nl}

The set K_{nl} is a combination of non-local constraints, meaning that in a fixed pair (i, j) we sum up all possible combinations of $p^x(i, j, k)$ for all $1 \leq k_1 \leq k \leq k_2 \leq S$. So, we can not project pointwise. Let us assume that we consider a vector $p \in \mathbb{R}^{S\times 2}$ at a fixed pair (i, j). We denote p by $p = (p_1^x, p_2^x, ..., p_S^x)$, where $p_k^x \in \mathbb{R}^2$ for all $k = 1, ..., S$. Further, we consider a fixed pair (k_1, k_2) with $1 \leq k_1 \leq k_2 \leq S$. The set K_{nl} is therefore defined as

$$K_{nl} = \left\{ \left| \sum_{k_1 \leq k \leq k_2} p_k^x \right| \leq \nu \right\}.$$

The projection of a vector $\tilde{p} \in \mathbb{R}^{S\times 2}$ for a fixed pair (k_1, k_2) onto the set K_{nl} can be expressed by the following optimization problem:

$$\min_{p\in\mathbb{R}^{2\times S}} \frac{1}{2}||p - \tilde{p}||_2^2 \tag{4.19a}$$

$$\text{subject to } \left| \sum_{k_1 \leq k \leq k_2} p_k^x \right| \leq \nu. \tag{4.19b}$$

We seek to find the minimal distance of $\tilde{p}$ to K_{nl} by additionally fulfilling the inequality constraint in (4.19b). Then, the following algorithm establishes the projection of a $\tilde{p}$

onto K_{nl} for a fixed pair (k_1, k_2).

Algorithm 4.14 (Soft-Shrinkage Scheme) *Choose $\tilde{p}, p \in \mathbb{R}^{S\times 2}$, $\tilde{p}_k, p_k \in \mathbb{R}^2$ for all $k = 1, ..., S$ and choose a pair (k_1, k_2), such that $1 \leq k_1 \leq k_2 \leq S$. If $\tilde{p} \notin K_{nl}$ use $\tilde{s}_{k_1,k_2}, s_{k_1,k_2} \in \mathbb{R}^2$ and compute $\tilde{s}_{k_1,k_2} = \sum_{k_1\leq k\leq k_2} \tilde{p}_k^x$ together with*

$$s_{k_1,k_2} = \begin{cases} \tilde{s}_{k_1,k_2} & \text{if } |\tilde{s}_{k_1,k_2}| \leq \nu \\ \dfrac{\nu}{|\tilde{s}_{k_1,k_2}|}\tilde{s}_{k_1,k_2} & \text{else} \end{cases}.$$

Then, iterate for all $k = 1, .., S$ and set

$$p_k^x = \begin{cases} \tilde{p}_k^x + \dfrac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} & \text{if } k_1 \leq k \leq k_2 \\ \tilde{p}_k^x & \text{else} \end{cases}. \tag{4.20}$$

If $\tilde{p}$ already satisfies the inequality constraint (4.19b) or equivalently $\tilde{p} \in K_{nl}$, set $p = \tilde{p}$.

Proposition 4.15 *The solution generated in Equation (4.20) by Algorithm 4.14, solves the optimization problem (4.19) optimally.*

To prove this proposition, we make use of the Karush-Kuhn-Tucker optimality conditions of Theorem 2.5.

Proof For fixed pairs (i, j) and (k_1, k_2) let $\tilde{p} \in \mathbb{R}^{S\times 2}$ and let $\tilde{s}_{k_1,k_2}, s_{k_1,k_2} \in \mathbb{R}^2$ be as in Algorithm 4.14. If $\tilde{p}$ already satisfies the inequality constraint in 4.19b, which means that $\tilde{p} \in K_{nl}$, we set $p^* = \tilde{p}$ as the solution of System (4.19). With this choice the quadratic function in (4.19a) is zero and for that minimal.
For the rest of the proof we assume that $\tilde{p} \notin K_{nl}$ and for this also $\tilde{s} \neq 0$, since $\nu > 0$. We know from Example 2.13 1. that a quadratic function, e.g. (4.19a), is strictly convex. This implies that if we find a local solution of System (4.19), it is already a global solution.
Relating to the inequality constraint in (4.19b) we define a function $c : \mathbb{R}^{S\times 2} \to \mathbb{R}$ by

$$c(p) = \frac{1}{2}\nu^2 - \frac{1}{2}\left|\sum_{k_1\leq k\leq k_2} p_k^x\right|^2.$$

Suppose $p^* \in \mathbb{R}^{S\times 2}$ is the unique solution of System (4.19) and we characterize p^* by $p_k^* = \tilde{p}_k + \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1}$ if $k_1 \leq k \leq k_2$ and $p_k^* = \tilde{p}_k$ else. For this p^*, we can show that the

constraint in (4.19b) is active:

$$\begin{aligned} c(p^*) &= \frac{1}{2}\nu^2 - \frac{1}{2}\Big|\sum_{k_1\le k\le k_2} p_k^*\Big|^2 \\ &= \frac{1}{2}\nu^2 - \frac{1}{2}\Big|\sum_{k_1\le k\le k_2}\left(\tilde{p}_k + \frac{s_{k_1,k_2}-\tilde{s}_{k_1,k_2}}{k_2-k_1+1}\right)\Big|^2 \\ &= \frac{1}{2}\nu^2 - \frac{1}{2}\Big|\underbrace{\sum_{k_1\le k\le k_2}\tilde{p}_k}_{=\tilde{s}_{k_1,k_2}} + \underbrace{\sum_{k_1\le k\le k_2}\frac{s_{k_1,k_2}-\tilde{s}_{k_1,k_2}}{k_2-k_1+1}}_{=(k_2-k_1+1)\left(\frac{s_{k_1,k_2}-\tilde{s}_{k_1,k_2}}{k_2-k_1+1}\right)}\Big|^2 \qquad (4.21) \\ &= \frac{1}{2}\nu^2 - \frac{1}{2}\underbrace{|s_{k_1,k_2}|^2}_{=\nu^2} = 0 \qquad (4.22) \end{aligned}$$

Now, we want to show that LICQ holds at p^*. Let us first evaluate the gradient of the function c in p^*. We observe

$$\nabla c(p^*) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -\sum\limits_{k_1\le k\le k_2} p_k^* \\ \vdots \\ -\sum\limits_{k_1\le k\le k_2} p_k^* \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

and we consider the i-th row for which $k_1 \le i \le k_2$ and compute

$$(\nabla c(p^*))_i = -\sum_{k_1\le k\le k_2}\left(\tilde{p}_k + \frac{s_{k_1,k_2}-\tilde{s}_{k_1,k_2}}{k_2-k_1+1}\right) = -s,$$

where we used the equality of Equations (4.21) and (4.22). The gradient of c in p^* is

then given by

$$\nabla c(p^*) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -s \\ \vdots \\ -s \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Since $\tilde{s} \neq 0$ it follows that $s \neq 0$ by definition of s and $\nabla c(p^*)$ is linear independent. It is left to show that there is a Lagrange multiplier $\lambda^* \in \mathbb{R}$ such that the KKT conditions hold for (p^*, λ^*). For this we first define the Lagrange function by

$$\mathcal{L}(p, \lambda) = \frac{1}{2}||p - \tilde{p}||_2^2 - \lambda \left(\frac{1}{2}\nu^2 - \frac{1}{2} \left| \sum_{k_1 \leq k \leq k_2} p_k \right|^2 \right).$$

We evaluate $\nabla \mathcal{L}(p^*, \lambda^*)$ and observe

$$\nabla \mathcal{L}(p^*, \lambda^*) = \begin{pmatrix} p_1^* - \tilde{p}_1 \\ \\ \vdots \\ \vdots \\ \vdots \\ \\ \\ p_S^* - \tilde{p}_S \end{pmatrix} - \lambda^* \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -\sum_{k_1 \leq k \leq k_2} p_k^* \\ \vdots \\ -\sum_{k_1 \leq k \leq k_2} p_k^* \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} p_1^* - \tilde{p}_1 \\ \\ \vdots \\ \vdots \\ \vdots \\ \\ \\ p_S^* - \tilde{p}_S \end{pmatrix} + \lambda^* \begin{pmatrix} 0 \\ \vdots \\ 0 \\ s \\ \vdots \\ s \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

If we consider the i-th row of $\nabla \mathcal{L}(p^*, \lambda^*)$ with $i < k_1$ or $i > k_2$ then we have that λ^* can be chosen arbitrarily, because

$$p_i^* - \tilde{p}_i + \lambda^* \cdot 0 = 0 \iff p_i^* = \tilde{p}_i.$$

In order to compute λ^* we need to consider the i-th row of $\nabla \mathcal{L}(p^*, \lambda^*)$ for which i satisfies

$k_1 \leq i \leq k_2$. We observe

$$\begin{aligned}
& p_i^* - \tilde{p}_i + \lambda^* s = 0 \\
\iff \quad & \left(\tilde{p}_i + \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} \right) - \tilde{p}_i + \lambda^* = 0 \\
\iff \quad & \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} + \lambda^* s_{k_1,k_2} = 0 \\
\overset{s=\frac{\nu}{|\tilde{s}_{k_1,k_2}|}\tilde{s}_{k_1,k_2}}{\iff} \quad & \frac{\frac{\nu}{|\tilde{s}_{k_1,k_2}|}\tilde{s}_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} + \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|}\tilde{s}_{k_1,k_2} = 0 \\
\iff \quad & \tilde{s}_{k_1,k_2} \left(\frac{\frac{\nu}{|\tilde{s}_{k_1,k_2}|} - 1}{k_2 - k_1 + 1} + \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|} \right) = 0
\end{aligned}$$

Since $\tilde{s}_{k_1,k_2} \neq 0$ and $|\tilde{s}_{k_1,k_2}| > \nu$, we can evaluate λ^* by

$$\begin{aligned}
& \frac{\frac{\nu}{|\tilde{s}_{k_1,k_2}|} - 1}{k_2 - k_1 + 1} + \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|} = 0 \\
\iff \quad & \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|} = \frac{1 - \frac{\nu}{|\tilde{s}_{k_1,k_2}|}}{k_2 - k_1 + 1} \\
\iff \quad & \lambda^* = \underbrace{\frac{1}{k_2 - k_1 + 1}}_{>0} \left(\underbrace{\frac{|\tilde{s}_{k_1,k_2}|}{\nu}}_{>1} - 1 \right) > 0.
\end{aligned}$$

For this $\lambda^* > 0$, the gradient of $\mathcal{L}(p^*, \lambda^*)$ vanishes. Because the inequality constraint is active at p^*, the complementary slackness condition is satisfied. According to the work of Stephen Boyd and Lieven Vandenberghe ([7], page 244) we know as our optimization problem is convex that the KKT conditions are already sufficient conditions and we proved Proposition 4.15. ■

We presented methods to project onto the sets C and K. In order to project onto K, by using Dykstra's projection algorithm, we showed how projections onto the subsets of K can efficiently be computed. As we will see in Chapter 5, this approach needs a huge amount of memory and is extremely slow, even on a GPU. In the next section we present an alternative approach which yields a faster computation time.

4.4 An Alternative Approach using Lagrange Multiplier

In this section we present an alternative formulation of the saddle-point problem in Equation (4.8). This approach also solves the convex relaxed Mumford-Shah functional and the primal-dual algorithm for this framework is orders of magnitudes faster as the previously presented approach. In this framework the projections onto C and K_p remain unchanged, but the difference is that we get rid of Dykstra's projection algorithm to

save computation time. In order to derive such a representation we decouple K_{nl} and introduce Lagrange multipliers. We will obtain another saddle-point problem which can then be solved with a modified version of the primal-dual algorithm. In the original problem

$$\min_{u \in C} \max_{p \in K} \langle Au, p \rangle, \tag{4.23}$$

we maximized over the whole set K defined in Equations (4.10) and (4.11). In order to rewrite this formulation we revisit the non-local constraint set defined in the fashion of Subsection 4.3.5 by

$$K_{nl} = \left\{ \left| \sum_{k_1 \leq k \leq k_2} p_k^x \right| \leq \nu \right\},$$

at all possible positions (i, j) for all $i = 1, ..., N$, $j = 1, ..., M$ and all combinations (k_1, k_2) with $1 \leq k_1 \leq k_2 \leq S$. Decoupling of K_{nl} then means that we substitute $\sum_{k_1 \leq k \leq k_2} p_k^x$ by s_{k_1,k_2} and introduce an additional constraint to take the bound on ν into account. We have

$$K_{nl} = \left\{ |s_{k_1,k_2}| \leq \nu \text{ subject to } s_{k_1,k_2} = \sum_{k_1 \leq k \leq k_2} p_k^x \right\}, \tag{4.24}$$

again for all pairs (i, j), all combinations $1 \leq k_1 \leq k \leq k_2 \leq S$, $s_{k_1,k_2} \in \mathbb{R}^2$ and $s \in \mathbb{R}^{N \times M \times I \times 2}$. Here, $I = \frac{S^2+S}{2}$ denotes the number of all non-local constraint sets. We further present an auxiliary variable $\mu_{k_1,k_2} \in \mathbb{R}^2$, which belongs to a $\mu \in \mathbb{R}^{N \times M \times I \times 2}$ and define a Lagrange function

$$\mathcal{L}(u, \mu, p, s) = \langle Au, p \rangle + \sum_{k_1=1}^{S} \sum_{k_2=k_1}^{S} \langle \mu_{k_1,k_2}, \sum_{k_1 \leq k \leq k_2} p_k^x - s_{k_1,k_2} \rangle, \tag{4.25}$$

in which we added an enforced term to our original problem corresponding to the constraint in K_{nl}. Taking into account that the constraint in (4.24) is an equality constraint we have that $\mu_{k_1,k_2} \in \mathbb{R}^2$ for all $1 \leq k_1 \leq k_2 \leq S$. Using the Lagrangian in (4.25) we have the following proposition:

Proposition 4.16 *Let (u^*, p^*) be the solution of (4.23) and (u^*, μ^*, p^*, s^*) be the solution of*

$$\min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \max_{\substack{p \in K_p \\ |s_{k_1,k_2}| \leq \nu}} \mathcal{L}(u, \mu, p, s). \tag{4.26}$$

Then, the equality

$$\langle Au^*, p^* \rangle = \mathcal{L}(u^*, \mu^*, p^*, s^*) \tag{4.27}$$

holds.

Proof Let u^*, μ^*, p^*, s^* be the optimal values of Equations (4.23) and (4.26).

We show $\langle Au^*, p^* \rangle \geq \mathcal{L}(u^*, \mu^*, p^*, s^*)$. Then we observe the inequality

$$\begin{aligned}
\mathcal{L}(u^*, \mu^*, p^*, s^*) &= \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \max_{\substack{p \in K_p \\ |s_{k_1,k_2}| \leq \nu}} \langle Au, p \rangle + \sum_{k_1=1}^{S} \sum_{k_2=k_1}^{S} \langle \mu_{k_1,k_2}, \sum_{k_1 \leq k \leq k_2} p_k^x - s_{k_1,k_2} \rangle \\
&\leq \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \langle Au, p^* \rangle + \sum_{k_1=1}^{S} \sum_{k_2=k_1}^{S} \langle \mu_{k_1,k_2}, \sum_{k_1 \leq k \leq k_2} p_k^{*^x} - s^*_{k_1,k_2} \rangle \qquad (4.28) \\
&= \begin{cases} \langle Au^*, p^* \rangle, & \text{if } \sum_{k_1 \leq k \leq k_2} p_k^{*^x} = s^*_{k_1,k_2} \\ -\infty & \text{else} \end{cases} \qquad (4.29) \\
&\leq \langle Au^*, p^* \rangle.
\end{aligned}$$

In Equality (4.29) we used that if $\sum_{k_1 \leq k \leq k_2} p_k^{*^x} = s^*_{k_1,k_2}$, the last term in (4.28) vanishes. But, if $\sum_{k_1 \leq k \leq k_2} p_k^{*^x} \neq s^*_{k_1,k_2}$ the minimum over all μ assures that the last term in (4.28) tends to $-\infty$.
Let us now show $\langle Au^*, p^* \rangle \leq \mathcal{L}(u^*, \mu^*, p^*, s^*)$: If p^* is the optimal value of Equation (4.23) it also satisfies $p^* \in K = K_p \cap K_{nl}$. But this implies $p^* \in K_p$ and $p^* \in K_{nl}$ and for that we can set $s^*_{k_1,k_2} = \sum_{k_1 \leq k \leq k_2} p_k^{*^x}$ for all possible pairs (k_1, k_2). Then, we compute

$$\begin{aligned}
\langle Au^*, p^* \rangle &= \min_{u \in C} \max_{\substack{p \in K_p \\ p \in K_{nl}}} \langle Au, p \rangle \\
&\leq \min_{u \in C} \langle Au, p^* \rangle \\
&= \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \langle Au, p^* \rangle + \sum_{k_1=1}^{S} \sum_{k_2=1}^{S} \langle \mu_{k_1,k_2}, \underbrace{\sum_{k_1 \leq k \leq k_2} p_k^{*^x} - s^*_{k_1,k_2}}_{=0} \rangle \\
&= \mathcal{L}(u^*, \mu^*, p^*, s^*).
\end{aligned}$$

■

As discussed in Section 3.1, doing a gradient descent in the primal and a gradient ascent in the dual variable simultaneously means that we need to estimate the direction of the steepest descent and ascent, respectively. For that we compute $\nabla \mathcal{L}(u, \mu, p, s)$ and use the partial derivatives with respect to u, μ, p and s in the corresponding update equations

of the primal-dual algorithm. We compute

$$\frac{\partial \mathcal{L}(u,\mu,p,s)}{\partial u} = A^T p \tag{4.30}$$

$$\frac{\partial \mathcal{L}(u,\mu,p,s)}{\partial s_{k_1,k_2}} = -\sum_{k_1=1}^{S}\sum_{k_2=1}^{S} \mu_{k_1,k_2} \tag{4.31}$$

$$\frac{\partial \mathcal{L}(u,\mu,p,s)}{\partial \mu_{k_1,k_2}} = \sum_{k_1=1}^{S}\sum_{k_2=1}^{S}\left(\sum_{k_1\le k\le k_2} p_k^x - s_{k_1,k_2}\right) \tag{4.32}$$

$$\frac{\partial \mathcal{L}(u,\mu,p,s)}{\partial p} = Au + \hat{p}. \tag{4.33}$$

As a last step we need to estimate $\hat{p}$. We therefore consider the partial derivative in the l-th component in the direction of p.

$$\begin{aligned}
\frac{\partial \mathcal{L}(u,\mu,p,s)}{\partial p_l} &= (Au)_l + \frac{\partial}{\partial p_l}\left(\sum_{k_1=1}^{S}\sum_{k_2=k_1}^{S}\langle \mu_{k_1,k_2}, \sum_{k_1\le k\le k_2} p_k^x - s_{k_1,k_2}\rangle\right)\\
&= (Au)_l + \frac{\partial}{\partial p_l}\left(\sum_{k_1=1}^{S}\sum_{k_2=k_1}^{S}\left(\langle \mu_{k_1,k_2}, \sum_{k_1\le k\le k_2} p_k^x\rangle - \langle \mu_{k_1,k_2}, s_{k_1,k_2}\rangle\right)\right)\\
&= (Au)_l + \frac{\partial}{\partial p_l}\left(\sum_{k_1=1}^{S}\sum_{k_2=k_1}^{S}\langle \mu_{k_1,k_2}, \sum_{k_1\le k\le k_2} p_k^x\rangle\right)\\
&= (Au)_l + \begin{pmatrix} \sum_{k_1=1}^{l}\sum_{k_2=l}^{S}\mu^1_{k_1,k_2} \\ \sum_{k_1=1}^{l}\sum_{k_2=l}^{S}\mu^2_{k_1,k_2} \\ 0 \end{pmatrix}.
\end{aligned}$$

With this it follows

$$\tilde{p} = \begin{pmatrix} \sum_{k_1=1}^{l}\sum_{k_2=l}^{S}\mu^1_{k_1,k_2} \\ \sum_{k_1=1}^{l}\sum_{k_2=l}^{S}\mu^2_{k_1,k_2} \\ 0 \end{pmatrix}.$$

Recalling that we computed $I = \frac{S^2+S}{2}$, having these four updates and setting $A = \nabla$, we observe a new formulation for our primal-dual algorithm by:

Algorithm 4.17 *Choose* $(u^0, p^0, \mu^0, s^0) \in C \times K_p \times \mathbb{R}^{2\times N\times M\times I} \times \mathbb{R}^{2\times N\times M\times I}$ *and let* $\bar{u}^0 = u^0, \bar{\mu}^0 = \mu^0 = 0, p^0 = 0$. *Set* $\tau_u = \frac{1}{6}, \tau_\mu = \frac{1}{150}, \sigma_p = \frac{1}{3+S}, \sigma_s = 1$. *Then, we let for each* $n \geq 0$

$$\begin{cases} p^{n+1} = \Pi_{K_p}\left(p^n + \sigma_p(A\bar{u}^n + \tilde{p})\right) \\ s^{n+1}_{k_1,k_2} = \Pi_{|\cdot|\leq\nu}\left(s^n_{k_1,k_2} - \sigma_s\left(\sum_{k_1=1}^{S}\sum_{k_2=1}^{S}\bar{\mu}^n_{k_1,k_2}\right)\right) \\ u^{n+1} = \Pi_C\left(u^n - \tau_u A^* p^{n+1}\right) \\ \mu^{n+1}_{k_1,k_2} = \mu^n_{k_1,k_2} + \tau_\mu\left(\sum_{k_1\leq k\leq k_2} p^x_k - s^{n+1}_{k_1,k_2}\right) \\ \bar{u}^{n+1} = 2u^{n+1} - u^n \\ \bar{\mu}^{n+1}_{k_1,k_2} = 2\mu^{n+1}_{k_1,k_2} - \mu^n_{k_1,k_2}. \end{cases} \tag{4.34}$$

Once implemented this variant of the primal-dual algorithm yields the same approximation as the ordinary primal-dual algorithm using Dykstra's projection algorithm in each iteration step. The big difference between the two frameworks is the performance. We discuss this in Chapter 5. In the following we additionally show how the approximation u can be extracted from the three-dimensional space into a two-dimensional object.

4.5 Computing the 0.5-Isosurface

By using convex relaxation methods for minimizing the Mumford-Shah functional, we compute the approximation u in a three-dimensional space. So, we need a framework to extract the two-dimensional approximation $\hat{u}$. Therefore, we compute the 0.5-isosurface. The idea is to interpolate the corresponding channel at each pixel position (i, j) for $k = 1, ..., S$ by thresholding at 0.5. Before we propose the whole procedure, let us first explain how the interpolation with a threshold at 0.5 can be derived.

Remark 4.18 (Linear Newton Interpolation) *Recalling the linear Newton interpolation scheme, we can compute a function* $u(x)$ *by a convex combination of two pairs* $(x_0, u(x_0))$ *and* $(x_1, u(x_1))$. *The formula is given by*

$$u(x) = u(x_0) + \frac{u(x_1 - u(x_0))}{x_1 - x_0}(x - x_0).$$

The idea is to iterate for all $k = 1, ..., S$ at a fixed pair (i, j), until we find a value at the position $x_0 = \frac{k}{S}$ which fulfills the strict inequality $u(x_0) > 0.5$ and additionally find a value $x_1 = \frac{k+1}{S}$ together with $u(x_1) \leq 0.5$. Then, we can compute the function u at 0.5, namely $u(0.5)$, using Newton's linear interpolation formula by

$$\begin{aligned} u(0.5) &= \frac{k}{S} + \frac{\frac{k+1}{S} - \frac{k}{S}}{x_1 - x_0}(0.5 - x_0) \\ &= \frac{k}{S} + \frac{1}{S}\frac{0.5 - x_0}{x_1 - x_0} \\ &= \frac{1}{S}\left(k + \frac{0.5 - x_0}{x_1 - x_0}\right). \end{aligned}$$

Overall, we propose the following algorithm for computing the 0.5-isosurface:

Algorithm 4.19 (0.5-isosurface) *Let (i, j) be a fixed pair of all locations $i = 1, ..., N$ and $j = 1, ..., M$. Iterate from $1, ..., S-1$ and evaluate $u(i, j, k)$ and $u(i, j, k+1)$, until*

$$u(i, j, k) > 0.5 \text{ and } u(i, j, k+1) \leq 0.5,$$

and set

$$\hat{u}(i, j) = \frac{1}{S}\left(k + \frac{0.5 - u(i, j, k)}{u(i, j, k+1) - u(i, j, k)}\right).$$

If we do not find such a pair, we set

$$\hat{u}(i, j) = u(i, j, S).$$

Then $\hat{u}$ denotes the two-dimensional approximation of the input image f.

4.6 Convergence Criterion

As a convergence criterion we adapted the idea of the real-time minimizer framework. But we need to take into account that u is a three-dimensional object in the underlying algorithm. This means we need to modify the stopping criterion. We evaluate once every ten iterations

$$||u^{n+1} - u^n|| \leq \varepsilon,$$

with $\varepsilon = 5 \cdot 10^{-5}$. Then, the norm is computed by

$$||\tilde{u}|| = \frac{1}{N \cdot M \cdot S} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{S} |\tilde{u}_{i,j,k}|.$$

If we consider color images, we need a fourth loop over all $l = 1, 2, 3$ to take all color channels into account.

In this chapter we proposed a method for minimizing the Mumford-Shah functional. We rewrote the minimzation into a saddle-point problem to be able to use the primal-dual algorithm. Additionally, we presented two frameworks how the proposed saddle-point problem can be solved. The suggested method in [21], namely using Dykstra's projection algorithm to project onto the set K, is extremely slow, as Chapter 5 reveals. Decoupling of the set K into K_p and K_{nl} and applying a Lagrange multiplier formalism leads to a reformulation of the original saddle-point problem, for which we also rewrote the primal-dual algorithm. As we will see in Chapter 5, this approach is far more applicable and leads to a better run-time.

5 Applications to Imaging

In this chapter we present applications to imaging for the proposed models. We consider image cartooning by using the real-time minimizer for the Mumford-Shah model. This framework together with the ROF and TVL1 model are taken into account when we show image denoising. A modified version of the ROF model is applicable for image inpainting. At the end of this chapter we compare the two approaches for minimizing the convex relaxed Mumford-Shah functional, namely using Dykstra's projection algorithm and the Lagrange multiplier method. We also show that the Mumford-Shah is the most edge preserving method of all proposed models and good in image segmentation.

5.1 General Setting

We present several results in this chapter for image approximation, denoising, inpainting and image segmentation. We run all our tests on two different computers. The serial code is implemented on a CPU (Central Processing Unit) using the programming language C++. We execute the code on a MacBook Air (13-inch, Early 2014) with a 1,4 GHz Intel Core i5 Processor and a 4 GB 1600 MHz DDR3 internal memory. The two approaches for minimizing the convex relaxed Mumford-Shah functional needed parallelization to be applicable. In order to the implementation of both approaches we additionally parallelized our serial code on a GPU (Graphical Processor Unit) using the Nvidia CUDA framework. For more information on CUDA we refer to [20]. The code is executed at the University of Regensburg on the RZ411 computer containing a Nvidia Quadro K4000 GPU running 64-bit Rex-Linux. We additionally made use of the OpenCV library to read and write images in all our implementations. We refer to [14] for more information on OpenCV.
In the following sections we also present code examples for the primal-dual algorithm in C++. These examples are not the actual functions we use in our program. They are just a possible implementation we propose in this thesis. Further, we will not provide CUDA examples in this thesis, because it would go beyond the scope. For our complete coding, including the CUDA implementation, we refer to our computer program iPaur (image Processing At University of Regensburg) found on Github ([6]).
In Figure 5.1 we show the used images in our tests. The images (a) to (d) are images of Lena (or Lenna) Soderberg. We degraded the original Lena image in (a) with five percent Gaussain noise and twenty-five percent salt and pepper noise, respectively, in (b) and (c). And we removed seventy percent of the data in image (d). The images (g) to (i) are taken from the publication of Pock et al. in [21]. Let us mentioned that the Ladama image is originally called La Dama con l'ermellino and a painting by Leonardo da Vinci. The Synthetic Test image of (h) is further degraded with five percent Gaussian noise,

seen in (i). The other two color valued images in (e) and (f) are taken from the Github repository online at [27]. This repository belongs to the publication of Strekalovskiy and Cremers in [26].

(a) Lena image.

(b) Gaussian noise.

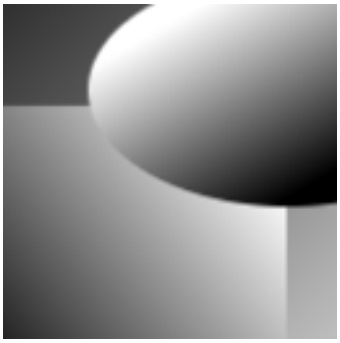
(c) S. & P. noise.

(d) Data loss.

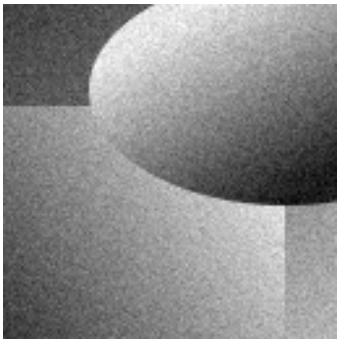
(e) Audrey Hepburn Image.

(f) Landscape image.

(g) Ladama image.



(h) Synthetic Test image.



(i) Gaussian noise.

Figure 5.1: (a) - (d) are grayscaled images with a size of 512×512. (g) - (i) are also grayscaled images with a size of 128×128. (e) is an RGB image of size 560×398 and (f) an RGB image of size 350×234.

We additionally introduce two important formulae used in the following sections. They can also be found in the book of Kristian Bredies and Dirk Lorenz in [9].

Definition 5.1 (MSE and PSNR) *Let $u, v \in \mathbb{R}^{N\times M}$ be two discrete images. Then the MSE is given by*

$$MSE(u,v) = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(u_{i,j} - v_{i,j})^2. \tag{5.1}$$

If $u_{i,j}, v_{i,j} \in [0, 255]$, then we define the peak-signal-noise-ratio (PSNR) by

$$PSNR(u,v) = 10\log_{10}\left(\frac{255^2}{MSE(u,v)}\right) \text{ db}. \tag{5.2}$$

The PSNR is used to compare the difference between two images, given in decibel. It is a scaling of the MSE and measures the possible maximal energy of an image u to the energy of the image v degraded with noise. If we derive a PSNR over 40 for two images u, v the human eye can not distinguish between the images u and v. For more information on this, we refer to [9].
To the end of this section let us mention some technical details on the programming language C++ and the examples we provide. From now on we let the number of color channels be determined by B. So, in the case of grayscaled images $B = 1$ and for color valued images, here RGB images, $B = 3$. Furthermore, we provide our functions as templated functions. This means that we do not determine the data type of our vectors beforehand. For more information on the C++ programming language and templating we refer to [18]. As a last thing we mention that if we have an array of length n the indices in C++ start at 0 and end at $n - 1$.

5.2 Linearized Storage of Images

In this section we show how images are stored ideally. Images in a discrete two-dimensional grid $\mathcal{G}$ of size $N \times M$ can be viewed as matrices. The first element is stored at the top left corner. Then, the next entries are stored row-wise from the left to the right side. The last element of this matrix is then at the bottom right. To save computation time we do not store two-dimensional images in matrix form, but use a linearized version. In a programming language like C++, an image u in matrix form would be accessed with

$$u[i][j] = value,$$

where i resembles the i-th row and j the j-th column. We store all data in one vector, so we allocate a vector u of the size $N \cdot M$ and attach each line of the matrix to this array. Then we access elements by

$$u[j + i \cdot M] = value.$$

This method works well for grayscaled images if we map from the discret pixel grid $\mathcal{G}$ (see also Equation (3.4)) for instance to $[0, 1]$ or $\{0, ..., 255\}$. Note, g, u and $\hat{u}$ in the proposed primal-dual algorithm are stored in this fashion. In the case of color images,

here RGB (red-green-blue) images, we can extend this kind of linearized storage easily. For that, we assume that each single color channel has its own pixel grid. We have for $k = 1, ..., B$

$$\mathcal{G}_k = \Big\{(i,j) : i = 1, ..., N \text{ and } j = 1, ..., M\Big\}. \tag{5.3}$$

Overall, we attach each channel with its grid separately to the vectors consecutively. Then, we have $u \in \mathbb{R}^{N \cdot M \cdot B}$ and access one element with $k = 1, ..., B$, by

$$u[j + i \cdot M + k \cdot M \cdot N] = value.$$

Furthermore, we need to consider the variables p in the primal-dual algorithm. These are also stored as vectors, but with the extension that for a fixed point (i, j, k) we have $p_{i,j,k} \in \mathbb{R}^2$. We have two possibilities how to handle the storage of these variables. The way we choose is to allocate two vectors of the size $N \cdot M \cdot B$ and call them p_x and p_y, respectively. Another possibility is to do the same as before: attach the values of p_y to these of p_x and observe one large array which can be accessed by

$$p[j + i \cdot M + k \cdot M \cdot N + l \cdot M \cdot N \cdot B] = value,$$

with $l = 1, 2$ or for grayscaled images

$$p[j + i \cdot M + l \cdot M \cdot N] = value.$$

For us, it seemed to be necessary to use the first method for obtaining a readable and maintainable code. With this setting we are ready to present the first considered model: The ROF model.

5.3 Image Approximation using the ROF Model

In this section we show approximations of input images. Besides we show how the primal-dual algorithm can be implemented efficiently and estimate well fitting parameters for this model.

5.3.1 Implementation Issues

In our framework we implemented the ROF model in C++ on a CPU and GPU. Since the serial code is quite fast we consider the serial version in this section, but also provide run-times of the parallelized version. We set $K = \nabla$ with the proposed discretization stated in Section 3.3. Further, we consider each vector, e.g. u or $\bar{u}$, being accessed with $u_{i,j,k}$. For the approximation of color images we have that $k = 1, .., B$, in the case of grayscaled images $k = B = 1$. Further, we allocate the vectors $u, \bar{u}, g \in [0, 255]^B$ and $p_x, p_y \in \mathbb{R}$. And we suggest to initialize $\bar{u} = u = g$ and $p_x = p_y = 0$ in the primal-dual algorithm.

Computing $p^{n+1} = (\mathrm{Id} + \sigma\,\partial F^*)^{-1}(p^n + \sigma\,\nabla\bar{u}^n)$

In each iteration step we need to estimate the gradient of $\bar{u}^n$. For this we allocate two variables dx and dy at a given point (i, j, k) and compute under the premise that we discretized ∇ by forward differences with Neumann boundary conditions

$$dx = \bar{u}_{i+1,j,k} - \bar{u}_{i,j,k} \quad \text{and} \quad dy = \bar{u}_{i,j+1,k} - \bar{u}_{i,j,k},$$

where we set $dx = 0$ if $i + 1 < N$ and $dy = 0$ if $j + 1 < M$. After that, we multiply both values with σ and then add the old estimate of p, namely p_x^n and p_y^n, to dx and dy, respectively. This means

$$dx = p^n_{x_{i,j,k}} + \sigma * dx \quad \text{and} \quad dy = p^n_{y_{i,j,k}} + \sigma * dy.$$

At the end we need to apply the proximity operator of the function F^* to the variables dx and dy and save the observed value in $p^{n+1}_{i,j,k}$. We compute

$$p^{n+1}_{x_{i,j,k}} = \frac{dx}{\max(1, norm)} \quad \text{and} \quad p^{n+1}_{y_{i,j,k}} = \frac{dy}{\max(1, norm)},$$

where $norm = \sqrt{dx * dx + dy * dy}$. Then a dual ascent function could look as follows:

```
template<typename T>
void dual_asc(T* p_x, T* p_y, T* u_bar, T sigma, int M, int N, int B) {
  T dx, dy, norm;
  int index;
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        index = j + i * M + k * N * M;
        dx = i+1<N ? u_bar[j + (i+1) * M + k * N * M]-u_bar[index] : 0;
        dy = j+1<M ? u_bar[(j+1) + i * M + k * N * M]-u_bar[index] : 0;
        dx = p_x[index] + sigma * dx;
        dy = p_y[index] + sigma * dy;
        norm = sqrt(dx*dx+dy*dy);
        p_x[index] = dx / max(1.f, norm);
        p_y[index] = dy / max(1.f, norm);
      }
    }
  }
}
```

These few lines of code are used to compute the update on $p^{n+1}_{i,j,k}$ for all $i = 1, ..., N$, $j = 1, ..., M$ and $k = 1, ...B$. To be able to use this function in another framework, e.g. for minimizing the TVL1 energy, we only need to exchange the last three lines in the inner for-loops, resembling the proximity operator for F^*.

Computing $u^{n+1} = (\mathrm{Id} - \tau\,\partial\, G)^{-1}(u^n + \tau\,\nabla^T p^{n+1})$

Again, we allocate values dx, dy and additionally sum, in which we store the sum of the partial derivatives. Note, that we change the minus sign in front of the transposed nabla operator, since $K^* = \nabla^T$ is defined as the negative divergence operator. Using backward differences with Dirichlet boundary conditions, like in Definition 3.5, we compute

$$dx = p_{x_{i,j,k}} - p_{x_{i-1,j,k}} \text{ and } dy = p_{y_{i,j,k}} - p_{y_{i,j-1,k}} \text{ and } sum = \tau \cdot (dx + dy).$$

We additionally take into account that if $i + 1 < N$ or $j + 1 < M$, we have

$$dx = -p_{x_{i-1,j,k}} \quad \text{or} \quad dy = -p_{y_{i,j-1,k}}$$

and if $i > 0$ or $j > 0$, we compute

$$dx = p_{x_{i,j,k}} \quad \text{or} \quad dy = p_{y_{i,j,k}}.$$

As we already multiplied the discrete divergence with the parameter τ we further add the previous estimate $u^n_{i,j,k}$ by

$$sum = u^n_{i,j,k} + sum.$$

In a last step we apply the proximity operator for the function G and observe

$$u^{n+1}_{i,j,k} = \frac{sum + \tau\lambda g_{i,j,k}}{1 + \tau\sigma}.$$

The primal descent function could then be implemented in the following fashion:

```
template<typename T>
void primal_desc(T* p_x, T* p_y, T* u, T* g, T tau, T lambda, int M, int N,
     int B) {
  T dx, dy, sum;
  int index;
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        index = j + i * M + k * N * M;
        dx = (i+1<N ? p_x[index] : 0.f)
           - (i>0 ? p_x[j + (i-1) * M + k * N * M] : 0.f);
        dy = (j+1<M ? p_y[index] : 0.f)
           - (j>0 ? p_y[(j-1) + i * M + k * N * M] : 0.f);
        sum = u[index] + tau * (dx + dy);
        u[index] = (sum + tau * lambda * g[index]) / (1.f + tau * lambda);
      }
    }
  }
}
```

It is only left to compute the extrapolation step. This is a straightforward computation, since we just add vectors $\bar{u}, u^n, u^{n+1} \in \mathbb{R}^{N \cdot M \cdot B}$, by

$$\bar{u}_{i,j,k}^{n+1} = u_{i,j,k}^{n+1} + \theta(u_{i,j,k}^{n+1} - u_{i,j,k}^{n}).$$

We can implement this update within one for-loop starting at the index 0 and ending at index $(N \cdot M \cdot B) - 1$. We observe the following C++ function:

```
template<typename T>
void extrapolation(T* u_bar, T* u, T* u_prev, T theta, int M, int N, int B)
{
  for (int i = 0; i < N*M*B; i++) {
    u_bar[i] = u[i] + theta * (u[i] - u_prev[i]);
    u_prev[i] = u[i];
  }
}
```

Note, we store the current estimate u^n in a vector *u_prev*, because we need the previous estimate in each step of the extrapolation function. It is only left to show how the stopping criterion for the ROF model can be implemented. Following our suggestion of Subsection 3.4.3, we set $eps = 1E-6$ and take the vector u and *u_prev* as arguments in the function. As these two vectors are an element of $\mathbb{R}^{N \cdot M \cdot B}$ we can compute the norm of $||u^{n+1} - u^n||$ within one for loop, as we did in the extrapolation function. Then u corresponds to u^{n+1} and *u_prev* to u^n. So, we allocate a temporary variable *energy* set to zero and sum up for all $i = 0, ..., (M * N * B) - 1$ by

$$energy = energy + abs(u[i] - u\_prev[i]).$$

As suggested in the first remark in the work of Pock and Chambolle ([3]) we compute σ by

$$\sigma = \frac{1}{\tau * L^2} = \frac{1}{\tau * 8}.$$

This means we need to pre-set the parameter τ and λ before we can run the algorithm. Further, we set $\theta = 1$. With this, we observe the following C++ function:

```
template<typename T>
T Stop(T* u, T* u_prev, int M, int N, int B) {
  T energy = 0.f;
  for (int i = 0; i < M*N*B; i++) {
    energy += abs(u[i] - u_prev[i]);
  }
  return (energy/(M*N));
}
```

Using all the presented functions, we observe our primal-dual algorithm for the ROF model in the following C++ function:

```
template<typename T>
void ROF(T* u, T* g, T lambda, T tau, int M, int N, int B, int iter) {
  T sigma = 1.f / (tau * 8.f);
  T theta = 1.f;
  T eps = 1E-6;
  T* u_bar = new T[M*N*B];
  T* u_prev = new T[M*N*B];
  T* p_x = new T[M*N*B];
  T* p_y = new T[M*N*B];

  for (int i = 1; i <= iter; i++) {
    dual_asc(p_x, p_y, u_bar, sigma, M, N, B);
    primal_desc(p_x, p_y, u, g, tau, lambda, M, N, B);
    extrapolation(u_bar, u, u_prev, theta, M, N, B);
    if (Stop(u, u_prev, M, N, B) <= eps) {
      break;
    }
  }

  delete [] u_bar;
  delete [] u_prev;
  delete [] p_x;
  delete [] p_y;
}
```

Now, that we know how the implementation of the ROF model can efficiently be realized, we want to estimate good choices for λ and τ. We do two separate parameter estimations: At first we look for a good fit for λ. Afterwards, we estimate the τ, for which the algorithm converges quickly with respect to the suggested λ.

5.3.2 Estimation of λ

In our tests, we run the algorithm several times with different values for λ, in which we evaluate the PSNR value with $PSNR(g, u)$. Here, g is the input image and u denotes the approximation of g. We choose λ to be in

$$\{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5\}.$$

The idea to find a possible fit for λ is to find a parameter which leads to a PSNR in the range $[30, 36]$. As we assume g to consist of data and noise, we seek for a $PSNR(g, u)$ large enough to get an approximation u close to the original image g, but also small enough to make sure that noise is removed. We find that λ depends on the input image. As Table 5.1 reveals, different λ for different images have various effects on $PSNR(g, u)$. However, the structure of an input image plays an important role. As the Landscape image has far more edges, curves and color variations, it needs another choice for λ compared to the Audrey Hepburn image with simpler structures, see therefore Figure 5.2. Despite this discrepancy, we suggest $\lambda = 0.1$ as a good setting in the ROF model. With a PSNR of 29 for the Landscape image this value is close to our desired range and

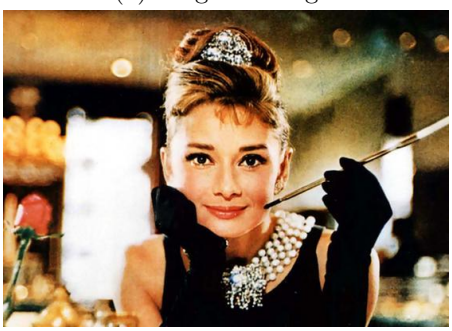
	Lena Image		Audrey Hepburn Image		Landscape Image	
λ	Iterations	PSNR	Iterations	PSNR	Iterations	PSNR
0.06	427	32 db	507	32 db	542	26 db
0.07	375	33 db	453	33 db	481	27 db
0.08	342	33 db	409	34 db	432	28 db
0.09	311	34 db	374	34 db	394	28 db
0.1	288	34 db	340	35 db	362	29 db

Table 5.1: The estimates for all λ tested in this framework.

also produces acceptable results. In the case of the Lena and Audrey Hepburn images this setting is well suited with a PSNR of 34 and 35, respectively.

(a) Original Image.

(b) $\lambda = 0.1$.



(c) Original Image.

(d) $\lambda = 0.1$.

Figure 5.2: Approximation of the Landscape and Audrey Hepburn image with the proposed λ.

5.3.3 Estimation of τ

We find that for $\lambda = 0.1$ the best choice for the time-step is $\tau = 0.73$. We observe this value by running tests for τ starting at 0.01 and increasing it by 0.01 until we reach 0.99. For this test procedure we make use of three different images: Lena (grayscaled), Hepburn (RGB) and Landscape (RGB). The suggested τ leads to a acceptable number

of iteration steps. This is also illustrated in Table 5.2. But, as it turns out, the time-step parameter τ couples with the image g and the parameter λ. Table 5.2 also reveals that the best estimates for τ vary a lot for each test image. The coupling of λ and τ can

	Lena Image		Hepburn Image		Landscape Image	
No.	τ	Iter	τ	Iter	τ	Iter
1	0.67	170	0.55	247	0.83	164
2	0.68	170	0.56	247	0.84	164
3	0.7	170	0.53	251	0.85	166
4	0.73	170	0.57	251	0.79	167
⋮						
19	0.87	173	0.73	257	0.94	172
⋮						
22	0.61	174	0.63	260	0.73	174

Table 5.2: Setting $\tau = 0.73$ provides a good guess for fast convergence.

be seen, if we set $\tau = 0.73$ and $\lambda = 0.2$ and approximate for instance the Landscape image. We only increase λ by a factor of 2, but the number of iterations for this setting drop to 121. If we decrease λ by a factor of 2, setting $\lambda = 0.05$, we obtain 288 iteration steps. We earlier discussed that a higher value for λ leads to an approximation u which is closer to the input image g. Our test shows that the algorithm converges faster if the data fidelity term is weighted higher. Overall, we see that finding a "perfect" time-step is not possible. So, we provide a good guess for which the algorithm is likely to converge fast.

	Iterations	CPU time	GPU time
Lena	170	1.13 s	0.025 s
Hepburn	257	4.13 s	0.049 s
Landscape	174	1.03 s	0.032 s

Table 5.3: Setting $\tau = 0.73$ and $\lambda = 0.1$ yields good results and a fast run-time. The parallelized version is up to 84 times faster than the serial version.

To this end, we suggest the setting $\theta = 1$, $\lambda = 0.1$ and $\tau = 0.73$ for the primal-dual algorithm, for which we derive the approximation u of g in a reasonable run-time, see also Table 5.3. In parallel we can achieve a performance increase up to a factor of 84.

5.4 Image Approximation using the TVL1 Model

We now turn our focus on the TVL1 model. As we did for the ROF model we provide the examples and parameter estimations depending on our serial C++ code in this section.

We additionally provide the run-times on the GPU. Fortunately, we can adapt a lot of operations from the previous section. The gradient operators and computation of the proximity operator for $F^*(p) = \delta_P(p)$ remain completely the same. The primal-dual algorithm with the extrapolation step is consistent and can be adapted. The only difference to the ROF model is the computation of $(\mathrm{Id} + \tau\,\partial G)^{-1}(\tilde{u})$. In Equation (3.15) we showed how proximity operator is computed. If we apply this computation to each color channel $k = 1, ..., B$ we add the index k to our notation. This means that our update on the variable u, or more precisely $u^{n+1}_{i,j,k}$, depends on the current estimate $\tilde{u}$ and the input image g. To save run-time, we pre-compute $fac = \tau * \lambda \quad \text{and} \quad est = u^n_{i,j,k} - g_{i,j,k}$. Since these values need to be computed in each if-statement, we calculate it once - before the if-statements occur - and make use of the values:

$$u^{n+1}_{i,j,k} = \begin{cases} u^n_{i,j,k} - fac & \text{if } est > fac \\ u^n_{i,j,k} + fac & \text{if } est < -fac \\ g_{i,j,k} & \text{if } |est| \leq fac \end{cases} .$$

Now, the only thing we need to change in the primal descent C++ function (*primal_desc*) is the last line of code in the inner for-loops. In this line, we compute the proximity operator, so we fit the code to the TVL1 model and obtain the following C++ function:

```
template<typename T>
void primal_desc(T* p_x, T* p_y, T* u, T* g, T tau, T lambda, int M, int N,
     int B) {
  T dx, dy, sum, est;
  T fac = tau*lambda;
  int index;
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        index = j + i * M + k * N * M;
        dx = (i+1<N ? p_x[index] : 0.f)
           - (i>0 ? p_x[j + (i-1) * M + k * N * M] : 0.f);
        dy = (j+1<M ? p_y[index] : 0.f)
           - (j>0 ? p_y[(j-1) + i * M + k * N * M] : 0.f);
        sum = u[index] + tau * (dx + dy);
        est = sum - g[index];
        if (est > fac) u[index] = sum - fac;
        if (est < -fac) u[index] = sum + fac;
        if (abs(est) <= fac) u[index] = g[index];
      }
    }
  }
}
```

With these few substitutions the TVL1 model is already set up. It is now left to provide good estimates for the input values λ and τ. We set again $\theta = 1$ and allocate $u, \bar{u}, g \in [0, 255]^B$, where B is the number of color channels.

5.4.1 Estimation of λ

For the estimation of λ we make the same assumptions as in Subsectoin 5.3.2. We choose

$$\lambda \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5\}$$

and seek for a value $PSNR(g, u)$ in the range of $[30, 36]$. We see in Table 5.4 that the parameter λ again couples with the input image. As in the ROF model, the structure of an image plays an important role. If we compare the PSNR values of the Landscape to those of the Audrey Hepburn image for some settings of λ, we see tremendous variations. Additionally, also taking the PSNR of the grayscaled Lena image into account, these differences are even more extreme. For instance if we set $\lambda = 0.7$ we observe a PSNR

	Lena Image		Hepburn Image		Landscape Image	
λ	Iter	PSNR	Iter	PSNR	Iter	PSNR
0.7	281	32 db	541	29 db	653	24 db
0.8	251	33 db	442	31 db	659	25 db
0.9	246	34 db	415	32 db	621	25 db
1.0	217	35 db	383	33 db	554	26 db
1.1	208	36 db	330	34 db	527	26 db
1.2	198	36 db	299	35 db	514	27 db

Table 5.4: Setting $\lambda = 0.7$ for grayscaled and $\lambda = 1.2$ for colored images provides a suitable framework.

of 32 for the Lena image. But the Hepburn image only attains a value of 29 and the Landscape image 24. This means that the PSNR varies in a range of 8 and for this finding a perfect fit is not possible. Overall, we suggest to set $\lambda = 0.7$ for grayscaled images and $\lambda = 1.2$ for color images. Relating to Table 5.4 this setting seems to be accurate. Let us check if there exists a time-step τ for which the algorithm convergences quickly with the suggested values for λ.

5.4.2 Estimation of τ

We apply the same procedure to estimate the best time-step parameter τ as in Subsection 5.3.3. We use again the images Lena, Audrey Hepburn and Landscape for the evaluation. There are several parameters which appear under the quickest run-times. The one which is most likely to guarantee a fast convergence is $\tau = 0.98$, cf. Table 5.5. In the parallel version we observe a tremendous run-time decrease up to a factor of 150. Even if we used grayscaled and colored images with $\lambda = 0.7$ and $\lambda = 1.2$, respectively, this τ appears to be a good choice. In Figure 5.3 (b) we show the result for our suggested framework by setting $\theta = 1, \lambda = 1.2$ and $\tau = 0.98$ using the Landscape image. Even though the TVL1 model yields good approximations, we clearly see that it is not able to preserve edges well enough if we choose λ too small. For instance in (c) of the same figure we see that some contours vanish and we see over-smoothing in the image if we set $\lambda = 0.7$.

	λ	Iterations	CPU time	GPU time
Lena	0.7	210	2.97 s	0.034 s
Hepburn	1.2	254	7.85 s	0.052 s
Landscape	1.2	210	3.69 s	0.040 s

Table 5.5: Setting $\tau = 0.98$ results in a fast run-time. Parallelization yields a performance increase up to a factor 150.

(a) The original Landscape image. (b) Approximation using $\lambda = 1.2$. (c) Approximation using $\lambda = 0.7$.

Figure 5.3: Approximation of the Landscape image with the TVL1 model.

Overall, the TVL1 model yields good approximations u and is able to remove Gaussian and salt and pepper noise from images. We will discuss this application in Section 5.7.

5.5 Image Approximation using the Real-Time Minimizer

In the case of the real-time minimizer for the Mumford-Shah model, which only has real-time applicability on a GPU, we can adapt most of the code from the ROF model. Even though it runs in real-time, we are more interested in the framework itself. As before our estimation is done with the serial C++ implementation and we additionally provide the run-times of the GPU version. The function G is almost the same as in the ROF model except the factor $\frac{\lambda}{2}$. So, we only exchange the line

$$u[index] = (sum + tau * lambda * g[index])/(1.f + tau * lambda)$$

by

$$u[index] = (sum + 2 * tau * g[index])/(1.f + 2 * tau)$$

in the primal descent function (*primal_desc*). In the dual ascent C++ function (*dual_asc*) we need to exchange two lines of code, where we take into account that the proximity

operator for $R^*_{MS}(\tilde{p})$ is computed by

$$p = (\mathrm{Id} + \sigma \partial R^*_{MS})^{-1}(\tilde{p}) \iff p_{i,j} = \begin{cases} \dfrac{\lambda}{\lambda+\sigma}\tilde{p}, & \text{if } |\tilde{p}| \leq \sqrt{\frac{\nu}{\lambda}\sigma(\sigma+2\lambda)} \\ 0 & \text{else} \end{cases}.$$

We pre-compute some values and use them afterwards in each calculation. Additionally, we add again the index k with $k = 1, ..., B$ to each variable. We set

$$fac = (2 * \lambda)/(\sigma + 2 * \lambda) \text{ and}$$
$$p\_abs = |p^n_{i,j,k}| = \sqrt{dx * dx + dy * dy}$$

and compute the bound by $bnd = \sqrt{(\nu/\lambda) * \sigma * (\sigma + 2 * \lambda)}$. Then we rewrite the proximity operator to

$$p^{n+1}_{i,j,k} = \begin{cases} fac * p^n_{i,j,k} & \text{if } p\_abs \leq bnd \\ 0 & \text{else} \end{cases}.$$

We change the computations, in which we compute p^{n+1}_x and p^{n+1}_y with respect to the proximity operator of $R^*_{MS}(\tilde{p}^n)$, and observe the new version of the C++ function by:

```
template<typename T>
void dual_asc(T* p_x, T* p_y, T* u_bar, T sigma, T lambda, T nu, int M, int
    N, int B) {
  T dx, dy;
  int index;
  T fac = (2 * lambda) / (sigma + 2 * lambda);
  T B = sqrt((nu / lambda) * sigma * (sigma + 2 * lambda));
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        index = j + i * M + k * N * M;
        dx = i+1<N ? u_bar[j + (i+1) * M + k * N * M]-u_bar[index] : 0;
        dy = j+1<M ? u_bar[(j+1) + i * M + k * N * M]-u_bar[index] : 0;
        dx = p_x[index] + sigma * dx;
        dy = p_y[index] + sigma * dy;
        p_abs = sqrt(dx*dx+dy*dy);
        p_x[index] = p_abs <= bnd ? fac * dx : 0;
        p_y[index] = p_abs <= bnd ? fac * dy : 0;
      }
    }
  }
}
```

Using the proposed stopping criterion of Subsection 3.6.2, we compute after each iteration step of the primal-dual algorithm

$$||u^{n+1} - u^n|| \leq \varepsilon,$$

with $\varepsilon = 5 \cdot 10^{-5}$. Strekalovskiy and Cremers suggest in [26] the same setting but evaluate the norm only once every ten iterations. As we used the second primal-dual algorithm

of Section 3.2, namely Algorithm 3.3, we additionally propose this version in C++ code:

```
template<typename T>
void RealTimeMinimizer(T* u, T* g, T lambda, T nu, int M, int N, int B, int
     iter) {
  T tau = 1.f / 4.f;
  T sigma = 1.f / 2.f;
  T theta = 2.f;
  T eps = 5*1E-5;
  T* u_bar = new T[M*N*B];
  T* u_prev = new T[M*N*B];
  T* p_x = new T[M*N*B];
  T* p_y = new T[M*N*B];

  for (int k = 1; k <= iter; k++) {
    dual_asc(p_x, p_y, u_bar, sigma, lambda, nu, M, N, B);
    primal_desc(p_x, p_y, u, g, tau, M, N, B);
    theta = (T)1 / sqrt(1 + 4 * tau); tau *= theta; sigma /= theta;
    extrapolation(u_bar, u, u_prev, theta, M, N, B);
    if (Stop(u, u_n, M, N, B) <= eps) {
      break;
    }
  }

  delete [] u_bar;
  delete [] u_prev;
  delete [] p_x;
  delete [] p_y;
}
```

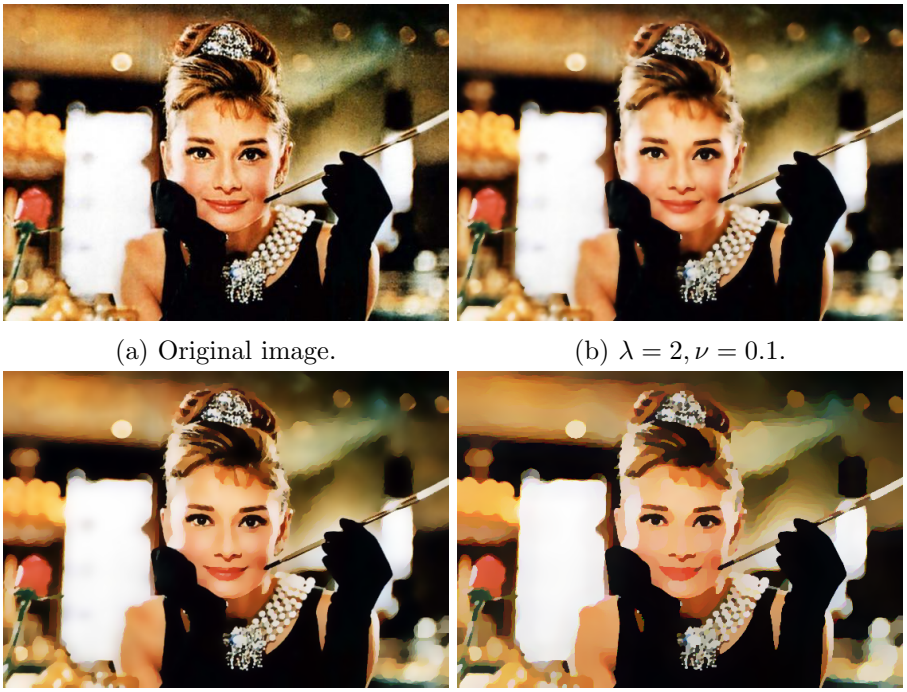
The difference in this framework is that the primal-dual function does not take the parameter τ as an argument, because Algorithm 3.3 suggests that this value is fixed by initialization. Instead the function takes the second parameter of the corresponding model, namely ν. Additionally, we compute the updates on θ, τ and σ in each iteration step of the primal-dual algorithm. For this particular framework we followed Strekalovskiy and Cremers in [26] and use that $u, \bar{u}, g \in [0,1]^B$, where B is the number of color channels.

5.5.1 Estimation of λ and ν

For this model we have two parameters which can be combined in a lot of ways. We set $\lambda \in \{2, 20, 500\}$ to determine if 2 or 20 fits better in the piecewise-smooth case and use 500 to evaluate the piecewise-constant case. Then, we need to find a suitable parameter ν for the different λ. So, we test each λ with the setting for ν being an element of

$$\{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}.$$

We use again the images of Lena and Audrey Hepburn and the Landscape image and seek for a $PSNR(g, u)$ in the range $[30, 36]$ in the piecewise-smooth case. In the piecewise-constant case, which we consider first, we choose settings with a PSNR in the range



(a) Original image. (b) $\lambda = 2, \nu = 0.1$.

(c) $\lambda = 20, \nu = 0.03$ (d) $\lambda = 500, \nu = 0.1$

Figure 5.4: Approximation of the Audrey Hepburn image with the piecewise-smooth (c) and piecewise-constant (d) Mumford-Shah model. In (b) one can see that $\lambda = 2$ is too weak and can cause blurring of the image.

[20, 25]. It turns out that several settings occur, which can be chosen in the piecewise-constant model. As possible ν all outcomes vary around $\nu = 0.1$, which we suggest to use. For the piecewise-smooth model we observe a lot of different estimations and the PSNR for each image provides different suggestions. Since this procedure is more a guessing than an evaluation, we compare the approximations u in terms of the approximations. We find that $\lambda = 2$ is too weak. These approximations are noisy and sometimes blurred. For this we conclude that a suitable setting for λ in the piecewise-smooth case is $\lambda = 20$. If we then choose $\nu = 0.03$, we observe PSNR values 31 (Lena), 25 (Landscape) and 30 (Audrey Hepburn). In terms of the Audrey Hepburn image, Figure 5.4 shows the outcomes for our suggested settings. Further, it reveals that $\lambda = 2$ is indeed not a perfect choice.

In Table 5.6 we find the run-times for our suggested setting. With a performance increase up to a factor of 47 the parallel version has indeed a real-time applicability. In this model there is no need to find a good estimate of the time-step τ, because this value is updated in each iteration seperately and is coupled with the parameter θ. As suggested in Algorithm 3.3 we pre-compute it with $\tau = \frac{1}{2d}$, where $d = 2$, since the dimension of Ω is two. Let us now turn our focus on applications to imaging. We start with one

	λ	ν	Iterations	CPU time	GPU time
Lena	20	0.03	21	0.085 s	0.004 s
Lena	500	0.1	12	0.050 s	0.002 s
Hepburn	20	0.03	40	0.360 s	0.008 s
Hepburn	500	0.1	66	0.620 s	0.013 s
Landscape	20	0.03	44	0.150 s	0.008 s
Landscape	500	0.1	54	0.190 s	0.010 s

Table 5.6: Estimated run-times on a CPU and GPU for our three test images. The parallel version is up to 47 times faster than the CPU version.

application of the real-time minimizer of the Mumford-Shah functional, namely image cartooning.

5.6 Image Cartooning

In this section we present a technique to turn an input image g into a cartooned image u. For this, we make use of the real-time minimizer for the Mumford-Shah functional. We further introduce a concept to highlight the edges in the cartooned images, first proposed in [26]. We make explicit use of the set K_{MS}, as modeled in Equation (3.19). We then present cartooned images - piecewise-constant approximations - with edge highlighting.

5.6.1 Edge Highlighting

The idea, proposed by Strekalovskiy and Cremers in [26], is to use the edge set K_{MS} in order to highlight edges in the approximation u. For that assume $(i,j) \in K_{MS}$, then we have $|(\nabla u)_{i,j}| \geq \sqrt{\frac{\nu}{\lambda}}$ by definition of K_{MS}. This notation is equivalent to

$$\frac{|(\nabla u)_{i,j}|}{\sqrt{\frac{\nu}{\lambda}}} \geq 1. \tag{5.4}$$

Further, we compute $|(\nabla u)_{i,j}| \leq \sqrt{2}$. This equation holds with the definition for ∇, cf. Definition 3.4, and having $u_{i,j} \in [0,1]$ for all $i = 1, ..., N$ and $j = 1, ..., M$. Then

$$\begin{aligned} |(\nabla u)_{i,j}| &= \sqrt{\underbrace{(u_{i+1,j} - u_{i,j})^2}_{\leq 1} + \underbrace{(u_{i,j+1} - u_{i,j})^2}_{\leq 1}} \\ &\leq \sqrt{1+1} \\ &= \sqrt{2} \end{aligned}$$

Together with Equation 5.4 it follows

$$1 \leq \frac{|(\nabla u)_{i,j}|}{\sqrt{\frac{\nu}{\lambda}}} \leq \frac{\sqrt{2}}{\sqrt{\frac{\nu}{\lambda}}}.$$

(a) Original image. (b) Without edge highlighting. (c) With edge highlighting.

(d) Original image. (e) Without edge highlighting. (f) With edge highlighting.

Figure 5.5: Cartooning of the Audrey Hepburn and Landscape image with $\lambda = 500$ and $\nu = 0.1$.

Applying the logarithm function, which is monotonically increasing, to each of the terms we get

$$0 \leq \log\left(\frac{|(\nabla u)_{i,j}|}{\sqrt{\frac{\nu}{\lambda}}}\right) \leq \log\left(\frac{\sqrt{2}}{\sqrt{\frac{\nu}{\lambda}}}\right).$$

If we now divide each term by the one on the rightmost, we observe

$$0 \leq \frac{\log\left(\frac{|(\nabla u)_{i,j}|}{\sqrt{\frac{\nu}{\lambda}}}\right)}{\log\left(\frac{\sqrt{2}}{\sqrt{\frac{\nu}{\lambda}}}\right)} \leq 1.$$

With these calculations, we now define a $v \in [0,1]$ by

$$v := \frac{\log\left(\frac{|(\nabla u)_{i,j}|}{\sqrt{\frac{\nu}{\lambda}}}\right)}{\log\left(\frac{\sqrt{2}}{\sqrt{\frac{\nu}{\lambda}}}\right)}.$$

Then, the parameter v only increases if $|(\nabla u)_{i,j}|$ increases and decreases if $|(\nabla u)_{i,j}|$ decreases, because $\sqrt{\frac{\nu}{\lambda}}$ is a constant factor and the logarithm function is monotonically increasing. The value v serves as an edge indicator. The idea is to multiply each pixel value u with

$$1 - v \in [0,1],$$

if $(i,j) \in K_{MS}$. On the other hand, if $(i,j) \notin K_{MS}$ the value u is supposed to remain unchanged. Then points in the image domain Ω with strong edges are painted darker. Using the proposed values $\lambda = 500$ and $\nu = 0.1$, as suggested in Section 5.5.1, we compare an approximation with and without edge highlighting. Figure 5.5 shows that this technique clearly yields darker edges. Since this highlighting takes place after the primal-dual algorithm, it does not slow down the code significantly.

We implement this technique of edge highlighting in our framework and test it against the pure piecewise-constant approximation. Summarizing this method in a C++-function, we get:

```
template<typename T>
void edging(T* u, T lambda, T nu, int M, int N, int B) {
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        int X = j + i * M + k * N * M;
        T dx = i + 1 < N ? u[j + (i+1) * M + k * N * M] - u[X] : 0;
        T dy = j + 1 < M ? u[j + 1 + i * M + k * N * M] - u[X] : 0;
        T norm = sqrt(dx*dx+dy*dy);
        T factor = sqrt(nu/lambda);
        if (norm >= factor) {
          T c = (T)1 / log(sqrt(2)/factor);
          u[X] = (1 - c * log(norm / factor)) * u[X];
        }
      }
    }
  }
}
```

In this section we showed that the Mumford-Shah real-time minimizer is good in cartooning images and has the advantage of the discrete modeled set K_{MS}, which can be used for edge highlighting. However, the real-time framework has another application, namely denoising. We present this case together with the ROF and TVL1 model, respectively, in the next section.

5.7 Image Denoising

The goal in image denoising is to remove noise from an input image g and get a smooth approximation u. We find different kinds of noise on images. The two most common are Gaussian and salt and pepper noise. Where Gaussian noise is relatively easy to remove, salt and pepper noise is more robust, since it resembles extrem values white and black on images. The ROF model and real-time minimizer for the Mumford-Shah model can handle Gaussian noise quite well, but are not able to remove salt and pepper noise accurately. However, the TVL1 model is able to manage both. We evaluate parameters λ, ν and τ for all models concerning the image denoising process. In this evaluation we follow the same procedure as in the previous sections. In this section we provide the best choices for the parameters. All estimations can be found online at [6].

5.7.1 Gaussian Noise

In the case of the ROF model we find another parameter as in Subsection 5.3.2. Setting $\lambda = 0.03$ yields good results and we observe a $PSNR(u, g)$ of 29, where u is the approximation and g corresponds to the original Lena image. For the real-time Mumford-Shah framework we find a lot of possible pairs (λ, ν). For this reason, we choose the pair which results in the visibly best results. So, we suggest to set $\lambda = 20$ and $\nu = 0.06$. At last we adapt the value $\lambda = 0.7$ for the TVL1 model from Subsection 5.4.1. We find that this value is a good fit to remove Gaussian noise with the TVL1 model. The approximations produced by the suggested values for λ can be seen in Figure 5.6. All three models remove Gaussian noise quite accurate. Whereas the real-time model for the Mumford-Shah functional does not need an estimation for τ, because this value is fixed, we further provide a suitable τ for the ROF and TVL1 model, respectively. In our tests we find that $\tau = 0.99$ with 359 is the best estimate for removing Gaussian noise with the ROF model. Additionally, we find in Table 5.7 some estimates for τ for the TVL1 model. If we set $\tau = 0.97$ we have a fast convergence for both: Removing Gaussian and salt and pepper noise. Because of this, we suggest this τ as our best fit for this model.

(a) Noisy image. (b) ROF: $\lambda = 0.03$. (c) TVL1: $\lambda = 0.7$. (d) MS: $\lambda = 20$ $\nu = 0.06$.

Figure 5.6: Removing Gaussian noise from the Lena image with the ROF, TVL1 and real-time Mumford-Shah frameworks.

5.7.2 Salt and Pepper Noise

Gaussian Noise				Salt & Pepper Noise			
τ	Iter	CPU time	GPU time	τ	Iter	CPU time	GPU time
0.97	435	15.00 s	0.176 s	0.98	337	3.96 s	0.054 s
0.96	436	14.90 s	0.177 s	0.97	339	3.99 s	0.054 s
0.95	437	14.98 s	0.177 s	0.96	341	4.00 s	0.054 s
0.94	438	15.05 s	0.178 s	0.95	344	4.05 s	0.054 s

Table 5.7: Setting $\tau = 0.97$ leads to fast convergence in both cases using the TVL1 model.

(a) Original image. (b) 25% noise. (c) $\lambda = 0.01$ (d) $\lambda = 0.02$

Figure 5.7: Salt and pepper denoising of the Lena image with the ROF model.

The first thing we try to evaluate: Finding a parameter λ for the ROF model, so that it can remove salt and pepper noise. In fact, we find a value, namely 0.01, see also Figure 5.7. Even though the result is far away from perfect, the ROF model is able to remove this kind of noise in some sense. If we seek for a τ in the case of denoising images degraded with salt and pepper noise, we find $\tau = 0.99$ with 724 iterations as the best fit. So, we suggest setting $\tau = 0.99$ for the ROF model. Furthermore, we are not able to find a pair (λ, ν) for the real-time minimizer of the Mumford-Shah model, which is able to remove salt and pepper noise. However, the TVL1 model, as mentioned, also handles salt and pepper noise. Figure 5.8 shows the denoising of salt and pepper noise with $\lambda = 0.7$. We see that the TVL1 model is very accurate in removing salt and pepper noise. If we test for a parameter τ to obtain fast convergence we find that $\tau = 0.97$ is the best choice, see also Table 5.7. Here, we see that this value for τ occurs under the four best estimates.

(a) Original Image. (b) 25% Noise. (c) $\lambda = 0.7$

Figure 5.8: Salt and pepper denoising of the Lena image with the TVL1 model.

In the last section we will compare these three approaches against the convex relaxed Mumford-Shah functional for image denoising. We will see that the Mumford-Shah functional is indeed the best choice for removing Gaussian noise, because this model is edge preserving. Further, note that denoising color valued images is also straightforward. We

can apply the proposed methods channel-wise and then derive denoised approximations of, for instance, RGB images.

5.8 Image Inpainting

Image inpainting is the process to restore or fill lost image data. Images which have a (huge) data loss can be inpainted by using a model based on the ROF energy. In this section we consider a modified version of the ROF model to derive a model for image inpainting. Let

$$\mathcal{D} = \{(i,j) \,|\, 1 \leq i \leq N, 1 \leq j \leq M\}$$

denote the set of all discrete locations in our image domain. Since we know that data is missing in the underlying image, we additionally define a set $\mathcal{I} \subseteq \mathcal{D}$. This set holds all indices for which we have a data loss. The key idea is to take these two sets into account in the data fidelity term. We still use the total variaton as a regularizer, because we want to penalize jumps in the approximation u. But, we approximate u only in terms of existing data. We have

$$\min_{u \in X} \left(||\nabla u||_1 + \frac{\lambda}{2} \sum_{(i,j) \in \mathcal{D} \setminus \mathcal{I}} (u_{i,j} - g_{i,j})^2 \right). \tag{5.5}$$

Remark 5.2 *According to Pock and Chambolle in [3] we have two choices for λ:*

1. *If we set $\lambda \in (0, \infty)$, the approximation u is inpainted and denoised. This means the data fidelity term not only assures to keep the original data into account, but also removes noise in regions without data loss.*

2. *Setting $\lambda = \infty$ only yields to the inpainting case. So, if we have noisy regions, where no data loss appears, we get no smoothing and noise removement in these areas.*

Using the Legendre-Fenchel conjugate to rewrite this minimization problem into a saddle-point problem remains the same as in Section 3.4, since we set $F(\nabla u) = ||\nabla u||_1$ and $G(u) = \frac{\lambda}{2} \sum_{(i,j) \in \mathcal{D} \setminus \mathcal{I}} (u_{i,j} - g_{i,j})^2$. We obtain

$$\min_{u \in X} \max_{p \in Y} \left(\langle \nabla u, p \rangle - \delta_P(p) + \frac{\lambda}{2} \sum_{(i,j) \in \mathcal{D} \setminus \mathcal{I}} (u_{i,j} - g_{i,j})^2 \right), \tag{5.6}$$

where P is as in Equation (3.10). Then we can identify $F^*(p) = \delta_P(p)$. But this also implies that the corresponding proximity operator for F^* remains the same and we get:

$$p = (\mathrm{Id} + \sigma \, \partial \, F^*)^{-1}(\tilde{p}) = \Pi_P(\tilde{p}) \iff p_{i,j} = \frac{\tilde{p}_{i,j}}{\max(1, |\tilde{p}_{i,j}|)}, \tag{5.7}$$

for all $i = 1, ..., N$ and $j = 1, ..., M$. If we want to apply our primal-dual algorithm to this model, we further need the proximity operator of the function G. In regions $\mathcal{D} \setminus \mathcal{I}$ this operator is the same as in the ROF model, since we have no changes in the data term. But we also need to take the regions $\mathcal{I}$ into account. As we set $G(u) = 0$ at each $(i, j) \in \mathcal{I}$, we compute the proximity operator by

$$(\mathrm{Id} + \tau \partial G)^{-1}(\tilde{u}) = \arg\min_{u \in \mathcal{I}} \frac{||u - \tilde{u}||_2^2}{2}.$$

Assume that $\hat{u} \in \arg\min_{u \in \mathcal{I}} \frac{||u-\tilde{u}||_2^2}{2}$, then with Proposition 2.24 we have

$$0 \in \partial \left(\frac{||u - \tilde{u}||_2^2}{2} \right) \iff u = \tilde{u}.$$

Overall, we have pointwise for all $i = 1, ..., N$ and $j = 1, ..., M$

$$u = (\mathrm{Id} + \tau \partial G)^{-1}(\tilde{u}) \iff u_{i,j} = \begin{cases} \tilde{u}_{i,j} & \text{if } (i,j) \in \mathcal{I} \\ \dfrac{\tilde{u}_{i,j} + \tau\lambda g_{i,j}}{1 + \tau\lambda} & \text{else} \end{cases}. \tag{5.8}$$

In order to extend this framework for colored images, we apply the primal-dual algorithm

(a) Original image. (b) Input image g. (c) $\lambda = 0.1, \tau = 0.97$

Figure 5.9: Inpainting of the Lena image with a data loss of seventy percent and simultanously denoising.

channel-wise. This means we add another index $k = 1, ..., B$ to all pairs (i, j) as we did in the other models before. In our implementation we do not need to change the code of the ROF model tremendously. We only need to add an additional if-statement in the primal descent function (*primal_desc*), in which we check if $(i, j, k) \in \mathcal{I}$. For this, we need to know where the discrete locations of $\mathcal{I}$ appear. We assumed that data loss appears if the color of the input image g at a position (i, j, k) is black. Then, we allocate an integer array $hash \in \mathbb{R}^{N \cdot M}$ and set each value of the array to 1 if $g_{i,j,k} = 0$ for all $k = 1, ..., B$ and 0 else. So, a black value appears if all color channels k at a position (i, j) are zero. This implies we compute the proximity operator of the function G by

using this array. If $hash_{i,j} = 1$ we set

$$u_{i,j,k} = \tilde{u}_{i,j,k} \quad \text{and} \quad u_{i,j,k} = \frac{\tilde{u}_{i,j,k} + \tau\lambda g_{i,j,k}}{1 + \tau\lambda}$$

else. Overall, this procedure can be summarized in a C++ function:

```
template<typename T>
void primal_desc(T* p_x, T* p_y, T* u, T* g, T tau, T lambda, int M, int N,
    int B) {
  T dx, dy, sum;
  int index;
  for (int k = 0; k < B; k++) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
        index = j + i * M + k * N * M;
        dx = (i+1<N ? p_x[index] : 0.f)
           - (i>0 ? p_x[j + (i-1) * M + k * N * M] : 0.f);
        dy = (j+1<M ? p_y[index] : 0.f)
           - (j>0 ? p_y[(j-1) + i * M + k * N * M] : 0.f);
        sum = u[index] + tau * (dx + dy);
        u[index] = hash[j + i * M] ? sum : (sum + tau * lambda * g[index])
            / (1.f + tau * lambda);
      }
    }
  }
}
```

(a) Original image. (b) Input image g. (c) $\lambda = \infty$

Figure 5.10: Inpainting of Lena image with a data loss of seventy percent and without denoising.

We also did not need to estimate a good fit for λ, since the fit we found in Section 5.3, namely $\lambda = 0.1$, worked perfectly. This is also illustrated in Figure 5.9. We additionally tested the case in which $\lambda = \infty$, see Figure 5.10. Indeed, the inpainting process works fine, but as stated above there was no smoothing in regions without data loss.

At last, we want to find the best choices for τ in both cases: if $\lambda \in (0, \infty)$ and if $\lambda = \infty$.

Not surprisingly, the best τ are - as in the other models - closer to 1. But, what we also find: if τ is too small, it turns out that the inpainting process is not successful, see Figure 5.11. We have no reason for this behaviour, but we assume that a small value for τ has a large influence in the proximity operator for G. So, the original data in $\mathcal{D} \setminus \mathcal{I}$ is weighted too less and for this, the algorithm is not able to fill lost data appropriately. Further, Table 5.8 reveals that the inpainting process takes more iterations, and for that

$\lambda = 0.1$				$\lambda = \infty$			
τ	Iterations	CPU time	GPU time	τ	Iterations	CPU time	GPU time
0.97	2114	22.60 s	0.303 s	0.89	1743	17.77 s	0.25 s
0.99	2245	22.25 s	0.322 s	0.88	1761	17.97 s	0.252 s
0.98	2254	22.58 s	0.322 s	0.86	1797	18.34 s	0.257 s
0.95	2275	22.77 s	0.326 s	0.98	2079	21.53 s	0.298 s
0.94	2350	23.70 s	0.337 s	0.79	2116	19.16 s	0.303 s

Table 5.8: This table shows several estimates for reasonable values τ using the ROF model in image inpainting.

a longer run-time, than pure image approximation or denoising. Even though, this run-time is still applicable in this powerful framework, even on a CPU. Of course, we get a tremendous performance increase if we parallelize this framework. This increase is up to a factor of 74. Run-time becomes a bigger issue, when talking about the convex relaxed

(a) Input image g. (b) $\lambda = 0.1, \tau = 0.01$ (c) $\lambda = \infty, \tau = 0.01$

Figure 5.11: Unsuccessful image inpainting of Lena image with $\tau = 0.01$.

Mumford-Shah functional. It is also able to remove Gaussian noise from images. We will see that it yields much exacter approximations and preserves edges far better than the ROF or TVL1 model, but is not comparable in run-time.

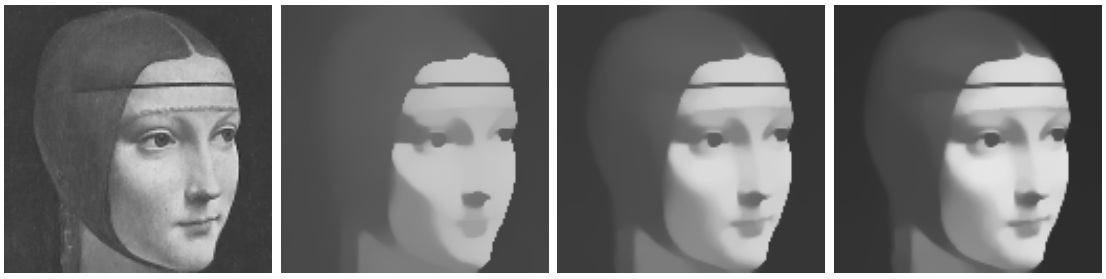
5.9 Minimizing the Mumford-Shah Functional

We show in this section that using the Lagrange Multiplier method for the convex relaxed Mumford-Shah model leads to a run-time speed up of a factor 450 compared

to the proposed method in [21]. Additionally, we show that this model is also accurate in removing Gaussian noise and can be used for image segmentation. As mentioned, the suggested method in [21] using Dykstra's projection algorithm, together with a soft-shrinkage scheme and clipping, is - even on a GPU - extremely slow. The parallel version takes several minutes to compute minimizers of the Mumford-Shah energy. We now present the results produced by these two frameworks. It turns out that both methods result in good approximations u and do not differ significantly. The big difference is the memory consumption and run-time. In [21] Pock et al. proposed their results using a 4GB GPU and $S = 32$ levels. We only had a 3GB GPU available, so we halved the number of levels to $S = 16$ for both algorithms. Fortunately, shrinking the number of added labels yields to a good approximation, too. We used our program for image approximation and denoising.

5.9.1 Best λ and ν estimation

It turns out that adapting the proposed values for λ and ν from the real-time minimizer of the Mumford-Shah functional does not lead to good approximations. This is because λ is swapped in our two definitions of the Mumford-Shah functional, see Equations (3.9) and (4.1). In the real-time framework we set $\lambda = 20$, so we try to shrink λ by a factor of ten. Testing $\lambda = 2$ and $\nu = 0.03$ yields to a good approximation. But as we test if



(a) Input image g. (b) $\lambda = 20, \nu = 0.03$ (c) $\lambda = 2$, $\nu = 0.03$ (d) $\lambda = 2, \nu = 0.01$

Figure 5.12: Finding a pair (λ, ν) with the Lena image for which the approximation is acceptable.

shrinking the parameter ν to 0.01 has any effect on the approximation, it turns out that shrinking ν leads to a slightly more detailed approximation. For that we suggest the pair $(\lambda, \nu) = (2, 0.01)$ for a piecewise-smooth approximation of the convex relaxed Mumford-Shah functional. Figure 5.12 demonstrates that this setting yields good approximations u and shows the outcomes of other parameter estimation mentioned in this subsection. The first comparison using the Dykstra framework against the Lagrange framework shows an approximation of the Ladama image. It is smooth and has sharp edges. Especially, in the ROF model - but also in the TVL1 model - it sometimes happens that edges are not as strong and sharp as in the original image. That is one big advantage of the Mumford-Shah functional. Because of its structure, namely using a edge set S_u, it is edge preserving. In Figure 5.13, we do not see a difference in the approximations

between Dykstra's projection algorithm and the approaches using Lagrange multipliers. But, when looking at Table 5.9, the Lagrange multiplier approach is more than 450 times faster and amounts only a fourth of memory usage. Note, that we do not need

(a) Input image g. (b) $\lambda = 2$, $\nu = 0.01$ (c) $\lambda = 2$, $\nu = 0.01$

Figure 5.13: Approximation of the La Dama con l'ermellino (Leonardo da Vinci) image using the Mumford-Shah functional.

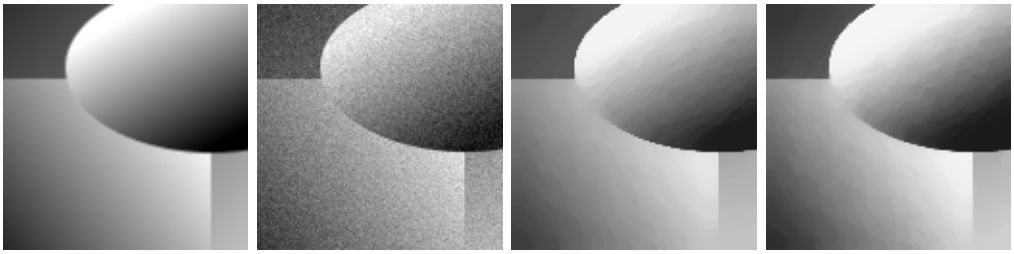
	La Dama con l'ermellino			Synthetic Test Image		
Approach	Iter	Memory	Run-Time	Iter	Memory	Run-Time
Lagrange	1000	0.228012 GB	10.42 s	1000	0.228012 GB	10.57 s
Dykstra	1000	1.040617 GB	4495 s	1000	1.040617 GB	4503 s

Table 5.9: We get a performance increase up to a factor of 450.

to estimate the parameters $\tau_u, \tau_\mu, \sigma_p$ and σ_s in the algorithm using Lagrange multiplier, because they are set at initialization. We could estimate a τ best suited, when using the primal-dual algorithm combined with Dykstra's projection algorithm. In practice we suggest to use the approach with the Lagrange multiplier method, only. Because of this, we do not seek for a best suited parameter τ. Instead, we want to show that despite the lack of performance, the Mumford-Shah model is the one which preserves edges in approximations best compared to the ROF, TVL1 and the real-time approach to the Mumford-Shah functional.

5.9.2 Denoising with the Mumford-Shah Functional

First, we present denoising using the Mumford-Shah functional. We use the Synthetic Test image to show that this algorithm is able to remove Gaussian noise and still preserves sharp edges and structures. But, when looking at the run time, the Lagrange Multiplier approach outperforms Dykstra's algorithm, see again Table 5.9. If we only use the Lagrange approach we are also able to denoise the Lena image, since this method can handle larger images quite well. Additionally, let us mention that the approximated images, the Synthetic Test image and the Ladama image, have a size of 128×128. This



(a) Original image. (b) Input image g. (c) $\lambda = 2, \nu = 0.01$ (d) $\lambda = 2, \nu = 0.01$

Figure 5.14: Denoising Gaussian noise with the two approaches for minimizing the Mumford-Shah model.

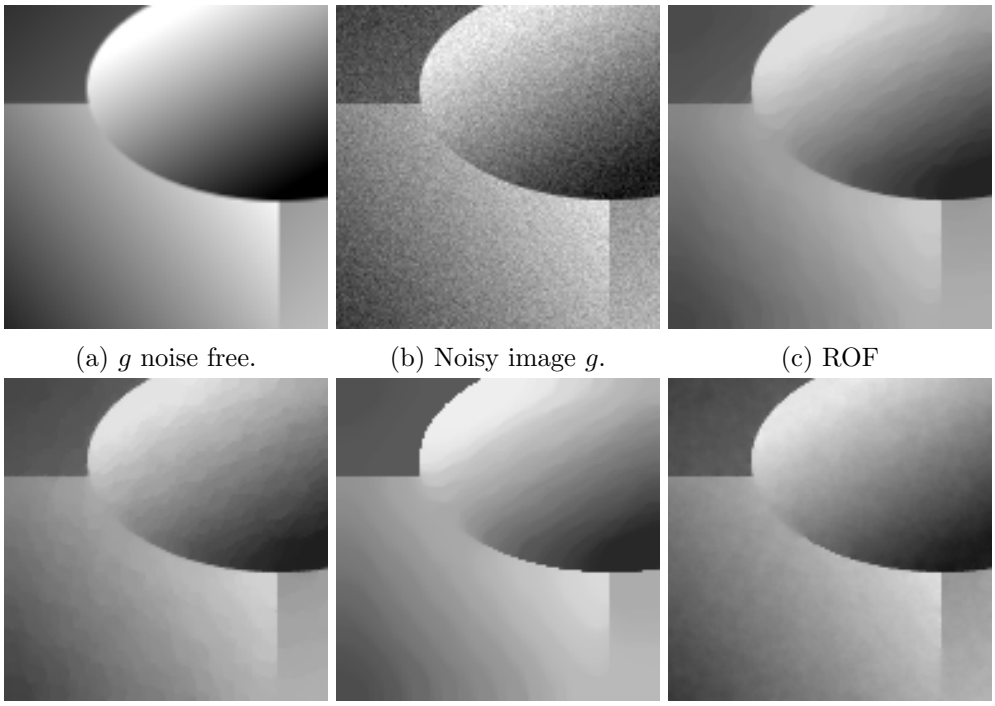
(a) Original image. (b) Gaussian Noise. (c) Approximation.

Figure 5.15: Approximation of the Lena image using the Mumford-Shah functional and 29 level.

means, having a run-time of about 10 seconds using the Lagrange multiplier approach seems a achievable run-time. On the other hand, considering larger images, for instance the Lena image with size 512×512 and $S = 16$, amounts a run-time of $41,73$ seconds. In this test we let the algorithm stop if convergence is given. The maximal number of label spaces we can apply for the Lena image is $S = 29$. This amounts 3.04488 GB memory and run-time is at $422,34$ seconds. The approximation can be viewed in Figure 5.15. However, it depends on the problem one considers. Even if this approach has no real-time applicability, it can remove noise accurately and can therefore be used in fields like medicine or scientific research.

5.9.3 Denoising Comparison

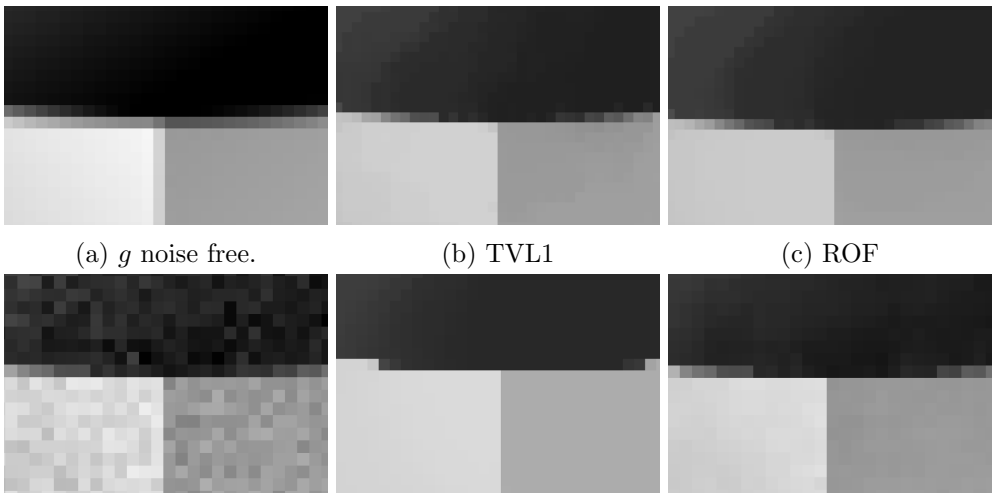
We present the comparison of the ROF, TVL1, real-time minimizer approach and convex relaxed Mumford-Shah model for denoising images. We show that the most robust model is indeed the Mumford-Shah functional. For this comparison we make use of the Lagrange Multiplier approach to the convex relaxed Mumford-Shah model and therefore set $S = 32$ to observe good approximations.



(a) g noise free. (b) Noisy image g. (c) ROF

(d) TVL1 (e) Lagrange MS (f) Real-Time MS

Figure 5.16: These pictures give an overview of denoised images using different models.



(a) g noise free. (b) TVL1 (c) ROF

(d) Noisy image g. (e) Lagrange MS (f) Real-Time MS

Figure 5.17: The convex relaxed Mumford-Shah model yields sharp edges in its approximation.

The real-time approach to the Mumford-Shah model handles this image quite well. But still, the results of this framework are not comparable to those of the convex relaxed formulation. The difference of the minimization of the Mumford-Shah functional to the minimization of the ROF and TVL1 functional is indeed that it is much more edge preserving. The edges are sharper and do not vanish. If we zoom into the images, see Figure 5.17, there is a clear partitioning of the regions. Also note that all noise is cleared out. Even if it is far more difficult to minimize the Mumford-Shah functional, it preserves edges perfectly. In the last subsection we additionally show that minimizing the Mumford-Shah functional is also a good model for image segmentation. We show that both methods, the real-time approach and the Lagrange multiplier approach for the convex relaxed model, yield good segmentations of a given input image.

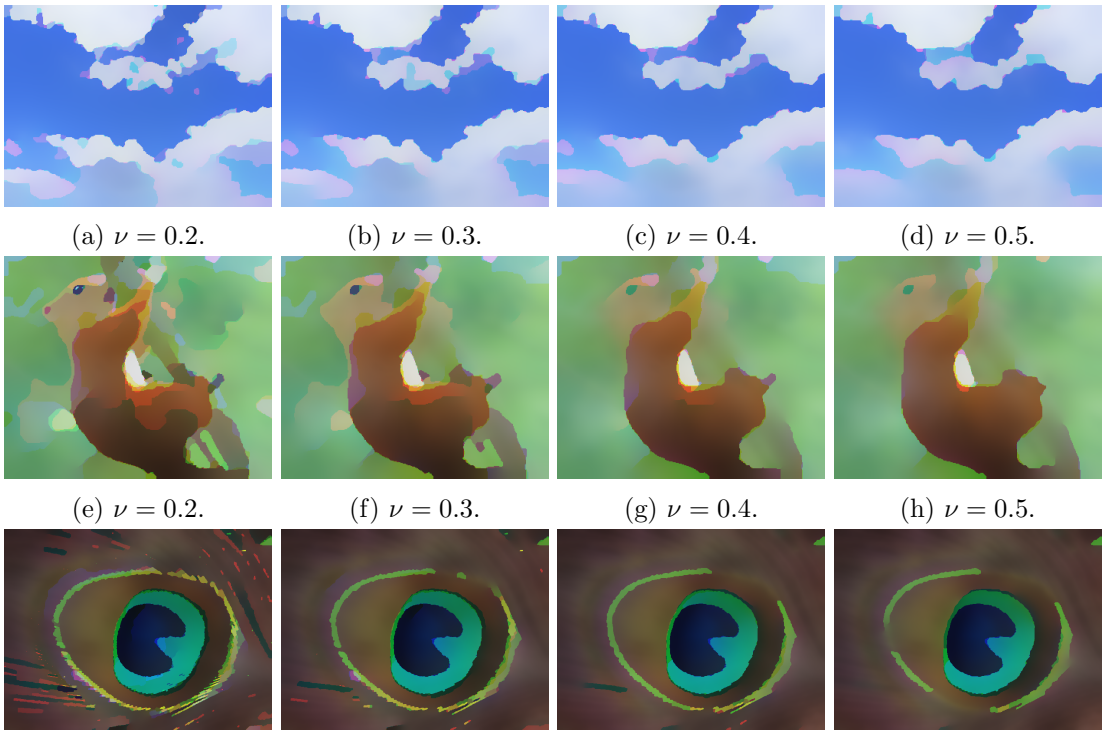
5.9.4 Image Segmentation

The last thing we want to show in this thesis is image segmentation. Since we use other images as in the above sections let us briefly introduce these images and give some references. Figure 5.18 (a) to (c) are three test images which are taken from the Github repository online found at [27] and they belong to the publication of Strekalovskiy and Cremers in [26].

(a) Blue image. (b) Squirrel image. (c) Feather image. (d) Tumor image.

Figure 5.18: Example images for image segmentation using the Mumford-Shah model. The sizes of the images are (a) 350×263, (b) 350×291, (c) 640×480 and (d) 400×535.

The image seen in (d) is a special medical image provided by Steven Keating (online at [15]). Steven Keating is a PhD student at MIT who was diagnosed with cancer in 2014. He had a brain tumor as large as a baseball. In order to find the best treatment for him and to provide useful data for science, he made all his medical data freely available. A lot of researchers used this data in order to help him to fight and defeat cancer. One part of his data are medical images of his brain tumor. We used one of these images in order to show that image segmentation is a useful application in medicine.



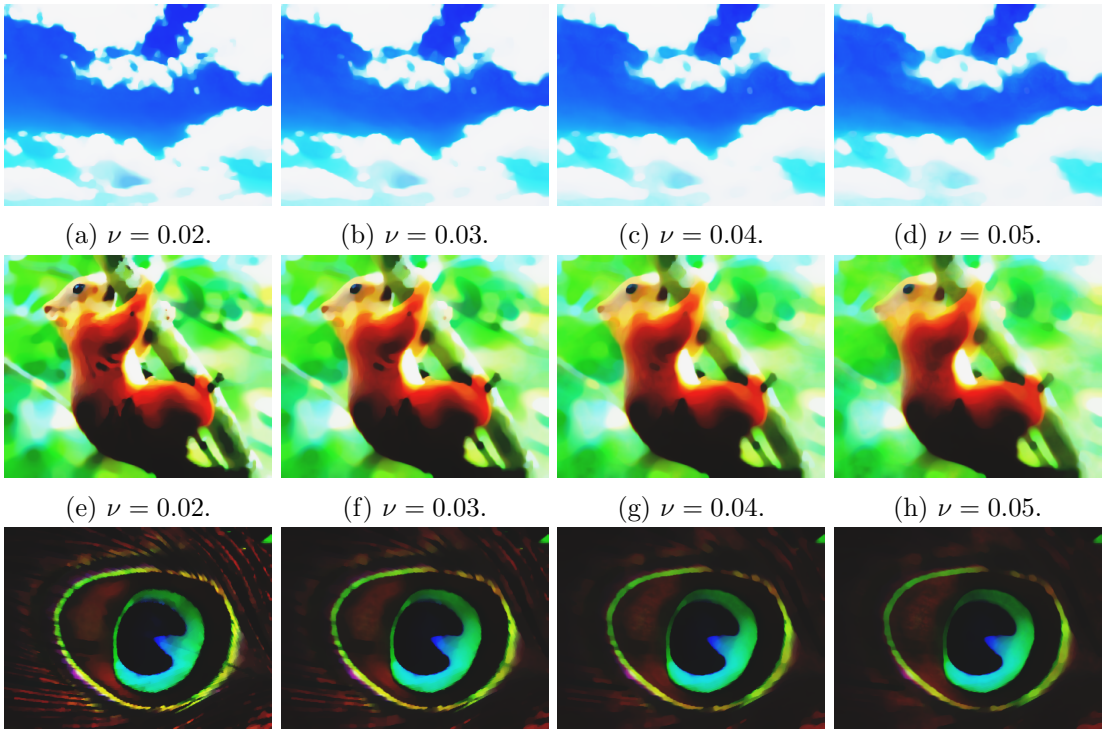
(a) $\nu = 0.2$. (b) $\nu = 0.3$. (c) $\nu = 0.4$. (d) $\nu = 0.5$.

(e) $\nu = 0.2$. (f) $\nu = 0.3$. (g) $\nu = 0.4$. (h) $\nu = 0.5$.

(i) $\nu = 0.2$. (j) $\nu = 0.3$. (k) $\nu = 0.4$. (l) $\nu = 0.5$.

Figure 5.19: Evolution process for some estimates of ν with the real-time approach and $\lambda = 500$.

Fortunately, both approaches to the Mumford-Shah functional, namely the real-time minimizer and the convex relaxation method together with the Lagrange multiplier formalism, can be used for image segmentation. The key idea is to approximate an input image g piecewise-constant, but also favor strong edges. In the real-time framework this can be derived with a high value for λ and a high value for ν, whereas in the Lagrange multiplier approach a small λ and a small ν accomplish image segmentation. So, we need to find pairs (λ, ν) which result in segmented approximations. From the previous sections we already know that we set $\lambda = 500$ for piecewise-constant approximations in the case of the real-time minimizer. In order to establish a piecewise-constant approximation for the Lagrange framework it turns out that we can set $\lambda = 0.02$. Then, it is only left to find suitable settings for ν to derive a unsupervised segmentation. Comparing the PSNR values would not make a lot of sense in this case, because our approximations will not be close to the input image in any way. They idea is to test a couple of ν and compare their outcomes by looking at the approximated images. In Figures 5.19 and 5.20 we see some evaluations for both approaches to the Mumford-Shah functional, where we set $S = 16$ in the convex relaxed Mumford-Shah model. With these images we suggest to set $\nu = 0.4$ for the real-time framework and $\nu = 0.04$ for the Lagrange framework. If we



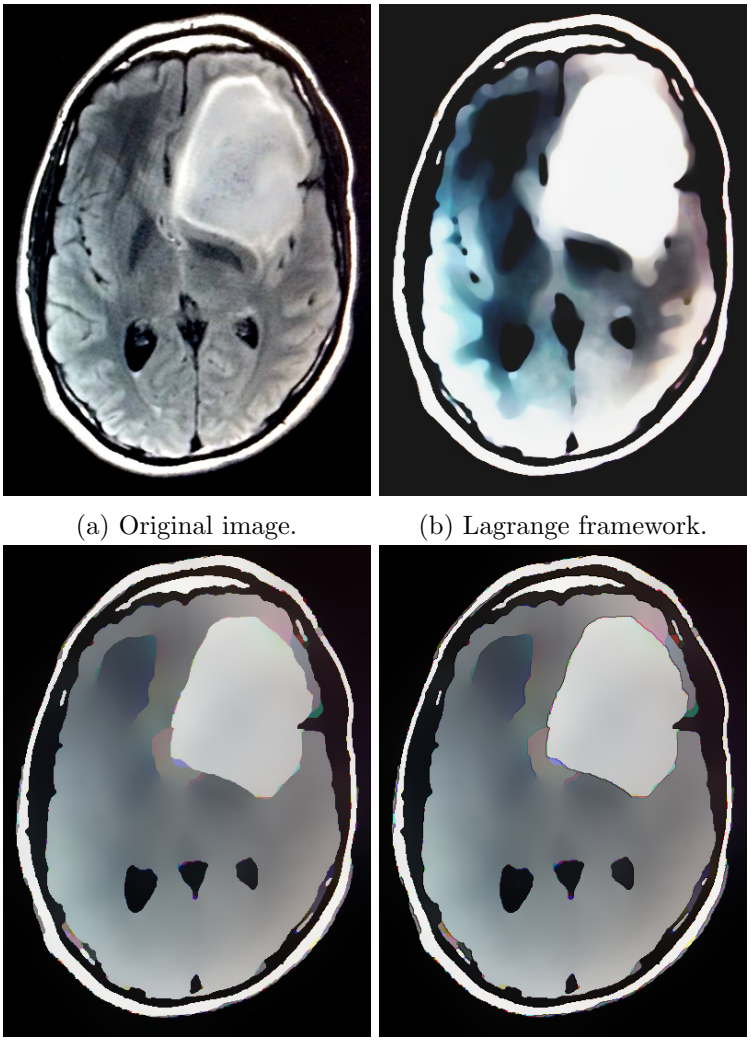
(a) $\nu = 0.02$. (b) $\nu = 0.03$. (c) $\nu = 0.04$. (d) $\nu = 0.05$.

(e) $\nu = 0.02$. (f) $\nu = 0.03$. (g) $\nu = 0.04$. (h) $\nu = 0.05$.

(i) $\nu = 0.02$. (j) $\nu = 0.03$. (k) $\nu = 0.04$. (l) $\nu = 0.05$.

Figure 5.20: Evolution process for some estimates of ν with the Lagrange framework and $\lambda = 0.02$.

apply both frameworks with the suggested parameters to the medical image of Steven Keating we observe the segmented images in Figure 5.21. For this particular image we can set $S = 32$ in the Lagrange framework to derive the possible best segmentation. We see a clear partitioning of the regions and the tumor - the big white part in the upper right - has sharp edges and is clearly separated from the other regions.



(a) Original image.

(b) Lagrange framework.

(c) Real-Time framework without edge highlighting.

(d) Real-Time framework with edge highlighting.

Figure 5.21: In (a) we see the original image. (b) shows the Lagrange approach with $\lambda = 0.02$ and $\nu = 0.04$. (c) and (d) are segmentations using the real-time framework with and without edge highlighting. Here, we set $\lambda = 500$ and $\nu = 0.4$.

In this chapter we presented a variety of possible applications to imaging with our proposed models. With the three models we are able to remove two kinds of noise, namely Gaussian and salt and pepper noise. We can turn an input image into a cartooned approximations. Further, we showed that image inpainting is an application to imaging as well as unsupervised image segmentation.

6 Conclusion

We presented three different models in image processing. These models have a high applicability in image processing and can handle a broad variety of problems. We saw that the ROF model is able to remove Gaussian noise and when slightly modified is able to inpaint lost data. Even though this model is not really edge preserving, it yields reasonable results. The other convex model we considered was the TVL1 model. It also has an advantage in its simple structure and removes Gaussian and salt and pepper noise. These two models are highly applicable in image processing and produce impressive results.
On the other hand, we computed minimizers for the non-convex Mumford-Shah functional. We proposed a framework which is able to minimize the energy of this functional in real-time. It additionally provides a discrete set with which we are able to highlight edges in the approximations. The framework itself is efficient in removing Gaussian noise and useful for unsupervised image segmentation. Unfortunately, we have no proof that this method yields the minimal value of the Mumford-Shah functional. Despite this fact, the method is a great approach to this non-convex problem.
The approach, applying convex relaxation techniques, leads possibly to the best approximations of an input image. But talking about applicability this framework is not only far away from real-time, it also has a tremendous lack of performance, even on a GPU. In order to find a minimzer of the convex relaxed Mumford-Shah functional the suggested method in the original work of Pock, Cremers, Bischof and Chambolle, using Dykstra's projection algorithm in each loop of the primal-dual algorithm, turns out to have a run-time of several minutes for very small images. For this, we reformulated the original problem and used Lagrange multipliers to derive a new formulation. With a performance increase of at least a factor 450, the convex relaxed Mumford-Shah functional can be minimized in a reasonable time.
Overall, we conclude that it depends on the underlying problem one considers. In the case of autonomously driving cars the convex relaxed Mumford-Shah functional has no application, because it can not be minimized in real-time. So, for the car industry the better choices would be the ROF or TVL1 model together with the real-time framework of the Mumford-Shah functional. But, in other fields where image processing is used, for instance in medicine or scientific research, all proposed models can be used and bring a various amount of applications. Especially, in medicine the convex relaxed Mumford-Shah functional is able to preserve contours and edges and for this does not erase necessary data.

Bibliography

[1] G. Alberti, G. Bouchitté, and G. Dal Maso. Recent convexity arguments in the calculus of variations. *"Lecture notes from the 3rd Int. Summer School on the Calculus of Variations, Pisa."*, 1998.

[2] G. Alberti, G. Bouchitté, and G. Dal Maso. Thecali- bration method for the mumford-shah functional and free- discontinuity problems. *"Calc. Var. Partial Differential Equations"*, 16(3):299–333, 2003.

[3] Chambolle Antonin and Pock Thomas. A first-order primal-dual algorithm for convex problems with applications to imaging. *"Journal of Mathematical Imaging and Vision"*, 40(1):120–145, 2011.

[4] B. Appleton and H. Talbot. Globally optimal geodesic active contours. *"J. Math. Imaging Vision"*, 23(1):67–86, 2005.

[5] K. J. Arrow, L. Hurwicz, and H. Uzawa. Studies in linear and non-linear programming. *"Stanford Mathematical Studies in the Social Sciences"*, II, 1958. with contributions by H. B. Chenery, S. M. Johnson, S. Karlin, T. Marschak, R. M. Solow.

[6] Michael Bauer. "ipaur project in the github repository of michael bauer". `https://www.github.com/BauerMichael/iPAUR`. "Computer program iPAUR".

[7] Stephen Boyd and Lieven Vandenberghe. *"Convex Optimization"*, volume 7. Cambridge University Press, New York, 2009. `http://stanford.edu/~boyd/cvxbook/`.

[8] J. P. Boyle and R. Dykstra. A method for finding projections onto the intersection of convex sets in hilbert spaces. *"Advances in order restricted statistical inference (Iowa City, Iowa, 1985)"*, 37(1):28–47, 1986.

[9] Kristian Bredies and Dirk Lorenz. *"Mathematische Bildverarbeitung, Einführung in Grundlagen und moderne Theorie"*, volume 1. Vieweg + Teubner — Springer Fachmedien Wiesbaden GmbH, 2011.

[10] John Canny. A computational approach to edge detection. volume PAMI-8, No. 6. 1986.

[11] A. Chambolle, V. Caselles, D. Cremers, M. Novaga, and T. Pock. An introduction to total variation for image analysis. In *"Theoretical Foundations and Numerical Methods for Sparse Recovery"*. De Gruyter, 2010.

[12] D. Cremers and K. Kolev. Multiview stereo and silhouette consistency via convex functionals over convex domains. *"IEEE Transactions on Pattern Analysis and Machine Intelligence"*, 33(6):1161–1174, 2011.

[13] Enrico Giusti. *"Minimal Surfaces and Functions of Bounded Variation"*, volume 80. Birkhäuser, Bosten, Basel, Stuttgart, 1984. Monographs in mathematics.

[14] ITSEEZ. "opencv". `http://opencv.org`.

[15] Steven Keating. "health data of steven keating". `http://stevenkeating.info/main.html`. Found when clicking at the button health.

[16] J. P. McKelvey. Simple transcendental expressions for the roots of cubic equations. *"Amer. J. Phys."*, 52:269–270, 1984.

[17] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *"Comm. Pure Appl. Math."*, 42:577–685, 1989.

[18] The C++ Resources Network. "cplusplus.com". `http://www.cplusplus.com`, 2016.

[19] Jorge Nocedal and Stephen J. Wright. *"Numerical Optimization"*, volume Seconde Edition. Springer Science + BusinessMedia, LLC., 2006.

[20] nVidia. "cuda education & training". `https://developer.nvidia.com/cuda-education-training`.

[21] T. Pock, D. Cremers, H. Bischof, and A. Chambolle. An algorithm for minimizing the piecewise smooth mumford-shah functional. In *"IEEE International Conference on Computer Vision (ICCV)"*, Kyoto, Japan, 2009.

[22] R. T. Rockafellar. *"Convex analysis"*, volume Reprint of the 1970 original, Princeton Paperbacks. Princeton, NJ, 1997. Princeton Landmarks in Mathematics. Princeton University Press.

[23] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *"Physica D"*, 60:259–268, 1992.

[24] Jitkomut Songsiri. Projection onto an l1-norm ball with application to identification of sparse autoregressive models. Department of Electrical Engineering, Faculty of Engineering, Chulalongkorn University, 254 Phayathai Rd., Pathumwan, Bangkok, 10330 Thailand, 2011.

[25] E. Strekalovskiy, A. Chambolle, and D. Cremers. Convex relaxation of vectorial problems with coupled regularization. *"SIAM Journal on Imaging Sciences"*, "7"("1"):"294–336", 2014.

[26] E. Strekalovskiy and D. Cremers. Real-time minimization of the piecewise smooth mumford-shah functional. In *"European Conference on Computer Vision (ECCV)"*, pages 127–141, 2014. `https://github.com/tum-vision/fastms`.

[27] Evgeny Strekalovskiy. "fastms github repository". `https://github.com/tum-vision/fastms.git`.

[28] Pock Thomas and Chambolle Antonin. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *"International Conference on Computer Vision (ICCV 2011)"*, 2011. To Appear.

[29] Eberhard Zeidler. *"Nonlinear Functional Analysis and its Applications"*, volume "I: Fixed-Point Theorems". Springer Verlag, 1986.

[30] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *"CAM Reports 08-34, UCLA, Center for Applied Math."*, 2008.

Erklärung der Selbständigkeit

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Außerdem bestätige ich ich hiermit, dass die vorgelegten Druckexemplare und die vorgelegte elektronische Version der Arbeit identisch sind, dass ich über wissenschaftlich korrektes Arbeiten und Zitieren aufgeklärt wurde und dass ich von den in §26 Abs. 5 vorgesehenen Rechtsfolgen Kenntnis habe.

Ort, Datum

Unterschrift