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 28.02.2016

Vorgelegt von:
Michael Bauer
Banater Str. 1
84061 Ergolsbach
Geboren am 23.08.1987 in Gräfelfing

Abstract

Studying three different functionals in image processing, namely ROF, TVL1 and Mumford-Shah, lead to a variety of interesting applications. These models are based on minimization problems and can efficiently be solved by using a primal-dual algorithm. Whereas minimizing the ROF and TVL1 functionals are convex optimization problems, minimization of the Mumford-Shah functional is a non-convex problem. In order to overcome the non-convexity, convex relaxation techniques are used. This leads to a convex saddle-point formulation of the problem and can therefore also be solved using the primal-dual algorithm. However, the algorithm itself converges slowly to the optimal solution in the case of the Mumford-Shah functional. 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. It creates at least a factor 450 run-time performance increase compared to the version based on Dykstra projections. Further, we consider an approach to the Mumford-Shah functional, which is able to compute the minimizer of this functional in real-time. With these three models we present a variety of applications to imaging, like image approximation, cartooning, denoising and inpainting.

Acknowledgements

The last step to obtain the title Master of Science in Computational Science, was this thesis. I could never have done it without the support of several people. Now, I get the chance to thank everybody who contributed directly or indirectly and gave me assistance, whenever I needed it.

For being patient, supportive, motivating and always positive I have to thank you so much Sylvia. For your patience also lot of thanks to my mother. For fruitful discussions, listening to all my talks at least once and correcting silly sentences I want to thank Leonie and Bernhard. For the great discussions about programming thank you Michael.

The basis of this thesis are the 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. Thank you so much for your support, input and the freedom you gave in this work.

Contents

List of Tables

List of Figures

1 Introduction

In the second decade of the twenty-first century autonomous driving seems to be the big 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 to a better. As this may comfort many people, it poses a lot of 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 car needs to decide in real-time. As one can imagine, it is extremely difficult to develop fast, stable and tractable methods.
To learn from a certain situation, the car and the computer, respectively, need 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 for this providing necessary data.
But it is not only the autonomously driving car which make use of images. Doctors use X-ray view or MRI scans in patient treatment, semiconductor companies use pictures of wafers seeking for damages on it, for example scratches or dark spots and unfortunately, images are also used in modern warfare. 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 provided the first tractable and stable edge detection algorithm, presented in 1986 (c.f. [10]).
Only three years after Canny published his work, two researchers, namely David Mumford and Jayant Shah, published a paper ([15]), whose impact is still present. To date their publication of the so called Mumford-Shah functional was cited over 4.664 times (as of February 14th 2016). They proposed to minimize the energy of the 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 to compute the minimizer is by using convex relaxation techniques. We can rewrite the non-convex minimization into a convex saddle-point problem. With this reformulation we are able to apply a primal-dual algorithm and therefore solve the optimization problem. Since, this approach leads to a long run-time, we will contribute another version of the proposed saddle-point formulation. It leads to a speed-up by a factor 450. The quickest method for minimizing the Mumford-Shah functional is prob-

ably the real-time framework proposed by Strekalovskiy and Cremers in 2014. This approach is more in the sense of modern technology and applicable for image cartooning and denoising, as well as unsupervised segmentation.
Another approach to image processing was proposed in 1992. The researchers Rudin, Osher and Fatemi focused on total variation based imaging. They stated a convex minimization problem, namely the ROF model. Applications, which arise from these models, are for instance, removing Gaussian noise from a given input image or image inpainting. Even, if this model is not as accurate as the Mumford-Shah model, we will show, that it is much easier to solve and tractable for real-time computations.
In the ROF model we find a data term based on the L^2 norm. By replacing this norm with the L^1 norm, we derive the so called TVL1 model, hence the name. It is again a convex model and the energy of the functional can therefore easily be minimized. We will show, that this is by far the best model to remove salt and pepper noise from input images. It is also accurate in removing Gaussian noise.
In this work, we present all of these models, compare them against each other and estimate applicable parameters used in the models itself. We further present the primal-dual algorithm, which is the framework to solve all minimization problems. Since, this algorithm is a solver for saddle-point problems, we also clarify how we turn a minimization into a saddle-point problem. All this is summarized in a computer program with the name *iPAUR*, which is available online. The desired hope is, that the applications of these models are used to solve the new challenges arising in industry or medicine.

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 [9], in german.

Definition 2.1 (Image) *Let $\Omega \subseteq \mathbb{R}^n$ be an image domain and C be a color space. A 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 our computers are discrete, because each singel 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 images, 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.*
- *In the case of grayscaled images we set $C = \{0, ..., 2^k - 1\}$, where the value k determines the bit depth. Usually, we find in computers $k = 8$, i.e. grayscaled images take on values in between 0 and 255, where $0 =$ black and $255 =$ white.*
- *A n-dimensional color image has $C = \{0, ..., 2^k - 1\}^n$, where - for instance - $n = 3$.*
- *As a last example 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 (red-green-blue) images with $n = 3$, $a = 0$ and $b = 1$.*

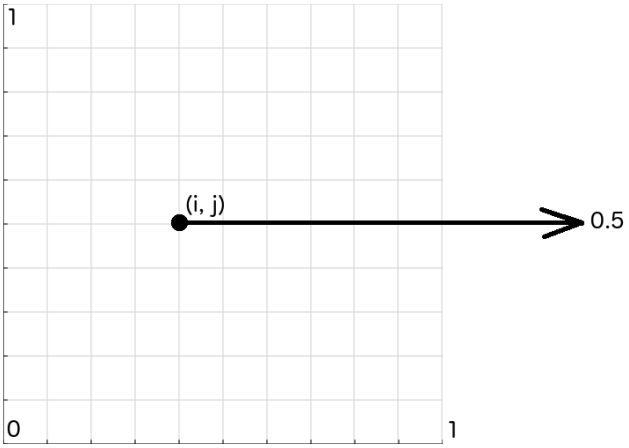


Figure 2.1: A function $u : \{0, 1, ..., 10\}^2 \to [0, 1]$, mapping a pixel (i, j) to the value $x \in [0, 1]$.

In this thesis we consider two cases 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 case a point (i, j, k) is called voxel.

In the following we use the notation $\mathbb{R}_\infty$ for the extended real-values, namely $\mathbb{R} \cup \{\infty\}$, and $\mathbb{R}_+$ is used equivalently to $\mathbb{R}_{>0}$.

2.2 Convex Optimization and Convex Analysis

In this section we cover a few topics which are related to convex analysis. Since we are facing convex optimization problems, we first define a convex program, where we mainly follow the book Convex Optimization by Stephen Boyd ([7]).

An optimization problem is of the form

$$\begin{array}{lll} \min\limits_{u \in \mathbb{R}^n} & F(u) & \\ \text{subject to} & G_i(u) \leq 0, & i = 1, ..., m \\ & H_j(u) = 0 & j = 1, ..., p, \end{array} \tag{2.1}$$

where

- $u \in \mathbb{R}^n$ is called the optimization variable,
- $F : \mathbb{R}^n \longrightarrow \mathbb{R}$ objective function or cost function,
- $G_i(u) \leq 0$ inequality constraints for all $i = 1, ..., m$,
- $H_j(u) = 0$ equality constraints for all $j = 1, ..., p$,
- $G_i : \mathbb{R}^n \longrightarrow \mathbb{R}$ for all $i = 1, ..., m$ the inequality constraint functions and

- $H_j(u) = 0$ for all $j = 1, ..., p$ the equality constraint functions.

If we have no constraints, meaning $m = p = 0$, we call the system 2.1 unconstrained. It is a convex optimization problem if all, the objectiv, equality and inequality functions, are convex. We call a point u feasible if it satisfies all constraints. The optimization problem 2.1 is said to be feasible if there exists at least one feasible point, and infeasible otherwise. We define the optimal value u^* of the system 2.1 by

$$u^* = \inf \{F(u) : G_i(u) \leq 0,\ i = 1, ..., m,\ H_j(u) = 0,\ j = 1, ..., p\},$$

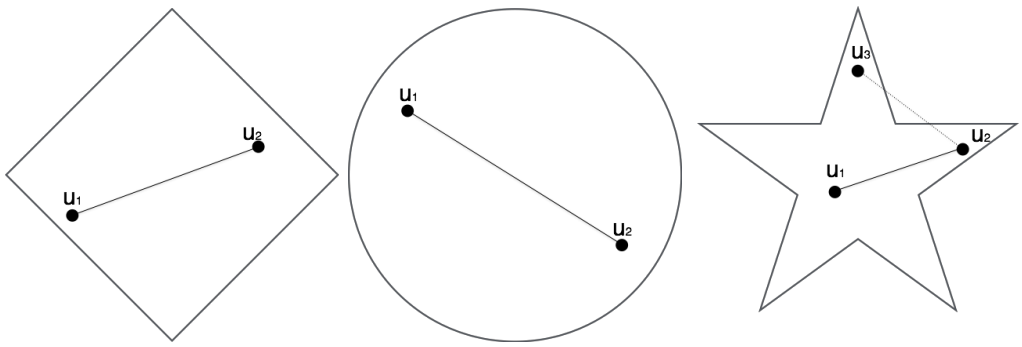
where we allow u^* to take on the extended values $\pm\infty$. We set $u^* = \infty$ if the problem is infeasible, since $\inf(\emptyset) = \infty$. If there are feasible points u_k with $F(u_k) \to -\infty$ as $k \to \infty$, then $u^* = -\infty$, and we say that problem 2.1 is unbounded below.

Since, convex optimization is based on convex analysis we define the class of convex functions and its properties. We set $X \subseteq \mathbb{R}^n$ and its dual space $Y = X^*$. Relating to Chambolle et al. in [11], we have the following basic definitions, propositions and theorems.

Definition 2.3 (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, we can always find two arbitrary points in C, such that the line segment $[u_1, u_2]$ with end points u_1, u_2 lies fully in C (see figure 2.2).



(a) l_1 unit sphere (b) l_2 unit sphere (c) Star: a non-convex set.

Figure 2.2: Three different sets, where (a) and (b) are convex, but (c) is not. (a) refers to the unit sphere of the l^1 norm, where (b) refers to the unit sphere of the l^2 norm. In (c) we see that the line segment of $[u_2, u_3]$ lies not in the set itself.

Definition 2.4 (Indicator Function, Support Function) *For any subset $C \subseteq X$ of a vector space, 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 2.5 (Convex Function, Proper, Lower-Semicontinuous) *Let $C \subseteq \mathbb{R}^n$ be a convex set. A function $F : C \longrightarrow \mathbb{R}_\infty$ 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*

$$F(u_1 t + u_2(1-t)) \leq F(u_1)t + F(u_2)(1-t) \tag{2.2}$$

  is satisfied. F is called strictly convex, if the inequality 2.2 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, c.f. example 2.8 and figure 2.3 (a). It is illustrated, together with the plot in (b) of the function

$$F(u) = \begin{cases} u^2 & x \in [-1,1] \\ -\frac{1}{2}u(u-4) & x \in (1,3], \end{cases} \tag{2.3}$$

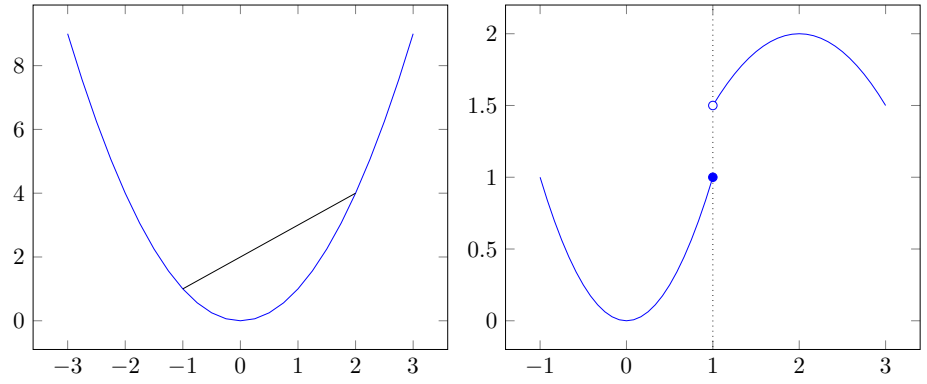
being lower-semicontinuous.

Proposition 2.6 *Let $F, G : X \longrightarrow \mathbb{R}_\infty$ be two convex functions. Then $F + G$ is also convex.*

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

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

■



(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.

Remark 2.7 (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 2.2.*

Example 2.8 *1. Let $A \in \mathbb{R}^{m\times n}$ be a real matrix and $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ a linear function with $F(u) = Au$. Then F is convex and concave, hence linear functions are affine.*

Proof *Choose $u_1, u_2 \in \mathbb{R}, t \in [0, 1]$. We get*

$$F(u_1 t + u_2(1-t)) = F(u_1 t) + F(u_2(1-t)) = F(u_1)t + F(u_2)(1-t)$$

by definition of linearity. ■

2. Let $A \in \mathbb{R}^{n\times n}$ be a real, symmetric, positiv definite matrix and $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ 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 (matrix) 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.*

3. *Each norm* $||\cdot|| : X \longrightarrow \mathbb{R}$ *on a normed vector space* X *is convex (see also figure 2.5 for the* l_1 *and* l_2 *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* $(1-t) \in [0,1]$*.* ■

Another definition of convex functions can sometimes be found in literature, for instance in [19]. Therefore, let us first introduce the epigraph and the domain of a function.

Definition 2.9 (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

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

Then we have the following definition for convex functions.

Definition 2.10 (Convex Function) *A function* $F : X \longrightarrow \mathbb{R}_\infty$ *is convex if* $\mathrm{dom}(F)$ *and* $\mathrm{epi}(F)$ *are convex sets.*

Definition 2.11 (Closed Function) *Let* $F : X \longrightarrow \mathbb{R}_\infty$ *be a convex function. Then* F *is closed if* $\mathrm{epi}(F)$ *is a closed set.*

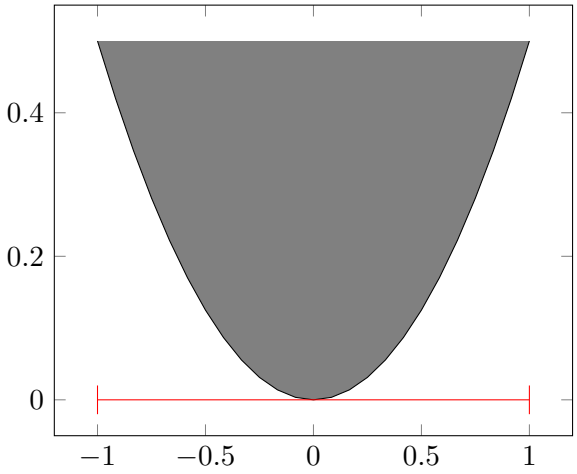


Figure 2.4: The domain (red) denoted as $\mathrm{dom}(F) = [-1, 1]$ and epigraph $\mathrm{epi}(F) = \{(u, t) \in [-1, 1] \times \mathbb{R} : t \geq \frac{1}{2}u^2\}$ (gray) of a function $F : [-1, 1] \longrightarrow \mathbb{R}$ with $F(u) = \frac{1}{2}u^2$.

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

Definition 2.12 (Legendre-Fenchel conjugate) *Let $F : X \longrightarrow \mathbb{R}_\infty$ be a convex function. We define the Legendre-Fenchel conjugate F^* of F for any $p \in Y$ by*

$$F^*(p) = \sup_{u \in X} \left(\langle p, u \rangle - F(u) \right). \tag{2.4}$$

Remark 2.13

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

The last theorem assures that we can rewrite equation 2.4 into

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

A proof of this theorem can be found in [19].

Example 2.15 *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 $F : X \to \mathbb{R}_\infty$ 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

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

which shows the desired equation.

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.

Proof *We have*

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

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 have

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

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 $u \in X$. But this implies

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

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

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

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

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* $H^{'}(x) = 0$ *implies* $x = y$. *Plugging* $||u||$ *into* H *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 $F^*(p) = \sup_{u\in X}\left(\langle u, p\rangle - \frac{||u||_2^2}{2}\right) \leq \frac{||p||_*^2}{2}$. *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. If such a function is convex, we have a generalization of the derivative, called subdifferential. For this, we want to define the subdifferential and corresponding to it, the subgradient.

Definition 2.16 (Subgradient, Subdifferential) *Let* $X \subseteq \mathbb{R}^n$ *be an open, convex set and* $F : X \longrightarrow \mathbb{R}_\infty$ *a convex function. A vector* y *is called subgradient of* F *in* $u_0 \in X$, *if*

$$F(u) \geq F(u_0) + \langle y, u - u_0\rangle \;\; \forall u \in X. \tag{2.5}$$

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 [19]. But, we state that in our computations this equality can always be used.

Proposition 2.17 *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 2.18 *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 $\mathbb{R}/\{0\}$, we compute the subgradient y

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

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

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

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

Finally, we summarize

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

2. Let $F(u) = ||u||_1$ be the convex l^1 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 example 1., we set $y_i = -1$ if $u_i < 0$ and $y_i = 1$ if $u_i > 0$. In the case, where $u_i = 0$, we observed $y_i \in [-1, 1]$. Overall, we set

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

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 2.19 *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 [19], along with a proof.

Proposition 2.20 *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 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 optimum. Further, we can verify that a solution to 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 model. This operator is defined in the next definition and is the basis of Moreau' theorem (c.f. theorem 2.22).

Definition 2.21 (Projection Operator, Proximity Operator, Moreau Envelop) *For any non-empty closed set C the projection operator for an arbitrary point $z \notin C$ on the set C is defined by*

$$\Pi_C(z) = \arg\min_{u \in C} \frac{1}{2}||z - u||_2^2, \tag{2.6}$$

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

$$\text{prox}_F^\alpha(z) = \arg\min_{u \in X} \frac{1}{2}||u - z||_2^2 + \alpha\, F(u), \tag{2.7}$$

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 [19], along with a proof.

Theorem 2.22 (Moreau's Theorem) *Let $F \in \Gamma_0(X)$ and $F^*(p) = \sup_{u \in X} \langle u, p\rangle - F(u)$ be its Legendre-Fenchel conjugate. Then*

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

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

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

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

$$z = u + p, \quad p \in \partial F(u), \tag{2.8}$$

and they are given by

$$u = \text{prox}_F^\alpha(z), \quad p = \text{prox}_{F^*}^\alpha(z). \tag{2.9}$$

Remark 2.23 *Another way to denote the proximity operator is the notation*

$$\text{prox}_F^\alpha = (\text{Id} + \alpha\partial F)^{-1}. \tag{2.10}$$

It can be derived by proposition 2.19. For that, assume that $u \in \text{prox}_F^\alpha$. Then

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

But this can be rewritten as

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

Furthermore, using equations 2.8, 2.9 and 2.10, we get

$$z = (\text{Id} + \alpha\,\partial F)^{-1}(z) + \alpha\left(\text{Id} + \frac{1}{\alpha}\partial F^*\right)^{-1}\left(\frac{z}{\alpha}\right). \tag{2.11}$$

This equation is best known as Moreau's identity.

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

Example 2.24

- *The l^p 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:*

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

According to [21] the l^2 or euclidean projection 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 $\Pi_{||\cdot||_\infty \leq 1}(p) = \min(1, \max(-1, p))$, 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 we set $F(u) = \delta_C(u)$ then we get*

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

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, which is enough for the considered classes of problems.

Definition 2.25 (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 2.26 *Let $u \in W_1^1(\Omega)$, then integration by parts and the fact that φ having compact support leads to*

$$\begin{aligned} -\langle u, \operatorname{div}(\varphi)\rangle &= -\int_\Omega u \operatorname{div}(\varphi)\, dx \\ &= -\left(\underbrace{\int_{\partial\Omega} u\, \varphi\, d\mathcal{H}^{n-1}}_{=0} - \int_\Omega \nabla u\, \varphi\, dx \right) = \int_\Omega \nabla u\, \varphi\, dx \\ &= \langle \nabla u, \varphi \rangle, \end{aligned} \tag{2.12}$$

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

$$\mathrm{TV}(u) = \int_\Omega |\nabla f| dx. \tag{2.13}$$

The idea to derive equation 2.13 is that we first evaluate the case where

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

This can be derived 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 L^1. This space lies close in the space of smooth functions with compact support and for that satisfies the assumptions.

Remark 2.27 *Note, that $W_1^1(\Omega)$ denotes the L^1-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 $u \in [a,b] \subseteq \mathbb{R}$ with $a, b \in \mathbb{R}$ and $a < b$.

Definition 2.28 (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*

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

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. But to make variation a bit more vivid, it fits perfectly.

Example 2.29 *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 explicitely make use of this class of functions, we want to give the definition(s) 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 2.30 (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 2.29 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 $u \in \Gamma_0(\Omega)$, since the total varition is convex and l.s.c. (see again [11]).

Definition 2.31 (Special functions of bounded variation [18]) *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.*

These classes of functions play indeed an important 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.

3 The ROF, TVL1 and Mumford-Shah Functional

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 convex optimization techniques to turn these minimization problems 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 - to-date - no proof, that this strategy yields the minimal value of the Mumford-Shah functional. Nonetheless, this approach leads to high-quality solutions. To compute the minimal values to all these optimization problems, we need some assumption and conventions to the models we are considering. We seek 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 u. We consider

$$minimize \text{ Functional} = minimize \text{ Data Fidelity Term} + \text{Regularizer Term}.$$

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 Pock and Chambolle ([3]). 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 $G : X \longrightarrow \mathbb{R}_+$ and $F^* : Y \longrightarrow \mathbb{R}_+$ 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:

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

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

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

and the dual problem by

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

These three different classes of problems are equivalent, if F itself is convex. The Legendre-Fenchel conjugate assures, with $F(Ku) = \sup_{p\in Y}(\langle p, Ku\rangle - F^*(p))$, that

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

Finding the optimal value $\hat{u}$ for the primal problem can be done by a gradient descent in u. On the other hand, if $\hat{p}$ is an optimal value, we find this 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 First-Order Primal-Dual Algorithm

According to Chambolle et al. in [11], the idea of the primal-dual algorithm goes back to Arrow and Hurwicz, see also [5] and the first time this algorithm was stated in such a framework was probably in [4]. As discussed 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

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

The scheme was first proposed in a paper of Zhu and Chan in [24]. Unfortunately, there is no proof of convergence for it. But in 2009 Pock, Cremers, Bischof and Chambolle published a paper ([18]), that we also consider in chapter 4, whose contribution was a provable extension of the above scheme. The idea here was to add 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, for instance, 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 [22]. Furthermore, we will not provide a proof of convergence for these algorithms. For details we refer to [3] and [18]. The general primal-dual algorithm is 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.

Remark 3.2 *In the publication of Pock and Chambolle ([3]), we find a convergence theorem, which guarantees convergences, if we find appropriate chosen parameters τ and σ. This holds for the setting $\theta = 1$. We will not provide this theorem, but discuss the main assertion of it. It states, that we need to choose τ, σ carefully by initializing the algorithm. As long as the inequality $\tau\sigma L^2 < 1$ is satisfied, where L is the bound on the norm of the linear operator K, convergence is guaranteed. The better the choice for these beforehand, the faster the algorithm converges. Estimating the best time-steps is ongoing research. A first approach was a preconditioning scheme, published in [23]. Of course, it is not best practice having an algorithm which is dependent on manually chosen parameters, which further depend on other parameters. But, on the other hand, two parameters, the time-step τ and parameter of the model λ, are highly 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]. Further, note with $\theta = 1$ the third line of the algorithm becomes*

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

As mentioned, the proposed algorithm probably goes back to Arrow-Hurwicz. To derive their original algorithm, we only need to choose $\theta = 0$. Then the last line of algorithm 3.1 becomes $\bar{u}^{n+1} = u^n$. And for that, one can drop $\bar{u}$ and derive the Arrow-Hurwicz method.

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 ([22]) 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

As images on our computers are discrete objects, with discrete pixel locations, we present the 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. And 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

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

with $p_{i,j} = \left(p^1_{i,j}, p^2_{i,j}\right)^T, q_{i,j} = \left(q^1_{i,j}, q^2_{i,j}\right)^T \in Y$ and also equipped with the euclidean norm. It is left to present the representation of the linear operator K. Relating to the definition of the total variation we had in equation 2.12 that $\langle \nabla u, p \rangle = -\langle u, \mathrm{div}\,(p) \rangle$. In a finite dimensional vector space ∇ resembles a matrix and we have $\langle \nabla u, p \rangle = \langle u, \nabla^T p \rangle$. For this, we set $\nabla^T = -\mathrm{div}$. According to the linear operator K we set $K = \nabla$ and $K^* = \nabla^T = -\mathrm{div}$.

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*

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

And the discrete divergence operator is defined in the following fashion:

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*

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

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*

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

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

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

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

3.4 The ROF Model

The first model we consider in this thesis is minimization of the ROF functional, named after Leonid I. Rudin, Stanley Osher and Emad Fatemi. They first proposed it in 1992 in [20]. It is the prototype when talking about variational methods in image processing. Before we define this model in detail, we need two important norms. The discrete isotropic total variation norm is given by

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

Additionally, we define the discrete maximum norm by

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

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 model is defined as the energy*

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

and we seek to compute the minimizer over all u.

The appearing parameter λ is used to handle the tradeoff between the regularizer, namely the total variation, and the data fidelity term. Using a larger value for λ we get an approximation u, which is closer to the input image g. If we choose λ to be small, then the weighting of the regularizer is higher and 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 leads us to:

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

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.

Remark 3.8 *In some literature, for instance [3], one finds a multiplicative factor h^2 in equation 3.8. This factor is due to discretization. Since, we assume our image domain Ω having its pixel values as discrete locations, we do not make use of the additional factor in none of our models.*

3.4.1 ROF Model as Saddle-Point Problem

To obtain the saddle-point formulation of equation 3.8, we identify $F(\nabla u) = ||\nabla u||_1$ and $G(u) = \frac{\lambda}{2}||u-g||_2^2$. Hence, we are facing the following optimization problem:

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

Using the convex conjugate, we have

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

and we observe, by plugging $F(\nabla u)$ 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 $F^*(p)$ looks like. From example 2.15 2. and the fact that the conjugate norm of the l_1 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 example 2.15 2. and 4. Plugging 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 turn our interest in solving equation 3.11, for which we need to compute the corresponding proximity operators.

3.4.2 The Proximity Operators of the ROF Model

In algorithm 3.1 we saw, how saddle-point problems can be solved efficiently. In two of the three steps of this scheme we needed the proximity operators

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

Within the ROF saddle-point problem 3.11 we have $G(u) = \frac{\lambda}{2}||u-g||_2^2$ and $F^*(p) = \delta_P(p)$. Applying equation 2.7 to F^* we get

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

This is nothing but the euclidean projection of a vector $\tilde{p} \notin P$ onto the convex set P. Since, the set P is the cross product of pointwise L^2 balls, we compute pointwise euclidean projections and get with example 2.24 1.

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

pointwise for all $i = 1, ..., N, j = 1, ..., M$. So, it only 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 2.19 we observe

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

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

Overall, the following equivalence holds

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

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

Now, that we computed the proximity operators for the ROF model, everything is set up and we are able to implement the primal-dual algorithm. The last thing which remains to show, is an appropriate stopping criterion for the algorithm.

3.4.3 Stopping Criterion

We have several possibilities to determine a valid stopping criterion. One possibility would be to compute

$$||u^{n+1} - u^n||_2 \leq \varepsilon,$$

as suggested in algorithm 3.3, which amounts little run-time and is easy to implement. But, we suggest to evaluate the energy of the functional in the primal-dual formulation in each iteration and check, whether there are not too much changes between two iterations. We compute

$$\left|\left(\frac{\lambda}{2}||u^{n+1}-g||_2^2+\langle\nabla u^{n+1},p\rangle\right)-\left(\frac{\lambda}{2}||u^{n}-g||_2^2+\langle\nabla u^{n},p\rangle\right)\right|\leq\varepsilon, \tag{3.13}$$

with $\varepsilon = 10^{-6}$. Computationally, this method leads to a slower run-time, since we do not only evaluate the euclidean norm in each iteration, we further compute the gradient and the inner product $\langle\nabla u, p\rangle$. To save run-time, we only evaluate the energies every ten iterations and compare them against each other, meaning we compute

$$\left|\left(\frac{\lambda}{2}||u^{n+10}-g||_2^2+\langle\nabla u^{n+10},p\rangle\right)-\left(\frac{\lambda}{2}||u^{n}-g||_2^2+\langle\nabla u^{n},p\rangle\right)\right|\leq\varepsilon,$$

with $\varepsilon = 10^{-6}$. It turns out, that this is a stable and tractable stopping criterion. It is the last component in our framework. Implementation issues and finding good choices for λ and τ are discussed in chapter 5. We will see, that this setting leads to good approximations u with less iteration steps of the primal-dual algorithm.

3.5 The TVL1 Model

The idea of the TVL1 functional is to replace the L^2 norm in the data fidelity term $||u-g||_2^2$ with the L^1 norm, hence the name. This norm is more robust in removing the so called salt and pepper noise, meaning that it removes single pixels with extrem values white or black. The functional is defined as follows:

Definition 3.9 (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 energy function is defined as*

$$E_{TVL1}(u) = \mathrm{TV}(u) + \lambda\int_\Omega |u-g|\,dx = \int_\Omega |\nabla u|\,dx + \lambda\int_\Omega |u-g|\,dx, \tag{3.14}$$

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 previous section for 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 of minimizing the TVL1 functional is represented by

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

where u is the approximation of an input image g.

3.5.1 TVL1 as Saddle-Point Problem

We can rewrite this minimization problem into a saddle-point problem, as we did for the ROF model. Let us first state, that our function $F(\nabla u)$, namely the total variation, remains the same as in the ROF model. We only have a change in G. We compute the convex conjugate of F, as in subsection 3.4.1, and obtain

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

Writing F^* as the indicator function with $F^* = \delta_P(p)$ and the set P of equation 3.10 we observe the primal-dual formulation

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

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

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

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

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

Proof 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

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

Using example 2.15 2. and the fact that the conjugate norm of the l_1 norm is the l_∞ norm, gives us

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

Or equivalently $G^*(q) = \delta_Q(q) + \langle q, g\rangle$. ■

The dual formulation of the TVL1 saddle-point problem is then given by

$$\begin{aligned} \max_{p\in Y} -(G^*(-K^*p) + F^*(p)) &= \max_{p\in Y} -\left(\delta_Q(-\nabla^T p) + \langle -\nabla^T p, g\rangle + \delta_P(p)\right) \\ &= \max_{p\in Y} -(\delta_Q(\operatorname{div}(p)) + \langle \operatorname{div}(p), g\rangle + \delta_P(p)) \quad (3.15) \end{aligned}$$

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

The Proximity Operators of the TVL1 Model

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

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

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 norms - a convex functional. Let $\hat{u} \in \arg\min_{u \in X} \mathcal{L}(u)$. Then with proposition 2.19 we have that

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

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 implies, we need to compute the subgradient of the absolute value function for all partial subderivatives $i = 1, ..., N$, $j = 1, ..., M$. In example 2.18 1. we saw, how the subgradient for the absolute-value function is computed. We observed

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

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

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

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

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

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

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

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

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

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

3. Finally, $y_{i,j} \in [-1,1]$. Since, we know that $\hat{u}_{i,j} = g_{i,j}$ we get

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

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

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

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

$$u = (\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u}) \Longleftrightarrow u_{i,j} = \begin{cases} \tilde{u}_{i,j} - \tau\lambda & \text{if } \tilde{u}_{i,j} - g_{i,j} > \tau\lambda, \\ \tilde{u}_{i,j} + \tau\lambda & \text{if } \tilde{u}_{i,j} - g_{i,j} < -\tau\lambda, \\ g_{i,j} & \text{if } |\tilde{u}_{i,j} - g_{i,j}| \leq \tau\lambda. \end{cases} \tag{3.16}$$

These computations are everything we need for the implementation of the primal-dual algorithm. It is left to give an appropriate stopping criterion for the algorithm.

3.5.2 Stopping Criterion

The stopping criterion for the TVL1 primal-dual algorithm is evaluated in the same fashion as in the case of the ROF model (see subsection 3.4.3). The only difference is that we do not compute the euclidean norm to evaluate the energy of the data fidelity term, but, as the model suggests, use the isotropic total variation norm instead. We evaluate every ten iteration steps

$$\left|\left(\lambda||u^{n+10} - g||_1 + \langle\nabla u^{n+10}, p\rangle\right) - \left(\lambda||u^n - g||_1 + \langle\nabla u^n, p\rangle\right)\right| \leq \varepsilon, \tag{3.17}$$

with $\varepsilon = 10^{-6}$, as before. In chapter 5 we show how λ and τ should be set to derive a stable framework and a fast convergence.

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 problems. With this, we are able to apply the primal-dual algorithm to the saddle-point problem and solve it. 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 at first.

Definition 3.10 (The Mumford-Shah Functional) *Let $\Omega \subseteq \mathbb{R}^2$ be a rectangular image domain. In order to approximate an input image $g : \Omega \longrightarrow \mathbb{R}$ in terms of a piecewise smooth function $u : \Omega \longrightarrow \mathbb{R}$, the Mumford-Shah Functional is given by*

$$E_{MS}(u) = \int_\Omega (u-g)^2 dx + R(u) = \int_\Omega (u-g)^2 dx + \lambda \int_{\Omega \setminus K} |\nabla u|^2 dx + \nu |K|, \qquad (3.18)$$

where $\nu, \lambda > 0$ are weighting parameters, $K = K_1 \cup ... \cup K_N$ and $|K|$ denotes the length of the curves in K.

Remark 3.11 *In this section we follow the representation of the Mumford-Shah functional of [22]. This is equivalent to equation 4.1.*

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 the scaling factor $\frac{\lambda}{2}$. The approximation u should be as close to the input image g as possible. The regularizer consists of two terms. 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 is not allowed to change too much in sets $\Omega \setminus K_k$. We call K_k the discontinuity sets (or jump sets) for all $k = 1, ..., D$ and curves u_k. The gradient is only taken into account in exactly these regions. The length of the discontinuity set K is also measured and taken into account in the energy $E(u)$. This means, that all curves K_k should be regular in the sense of measure theory. 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 if we set $\nabla u = 0$ in $\Omega \setminus K$. Then the model is known as the piecewise-constant Mumford-Shah model. In [22] they suggest to rewrite the regularizer in the discrete setting by using the following function:

$$R_{MS}(f) = \min(\lambda |f|^2, \nu). \qquad (3.19)$$

Then minimizing the discrete Mumford-Shah energy can be expressed by

$$\begin{aligned} \min_{u \in X} E_{MS}(u) &= \min_{u \in X} F(\nabla u) + G(u) \\ &= \min_{u \in X} \left(\left(\sum_{i=1}^{N} \sum_{j=1}^{M} R_{MS}((\nabla u)_{i,j}) \right) + ||u-g||_2^2 \right) \\ &= \min_{u \in X} \left(\sum_{i=1}^{N} \sum_{j=1}^{M} R_{MS}((\nabla u)_{i,j}) + (u_{i,j} - g_{i,j})^2 \right). \qquad (3.20) \end{aligned}$$

According to [22] 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 3.19

attains ν. In other words, if the gradient $(\nabla u)_{i,j}$ is large enough, we have for the explicit set K_{MS}:

$$K_{MS} = \left\{ (i,j) \in \Omega : |(\nabla u)_{i,j}|^2 \geq \sqrt{\frac{\nu}{\lambda}} \right\}. \tag{3.21}$$

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

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

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 3.12 *In the piecewise-constant case, where $\lambda \to \infty$, equation 3.19 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

Again, we formulate this model as a saddle-point problem to be able to apply the real-time primal-dual algorithm 3.3. In the sense of our notations from the previous section we have

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

This is the primal formulation for the discrete Mumford-Shah model. In the earlier sections we could easily apply the Legendre-Fenchel conjugate on the function F, since it is convex and for that we had $F^{**} = F$. Unfortunately, the function F, namely R_{MS}, is non-convex in this model. We could still compute the convex conjugate, but would not be able to compute an appropriate proximity operator for R_{MS}^*. Because of this reason, we assume this model to have an arbitrary regularizer R, which we further assume to be convex. We consider

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

Applying now the Legendre-Fenchel conjugate on R we get the primal-dual formulation by

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

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

Since, we do not compute R^* directly, we 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 scaling factor $\frac{\lambda}{2}$. For the convex function $\mathcal{L}(u) = \frac{||u-\tilde{u}||_2^2}{2} + \tau \frac{||u-g||_2^2}{2}$ we had

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

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

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

We then get pointwise for all $i = 1, ..., N$ and $j = 1, ..., M$

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

As we originally consider 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 2.11, which means we compute

$$p = (\mathrm{Id} + \sigma \partial R^*)^{-1}(\tilde{p}) = \tilde{p} - \sigma \left(\mathrm{Id} + \frac{1}{\sigma} \partial R\right)^{-1} \left(\frac{\tilde{p}}{\sigma}\right). \tag{3.22}$$

Then, we can plug 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. As the objective function of this minimization problem is non-convex, we can not make use of proposition 2.19. Despite that fact, we evaluate the partial derivative in the (i,j)-th direction. We observe

$$\partial_{i,j} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) \right) = x_{i,j} - \tilde{x}_{i,j} + \gamma R_{MS}(x_{i,j}) = 0. \tag{3.23}$$

The partial derivative of R_{MS} further depends on which value this 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 do a case by case analysis:

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 3.23 $x_{i,j} = \tilde{x}_{i,j}$. If this $x_{i,j}$ is the minimal value in the (i,j)-th component, we observe

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

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 3.23

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

Then, we attain the minimum at

$$\begin{aligned}
\min_{x_{i,j}} \frac{|x_{i,j} - \tilde{x}_{i,j}|^2}{2} + \gamma R_{MS}(x_{i,j}) &= \frac{|\frac{\tilde{x}_{i,j}}{1+2\gamma\lambda} - \tilde{x}_{i,j}|^2}{2} + \gamma\lambda |\frac{\tilde{x}_{i,j}}{1 + 2\gamma\lambda}|^2 \\
&= \frac{4\gamma^2\lambda^2}{2(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 + \frac{\gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{2\gamma^2\lambda^2 + \gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{(1+2\gamma\lambda)\gamma\lambda}{(1+2\gamma\lambda)^2} |\tilde{x}_{i,j}|^2 \\
&= \frac{\gamma\lambda}{1+2\gamma\lambda} |\tilde{x}_{i,j}|^2. \qquad (3.24)
\end{aligned}$$

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 3.1, we see, that in a two-dimensional case, the objective function can attain several minima. We need to check

$$\begin{aligned}
\frac{\gamma\lambda}{1+2\gamma\lambda} |\tilde{x}_{i,j}|^2 \leq \gamma\nu &\iff \frac{\lambda}{1+2\gamma\lambda} |\tilde{x}_{i,j}|^2 \leq \nu \qquad (3.25) \\
&\iff |\tilde{x}_{i,j}|^2 \leq \frac{\nu}{\lambda}(1+2\gamma\lambda) \\
&\iff |\tilde{x}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}
\end{aligned}$$

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

The proximity operator for an image approximation $u \in \mathbb{R}^{N\times M}$ of the function R_{MS} is therefore given pointwise by

$$x = (\mathrm{Id} + \gamma R_{MS})^{-1}(\tilde{x}) \Longleftrightarrow x_{i,j} = \begin{cases} \dfrac{1}{1+2\gamma\lambda}\tilde{x}_{i,j} & \text{if } |\tilde{x}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)} \\ \tilde{x}_{i,j} & \text{else,} \end{cases} \tag{3.26}$$

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

To derive the representation of the proximal operator for R^*_{MS} we now plug the operator of equation 3.26 into Moreau's identity formula of equation 3.22. Again, we need to distinguish two cases.

1. Assume that $|\tilde{p}| > \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}$. Then the proximity operator at a point (i,j) is given by
$$p = \sigma\left(\mathrm{Id} + \frac{1}{\sigma}\partial R_{MS}\right)^{-1}\left(\frac{\tilde{p}}{\sigma}\right) \Longleftrightarrow p_{i,j} = \tilde{p}_{i,j}.$$
With Moreau we get in this point (i,j)
$$p = (\mathrm{Id} + \sigma\partial R^*_{MS})^{-1}(\tilde{p}) \Longleftrightarrow p_{i,j} = 0.$$

2. Now, we assume that $|\tilde{p}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda}(1+2\gamma\lambda)}$ at some point (i,j). This leads to
$$\begin{aligned} p = (\mathrm{Id} + \sigma\partial R^*_{MS})^{-1}(\tilde{p}) \quad &= \quad \tilde{p} - \sigma\left(\mathrm{Id} + \frac{1}{\sigma}\partial R_{MS}\right)^{-1}\left(\frac{\tilde{p}}{\sigma}\right) \\ &\Longleftrightarrow \quad p_{i,j} = \tilde{p}_{i,j} - \sigma\frac{\frac{\tilde{p}_{i,j}}{\sigma}}{1+2\frac{1}{\sigma}\lambda} \\ &= \quad \tilde{p}_{i,j} - \frac{\sigma}{\sigma+2\lambda}\tilde{p}_{i,j} \\ &= \quad \left(1 - \frac{\sigma}{\sigma+2\lambda}\right)\tilde{p}_{i,j} \\ &= \quad \left(\frac{\sigma+2\lambda-\sigma}{\sigma+2\lambda}\right)\tilde{p}_{i,j} \\ &= \quad \frac{2\lambda}{\sigma+2\lambda}\tilde{p}_{i,j}. \end{aligned}$$

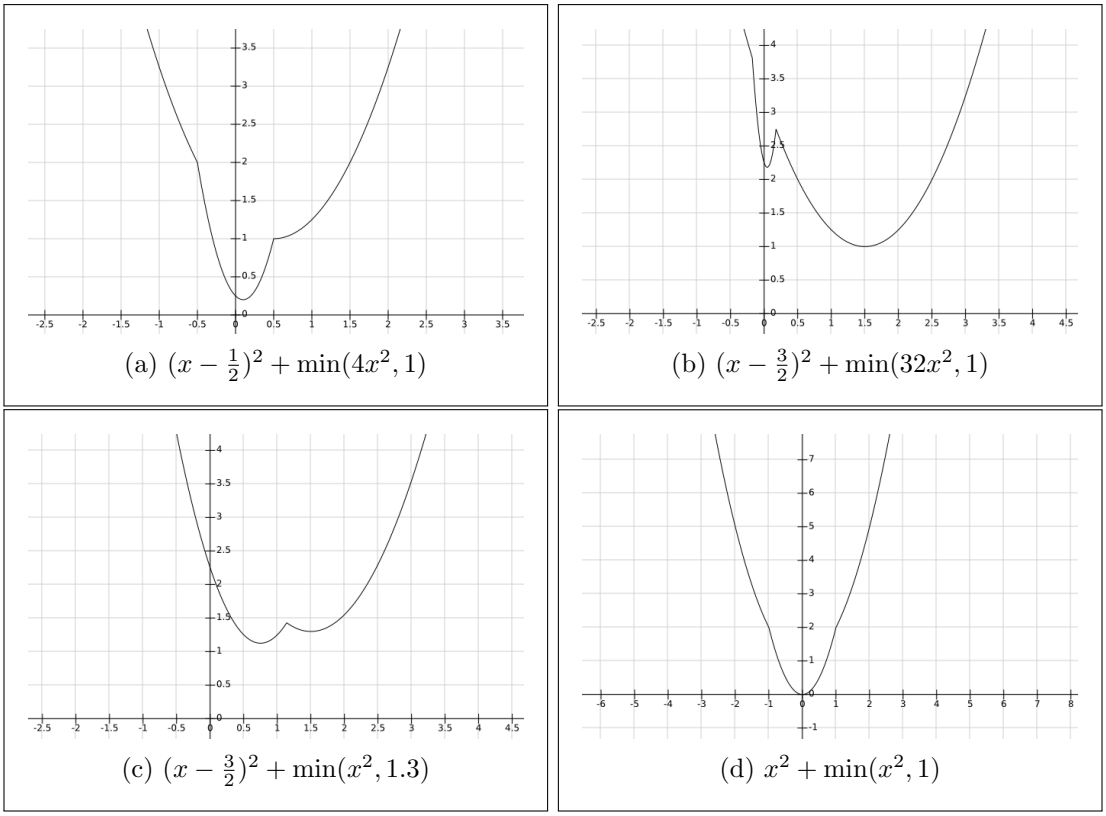
As we did for the proximity operator for R_{MS}, we also need to show for which condition these two cases hold. Using equation 3.24, inequality 3.25 and the fact that we have $\gamma = \frac{1}{\sigma}$, gives us

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

Overall, the proximity operator for R^*_{MS} is defined by

$$p = (\mathrm{Id} + \sigma \partial R^*_{MS})^{-1}(\tilde{p}) \Longleftrightarrow p_{i,j} = \begin{cases} \dfrac{\lambda}{\lambda + \sigma} \tilde{p}_{i,j}, & \text{if } |\tilde{p}_{i,j}| \leq \sqrt{\frac{\nu}{\lambda} \sigma (\sigma + 2\lambda)}, \\ 0 & \text{else,} \end{cases} \tag{3.27}$$

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



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

According to [22], there exists no theorem of convergence for this framework. Despite that fact, they show the boundedness of u^n in the piecewise smooth case, meaning $\lambda < \infty$. This is summarized in the following proposition.

Proposition 3.13 *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 [22]. Note, that in the case of this suggested framework, we also followed [22] with their proposed stopping criterion,

evaluating once every ten iterations

$$||u^{n+1} - u^n|| \leq \varepsilon, \tag{3.28}$$

with $\varepsilon = 5 \cdot 10^{-5}$ and define the underlying norm by

$$||\tilde{u}|| := \frac{1}{|\Omega|} \sum_{k=1}^{3} \sum_{i=1}^{N} \sum_{j=1}^{M} |\tilde{u}_{i,j,k}|.$$

Here, k resembles the number of color channels. We let $k = 1, 2, 3$, for color images. If we consider grayscaled images we have $k = 1$.

In this chapter we provided a framework to solve variational problems, as well as a possibility to minimize the Mumford-Shah functional. Applying this framework to images and compare the results is part of chapter 5. First, we want to show 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 3.10 to be consistent with the publication of Pock, Cremers, Bischof and Chambolle in [18], which we mainly follow in this chapter. If other results are taken into account, we make this clear.

Definition 4.1 (Mumford-Shah Functional) *Let $\Omega \subseteq \mathbb{R}^2$ be a rectangular image domain. In order to approximate an input image $f : \Omega \longrightarrow [0,1]$ in terms of a piecewise smooth function $u : \Omega \longrightarrow [0,1]$, Mumford and Shah suggested to minimize the functional*

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

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.

The difference to chapter 3 is, that we interchanged the set $K = K_1 \cup ... \cup K_N$ 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. In the case, where $u \in \Omega \subseteq \mathbb{R}^2$, the one-dimensional Hausdorff-measure of S_u is nothing but $|S_u|$. Another difference to definition 3.10 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 now the term where the image f enters the functional. 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. They applied convex relaxation techniques to approximate the Mumford-Shah functional by a convex maximization problem.

4.1 Convex Relaxation

In order to state a convex relaxed model, we need to 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 adds another label space and we then act in a three-dimensional space.

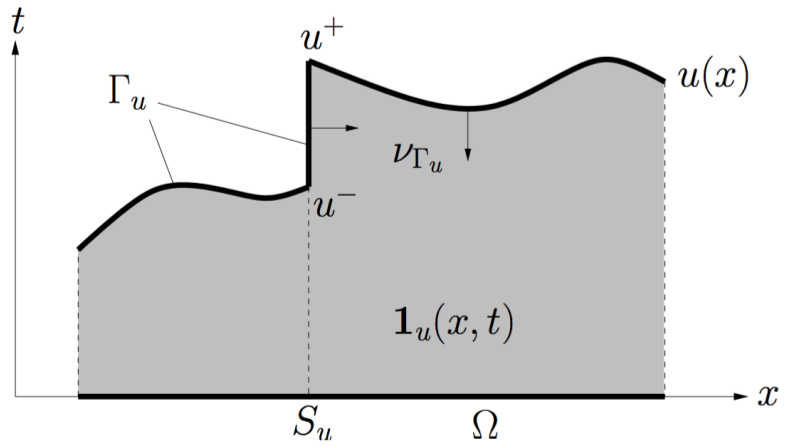


Figure 4.1: This picture (taken from [18]) shows the characteristic function $\mathbb{1}_u(x,t)$ of a function $u(x) \in SBV(\Omega)$.

The gray 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 this theorem in the fashion of [18].

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*

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

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

- *In the theorem we used, that we represent the vector φ by $\varphi(x,t) = \big(\varphi^x(x,t), \varphi^t(x,t)\big)^T$, 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.*

Our goal is to minimize 4.3. We seek to solve

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

but, 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:

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

According to [18] there is no proof, that finding an optimal pair (v^*, φ^*) for the optimization problem 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.

In the next section we consider the optimization problem 4.7 in the discrete version. Additionally, we propose the corresponding operators and compute the proximity operators.

4.2 Discrete Setting

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 [18] they consider $\Omega = [0,1]^2$ and for that the subgraph of u to be in $[0,1]^3$. This would imply that we discetize these two spaces by adding a step-size h, where for instance $h_N = \frac{1}{N}$ or $h_M = \frac{1}{M}$. Further, they consider all their operators without having this additional step-size. To be not confusing, we propose all our spaces and operators without a step size. Then the image domain is $\Omega = \{1, 2, ..., N\} \times \{1, 2, ..., M\}$ 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 continous 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

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

This notation looks now familiar to us. Here, A is the linear operator, earlier denoted with K. The set C of the minimization is defined as

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

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. We also model in the set C, that the approximation u maps into $[0, 1]$. The discrete version of the 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, $p^x \in \mathbb{R}^{N \cdot M \cdot S \cdot 2}$ and $p^t \in \mathbb{R}^{N \cdot M \cdot S}$. 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) for all $k_1, k_2 = 1, ..., S$. 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. Namely, one has that for a fixed voxel (i, j, k) one can compute $\frac{S(S-1)}{2} + S + 1$ 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

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

The large amount of several convex sets will lead us to a long run-time in the suggested framework in [18]. 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, i.e. is set to zero. The bound on the norm L depends on the discrete gradient operator, like in propositions 3.6 or 4.8. In [18] they proposed to set the discrete version of t to $\frac{k}{L}$, which is a mistake.*

We further need to define the representation of the linear operator A. It is the same as found in section 3.3, namely the discrete gradient operator, but extended to the additional label space.

Definition 4.6 (Discrete gradient operator) *We define the discrete gradient 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*

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

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

Again we have $\nabla^T : Y \longrightarrow X$ as the discrete divergence operator, with $-\nabla^T = \text{div}$, as seen before.

Definition 4.7 (Discrete divergence operator) *We define the discrete divergence 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*

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

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

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 is the same as in section 3.3 by adding the additional discretization variable $p^3_{i,j,k}$. ■

4.3 Primal and Dual Formulation

Recalling equation 4.8 and setting $A = \nabla$, we have that a minimizer of the Mumford-Shah functional can be computed by solving

$$\min_{u\in C}\max_{p\in K}\langle \nabla u, p\rangle = \min_{u\in C}\max_{p\in K}\langle \nabla u, p\rangle - \delta_K(p) + \delta_C(u). \tag{4.12}$$

As in the chapter before, we first want to formulate this saddle-point problem in the primal and the dual version. Afterwards, we want to compute the proximity operators for the corresponding functions $G(u) = \delta_C(u)$ and $F^*(p) = \delta_K(p)$.

First we compute the primal formulation of the Mumford-Shah model. Therefore, it is left to determine F from F^*. Assume the function $F(g)$ to be convex, then the Legendre-Fenchel conjugate is given by

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

For F convex we observe on the other hand, with equality $F^{**} = F$ holding, that

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

We want to compute $F(g)$ and show that the function F is indeed convex. Computing F means solving the convex problem

$$\sup_{p\in Y}\left(\langle g, p\rangle - \delta_K(p)\right) = \sup_{p\in K}\langle g, p\rangle.$$

Assume, $\hat{p} \in -\min_{p\in K}\langle p, g\rangle$, then we have with proposition 2.19

$$\max_{p\in K}\langle g, p\rangle \Longleftrightarrow -\min_{p\in K}\langle p, g\rangle \Longleftrightarrow 0 \in \partial_p\left(\langle g, \hat{p}\rangle\right) \Longleftrightarrow 0 \in g \Longrightarrow g = 0.$$

But, this means, if $g = 0$ then $F(g) = 0$. If $g < 0$ holds, then the only choice to attain a maximal value is to let p go to $-\infty$. But if $g > 0$ then $p \to \infty$, reversely. For an arbitray argument of F, we set

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

This is nothing but the indicator function of a single point and for this a convex function. Then for the primal formulation we get

$$\begin{aligned} \min_{u\in X} F(\nabla u) + G(u) \quad &\Longleftrightarrow \quad \min_{u\in C} G(u) \\ &\text{s.t.} \quad \nabla u = 0. \end{aligned}$$

Finally, we also want to evaluate the dual formulation of the Mumford-Shah saddle-point problem. We need to compute the Legendre-Fenchel conjugate of G. Since G is the indicator function of the set C we can make use of example 2.15 1. and observe

$$G^*(p) = \delta_C^*(p) = \sup_{u\in C}\langle p, u\rangle,$$

which is the support function of C. Overall, with $K^* = \nabla^T = -\text{div}$ we obtain the dual problem by

$$\max_{p \in Y} -(G^*(-K^*p) + F^*(p)) = \max_{p \in Y} -(\sup_{u \in C} \langle -\nabla^T p, u \rangle + \delta_K(p)) = \max_{p \in K} -(\sup_{u \in C} \langle \text{div}(p), u \rangle). \tag{4.13}$$

Now, assume that after the n-th iteration of the primal-dual algorithm we computed a $p^n \in K$. Then we can rewrite equation 4.13 into

$$-\max_{u \in C} \langle \nabla^T p^n, u \rangle = \min_{u \in C} \langle \nabla^T p^n, u \rangle. \tag{4.14}$$

It is left to compute the proximity operators. Since, the proximity operator of the indicator function is a euclidean projection onto the corresponding set, we need to describe how a projection on C and K can be derived. But first, we rewrite the primal-dual algorithm 3.1 to be consistent with [18].

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 $n \geq 0$*

$$\begin{cases} p^{n+1} = \Pi_K(p^n + \sigma K \bar{u}^n) \\ u^{n+1} = \Pi_C(u^n - \tau K^* p^{n+1}) \\ \bar{u}^{n+1} = 2u^{n+1} - u^n. \end{cases} \tag{4.15}$$

Here we denote Π_K and Π_C as the (euclidean) projections on the sets K and C. How we compute such projections will be discussed in the next section.

4.4 Projection onto the sets C and K

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

4.4.1 Projection onto C

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

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

As this projection goes pointwise, 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 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) *The projection of a point $u^n \in \mathbb{R}$ onto the intervall $[a, b]$, where $a, b \in \mathbb{R}$ and $a < b$ is given by*

$$u^{n+1} = \min\{b, \max\{a, u^n\}\} \tag{4.16}$$

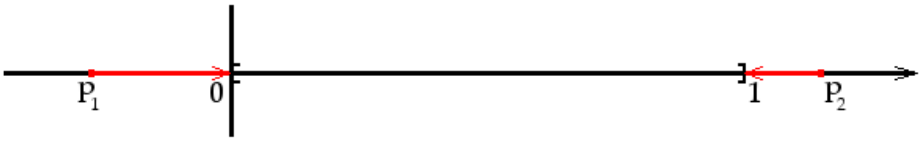


Figure 4.2: A point $p_1 < 0$ is clipped to 0, whereas a point $p_2 > 1$ is clipped to 1.

Remark 4.11

- *By projecting onto C, we also need to take care about the limits in 4.6. For that we set $u(i, j, 1) = 1$ and $u(i, j, S) = 0$ in each projection.*
- *In the case of our framework - acting in a discrete cube $\mathcal{G}$ - equation 4.16 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 $k = 1, ..., S$.

4.4.2 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. 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. We follow the notation of Cremers and Kolev in [12].

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 $i = 1, ..., P$ and $k = 1, 2, ...$, where k denotes the number of iterations. Then the algorithm finds the (only) $u^* \in X$. 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):*

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

Now, that we know, how the projection onto the entire set K can be established, we need to discuss how a projection onto the subsets of K can be computed.

4.4.3 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

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

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 additionally for all combinations (k_1, k_2) with $1 \leq k_1 \leq k_2 \leq S$. We will now deduce the projection onto these two sets. Let us start with the projection onto the parabola constraint.

4.4.4 Projection onto K_p

Since, the projection onto the set K_p goes pointwise we want to 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 as

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

We are seeking to minimize $\mathcal{L}(p, \mu)$ over all p and μ. The minimization problem we consider is convex, because the cost function is convex, the inequality constraint is convex and the feasible set $\mathbb{R}^n \times \mathbb{R}$ is also convex. For that, 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

$$\nabla \mathcal{L}(p, \mu) = \begin{pmatrix} \partial_{p^x} \mathcal{L}(p, \mu) \\ \partial_{p^t} \mathcal{L}(p, \mu) \\ \partial_{\mu} \mathcal{L}(p, \mu) \end{pmatrix} = \begin{pmatrix} p^x - p_0^x + \mu 2 \alpha p^x \\ p^t - p_0^t - \mu \\ p^t - \alpha ||p^x||_2^2 \end{pmatrix} = 0. \tag{4.17}$$

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

$$p_0^x = (\mu 2\alpha + 1)p^x \iff p^x = \frac{p_0^x}{\mu 2\alpha + 1}, \tag{4.18}$$

and the second equation leads us to

$$p^t = p_0^t + \mu. \tag{4.19}$$

In the following we discuss two different possibilities how the system 4.17 can be solved.

1. Plugging the equalities 4.18 and 4.19 into the third line of equation 4.17, we observe

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

   To solve this, 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

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

   for $k = 1, 2, ...$, until convergence. 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] they suggest setting $\mu^0 = \max\{0, -\frac{2p_0^t}{3}\}$, where they state that Newton's method converges within 7-10 iterations to a quite accurate solution.
   The projected vector p of our problem is then given by

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

which we get from equations 4.18 and 4.19. We did not apply this method to our implementation, since the primal-dual algorithm will be extremley 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 a exact solution, which can be computed within one loop of straightforward computations.

2. For the second approach we note that 4.18 and 4.19 hold and the third equation in 4.17 can be expressed by

$$p^t = \alpha||p^x||_2^2 \Longleftrightarrow p_0^t + \mu = \alpha||p^x||_2^2. \tag{4.20}$$

With equation 4.18 we can compute the solution of μ by using

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

We then plug the solution of μ in 4.20 and observe

$$\begin{aligned}
\alpha||p^x||_2^2 \quad &= \quad p_0^t + \frac{1}{2\alpha}\left(\frac{||p_0^x||_2}{||p^x||_2}\right) - \frac{1}{2\alpha} \\
&\overset{\cdot 2\alpha||p^x||_2}{\Longleftrightarrow} \quad 2\alpha^2||p^x||_2^3 = 2\alpha||p^x||_2 p_0^t + ||p_0^x||_2 - ||p^x||_2 \\
&\iff \quad 2\alpha^2||p^x||_2^3 + 2\alpha||p^x||_2 - 2\alpha p_0^t||p^x||_2 - ||p_0^x||_2 = 0. \\
&\iff \quad 2\alpha^2||p^x||_2^3 + (1 - 2\alpha p_0^t)||p^x||_2 - ||p_0^x||_2 = 0. \\
&\overset{\cdot 4\alpha}{\Longleftrightarrow} \quad 8\alpha^3||p^x||_2^3 + 4\alpha(1 - 2\alpha p_0^t)||p^x||_2 - 4\alpha||p_0^x||_2 = 0. \\
&\iff \quad (2\alpha||p^x||_2)^3 + 2(1 - 2\alpha p_0^t)2\alpha||p^x||_2 - 4\alpha||p_0^x||_2 = 0. \\
&\iff \quad t^3 + 3bt - 2a = 0,
\end{aligned} \tag{4.21}$$

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 4.21 in t can efficiently be solved using the analytical formula for solving cubic equations published by J. P. McKelvey in 1984 in [14].

The result of this work is summarized in the following algorithm. Note, that we already computed the factors a and b. The others follow with [14].

Algorithm 4.13 *If already $p_0^t \geq \alpha||p_0^x||_2^2$, the solution is given by $(p^x, p^t) = (p_0^x, p_0^t)$. Otherwise, with $a = 2\alpha||p_0^x||_2$, $b = \frac{2}{3}(1 - 2\alpha p_0^t)$, and*

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

we 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 computation time is fast. It is left to show, how we project onto K_{nl}.

4.4.5 Projection onto K_{nl}

This set is a combination of non-local constraints, meaning that in a fixed voxel (i, j, k) we sum up for all possible combinations $k_1 \leq k \leq k_2$ for all $k, k_1, k_2 = 1, ..., S$. For that reason, we can not project pointwise. We make use of a so called soft-shrinkage scheme. Let us first present the algorithm and then give a proof of concept.

Algorithm 4.14 (Soft-Shrinkage Scheme) *Let $(p^x(i,j,k))^n \in \mathbb{R}^2$ for all iterations $n = 1, 2, ...$ Then the projection $(p^x(i,j,k))^{n+1}$ of $(p^x(i,j,k))^n$ - for a fixed pair (k_1, k_2) with $1 \leq k_1 \leq k \leq k_2 \leq S$ - is computed by:*

$$(p^x(i,j,k))^{n+1} = \begin{cases} (p^x(i,j,k))^n + \dfrac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} & \text{if } k_1 \leq k \leq k_2, \\ (p^x(i,j,k))^n & \text{else,} \end{cases}$$

where $s_{k_1,k_2}, \tilde{s}_{k_1,k_2} \in \mathbb{R}^2$ with

$$\tilde{s}_{k_1,k_2} = \sum_{k_1 \leq k \leq k_2} (p^x(i,j,k))^n$$

and

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

Since, this procedure needs a clarification, we need the KKT conditions. Let us introduce these conditions by following the notation, definition and theorem of [16]. We consider the optimization problem:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0 & i \in \mathcal{E}, \\ c_i(x) \geq 0 & i \in \mathcal{I}, \end{cases} \tag{4.22}$$

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. We further want to define the feasible set Ω. As stated in section 2.2, this set 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 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 4.15 (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 a $i \in \mathcal{I}$. The definition of the active set is an important basis for the following definition.

Definition 4.16 (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.

Then, we can propose the first-order necessary conditions, which are also well known as the Karush-Kuhn-Tucker (KKT) optimality conditions.

Theorem 4.17 (Karush-Kuhn-Tucker Optimality Conditions) *Suppose that x^* is a local solution of equation 4.22, 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^*).*

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{4.23a}$$
$$c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E}, \tag{4.23b}$$
$$c_i(x^*) \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{4.23c}$$
$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{4.23d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E} \cup \mathcal{I}. \tag{4.23e}$$

Remark 4.18 *The last condition 4.23e in the 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.*

A proof of theorem 4.17 can for instance be found in [16]. Let us now proof, that algorithm 4.14 is valid, by using the KKT conditions.

Proof Let $\tilde{s}_{k_1,k_2}, s_{k_1,k_2}, p^x(i,j,k), \tilde{p}^x(i,j,k) \in \mathbb{R}^2$ and $p, \tilde{p} \in \mathbb{R}^{S\times 2}$. Further, let $\tilde{s}_{k_1,k_2}$ and s_{k_1,k_2} have the same form as in algorithm 4.14. At a fixed point (i,j) and a fixed pair (k_1,k_2) we face the following optimization problem:

$$\min_{p\in\mathbb{R}^{2\times S}} \frac{1}{2}||p-\tilde{p}||_2^2 \tag{4.24a}$$
$$\text{subject to } \left|\sum_{k_1\leq k\leq k_2} p^x(i,j,k)\right| \leq \nu. \tag{4.24b}$$

The problem states that we try to find the closest p to $\tilde{p}$ whose components fullfil the inequality constraint in 4.24b, which can also be written as

$$\frac{1}{2}\nu^2 - \frac{1}{2}\left|\sum_{k_1\leq k\leq k_2} p^x(i,j,k)\right|^2 \geq 0.$$

In the following, we drop the indices (i,j,k) and the superscript x, since we assume to be at a fixed location (i,j). For this, we denote $p^x(i,j,k)$ by p_k and $\tilde{p}^x(i,j,k)$ by $\tilde{p}_k$. Let us introduce a Lagrange multiplier variable $\lambda \in \mathbb{R}_{\geq 0}$ and observe the Lagrange function

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

Now, assume that $p^* \in \mathbb{R}^{S\times 2}$ is a feasible point, with $\mathbb{R}^2 \ni p_k^* = \tilde{p}_k + \frac{s-\tilde{s}}{k_2-k_1+1}$ in its k-th component with $|\tilde{s}_{k_1,k_2}| > \nu$, satisfying LICQ and being a local solution of system 4.24, together with a optimal λ^*. Then we get with equation 4.23a:

$$\nabla_p \mathcal{L}(p^*,\lambda^*) = \nabla_p f(p^*) - \lambda \nabla_p g(p^*) = \underbrace{\begin{pmatrix} p_1^* - \tilde{p}_1 \\ \vdots \\ \\ \vdots \\ \\ \vdots \\ p_S^* - \tilde{p}_S \end{pmatrix}}_{\in\mathbb{R}^S} - \lambda^* \underbrace{\begin{pmatrix} 0 \\ \vdots \\ -\sum\limits_{k_1\le k\le k_2} p_k^* \\ \vdots \\ -\sum\limits_{k_1\le k\le k_2} p_k^* \\ 0 \\ \vdots \end{pmatrix}}_{\in\mathbb{R}^S} = 0,$$

where the zeroes in the last vector are obtained for all components where $k < k_1$ and $k > k_2$. This means we set

$$p_k^* = \tilde{p}_k \text{ if } k < k_1 \text{ and } k > k_2.$$

Considering the i-th line, in which $(\nabla_p g(p))_i \neq 0$, we get

$$p_i^* - \tilde{p}_i + \lambda^* \sum_{k_1\le k\le k_2} p_k^* = 0.$$

Further, we observe the following identity for the feasible point p^*:

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

This means, that the inequality constraint is active for p_k^*. It is left to show, if the LICQ condition is also fullfild. We can again make us of the equalities in $(*)$ and $(**)$ of the previous calculation:

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

Because, we only observe one vector, LICQ is always true for this p^*. It is left to compute λ^* and show that it is greater or equal to zero. We get in the i-th line of $\nabla_p \mathcal{L}(p^*, \lambda^*)$:

$$
\begin{aligned}
& p_i^* - \tilde{p}_i + \lambda^* \sum_{k_1 \le k \le k_2} p_k^* = 0 \\
\iff \quad & \tilde{p}_i + \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} - \tilde{p}_i + \lambda^* \left(\underbrace{\sum_{k_1 \le k \le k_2} \left(\tilde{p}_k + \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} \right)}_{= s_{k_1,k_2}, \text{ with } (*)\&(**)} \right) = 0 \\
\iff \quad & \frac{s_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} + \lambda^* s_{k_1,k_2} = 0 \\
\iff \quad & \frac{\frac{\nu}{|\tilde{s}_{k_1,k_2}|} \tilde{s}_{k_1,k_2} - \tilde{s}_{k_1,k_2}}{k_2 - k_1 + 1} + \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|} \tilde{s}_{k_1,k_2} = 0 \\
\iff \quad & \tilde{s}_{k_1,k_2} \left(\frac{\frac{\nu}{|\tilde{s}_{k_1,k_2}|} - 1}{k_2 - k_1 + 1} + \lambda^* \frac{\nu}{|\tilde{s}_{k_1,k_2}|} \right) = 0
\end{aligned}
$$

Since, $|\tilde{s}_{k_1,k_2}| > \nu$ we can solve the last equality with

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

Applying this procedure to each pair (k_1, k_2) and all pixel positions (i, j) with $i = 1, ..., N, j = 1, ..., M$, we observe the algorithm. ■

We presented projection methods to project onto the sets C and K. As we will see in chapter 5 this approach needs a huge amount of memory and is extremley slow, even on a GPU. In the next section, we will present an alternative approach, which yields a faster computation time.

4.5 An Alternative Approach using Lagrange Multiplier

We now present an alternative approach to solve the discrete Mumford-Shah functional of subsection 4.3. The idea is to decouple the set K_{nl} to derive another saddle-point problem, which can be solved with the primal-dual algorithm 3.1. Recalling the original problem

$$\min_{u \in C} \max_{p \in K} \langle Au, p \rangle, \tag{4.25}$$

where we maximized over the whole set K, defined in equations 4.10 and 4.11. As discussed in section 4.2, this set is an intersection of several convex sets. We projected onto the non-local constraint, defined as

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

for all $i = 1, ..., N$, $j = 1, ..., M$ and all combinations (k_1, k_2) with $1 \leq k_1 \leq k_2 \leq S$, using a soft-shrinkage scheme. This scheme was part of Dykstra's projection algorithm. This meant an iterative scheme in each primal-dual iteration step with several projections onto K_{nl}. We want to derive a program with a speed up compared to the originally suggested framework in [18]. We do this by decoupling the non-local constraint. We note that in the following we drop the index representation for p^x. Instead of writing $p^x(i,j,k)$ we denote this by p^x_k for a pair (i,j) and all k. Decoupling means, that we substitute $\sum_{k_1 \leq k \leq k_2} p^x_k$ by s_{k_1,k_2} and introduce an additional constraint to take the bound on ν into account. We have

$$K_{nl} = \left\{ |s_{k_1,k_2}| \leq \nu \text{ subject to } s_{k_1,k_2} = \sum_{k_1 \leq k \leq k_2} p^x_k \right\}, \tag{4.26}$$

for all combinations $1 \leq k_1 \leq k \leq k_2 \leq S$, $s_{k_1,k_2} \in \mathbb{R}^2$ and $s \in \mathbb{R}^{N \times M \times I \times 2}$, respectively. Here, I denotes the number, which is computed by $I = \frac{S(S-1)}{2} + S$. This is 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 new function

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

in which we added an enforced term to our original problem, corresponding to the constraint in K_{nl}. Taking into account, that the constraint is an equality constraint, we have that $\mu_{k_1,k_2} \in \mathbb{R}^2$ for all $1 \leq k_1 \leq k_2 \leq S$. Additionally, we need to make sure, that the inequality $|s_{k_1,k_2}| \leq \nu$ holds. Then we obtain the following equivalence:

$$\min_{u \in C} \max_{p \in K} \langle Au, p \rangle \Longleftrightarrow \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \max_{\substack{p \in K_p \\ |s_{k_1,k_2}| \leq \nu}} \mathcal{L}(u, \mu, p, s),$$

which complies to

$$\min_{u \in C} \max_{p \in K} \langle \nabla u, p \rangle \Longleftrightarrow \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \max_{\substack{p \in K_p \\ |s_{k_1,k_2}| \leq \nu}} \langle \nabla u, p \rangle + \sum_{k_1=1}^{S} \sum_{k_2=1}^{S} \langle \mu_{k_1,k_2}, \sum_{k_1 \leq k \leq k_2} p^x_k - s_{k_1,k_2} \rangle,$$

if we use $A = \nabla$ as before. Let us proof this equivalence in the general case, where A resembles a linear operator.

Proof As we did not change either the properties on u nor the set C, we assume for the rest of the proof, that we found an optimal value u^*. Further, assume we have a optimal value s^*, for which $|s^*_{k_1,k_2}| \leq \nu$ for all possible combinations (k_1, k_2). Then, we do a case analysis:

1. If equality $\sum_{k_1 \leq k \leq k_2} p^x_k = s^*_{k_1,k_2}$ holds, the saddle-point problem reduces to
$$\max_{p \in K_p} \langle Au^*, p \rangle,$$
and with $\left|\sum_{k_1 \leq k \leq k_2} p^x_k\right| = |s_{k_1,k_2}| \leq \nu$, it is only left to project onto the parabola, like in the original problem. If the equality does not hold, we distinguish between two other cases:
   a) Let $\left|\sum_{k_1 \leq k \leq k_2} p^x_k\right| < |s^*_{k_1,k_2}|$, but as $|s^*_{k_1,k_2}| \leq \nu$ we already obtain, that $\left|\sum_{k_1 \leq k \leq k_2} p^x_k\right| \leq \nu$. Since, the conditions according to the set K_{nl} are met, we set $\mu_{k_1,k_2} = 0$. We get again
   $$\max_{p \in K_p} \langle Au^*, p \rangle.$$
   b) Now, let $\left|\sum_{k_1 \leq k \leq k_2} p^x_k\right| > |s^*_{k_1,k_2}|$. From our assumption that $|s^*_{k_1,k_2}| \leq \nu$, the only choice we have is to set $\sum_{k_1 \leq k \leq k_2} p^x_k = s^*_{k_1,k_2}$. With this we observe again
   $$\max_{p \in K_p} \langle Au^*, p \rangle,$$
   and the μ_{k_1,k_2} can be chosen arbitrarily. This shows, the equivalence of the two problems.

■

As discussed in section 3.1, solving the saddle-point problem is equivalent to solving the primal problem, a minimization problem in the primal variable, or solving the dual problem, a maximization problem in the dual variable. Doing a gradient descent in the primal and a gradient ascent in the dual variable means, 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. We compute

$$\frac{\partial \mathcal{L}(u, \mu, p, s)}{\partial u} = A^T p \tag{4.27}$$

$$\frac{\partial \mathcal{L}(u, \mu, p, s)}{\partial s_{k_1,k_2}} = -\sum_{k_1=1}^{S} \sum_{k_2=1}^{S} \mu_{k_1,k_2} \tag{4.28}$$

$$\frac{\partial \mathcal{L}(u, \mu, p, s)}{\partial \mu_{k_1,k_2}} = \sum_{k_1=1}^{S} \sum_{k_2=1}^{S} \left(\sum_{k_1 \leq k \leq k_2} p^x_k - s_{k_1,k_2} \right) \tag{4.29}$$

$$\frac{\partial \mathcal{L}(u, \mu, p, s)}{\partial p} = Au + \hat{p}. \tag{4.30}$$

As a last step we need to verify, how $\hat{p}$ is computed. We therefore consider the partial derivative in the l-th component in the direction p.

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

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 $I = \frac{S(S-1)}{2} + S$ and having these four estimated updates, we observe the new formulation for our primal-dual algorithm:

Algorithm 4.19 *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* $\bar{x}^0 = u^0, \bar{\mu}^0 = \mu^0$. *We choose* $\tau_u = \frac{1}{6}, \tau_\mu = \frac{1}{2+Z}, \sigma_p = \frac{1}{3+S}, \sigma_s = 1$. *Then, we let for each* $n \ge 0$

$$\begin{cases} p^{n+1} = \Pi_{K_p}(p^n + \sigma_p(A\bar{u}^n + \tilde{p})) \\ s^{n+1}_{k_1,k_2} = \Pi_{|\cdot|\le\nu}(s^n_{k_1,k_2} - \sigma_s\bar{\mu}^n_{k_1,k_2}) \\ u^{n+1} = \Pi_C(u^n - \tau_u A^* p^{n+1}) \\ \mu^{n+1}_{k_1,k_2} = \mu^n_{k_1,k_2} + \tau_\mu\left(\sum_{k_1\le k\le k_2} p^x_k - s^{n+1}_{k_1,k_2}\right) \\ \bar{u}^{n+1} = 2u^{n+1} - u^n \\ \bar{\mu}^{n+1}_{k_1,k_2} = 2\mu^{n+1}_{k_1,k_2} - \mu^n_{k_1,k_2}. \end{cases} \tag{4.31}$$

We suggest to set Z in the computation of τ_μ to a value of 150. In our framework we find this as a viable value. It is left to show, how we can extract the approximation u from the three-dimensional cube.

4.6 Computing the 0.5-Isosurface

We approximated the image g in a cube and stored the result in the variable u. As u is a three-dimensional object, we need a framework to extract the two-dimensional approximation. Therefore, we make use of the 0.5-isosurface. The idea is to interpolate at each pixel position (i, j) the corresponding channel 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 4.20 (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 was 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 fullfils the 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 to compute the 0.5-isosurface:

Algorithm 4.21 (0.5-isosurface) *At a fixed pair (i, j), holding for all $i = 1, ..., N$ and $j = 1, ..., M$, iterate from $1, ..., S-1$ and evaluate $u(i, j, k)$ and $u(i, j, k+1)$, respectively. If we find a combination, such that*

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

we set

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

and stop the loop. If we did not find such a pair, we set

$$\hat{u}(i, j) = u(i, j, k + 1).$$

Here $\hat{u}$ denotes the two-dimensional approximation of our input image g.

Note, that this algorithm is also applicable in other frameworks, so it is not bounded to our specific problem.

4.7 Determination of Convergence

As a convergence criterion, we adapted the idea of the real-time minimizer framework. But, we need to take into account that for the variable u in the underlying algorithm, we have $u \in C$. This means, we need to modify the stopping criterion a bit. We evaluate once every ten iterations

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

with $\varepsilon = 5 \cdot 10^{-5}$. In this framework, 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 valued images, we need a fourth loop over all $l = 1, 2, 3$ to take all color channels into account.

Decoupling of the set K into K_p and K_{nl}, applying a Lagrange formalism and rewriting the original saddle-point problem yields to a new version to minimize the Mumford-Shah functional. As we will see in chapter 5, this approach is far more applicable and leads to a better run-time.

5 Applications to Imaging

We now present applications to imaging for the proposed models. We consider image cartooning by using the real-time minimizer for the Mumford-Shah model. The ROF and TVL1 model are taken into account, when we show denoising. A modified version of the ROF model is considered in the image inpainting case, which is the most impressive application of the proposed models. At the end of this chapter we compare the two approaches to minimize the convex relaxed Mumford-Shah functional, namely using Dykstra's projection algorithm and the Lagrange Multiplier method. All input images, which are used in this chapter, can be found in the appendix, together with their size.

5.1 Linearized Storage of Images and PSNR

Images in a discrete two-dimensional grid $\mathcal{G}$ of size $N \times M$ can be viewed as matrices. The first element, which can be accessed is the one at the top left corner. Then the values are stored row-wise from the left to the right side. The last element of this matrix is then at the bottom right. 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 of the pixel grid and j the j-th element in the corresponding column. It also admits, that we allocate a pointer to a pointer array. To access one element, we would need to look up two points in memory. This causes an overhead in each element access. To save computation time we store all data in one vector, where the access of one element is linear. We allocate a vector u of the size $N \cdot M$. We attach each line of the image matrix to this array. Then we access elements by

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

This method works well for grayscaled images, where we approximate an input image $g : \mathcal{G} \longrightarrow D = [0, 1]$ by a function $u : \mathcal{G} \longrightarrow D$. So we map from the discret pixel grid $\mathcal{G}$ (see also notation 3.4) into the range from 0 to 1. Instead of using $[0, 1]$, we could also set $D = \{0, ..., 255\}$, where we assume our image and approximation to have 8-bit depth (see remark 2.2). Additionally note, that the vector $\bar{u}$ is stored in the same fashion, in the proposed primal-dual algorithm.

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 $k = 1, 2, 3$

$$\mathcal{G}_k = \Big\{(i,j) : i = 1, ..., N \text{ and } j = 1, ..., M\Big\}. \tag{5.1}$$

Overall, we attach each row of each grid separately to the vector u consecutively. Then, we have that $u \in \mathbb{R}^{N \cdot M \cdot C}$, where C resembles the number of color channels, and we access one element 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) (or (i, j, k) in the case of color images) we have $p_{i,j(,k)} \in \mathbb{R}^2$. We have two possibilities how to handle the storage of these variables. The way, which we choose to use, is to allocate two vectors of the size $N \cdot M(\cdot C)$ and call them for instance 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

$$p[j + i \cdot M + k \cdot M \cdot N + l \cdot M \cdot N \cdot C] = value;$$

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

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

For us, it seemed to be necessary to use the first method, to obtain a readable and maintainable code. To this end, we additionally want to introduce two important formulas used in the following sections. They can also be found in [9], in german.

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*

$$MSE(u, v) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} (u_{i,j} - v_{i,j})^2. \tag{5.2}$$

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

$$PSNR(u, v) = 10 \log_{10} \left(\frac{255^2}{MSE(u, v)} \right) \text{ db}. \tag{5.3}$$

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 to the energy of the image with given 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 this we say they are almost similar. For more information on this, we refer to [9].

5.2 Image Approximation using the ROF Model

In this section we show approximations of input images. Besides we compare run-time and different parameters. We also provide the best estimates of parameters for each model.

5.2.1 Implementation Issues

Let us first discuss some implementation issues. In our framework we implemented the ROF model in C++ on a CPU. Since, the algorithm is quite fast, even on a CPU, parallelization was not necessary.

We set $K = \nabla$ with the proposed discretization stated in section 3.3. Further, we consider u, $\bar{u}$, etc. being accessed with $u_{i,j,k}$. For the approximation of color images we have that $k = 1, 2, 3$, in the case of grayscaled images $k = 1$. Further, we allocate $u, \bar{u}, g$, such that $u, \bar{u}, g \in [0, 255]^C$, where C is again the number of color channels.

Initialization

According to [3] the algorithm is independent of its initialization. The better the initialization at the beginning, the faster the convergence to a optimal solution. We ran numerous tests to find good initialization values, e.g. setting u^0 to zero or using the maximal value. It turned out, that the best choice is the input image g itself. For that, we suggest this as the best initialization criterion and make use of it in all our tests.

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

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

At the end we only 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 $norm = sqrt(dx * dx + dy * dy)$.

Algorithm 5.2 (Dual Ascent)

```
template<typename T>
void dual_asc(T* p_x, T* p_y, T* u_bar, T sigma, int M, int N, int C) {
  T dx, dy, norm;
  int index;
  for (int k = 0; k < C; 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 / fmax(1.f, norm);
        p_y[index] = dy / fmax(1.f, norm);
      }
    }
  }
}
```

where the template value T is mostly set as float.

These few lines of code are used to compute the update on $p_{i,j,k}^{n+1}$ for all $i = 1, ..., N$, $j = 1, ..., M$ and $k = 1(, 2, 3)$. To be able to use this function in another frameworks, like for minimizing the TVL1 energy, we only need to change the last two lines in the for-loops, resembling the proximity operator for F^*.

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

We follow a similar procedure as before: we allocate values dx, dy and 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 $K^* = \nabla^T = -\mathrm{div}$. 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, it is left to add the previous estimate $u_{i,j,k}^n$ by

$$sum = u_{i,j,k}^n + sum.$$

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

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

Algorithm 5.3 (Primal Descent)
```
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 C) {
  T dx, dy, sum;
  int index;
  for (int k = 0; k < C; 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) -
             (i>0 ? p_y[(j-1) + i * M + k * N * M] : 0.f);
        sum = tau * (dx + dy);
        sum += u[index];
        u[index] = (sum + tau * lambda * g[index]) / (1.f + tau * lambda);
      }
    }
  }
}
```

where the template value T is mostly set as float.

It is only left to compute the extrapolation step. This is a straightforward computation, since we just add vectors, by

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

We can implement this update within one for-loop, since we have that $\bar{u}^{n+1}, u^{n+1}, u^n \in \mathbb{R}^{N \cdot M \cdot C}$.

Algorithm 5.4 (Extrapolation)
```
template<typename T>
void extrapolation(T* u_bar, T* u, T* u_prev, T theta, int M, int N, int C) {
  for (int i = 0; i < N*M*C; i++) {
    u_bar[i] = u[i] + theta * (u[i] - u_prev[i]);
    u_prev[i] = u[i];
  }
}
```

where the template value T is mostly set as float.

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 we can implement a stopping criterion for the ROF model. Following our suggestion of section 3.4.3, we allocate a $eps = 1E-6$ and two variables to store the energy in the n-th interation step, nrj_n, and in the n+10-th iteration step, nrj_10. Using these variables and the previously defined functions, we observe our primal-dual algorithm in the following fashion:

Algorithm 5.5 (Primal-Dual Algorithm)
```
template<typename T>
void ROF(T* u, T* g, T lambda, T tau, int M, int N, int C, int iter) {
  T sigma = 1.f / (tau * 8.f);
```

```
  T theta = 2.f;
  T eps = 1E-6;
  T nrj_n = Stop(u, g, p_x, p_y, lambda);
  T nrj_10;
  T* u_bar = new T[M*N*C];
  T* u_prev = new T[M*N*C];
  T* p_x = new T[M*N*C];
  T* p_y = new T[M*N*C];

  for (int i = 1; i <= iter; i++) {
    void dual_asc(p_x, p_y, u_bar, sigma, M, N, C);
    void primal_desc(p_x, p_y, u, g, tau, lambda, M, N, C);
    void extrapolation(u_bar, u, u_prev, theta, M, N, C);
    if (k%10 == 0) {
      nrj_10 = Stop(u, g, p_x, p_y, lambda);
      if (abs(nrj_n - nrj_10) <= eps) {
        break;
      } else {
        nrj_n = nrj_10;
      }
    }
  }

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

where the template value T is mostly set as float.

As suggested, in the first remark in [3] we compute σ by

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

This means we only need to pre-set the parameter τ and λ, before we can run the algorithm. We also set $\theta = 1$ to derive the version suggested in the work of Pock and Chambolle ([3]).

Further, we need to define the function, which evaluates the current primal-dual ROF energy. The energy itself is given by

$$E(u^n) = \frac{\lambda}{2}||u^n - g||_2^2 + \langle \nabla u^n, p \rangle.$$

Computationally, we allocate a value $energy = 0$ and temporary variables dx and dy. The last two are used to compute the gradient and the multiplication in the inner product for one pair i, j, k. In each step of the three inner for-loops, we add to the *energy* variable

$$energy+ = (dx + dy) \text{ and } energy+ = (lambda/2 * pow(u[X] - g[X], 2)),$$

where X is computed by $X = j + i * M + k * N * M$. Summarized in a C++-function, we observe:

Algorithm 5.6 (Stopping Criterion)

```
template<typename T>
T Stop(T* u, T* g, T* p_x, T* p_y, T lambda, int M, int N, int C) {
    T energy = 0.f;
    T dx, dy;
    int X;
    for (int k = 0; k < C; k++) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                X = j + i * M + k * N * M;
                dx = i + 1 < N ? u[j + (i+1) * M + k * N * M] - u[X] : 0;
                dy = j + 1 < M ? u[j + 1 + i * M + k * N * M] - u[X] : 0;
                dx *= p_x[X];
                dy *= p_y[X];
                energy += (dx + dy);
                energy += (lambda/2 * pow(u[X] - g[X], 2));
            }
        }
    }
    return energy;
}
```

where the template value T is mostly set as float.

Now, that we know how the implementation of the ROF model can efficiently be realized, we want to compare some outcomes of the algorithm. We started two separate parameter estimations: first we looked for the perfect λ for a minimal energy, a fast convergence and a visible good approximation u. Afterwards, we estimated the τ, for which the algorithm converges quickly with respect to the optimal λ.

5.2.2 Estimation of λ

To estimate the possible best λ we start testing all values in the range from 0.001 to 0.01 by adding in each approximation a factor 0.001 to the parameter λ. In figure 5.1 one can see, that an increasing value for λ makes the approximation more vivid. But still, the highest value, namely $\lambda = 0.01$, yields an over-smoothed version.

It seems not to be appropriable to go one one-thousandth steps and we already learned, that a small parameter does not lead to the desired results. We turn our interest to the other extrem setting. We use the range from 0.1 to 1 in one-tenth steps. It not only leads to the perfect fit, it also produces good approximations u of our input image g. This can be seen in figure 5.2 and the estimated values in table 5.1.

We only consider approximations u, which have a PSNR less than 40, compared to the input image g. Then there are only the choices 0.1, 0.2 and 0.3. Because the PSNR for $\lambda = 0.3$ is at 39, we also dropped this value. As the energy in the case $\lambda = 0.1$ is significantly smaller as for $\lambda = 0.2$, we choose $\lambda = 0.1$ as our best estimate.

To the end of this subsection let us mention that we did not discretize the domain Ω in this framework. For this, we observe other values for λ as proposed in [3].

(a) $\lambda = 0.001$ (b) $\lambda = 0.002$ (c) $\lambda = 0.003$ (d) $\lambda = 0.004$ (e) $\lambda = 0.005$

(f) $\lambda = 0.006$ (g) $\lambda = 0.007$ (h) $\lambda = 0.008$ (i) $\lambda = 0.009$ (j) $\lambda = 0.01$

Figure 5.1: Approximation of the Lena image with the ROF model. The higher λ the closer the approximation to the original image.

(a) $\lambda = 0.1$ (b) $\lambda = 0.2$ (c) $\lambda = 0.3$ (d) $\lambda = 0.4$ (e) $\lambda = 0.5$

(f) $\lambda = 0.6$ (g) $\lambda = 0.7$ (h) $\lambda = 0.8$ (i) $\lambda = 0.9$ (j) $\lambda = 1$

Figure 5.2: Approximation of the Lena image with the ROF Model. Again, the higher λ the closer the approximation to the original image.

5.2.3 Estimation of τ

We find, that for $\lambda = 0.1$ the best choice for the time-step is $\tau = 0.63$. This value rises 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 Van Gogh (RGB). As table 5.2 reveals, the suggested value for τ is always under the first ten estimates, where the table was sorted by the number of iterations and the run-time of the algorithm. We also learn from this table, that the time-step τ couples with the input image. This means, that $\tau = 0.63$ is a good guess and likely to

λ	MSE	PSNR	Energy
0.1	23	34	1,384,210
0.2	12	38	1,595,940
0.3	8	39	1,717,590
0.4	6	41	1,802,040
0.5	4	42	1,865,850
0.6	3	43	1,916,280
0.7	3	44	1,957,350
0.8	2	44	1,991,410
0.9	2	45	2,020,130
1.0	2	46	2,044,670

Table 5.1: The estimates for all λ in $\{0.1, 0.2, ..., 1.0\}$.

yield a fast convergence, but does not hold for each and every image. This coupling is not only between the input image g and τ, we further find it between λ and τ. One example, how this coupling influences the algorithm is, If we let $\lambda = 0.01$ and using $\tau = 0.63$ we observe 1110 iterations in the primal-dual algorithm. If we compare this with 240 iterations for $\lambda = 0.1$ (table 5.2), we see that this coupling has a big influence on finding valid estimates for λ and τ beforehand.

Lena Image				Hepburn Image			Van Gogh Image		
No.	τ	Iter	Run-Time	τ	Iter	Run-Time	τ	Iter	Run-Time
1	0.63	100	0.84	0.65	130	2.75	0.63	90	0.87
2	0.74	120	1.03	0.9	150	3.13	0.68	110	1.07
3	0.86	120	1.09	0.85	160	3.35	0.76	120	1.14
4	0.94	140	1.11	0.78	200	4.23	0.86	140	1.36
$\vdots$									
10	0.63	240	2.16	0.63	260	5.42	0.78	190	1.72

Table 5.2: Setting $\tau = 0.63$ provides a good guess for fast convergence.

Knowing nothing about the best time-step τ by initialization of the algorithm, makes the proposed approach a bit incosistent. The coupling makes it not easy to determine the best parameter for τ. This problem is still ongoing research. Nonetheless, having a stable framework like this, to minimize the energy for the ROF model, is a great approach and yields good approximations u of the input images g, as figure 5.3 shows for the Audrey Hepburn image.

Overall, we suggest for the primal-dual algorithm the setting $\theta = 1$, $\lambda = 0.1$ and $\tau = 0.63$, for which we derive the approximation u of g in a reasonable run-time and visibly good results.

(a) The original image of Audrey Hepburn. (b) Approximation using the proposed parameter. (c) Approximation using $\lambda = 0.01$.

Figure 5.3: Approximation of the Audrey Hepburn image with the ROF model. In (b) we set $\lambda = 0.1, \tau = 0.63$, where (c) resembles $\lambda = 0.01, \tau = 0.63$.

5.3 Image Approximation using the TVL1 Model

We now turn our focus on the TVL1 model. As this model has a fast CPU version, we also do not provide a parallelized version for a GPU. Additionally, 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 used. The only difference to the ROF model, is the computation of $(\mathrm{Id} + \tau\,\partial\, G)^{-1}(\tilde{u})$ and we need to change the function for our stopping criterion. In equation 3.16 we showed, that this proximity operator is of the form

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

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

$$fac = tau * lambda \;\text{ and }\; est = u^n_{i,j,k} - g_{i,j,k}.$$

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:

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

Now, the only thing we need to change in in algorithm 5.3 is the last line of code in the inner for-loop. In this line we compute the proximity operator, so we fit the code to the TVL1 model and obtain:

Algorithm 5.7 (Primal Descent)

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

where the template value T is mostly set as float.

Because, we changed the L^2 norm in the data fidelity term of the ROF functional to the L^1 norm, we need to take this into account in the stopping criterion. We only need to substitute the line of code, where the data fidelity term enters the computation of the energy itself. According to the C++-function in 5.6, we only change the last line of code in the inner for-loops, from

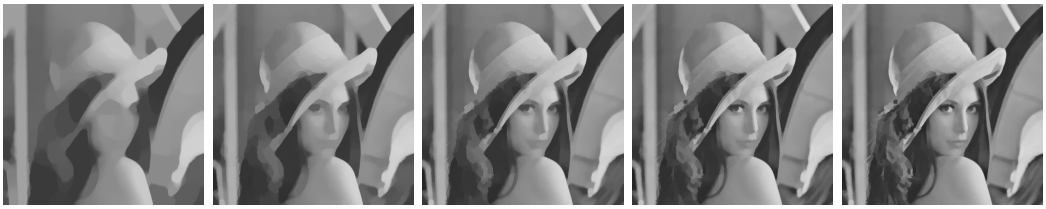
$$energy+ = (lambda/2*pow(u[X]-g[X],2)) \text{ to } energy+ = (lambda*(abs(u[X]-g[X]))).$$

These substitutions are all it takes and we are completely set up with the TVL1 model. It is now left to provide good estimates for the input values λ and τ. We set again $\theta = 1$ and allocate $u, \bar{u}, g \in [0, 255]^C$, where C is the number of color channels.

5.3.1 Estimation of λ

For the TVL1 model, estimating a good λ was an easy task compared to the ROF model. The reason for this is, that in [3] they proposed this model without the scaling factor h^2, like we do in this work. They already suggested to set $\lambda = 0.7$. We adapted this idea and tested all values from 0.1 to 1 by increasing λ for each approximation by 0.1. This evolution process is shown in figure 5.4.

As one can see, the value $\lambda = 0.7$ yields reasonable results with less noise, no over-smoothing and sharp edges. The values 0.6 and 0.8 would also fit, but we decided to follow Pock and Chambolle ([3]) with their suggestion.



(a) $\lambda = 0.1$ (b) $\lambda = 0.2$ (c) $\lambda = 0.3$ (d) $\lambda = 0.4$ (e) $\lambda = 0.5$

(f) $\lambda = 0.6$ (g) $\lambda = 0.7$ (h) $\lambda = 0.8$ (i) $\lambda = 0.9$ (j) $\lambda = 1$

Figure 5.4: Approximation of the Lena image with the TVL1 Model.

5.3.2 Estimation of τ

We apply the same procedure to estimate the best time-step parameter τ as in subsection 5.2.3. We use again the images Lena, Hepburn and Van Gogh for the evaluation. There is only one parameter, which appeares under the sixth fastest convergence rates: $\tau = 0.97$, c.f. table 5.3. For this we suggest to set $\tau = 0.97$. Unfortunatelly, we have again a coupling of the image, λ and τ. This means, that using the estimate $\tau = 0.97$ does not guarantee fast convergence with every image.

	Lena Image		Hepburn Image		Van Gogh Image	
τ	Iter	Run-Time	Iter	Run-Time	Iter	Run-Time
0.97	370	3.44	580	12.66	1500	15.47

Table 5.3: The results for the best estimate $\tau = 0.97$.

In figure 5.5 we show the result for our suggested framework by setting $\theta = 1, \lambda = 0.7$ and $\tau = 0.97$ using the Landscape image. The underlying image is color valued and it only took 530 iterations until convergence. In run-time it took about 4.4 seconds and is for that quite fast. Even though, the TVL1 model yields good approximations, we clearly see, that it is not able to preserve edges well enough. Some contours vanish and we see some over-smoothing in the image itself. If we set $\lambda = 1.2$, we observe much better results, since the higher parameter for λ assures, that the approximation is closer to the input image g. With this λ and again $\tau = 0.97$, we observe 850 iteration steps in the primal-dual algorithm. Again, the coupling of λ, τ and the image g does not ensure a fast convergence.

The TVL1 model yields good approximations u and as we will see is able to remove Gaussian and salt and pepper noise from images. But, we are also interested in preserving edges in the approximations. For that, the convex relaxed Mumford-Shah model, as seen

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

Figure 5.5: Approximation of the Landscape image with the TVL1 model.

in section 5.8, is therefore the right choice.

5.4 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 again adapt most of the code from the ROF model. Even though, it runs in real-time, we were more interested in the framework itself. For that, we only provide a CPU implementation, which also has a good performance.

The function G is almost the same as in the ROF model, except the scaling parameter $\frac{\lambda}{2}$. But for that, we only need to exchange one line of code. The line

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

becomes

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

Everything else in algorithm 5.3 remains the same. In algorithm 5.2 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

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

For the implementation we make use of the run-time saving method to pre-compute some values and use them afterwards in each calculation. We set $fac = (2 * lambda)/(sigma + 2 * lambda)$, $p_{abs} = |p^n_{i,j,k}| = sqrt(dx * dx + dy * dy)$ and compute the bound by $B = sqrt((nu/lambda) * sigma * (sigma + 2 * lambda))$. Then we rewrite the proximity operator to

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

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

Algorithm 5.8 (Dual Ascent)

```
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 C) {
  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 < C; 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 <= B ? fac * dx : 0;
        p_y[index] = p_abs <= B ? fac * dy : 0;
      }
    }
  }
}
```

where the template value T is mostly set as float.

The function to determine convergence, needs to be rewritten. Using the proposed method in equation 3.28 we compute after each iteration step of the primal-dual algorithm

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

with $\varepsilon = 5 \cdot 10^{-5}$ and with the definition of the underlying norm

$$||u^{n+1} - u^n|| = \frac{1}{N \cdot M} \sum_{k=1}^{C} \sum_{i=1}^{N} \sum_{j=1}^{M} |u_{i,j,k}^{n+1} - u_{i,j,k}^{n}|.$$

As in the extrapolation function, we can compute the norm within one for loop for letting $u^n, u^{n+1} \in \mathbb{R}^{N \cdot M \cdot C}$. Overall, we summarize this in the following function:

Algorithm 5.9 (Stopping Criterion)

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

where the template value T is mostly set as float.

As we used the second algorithm of section 3.2, namely algorithm 3.3, we additionally propose this version in C++ code:

Algorithm 5.10 (Primal-Dual Algorithm)

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

  for (int k = 1; k <= iter; k++) {
    void dual_asc(p_x, p_y, u_bar, sigma, lambda, nu, M, N, C);
    void primal_desc(p_x, p_y, u, g, tau, M, N, C);
    theta = (T)1 / sqrt(1 + 4 * tau); tau *= theta; sigma /= theta;
    void extrapolation(u_bar, u, u_prev, theta, M, N, C);
    if (k%10 == 0) {
      stop = h * ;
      if (Stop(u, u_n, M, N, C) <= 5 * 1E-5) {
        break;
      }
    }
  }

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

where the template value T is mostly set as float.

The difference in this framework is, that the primal-dual function does not take the parameter τ as an argument, since 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. It is left to present the estimation for the parameters λ and ν. For this particular framework we followed Strekalovskiy and Cremers in [22] and set $u, \bar{u}, g \in [0,1]^C$, where C is again the number of color channels.

5.4.1 Best λ and ν estimation

For this model we have two parameters, which can be combined in a lot of ways. Finding parameters λ and ν take several estimation steps. We will not provide the whole procedure of this process, since it could fill a huge amount of pages. But, we discuss our estimation in a short term and the corresponding results.

Initially we set $\nu = 0.1$ and seek for λ using the Audrey Hepburn image. For the piecewise-constant model, we try several possible λ in between 100 and 1000. It turns out, that there are no big changes if $\lambda \geq 500$ and so we use the setting $\lambda = 500$. In the piecewise-smooth model, we start with the guess $\lambda = 0.1$, which is too small. By testing each λ by an increased value of 0.1, we find, that at about 1.8 to 2.2, there is a good

chance for a good fit. Of course, testing then the value ν, has also an influence on λ again. Nonetheless, we fix $\lambda = 2$ and test for ν. For the piecewise-smooth case, we start with 0.1 to 1 in 0.1-steps. We focus on the PSNR value. We assume that it should be over 30, but not too close to 40. A good fit for this setting is $\nu = 0.2$. In figure 5.6, we see, that this setting indeed yields a good smooth approximations u. Learning that ν lies in the range of 0.1 to 1, we adapt the value for ν for the piecewise-constant model. Unfortunately, it turns out, that this setting needs a higher value for ν, to be able to cope with the increased value for λ in the minimum function of R_{MS}. Testing again each ν in the range of 0.1 to 1 by increasing it by 0.1 in each estimate, we find that $\nu = 0.7$ is a good choice. See again figure 5.6 for this setting.

λ	ν	Iterations	Run-Time	PSNR
2	0.2	80	1.37	32
500	0.7	430	7.2	23

Table 5.4: Best estimates in the piecewise-constant and piecewise-smooth case.

The corresponding images to the proposed values for λ and ν can be seen in figure 5.6. The estimated values for the PSNR, number of iterations, run-time and the parameters suggested, can be found in table 5.4.

(a) Original image. (b) $\alpha = 2, \lambda = 0.2$ (c) $\alpha = 500, \lambda = 0.7$

Figure 5.6: Approximation of the Audrey Hepburn image with the piecewise-smooth and piecewise-constant Mumford-Shah model.

Further, 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 5.10 we pre-computed it with $\tau = \frac{1}{2d}$, where $d = 2$, since the dimension of Ω is two.

Now, that we are set up with the pure image approximation and the corresponding implementation issues, let us turn our focus on applications to imaging. We start with one application of the real-time minimizer of the Mumford-Shah functional, namely image cartooning.

5.5 Image Cartooning

In this section we present a technique to turn an input image g into a cartooned image u. For this we will make use of the real-time minimizer for the Mumford-Shah functional, presented in section 3.6. We further introduce a concept to highlight the edges in the cartooned images, first proposed in [22]. For this we make explicit use of the set K_{MS}, as modeled in equation 3.21. We then present cartooned images - piecewise-constant approximations - with edge highlighting.

5.5.1 Edge Highlighting

The idea, proposed by Strekalovskiy and Cremers in [22], is to use the edge set K_{MS} in order to highlight edges in the approximation u.

For that, assume that $(i,j) \in K_{MS}$ then we have $|(\nabla u)_{i,j}| \geq \sqrt{\frac{\nu}{\lambda}}$ by definition. This notation is equivalent to

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

Further, we compute $|(\nabla u)_{i,j}| \leq \sqrt{4} = 2$. This equation holds with the definition for ∇, c.f. definition 3.4, since

$$\begin{aligned} |(\nabla u)_{i,j}| &= \sqrt{(u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2} \\ &= \sqrt{u_{i+1,j}^2 - 2u_{i+1,j}u_{i,j} + u_{i,j}^2 + u_{i,j+1}^2 - 2u_{i,j+1}u_{i,j} + u_{i,j}^2} \\ &= \sqrt{2u_{i,j}^2 + u_{i+1,j}^2 + u_{i,j+1}^2 - 2u_{i,j}\left(u_{i+1,j} - u_{i,j+1}\right)} \\ &\overbrace{\leq}^{u \in [0,1]} \sqrt{2+1+1} = \sqrt{4} = 2. \end{aligned} \tag{5.4}$$

And for that

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

Applying the logarithm function to each of the terms we get

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

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

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

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

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

then, since ν and λ are constant values, this 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 then to multiply each pixel value u by

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

if $(i,j) \in K_{MS}$. On the other hand, if $(i,j) \notin K_{MS}$ the value u will not be changed. Then points in the image domain Ω with strong edges are painted darker.

We implemented this technique of edge highlighting in our framework and tested it against the pure piecewise-constant approximation. Using the proposed values $\lambda = 500$ and $\nu = 0.7$, as suggested in section 5.4.1, we visiually compared an approximation with and without edge highlighting.

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

Figure 5.7: Cartooning of the Hepburn image with $\lambda = 500$ and $\nu = 0.7$.

Figure 5.7 shows, that this technique clearly yields darker edges. Since, this highlighting takes place after applying the primal-dual algorithm, it does not slow down the code significantly. Summarizing this method in a C++-function, we have:

Algorithm 5.11 (Edge Highlighting)

```
template<typename T>
void edging(T* u, T lambda, T nu, int M, int N, int C) {
  for (int k = 0; k < C; 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(2/factor);
            u[X] = (1 - c * log(norm / factor)) * u[X];
          }
        }
      }
    }
}
```

where the template value T is mostly set as float.

In this section we showed, that the Mumford-Shah realtime minimizer is good in cartooning images and has the advantage of the discrete modeled set K_{MS}, which can be used for edge highlighting. But, 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.

5.6 Image Denoising

As discussed, an image g consists of data and noise. Our goal is to remove the 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 the so called salt and pepper noise. Where Gaussian noise is relatively easy to remove, salt and pepper noise is more robust, since it resembles extrem values 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.

(a) Original image. (b) 25% noise. (c) $\lambda = 0.007$ (d) $\lambda = 0.008$

Figure 5.8: Salt and pepper denoising of the Lena image with the ROF Model.

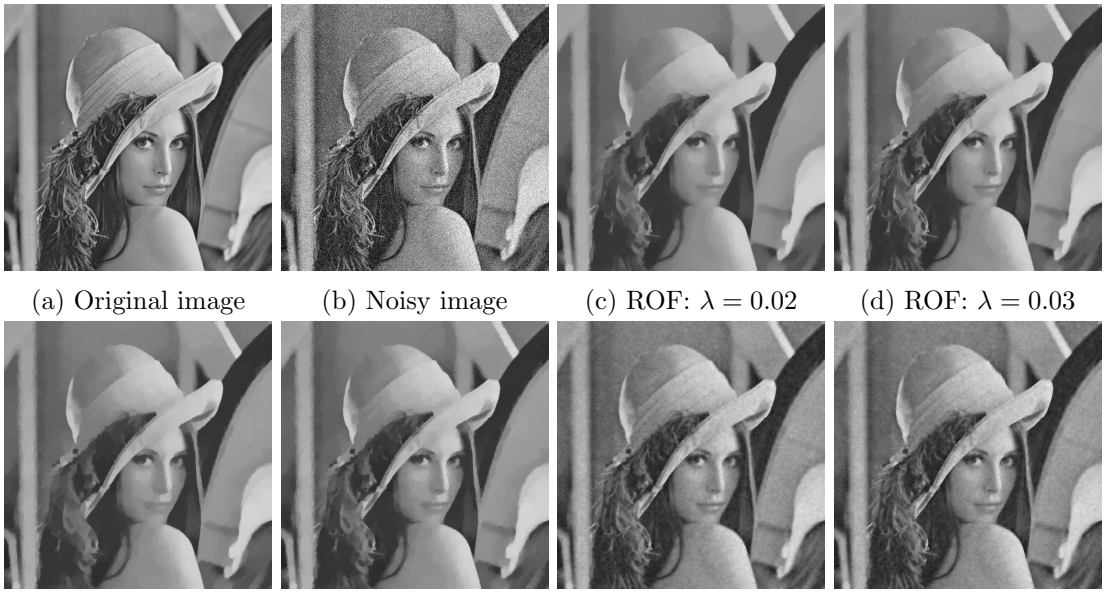
By evaluating a good λ for both the ROF and TVL1 model, we tried our estimates of sections 5.2 and 5.3. It turned out, that these λ also fit perfectly to remove noise. The only thing we further tried to evaluate: finding a parameter λ for the ROF model, so that it can remove salt and pepper noise. In fact, we found two values by testing all possible λ in the range of 0.001 to 1 by increasing λ in each test by 0.001. Even though, the results u are not as good as in the TVL1 model (see figure 5.9), the ROF model (figure 5.8) is able to remove this kind of noise in some sense, but as mentioned, not

accurately. We also tested the real-time minimizer for the Mumford-Shah model, but could not find any pair of parameter, able to handle salt and pepper noise.

(a) Original image. (b) 25% noise. (c) $\lambda = 0.7$

Figure 5.9: Salt and pepper denoising of the Lena image with the TVL1 Model.

Comparing the denoising of images afflicted with Gaussian noise, we also compare the ROF model against the TVL1 model and the Mumford-Shah real-time model, figure 5.10.



(a) Original image (b) Noisy image (c) ROF: $\lambda = 0.02$ (d) ROF: $\lambda = 0.03$

(e) TVL1: $\lambda = 0.6$ (f) TVL1: $\lambda = 0.7$ (g) MS: $\lambda = 2, \nu = 0.7$ (h) MS: $\lambda = 2, \nu = 0.5$

Figure 5.10: Removing Gaussian noise from the Lena image with the ROF, TVL1 and Mumford-Shah model.

We see, that the Mumford-Shah model yields not the best approximations. From the evaluation of λ and ν, we already know, that we need $\lambda = 2$ for the piecewise-smooth approximation. By finding a suitable value ν it turns out, that the choice $\nu = 0.5$, as used

in the piecewise-constant case, fits well. But also $\nu = 0.7$, as suggested for piecewise-smooth approximations, yields acceptable results. Even, if the noise is removed, the approximation is not comparable to those of the ROF and TVL1 model. These yield denoised approximations, which still preserve edges and areas accurately. In the last section, where we take a look at the convex relaxed Mumford-Shah model, we will see that this formulation is a good choice for removing Gaussian noise. The proposed real-time approach is not well suited for this class of problems.

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.

5.7 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 $\mathcal{I} \subseteq \mathcal{D}$, which 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

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

Remark 5.12 *According to Pock and Chambolle in [3], we have two choices for setting 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 these regions.*

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

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

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 remains the same and we get:

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

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 again 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 2.19 we have

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

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

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

To extend it for colored images, we apply this procedure channel-wise. Computationally, we do not need to change the code that much. We only need to add an additional if-statement in the dual ascent function. But, of course we need to know where the discrete locations of $\mathcal{I}$ are. A common method would be to use clustering methods to determine the locations (i, j), where data is lost. We mainly followed [3], where they already know the indices of the set $\mathcal{I}$ beforehand.

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

Figure 5.11: Inpainting of the Lena image with a data loss of seventy percent and simultanously denoising.

We also did not need to estimate a good fit for λ, since the fit we found in section 5.2, namely $\lambda = 0.1$, worked perfectly. This is also illustrated in figure 5.11. We additionally tested the case in which $\lambda = \infty$, see figure 5.12. Indeed, the inpainting process works fine, but as stated above, there was no smoothing in regions without data loss.

(a) Original image. (b) Input image g. (c) $\lambda = \infty$

Figure 5.12: Inpainting of Lena image with a data loss of seventy percent and without denoising, namely $\lambda = \infty$.

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 found: if τ is too small, it turns out, that the inpainting process was not successful, see figure 5.13. 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.

$\lambda \in (0, \infty)$			$\lambda = \infty$		
τ	Iterations	Run-Time	τ	Iterations	Run-Time
0.97	2114	22.60	0,89	1743	17.77
0.99	2245	22.25	0,88	1761	17.97
0.98	2254	22.58	0,86	1797	18.34
0.95	2275	22.77	0,98	2079	21.53
0.94	2350	23.70	0,79	2116	19.16

Table 5.5: This table shows several estimates for reasonable values τ using the ROF model in image inpainting.

Further, table 5.5 reveals, that the inpainting process takes more iterations, and for that 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.

Run-time becomes a bigger issue, when talking about the convex relaxed Mumford-Shah functional. It is also able to remove Gaussian noise from images. Extending the model, one could also do image inpainting and segmentation with it. We will see, that

(a) Input image g. (b) $\lambda = 0.1, \tau = 0.01$ (c) $\lambda = \infty, \tau = 0.01$

Figure 5.13: Unsuccessful image inpainting of Lena image with $\tau = 0.01$.

it yields much exacter approximations and preserves edges far better than the ROF or TVL1 model, but is not comparable in run-time, although it is parallelized on a GPU.

5.8 The Convex Relaxed Mumford-Shah Model

We proof 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 [18]. We compare the two presented frameworks, namely Dykstra's projection algorithm and the Lagrange multiplier approach to solve this minimization problem. Additionally, we show, that this model is also accurate in removing Gaussian noise. In chapter 4 we presented the Mumford-Shah energy

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

which we then rewrote into a saddle-point formulation of the form

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

Overall, we aimed to solve the discrete saddle-point problem

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

with $A = \nabla$. As mentioned, the suggested method in [18] 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. In section 4.5 we proposed a decoupling of the non-local constraint set K_{nl}, used Lagrange Multipliers and proved that we can rewrite the saddle-point formulation into

$$\min_{u \in C} \max_{p \in K} \langle \nabla u, p \rangle \Longleftrightarrow \min_{\substack{u \in C \\ \mu_{k_1,k_2}}} \max_{\substack{p \in K_p \\ |s_{k_1,k_2}| \leq \nu}} \langle \nabla u, p \rangle + \sum_{k_1=1}^{S} \sum_{k_2=1}^{S} \langle \mu_{k_1,k_2}, \sum_{k_1 \leq k \leq k_2} p_k^x - s_{k_1,k_2} \rangle.$$

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 [18] they proposed their results using a 4GB GPU and $S = 32$ levels. As we only had a 3GB GPU available, we halved this number of levels to $S = 16$ for both algorithms to observe comparable results. Fortunately, shrinking the number of added labels yields to good estimates u, too. We used our program for image approximation and denoising. Extending it, the Mumford-Shah functional could also be used for inpainting and segmentation.

5.8.1 Implementation Issues

The implementation to minimize the convex relaxed Mumford-Shah functional with the suggested framework using Dykstra's projection algorithm is quite difficult. One needs to be extremely careful with each line of code. Besides, the lack of run-time (see for instance table 5.6) is not worth discussing the implementation issues. For an overview of the code, we refer to [6]. Instead, we provide the necessary computations needed for the Lagrange multiplier approach. This method leads to a acceptable run-time for some applications. Since, the implementation itself is on a GPU, the corresponding C++ code looks slightly different to those of the previous sections. Instead of for-loops, we let each process do one computation on the GPU. For that, the number of loops reduces, but the computations itself remain the same. The advantage of parallelization in this method is the tremendous speed up in run-time. For more information on the CUDA framework we refer to [17]. Let us start by explaining, which variable needs to be allocated in which size.

Since, we are acting in a three-dimensional grid of size $N \times M \times S$ and make again use of the linearized storage, we allocate $u, ubar, uprev \in \mathbb{R}^{N \cdot M \cdot S \cdot T}$. In this subsection T resembles the number of color channels of the image. So, we apply all our operations channel-wise to our images. Further, we have that at a voxel (i, j, k) we know, that $p_{i,j,k} \in \mathbb{R}^3$. For that, we follow the same procedure as in subsection 5.2.1, where we allocate three seperate vectors for p, namely $p1, p2, p3 \in \mathbb{R}^{N \cdot M \cdot S}$. As we extended our original saddle-point formulation with Lagrange multipliers, we also need to allocate the vectors for μ and s. Recalling, that we earlier computed $I = \frac{S(S-1)}{2} + S$, and have that for a fixed position (i, j) and a fixed combination (k_1, k_2) we set $\mu_{k_1,k_2}(i, j), s_{k_1,k_2}(i, j) \in \mathbb{R}^2$, then we allocate $mu1, mu2, mubar1, mubar2, muprev1, muprev2, s1, s2 \in \mathbb{R}^{N \cdot M \cdot I}$. At last, we allocate the input image $f \in \mathbb{R}^{N \cdot M}$. We initialize all vectors with zero, except u and $ubar$. Here, we allocate each level with the original image f.

Finally, we need to preset the time-steps, which we take as given in this framework. One could modify these steps, to obtain faster convergence, or even better results. Since, we do not know the impact of the parameters by hard, we suggest to use these as they work perfect in this setting. We compute

$$\tau_u = \frac{1}{6}, \;\; \tau_\mu = \frac{1}{2+Z}, \;\; \sigma_p = \frac{1}{3+S} \;\text{ and }\; \sigma_s = 1,$$

where $Z = 150$.

Let us start with the last two lines of algorithm 4.19. These are the interpolation steps for the primal variables. We can completely adapt the extrapolation function of 5.4. In this case, we need two loops: one for the variables of u and those of μ. The first loop, is of length $\cdot S \cdot T$, whereas the second loop has $\cdot I \cdot T$. And we parallalize the locations (i, j) for all $i = 1, ..., N$ and $j = 1, ..., M$. We observe the following CUDA kernel:

Algorithm 5.13 (Extrapolation)

```
__global__ void extrapolate(float* ubar, float* mubar1, floa
{
  int j = threadIdx.x + blockDim.x * blockIdx.x;
  int i = threadIdx.y + blockDim.y * blockIdx.y;

  if (j < M && i < N) {
    int K, J;
    for (int c = 0; c < T; c++) {
      for (int k = 0; k < I; k++) {
        K = j+i*M+k*N*M+c*N*M*S;
        J = j+i*M+k*N*M+c*N*M*I;
        if (k<S) {
          ubar[K] = 2.f*u[K]-un[K];
        }
        mubar1[J] = 2.f*mu1[J]-muprev1[J];
        mubar2[J] = 2.f*mu2[J]-muprev2[J];
      }
    }
  }
}
```

Note, that this time, we did not store the old estimate on u in $uprev$. This is done in the next function, namely computing the update on the variable μ_{k_1,k_2}. In this step, we do not need any projection. We only do straightforward computations. For a fixed pair (k_1, k_2) at a fixed (i, j), we evaluate

$$\sum_{k_1 \leq k \leq k_2} p_k^x.$$

This means for our framework, we allocate two temporary variables $t1, t2$ and sum up all values of $p1$ and $p2$ in the range of k_1 and k_2. Since, we update μ_{k_1,k_2} for all possible combinations (k_1, k_2) and for all pixel locations (i, j), $t1$ and $t2$ are computed at each pair $(i, j, (k_1, k_2))$. Additionally, we subract these two temporary values of each s_{k_1,k_2} in every iteration step, multiply the outcomes with τ_μ and add the old estimate of μ_{k_1,k_2} to it. This technical description is summarized in the following CUDA kernel:

Algorithm 5.14 (Update on μ)

```
__global__ void mu(float* mu1, float* mu2, float* muprev1, flo
{
  int j = threadIdx.x + blockDim.x * blockIdx.x;
  int i = threadIdx.y + blockDim.y * blockIdx.y;

  if (j < M && i < N) {
    float taumu = 1.f / (2.f + 150.f);
```

```
    float t1, t2, c1, c2;
    int J, K, L;
    for (int c = 0; c < T; c++) {
      K = 0;
      for (int k1 = 0; k1 < S; k1++) {
        for (int k2 = k1; k2 < S; k2++) {
          L = j+i*M+K*N*M+c*N*M*I;
          c1 = mu1[L]; c2 = mu2[L];
          t1 = 0.f; t2 = 0.f;
          for (int k = k1; k <= k2; k++) {
            J = j+i*M+k*N*M+c*N*M*S;
            t1+=p1[J];
            t2+=p2[J];
          }
          mu1[L] = c1+taumu*(s1[L]-t1);
          mu2[L] = c2+taumu*(s2[L]-t2);
          muprev1[L] = c1;
          muprev2[L] = c2;
          K++;
        }
      }
    }
  }
}
```

Here, we stored the old estimates of $mu1, mu2$ in the variables $muprev1$ and $muprev2$, respectively. The updates on the variable u is rather simple, since it is the same computation as in algorithm 5.3, in which we stored the temporary data in a variable called sum. The only thing we need to add is the derivative in the direction of S, following definition 4.7. And, of course the proximity operator, here euclidean projection, is substituted to take the clipping into account. This means, we have to change the code line

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

to

$$u[index] = fmin(1.f, fmax(0.f, sum))$$

where we compute the index values by $index = j + i \cdot M + k \cdot N \cdot M + c \cdot N \cdot M \cdot S$. Further note, that we set $u[index] = 1.f$ if $k = 0$ and $u[index] = 0.f$ if $k = S - 1$ to completely follow remark 4.11.

Algorithm 5.15 (Clipping)
```
__global__ void clipping(float* u, float* uprev, float* p1, float*
{
  int j = threadIdx.x + blockDim.x * blockIdx.x;
  int i = threadIdx.y + blockDim.y * blockIdx.y;
  int k = threadIdx.z + blockDim.z * blockIdx.z;

  if (j < M && i < N && k < S) {
    int J;
    float cur;
    float d1,d2,d3,D;
```

```
    for (int c = 0; c < T; c++) {
      J = j+i*M+k*N*M+c*N*M*S;
      cur = u[J];
      uprev[J] = cur;
      d1 = (i+1<N ? p1[J] : 0.f) - (i>0 ? p1[j+(i-1)*M+k*N*M+c*N*M*S] : 0.f);
      d2 = (j+1<M ? p2[J] : 0.f) - (j>0 ? p2[(j-1)+i*M+k*N*M+c*N*M*S] : 0.f);
      d3 = (k+1<S ? p3[J] : 0.f) - (k>0 ? p3[j+i*M+(k-1)*N*M+c*N*M*S] : 0.f);
      D = cur+tauu*(d1+d2+d3);
      if (k==0) {
        u[J]=1.f;
      } else if (k==S-1) {
        u[J]=0.f;
      } else {
        u[J]=fmin(1.f, fmax(0.f, D));
      }
    }
  }
}
```

The updates on the variables s_{k_1,k_2} are computed quite easy. At a fixed position (i, j) we can sum up over all possible combinations (k_1, k_2) by simply running a for-loop from 0 to $I - 1$. So, we allocate temporary variables $m1$ and $m2$ and compute for an index $K = j + i * M + k * N * M + c * N * M * I$

$$m1 = s1[K] - sigmas * mubar1[K] \text{ and } m2 = s2[K] - sigmas * mubar2[K].$$

With these, we can determine the norm by $norm = sqrt(m1 * m1 + m2 * m2)$ and project $m1$ and $m2$, respectively, onto the ν-ball and for that update $s1$ and $s2$, by

$$s1[K] = (norm <= nu)?m1 : nu*m1/norm \text{ and } s2[K] = (norm <= nu)?m2 : nu*m2/norm.$$

Then, we obtain the following CUDA kernel:

Algorithm 5.16 (Euclidean Projection)
```
__global__ void euclidean_projection(float* s1, floa
{
  int j = threadIdx.x + blockDim.x * blockIdx.x;
  int i = threadIdx.y + blockDim.y * blockIdx.y;

  if (j < M && i < N) {
    float m1, m2;
    float norm;
    int K;
    for (int c = 0; c < T; c++) {
      for (int k = 0; k < I; k++) {
        K = j+i*M+k*N*M+c*N*M*I;
        m1=s1[K]-sigmas*mubar1[K];
        m2=s2[K]-sigmas*mubar2[K];
        norm=sqrtf(m1*m1+m2*m2);
        s1[K]=(norm<=nu) ? m1 : nu*m1/norm;
        s2[K]=(norm<=nu) ? m2 : nu*m2/norm;
      }
```

```
    }
  }
}
```

The last update function, we need to define is more involved than the others, since we need to evaluate $\tilde{p} \in \mathbb{R}^3$ (c.f. algorithm 4.19), compute the discrete gradient and do the projection onto the parabola. Let us start by the calculation of $\tilde{p}$. For this we allocate two temporary variables $m1sum$ and $m2sum$. We use the estimates of $mu1$ and $mu2$ and sum up all values from 0 to $S-1$ for an index K, where $K \geq k_1$ and $K \leq k_2$. So, at a certain index $j + i * M + K * N * M + c * N * M * I$, we compute

$$mu1sum+ = mu1[j+i*M+K*N*M+c*N*M*I] \text{ and } mu2sum+ = mu2[j+i*M+K*N*M+c*N*M*I].$$

Then, $m1sum$ and $m2sum$ resembles the vector $\tilde{p}$. Note, that the third component of this vector is always zero and for that we do not an additional variable.

In addition, we need to evaluate the gradient of $ubar$. This can be done in the same manner as in algorithm 5.2. But, we further need to take the gradient in the direction of S into account. Then, we can add each element of $\tilde{p}$ to the gradient of $ubar$ and multiply this observed vector with σ_p. Afterwards, we add to this vector to the previous estimate of p. For three temporary variables $u1, u2$ and $u3$, which already hold the values of $\nabla ubar$, we have

$$u1 = p1[J]+sigmap*(u1+mu1sum) \text{ and } u2 = p2[J]+sigmap*(u2+mu2sum) \text{ and } u3 = p3[J]+sigmap*u3$$

Note, that the index J can be computed by $j + i * M + k * N * M + c * N * M * S$.

It is left to provide the function to project onto the parabola. First, we store at each pixel (i, j) the image value of f in a temporary variable img. We compute the bound in the local constraint by a temporary variable B, with

$$B = 0.25f * (u1 * u1 + u2 * u2) - lambda * pow((k+1)/S - img, S).$$

Here, we have $k+1$, because indices in C++ start by zero and our pixel grid we have $k = 1, ..., S$. Then, if $u3 \geq B$ we set

$$p1[J] = u1 \text{ and } p2[J] = u2 \text{ and } p3[J] = u3.$$

But, if $u3 < B$ we compute the projection of $u3$ onto the parabola. We compute this projection in a single device function. Following the straightforward calculation of algorithm 4.13, we observe the CUDA device function:

Algorithm 5.17 (Projection onto Parabola)

```
__device__ void on_parabola(float* p1,float* p2
{
  float y = u3 + lambda * pow((k+1)/S - img, 2);
  float norm = sqrtf(u1*u1+u2*u2);
  float v = 0.f;
  float a = 2.f * 0.25f * norm;
  float b = 2.f / 3.f * (1.f - 2.f * 0.25f * y);
```

```
  float d = b < 0 ? (a - pow(sqrt(-b), 3)) * (a + pow(sqrt(-b), 3)) : a*a + b*b*b;
  float c = pow((a + sqrt(d)), 1.f/3.f);
  if (d >= 0) {
    v = c == 0 ? 0.f : c - b / c;
  } else {
    v = 2.f * sqrt(-b) * cos((1.f / 3.f) * acos(a / (pow(sqrt(-b), 3))));
  }
  p1[J] = norm == 0 ? 0.f : (v / (2.0 * 0.25f)) * u1 / norm;
  p2[J] = norm == 0 ? 0.f : (v / (2.0 * 0.25f)) * u2 / norm;
  p3[J] = 0.25f * (p1[J]*p1[J] + p2[J]*p2[J]) - lambda * pow((k+1)/S - img, S);
}
```

This means, if $u3 < B$ we call this function by

$$on_parabola(p1, p2, p3, u1, u2, u3, img, lambda, k, J, S),$$

where again img is the current value of f at a position (i, j). Overall, we observe the update on the variable p by the following function:

Algorithm 5.18 (Parabola Function)
```
__global__ void parabola(float* p1, float* p2, float* p3
{
  int j = threadIdx.x + blockDim.x * blockIdx.x;
  int i = threadIdx.y + blockDim.y * blockIdx.y;
  int k = threadIdx.z + blockDim.z * blockIdx.z;

  if (j < M && i < N && k < S) {
    for (int c = 0; c < T; c++) {
      int L = j+i*M+c*N*M;
      int J = j+i*M+k*N*M+c*N*M*S;

      float B;

      float img = f[L];
      float mu1sum = 0.f;
      float mu2sum = 0.f;

      float val = ubar[J];
      float u1 = (i+1<N) ? (ubar[j+(i+1)*M+k*N*M+c*N*M*S]-val) : 0.f;
      float u2 = (j+1<M) ? (ubar[(j+1)+i*M+k*N*M+c*N*M*S]-val) : 0.f;
      float u3 = (k+1<S) ? (ubar[j+i*M+(k+1)*N*M+c*N*M*S]-val) : 0.f;

      int K = 0;
      for (int k1 = 0; k1 < S; k1++) {
        for (int k2 = k1; k2 < S; k2++) {
          if (k <= k2 && k >= k1) {
            mu1sum+=mu1[j+i*M+K*N*M+c*N*M*I];
            mu2sum+=mu2[j+i*M+K*N*M+c*N*M*I];
          }
          K++;
        }
      }
```

```
      u1 = p1[J]+sigmap*(u1+mu1sum);
      u2 = p2[J]+sigmap*(u2+mu2sum);
      u3 = p3[J]+sigmap*u3;

      B = 0.25f * (u1*u1 + u2*u2) - lambda * pow((k+1)/S - img, S);
      if (u3 < B) {
        on_parabola(p1,p2,p3,u1,u2,u3,img,lambda,k+1,J,S);
      } else {
        p1[J] = u1;
        p2[J] = u2;
        p3[J] = u3;
      }
    }
  }
}
```

The last function, which can be parallelized, is the computation of the 0.5-isosurface. Following the procedure proposed in algorithm 4.21, we observe:

Algorithm 5.19 (0.5-Isosurface)
```
__global__ void isosurface(float* f, float* u, int M, int N, i
{
    int j = threadIdx.x + blockDim.x * blockIdx.x;
    int i = threadIdx.y + blockDim.y * blockIdx.y;

    if (j < M && i < N) {
        float uk0, uk1, val;
        for (int c = 0; c < T; c++) {
            for (int k = 0; k < S-1; k++) {
                uk0 = u[j+i*M+k*N*M+c*N*M*S];
                uk1 = u[j+i*M+(k+1)*N*M+c*N*M*S];
                if (uk0 > 0.5 && uk1 <= 0.5) {
                    val = ((k+1) + (0.5 - uk0) / (uk1 - uk0)) / S;
                    break;
                } else {
                    val = uk1;
                }
            }
            f[j+i*M+c*N*M] = val;
        }
    }
}
```

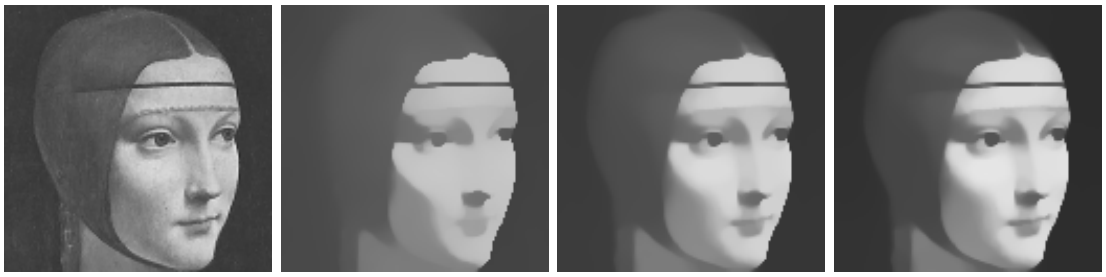
As a last step, we provide the loop, which resembles the primal-dual algorithm:

Algorithm 5.20 (Primal-Dual Loop)
```
for (iter = 0; iter < repeats; iter++) {
  parabola <<<grid, block>>> (d_p1,d_p2,d_p3,d_mu1,d_mu2,d_ubar,d_f,sigmap,lambda,M,N,S,I,
  l2projection <<<grid_iso, block_iso>>> (d_s1,d_s2,d_mubar1,d_mubar2,sigmas,nu,M,N,I,T);
  clipping <<<grid, block>>> (d_u,d_uprev,d_p1,d_p2,d_p3,tauu,M,N,S,T);
  mu <<<grid_iso, block_iso>>> (d_mu1,d_mu2,d_muprev1,d_muprev2,d_s1,d_s2,d_p1,d_p2,M,N,S,
  extrapolate <<<grid_iso, block_iso>>> (d_ubar,d_mubar1,d_mubar2,d_uprev,d_muprev1,d_mup
}
isosurface <<<grid_iso, block_iso>>> (d_f,d_u,M,N,S,T);
```

The prefactor $d\_$ at each variable tags the variables as device functions in CUDA. To have a difference of host and device functions, we add a d or h as prefactor. For the implementation of the stopping criterion, we refer to our computer program $iPAUR$ online at [6]. It is implemented as for the real-time minimizer framework, but with the additional label space taken into account. We now, want to evaluate the parameters λ and ν, for which this algorithm approximates input images f optimal.

5.8.2 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. As these parameters lead to a more piecewise-constant approach, we shrink both values, λ and ν, by a factor of ten. Testing $\lambda = 0.2$ and $\nu = 0.02$ yields to a almost optimal approximation u. To establish sharp edges and full colors, we only shrink both parameters by a factor of two. Then, we test $\lambda = 0.1$ and $\nu = 0.01$. As figure 5.14 demonstrates, this approach amounts to almost perfect piecewise-smooth approximations u. The figure further shows the other tested parameters of this estimation process.



(a) Input image g. (b) $\lambda = 2, \nu = 0.2$ (c) $\lambda = 0.2$, $\nu = 0.02$ (d) $\lambda = 0.1, \nu = 0.01$

Figure 5.14: Finding a pair (λ, ν) with the La Dama con l'ermellino (Leonardo da Vinci) image, for which the approximation is acceptable.

The first comparison using the Dykstra framework against the Lagrange framework, yields an approximation of the La Dama con l'ermellino (Leonardo da Vinci) 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 the edge set S_u, it is edge preserving. We do not see a difference in the approximations between the two approaches in figure 5.15. But, when looking at table 5.6, the Lagrange Multiplier approach is more than 450 times faster and amounts only a fourth of memory usage.

Note, that we do not need 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,

(a) Input image g. (b) $\lambda = 0.1$, $\nu = 0.01$ (c) $\lambda = 0.1$, $\nu = 0.01$

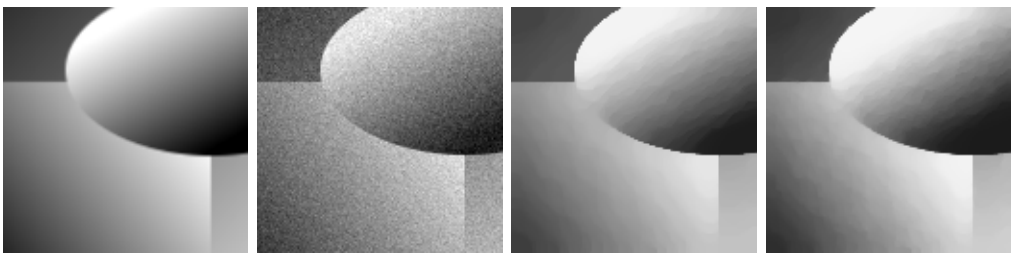
Figure 5.15: Approximation of the La Dama con l'ermellino (Leonardo da Vinci) image using the Mumford-Shah functional.

La Dama con l'ermellino			Synthetic Test Image		
Iter	Memory	Run-Time	Iter	Memory	Run-Time
1000	0.239981 GB	26.37 s	1000	0.239981 GB	27.57 s
1000	1.040617 GB	190.14 s	1000	1.040617 GB	190.33 s

Table 5.6: The run-time of the Lagrange Multiplier approach is much better, than using Dykstra's projection algorithm.

we want to show, that despite the lack of run-time, the Mumford-Shah model is the one, which preserves edges in an approximation u best, compared to ROF, TVL1 and the real-time approach to the Mumford-Shah functional.

5.8.3 Denoising with the Mumford-Shah Functional



(a) Original image. (b) Input image g. (c) $\lambda = 0.1, \nu = 0.01$ (d) $\lambda = 0.1, \nu = 0.01$

Figure 5.16: Denoising Gaussian noise with the two approaches for minimizing the Mumford-Shah model.

First of all, we present denoising using the Mumford-Shah functinal. We use a synthetic test image, to show, that this algorithm is able to remove Gaussian noise and

still preserves sharp edges and structures. But still, when looking at the run time, the Lagrange Multiplier approach outperforms Dykstra's algorithm, see again table 5.6.

(a) Original image. (b) Approximation.

Figure 5.17: Approximation of the Lena image using the Mumford-Shah functional and 25 label spaces.

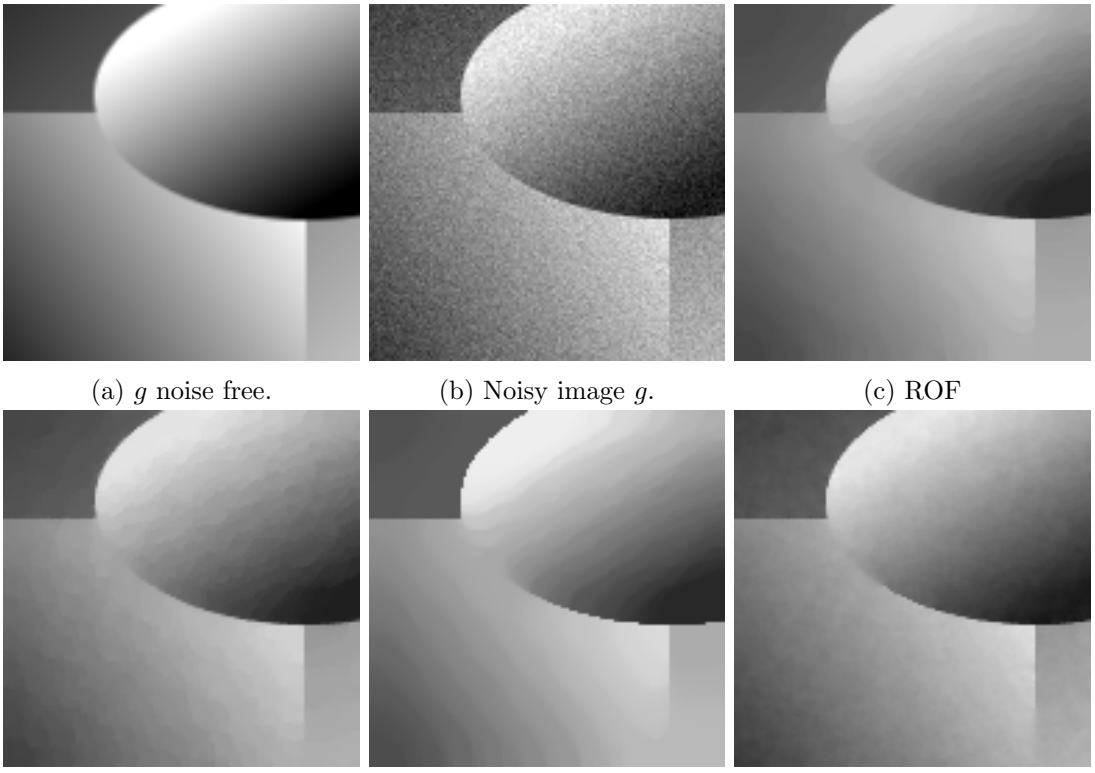
Additionally, we mention, that the approximated images have a size of 128×128. This means having a run-time of about 25-26 seconds, using the Lagrange multiplier approach, seems applicable. On the other hand, considering larger images, for instance the Lena image with size 512×512 and $S = 16$, amounts a run-time of 196 seconds. This is a tremendous increase and far away from real-time computation. The maximal number of label spaces, we can apply for this particular image are $S = 25$. This amounts 3.04488 GB memory. So, for this case we completely use the memory of the GPU. Run-time is at 715 seconds. The approximation can be viewed in figure 5.17.

5.8.4 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, but as discussed has tremendous run-time issues. 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 best suited approximations.

Model	tau	λ	ν	Iterations	Run-Time
ROF	0.63	0.03	na	530	1.02 s
TVL1	0.98	0.7	na	180	0.36 s
Realtime MS	0.25	2	0.5	60	1.31 s
Lagrange MS	na	0.1	0.01	2054	94 s

Table 5.7: Even though, the convex relaxed Mumford-Shah model provides the best approximation, the other methods outperform the algorithm.



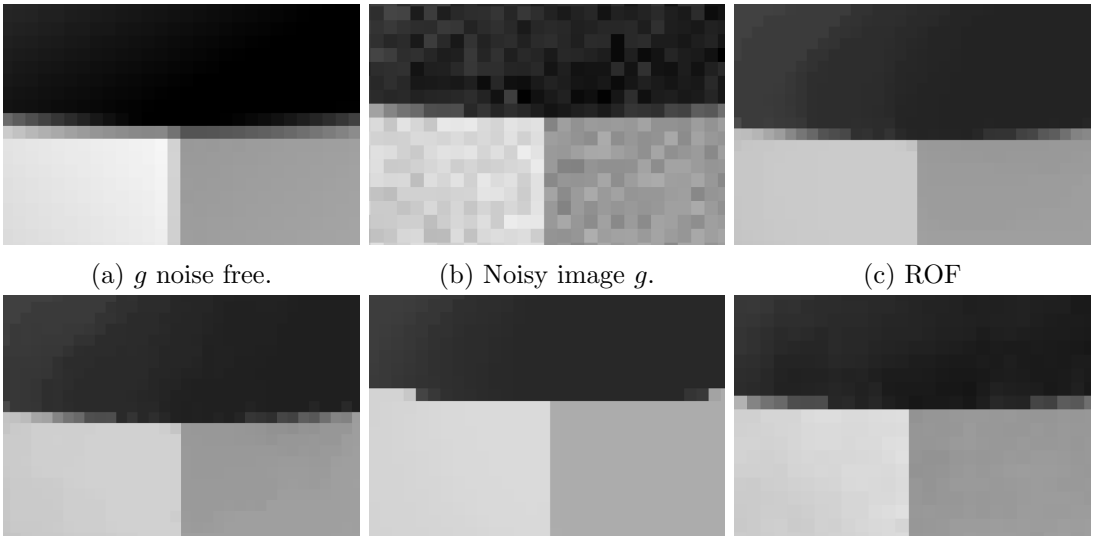
(a) g noise free. (b) Noisy image g. (c) ROF

(d) TVL1 (e) Lagrange MS (f) Real-Time MS

Figure 5.18: These pictures give an overview of denoised images using different models.

Surprisingly, the real-time approach to the Mumford-Shah model handles this special image quite well. But still, the results of this framework are not comparable to those of the original convex relaxed formulation. The difference of this approach to the ROF and TVL1 model is indeed, that it is much more edge preserving. The edges are sharper and do not vanish. If we zoom into the images, as figure 5.19 reveals, there is a clear partitioning of the regions. If we take a look at the run-times, for this particular image of size 128×128, we see huge differences. Especially, when taking into account that the Lagrange Multiplier approach to the Mumford-Shah functional is parallelized on a GPU and the other run-times are all achieved on a CPU, see table 5.7.

Overall, we conclude that the choice of the model depends on the underlying problem. If one seeks for fast, possibly real-time, solutions then the convex relaxed Mumford-Shah model is not the best choice. On the other hand, if structures and contours are of importance for a given statement of the problem, then the only choice can be the Mumford-Shah model.



(a) g noise free. (b) Noisy image g. (c) ROF

(d) TVL1 (e) Lagrange MS (f) Real-Time MS

Figure 5.19: The convex relaxed Mumford-Shah model yields sharp edges in its approximation.

6 Conclusion

We presented three different models in image processing. The convex functionals, whose energy should be minimized, are easy to implement. They have a high applicability to image processing and can handle a broad variety of problems. We saw, that a simple model as the ROF model is able to remove Gaussian noise and slightly modified 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, which also has its advantage in its simple structure and is able to remove salt and pepper noise. These two models are highly applicable in image processing and solving it via the primal-dual algorithm is parallelizable. For this, it is possible to run the program in real-time.

On the other hand, we visited the non-convex Mumford-Shah functional we seek to minimize. We proposed a framework, which is able to minimize the energy of this functional in real-time. Further, it provides a discrete set, with which one is able to highlight edges in the approximations. The framework itself is also efficient in removing Gaussian noise. Unfortunately, to-date we have no proof, that this method yields the minimal value of the Mumford-Shah functional. Despite this fact, the celebrated method is a great approach to this non-convex problem.

Mathematically, minimizing the Mumford-Shah is of high interest. The also celebrated approach, by 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 has a tremendous lack of run-time. To find a minimzer of the convex relaxed Mumford-Shah functional, the suggested method is to use Dykstra's projection method. As it turns out, run-time of this algorithm lies in minutes for very small images. For that, we reformulated the original problem and used Lagrange multipliers to derive a new formulation. Even, if we observe a speed up of a factor 450, the program is still far away from real-time. Nonetheless, our proposed approach is far more applicable. It can not be used in real-time computations, but for instance in medicine, where a timespan of thirty seconds to ten minutes is acceptable.

It depends on the underlying problem one considers. The Mumford-Shah functional is able to preserve edges, remove noise, inpaint images and can be used for image segmentation. It combines all possible applications to imaging. But, the approach, which can handle all these applications has a bad run-time. The real-time approach is not able to deal with all problems. We suggest using the ROF and TVL1 model, respectively, since they can handle almost all applications in image processing. Further, they are easy to solve, implement and modify. As their corresponding primal-dual algorithm is also parallelizable, solving these models can be done in real-time. Coming back to our original statement, that image processing is used for autonomously driving cars, the only choice can be these two models.

Bibliography

[1] G. Alberti, G. Bouchitt, 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. Bouchitt, 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. 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, Einfhrung 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. Birkhuser, Bosten, Basel, Stuttgart, 1984. Monographs in mathematics.

[14] J. P. McKelvey. Simple transcendental expressions for the roots of cubic equations. *"Amer. J. Phys."*, 52:269–270, 1984.

[15] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *"Comm. Pure Appl. Math."*, 42:577–685, 1989.

[16] Jorge Nocedal and Stephen J. Wright. *"Numerical Optimization"*, volume Seconde Edition. Springer Science + BusinessMedia, LLC., 2006.

[17] nVidia. Cuda education & training. `https://developer.nvidia.com/cuda-education-training`.

[18] 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.

[19] R. T. Rockafellar. *"Convex analysis"*, volume Reprint of the 1970 original, Princeton Paperbacks. Princeton, NJ, 1997. Princeton Landmarks in Mathematics. Princeton University Press.

[20] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *"Physica D"*, 60:259–268, 1992.

[21] 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.

[22] 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`.

[23] 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.

[24] 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.