Published in Image Processing On Line on 2020–09–29.
Submitted on 2020–02–21, accepted on 2020–07–31.
ISSN 2105–1232 © 2020 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2020.295

PALMS Image Partitioning – A new Parallel Algorithm for the Piecewise Affine-Linear Mumford-Shah Model

Lukas Kiefer[1,2], Martin Storath[3], Andreas Weinmann[2]

[1]Mathematical Imaging Group, Heidelberg University, Germany (lukas.kiefer2@gmail.com)
[2]Department of Mathematics and Natural Sciences, Hochschule Darmstadt, Germany
[3]Department of Applied Natural Sciences and Humanities, Hochschule Würzburg-Schweinfurt, Germany

Abstract

We present a method for computing approximate solutions of the piecewise affine-linear Mumford-Shah model – PALMS Image Partitioning. The piecewise affine-linear Mumford-Shah model is a variational approach to image partitioning. The underlying algorithm is based on a splitting approach using ADMM. The emerging subproblems are solved exactly and efficiently. We detail the solver for these subproblems which is based on dynamic programming and incorporates an acceleration strategy. The subproblems are solved in parallel in our implementation to provide an efficient overall algorithm. We conduct extended studies on the effects of the algorithmic parameters. Thereby, the implemented algorithm is further optimized w.r.t. runtime and efficiency. Finally, we underpin the efficiency of the algorithm by a comparison with the state-of-the-art which shows that the presented algorithm has lower computation times and yields lower mean functional values.

Source Code

The MATLAB and C++ source code for the presented algorithm is available at the associated web page[1]. Compilation and usage instructions are included in a `README.txt` file.

Keywords: image partitioning; unsupervised segmentation; piecewise affine-linear Mumford-Shah model; splitting approach; dynamic programming; parallelization

1 Introduction

Image partitioning is the task of dividing the domain of an image into regions of homogeneous image characteristic [11, 12]. Variational methods have been successfully used for this task [15, 7, 5, 9]. Typically, they impose regularity on both the regions and the boundaries of the segments. More precisely, the partition is modeled as the result of a minimization problem, where the corresponding functional imposes penalties for rough boundaries and/or lacking smoothness. The piecewise constant Mumford-Shah model is a particularly popular variational approach to image partitioning [7, 5, 16,

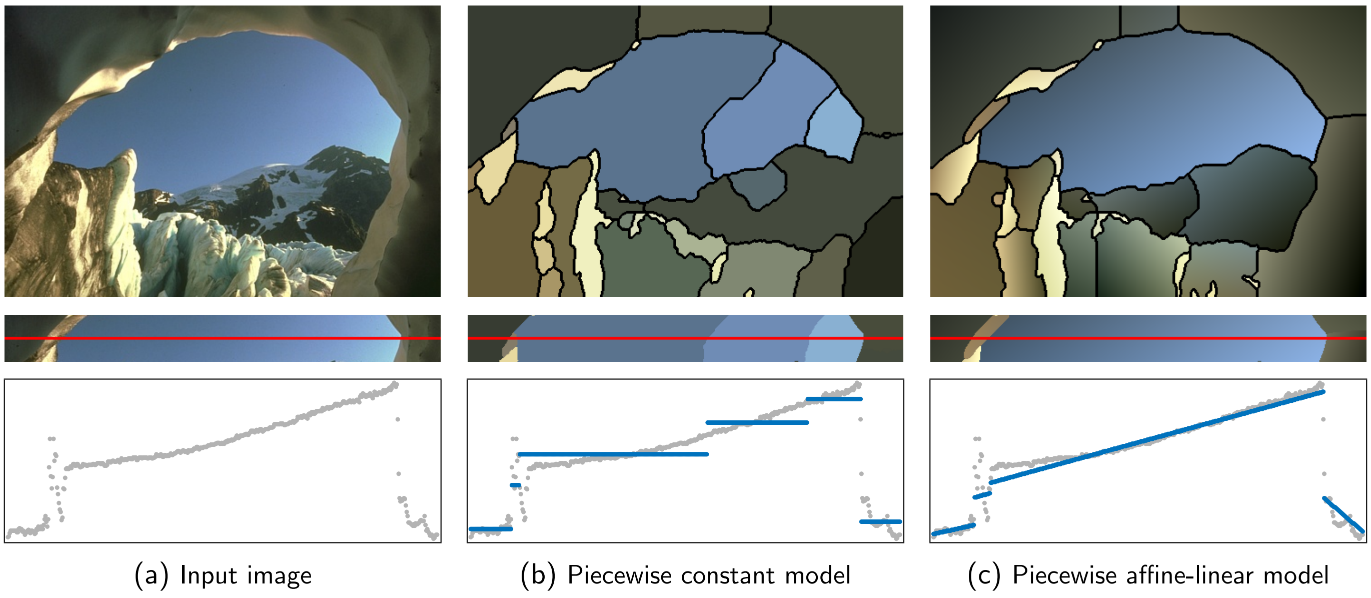


Figure 1: Comparison of the classical piecewise constant and the piecewise affine-linear Mumford-Shah model. The piecewise constant model oversegments the sky, while the right boundary of the cave is merged with the mountain. The piecewise affine-linear model yields an improved partitioning. The scanline plots of the blue channel reveal that the piecewise constant model produces spurious segments to account for the color gradient in the sky; the piecewise affine-linear model adjusts to the color gradient.

21]. It partitions the image domain into segments of constant color intensity such that the total boundary length is small and the corresponding piecewise constant image is close to the input image.

The restriction to piecewise constant results can be limiting. For instance, for images with color gradients the piecewise constant model can produce extra (spurious) segments and the results become oversegmented; see Figure 1. A natural way to allow for linear trends in an image is to consider a model which allows for affine-linear segments. Taking this into account amounts to the piecewise affine-linear Mumford-Shah model which, given an input image f, corresponds to the minimization problem

$$\begin{aligned} &\operatorname*{argmin}_{u,\mathcal{P}} \sum_{P\in\mathcal{P}} \left\{\frac{\gamma}{2}\text{length}(\partial P) + \int_P |u(x) - f(x)|^2\,dx\right\}, \\ &\text{subject to } u|_P \text{ is affine-linear for all } P \in \mathcal{P}. \end{aligned} \tag{1}$$

The minimization takes place w.r.t. (non-rough) partitions $\mathcal{P}$ of the image domain $\Omega \subset \mathbb{R}^2$ and the corresponding piecewise affine-linear functions u on Ω. A partition $\mathcal{P}$ of Ω is understood as a set of pairwise disjoint connected subsets P, i.e., the segments of Ω whose union equals Ω. The parameter $\gamma > 0$ controls the balance between closeness to the input image and total boundary length of the segments; that is, for large γ only few segments are created. In particular, for $\gamma \to \infty$ the solution of (1) is the affine-linear approximation of f and for $\gamma \to 0$ the input image is recovered. In Figure 1, we compare the piecewise constant model and the piecewise affine-linear model for a natural image with color gradient. While the piecewise constant model introduces spurious segments to approximate steep color gradients, the piecewise affine-linear model provides an improved result. In Figure 2, we illustrate different choices of the model parameter γ.

[1]https://doi.org/10.5201/ipol.2020.295

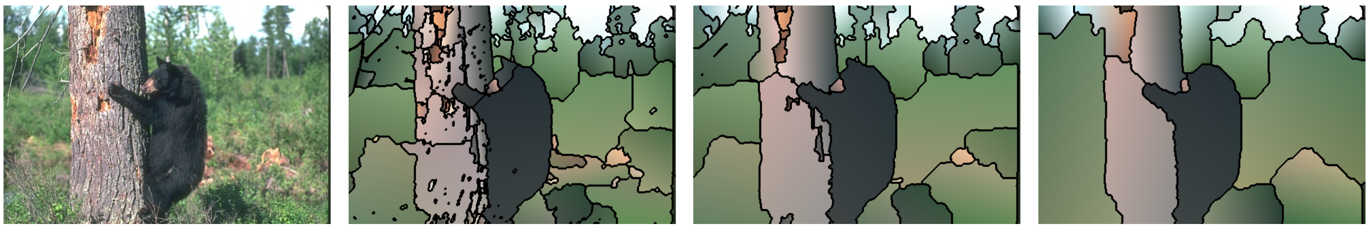


Figure 2: Effect of the model parameter γ. Large values of γ give few segments, while small choices lead to more segments and closeness to the data. Furthermore, in the limit case $\gamma \to 0$ the result reproduces the input image and for $\gamma \to \infty$ the result corresponds to the affine-linear approximation of the input image, respectively.

1.1 Contributions

In this paper, we present an optimized implementation of the algorithmic approach to the piecewise affine-linear Mumford-Shah derived by the authors in [13]. We elaborate on the employed solver of the arising subproblems which is based on dynamic programming. Towards an optimized implementation we include a pruning strategy in the dynamic programming approach to the subproblems. In extended numerical studies, we analyze the various algorithmic parameters. Furthermore, we study the speedup by including parallelization. The improvements are illustrated by numerical experiments. To sum up the contributions are as follows: (i) We give a detailed description of the dynamic programming approach to the subproblems. (ii) We achieve an acceleration of the computations by including a pruning strategy. (iii) We conduct extended studies on the effects of different choices of penalty progression, neighborhood system, parallelization and initialization. (iv) As a result of (ii)-(iii), the provided implementation is further optimized w.r.t. runtime and efficiency compared to [13].

1.2 Model

The authors' approach in [13] is based on a reformulation of (1) in terms of Taylor jets and a corresponding discretization. For the reader's convenience we give a brief recap.

Basics on jets. The Taylor jet $\mathcal{J}u$ of order k of a function $u : \Omega \to \mathbb{R}$ is a field of Taylor polynomials of order k. More precisely, $\mathcal{J}u : \Omega \to \Pi_k$ maps the domain Ω to the space of k-th order bivariate polynomials and $\mathcal{J}u(x)$ is given by the k-th Taylor polynomial of u at x. We focus on the case $k = 2$ which amounts to the first order Taylor jet. In this case $\mathcal{J}u(x)$ is the first order Taylor polynomial whose pointwise evaluation is given by

$$\mathcal{J}u(x)(z) = u(x) + \tfrac{\partial u(x)}{\partial x_1}(z_1 - x_1) + \tfrac{\partial u(x)}{\partial x_2}(z_2 - x_2), \tag{2}$$

for $z \in \mathbb{R}^2$. From (2) it follows immediately that the jet $\mathcal{J}u$ recovers the function u by $u(x) = \mathcal{J}u(x)(x)$. Since we are only interested in the case $k = 2$ in this paper, we will refer to the first order Taylor jet of a function u as *the jet of* u.

Jet formulation of the piecewise affine-linear Mumford-Shah model. It is straightforward to see that a function $u : \Omega \to \mathbb{R}$ is piecewise affine-linear if and only if its jet $\mathcal{J}u$ is a piecewise constant field. Therefore we may formulate (1) as

$$\begin{gathered}\operatorname*{argmin}_{u,\mathcal{P}} \sum_{P \in \mathcal{P}} \Big\{ \frac{\gamma}{2} \mathrm{length}(\partial P) + \int_P |u(x) - f(x)|^2 \, dx \Big\}, \\ \text{subject to } \mathcal{J}u|_P \text{ is constant for all } P \in \mathcal{P}.\end{gathered} \tag{3}$$

Moreover, by denoting the length of the jump set of the jet $\mathcal{J}u$ by $\|\nabla \mathcal{J}u\|_0$ we can formulate (3) in terms of u only

$$u^* = \operatorname*{argmin}_u \gamma \|\nabla \mathcal{J}u\|_0 + \int_\Omega |u(x) - f(x)|^2 \, dx. \tag{4}$$

(The factor 1/2 from (3) vanishes since we do not have to compensate for double counting the boundaries anymore.) Solutions of (4) are piecewise affine-linear because $\|\nabla \mathcal{J}u\|_0 < \infty$ if and only if the jet $\mathcal{J}u$ of u is piecewise constant. A basic but important observation is that there is a one-to-one correspondence between piecewise constant (first order) polynomial fields and the jets of piecewise affine-linear functions. First, let us denote by $\mathrm{PC}(\Omega, \Pi_1)$ the space of piecewise constant fields of first order polynomials. For each $J \in \mathrm{PC}(\Omega, \Pi_1)$ there is the corresponding piecewise affine-linear function $u(x) = J(x)(x)$ such that $\mathcal{J}u = J$. This allows us –instead of (4)– to consider the following equivalent minimization problem over polynomial fields

$$J^* = \operatorname*{argmin}_{J \in \mathrm{PC}(\Omega, \Pi_1)} \gamma \|\nabla J\|_0 + \int_\Omega |J(x)(x) - f(x)|^2 \, dx. \tag{5}$$

The connected components of the level sets of J correspond to the segments of the corresponding partition $\mathcal{P}$ and $u^*(x) = J^*(x)(x)$.

The discrete problem. Let $\Omega' = \{1, \ldots, m\} \times \{1, \ldots, n\}$ be the discrete image domain. In the discrete setting, the arguments to optimize for are fields $J : \Omega' \to \Pi_1$ of affine-linear polynomials on the discrete domain Ω'. We discretize the functional in (5) as follows. A common discretization of the length penalty term in (5) is given by (see [6, 3, 21])

$$\|\nabla J\|_0 = \sum_{s=1}^{S} \omega_s \|\nabla_{d_s} J\|_0. \tag{6}$$

(Please note that we use the same symbol for both the length term in the discrete and the continuous case.) Here, J is a field on the discrete domain Ω' as above and the right hand side amounts to the number of changes of the field J in the direction $d_s \in \mathbb{Z}^2$,

$$\|\nabla_{d_s} J\|_0 = \#\big\{x \in \Omega' : \, J(x) \neq J(x + d_s), \, x + d_s \in \Omega'\big\}. \tag{7}$$

The directions d_s form a neighborhood system $\{d_1, \ldots, d_S\}$, $S \geq 2$. The simplest choice is the system of unit vectors $\{e_1, e_2\}$ with the weights $\omega_{1,2} = 1$. To reduce anisotropy effects, we focus on the more isotropic discretization $\{e_1, e_2, e_1 + e_2, e_1 - e_2\}$ together with the weights $\omega_{1,2} = \sqrt{2} - 1$ and $\omega_{3,4} = 1 - \frac{\sqrt{2}}{2}$ used in [21] which coincide up to normalization with those of [6, 3]. Altogether, the discretized problem is given by

$$\operatorname*{argmin}_{J : \Omega' \to \Pi_1} \gamma \sum_{s=1}^{S} \omega_s \|\nabla_{d_s} J\|_0 + \sum_{x \in \Omega} |J(x)(x) - f(x)|^2. \tag{8}$$

The first term in (8) makes the problem nonsmooth, nonconvex and even NP-hard [22, 5]. We use a splitting method based on the ADMM to approach (8). The arising subproblems are solved exactly and efficiently by a tailored dynamic programming scheme. Furthermore, those subproblems are highly parallelizable. Note that in the following we use the symbol Ω to denote the discrete domain.

2 Algorithmic Approach

We present the algorithmic approach to the piecewise affine-linear Mumford-Shah model (8) as developed by the authors in [13]. That is, we employ a splitting approach based on ADMM. (We note that ADMM approaches often work well for various nonconvex problems; see, e.g., [23, 25, 8, 20].) To this end, we reformulate the original unconstrained problem (8) as an equivalent constrained problem with additional splitting variables. It turns out that the subproblems arising in the ADMM scheme for this constrained problem can be solved exactly and efficiently. In addition to [13], we give a more detailed description of the solver of these subproblems. We further incorporate a pruning strategy to speed up computations. Finally, we give more details about the extension to multichannel images briefly described in [13]. In particular, we illustrate the advantages over its channelwise counterpart in terms of quality and runtime.

2.1 Splitting Approach based on ADMM

In order to access the discrete length term (6) more directly, we reformulate (8) as a constrained problem. To this end, we introduce splitting jet variables J^s for each direction d_s subject to the constraint that they are equal

$$\begin{aligned}&\operatorname*{argmin}_{J^1,\dots,J^S} \sum_{s=1}^{S} \left\{ \gamma\omega_s \|\nabla_{d_s} J^s\|_0 + \sum_{x\in\Omega} \frac{1}{S} |J^s(x)(x) - f(x)|^2 \right\},\\ &\text{subject to } J^1 = J^2 = \ldots = J^S.\end{aligned} \tag{9}$$

Note that due to the equality constraints, (9) is equivalent to the original problem (8). To improve readability we introduce abbreviations for the offset values $J^s(x)(x)$ and the slopes in coordinate directions of J^s, respectively. (The first order polynomials $J^s(x)$ are uniquely defined by these.) In particular, we define for the coordinate directions $e_1 = (1,0)^T$ and $e_2 = (0,1)^T$,

$$u^s(x) := J^s(x)(x), \qquad \text{the function value of } J^s(x) \text{ at } x, \tag{10}$$

$$a^s(x) := J^s(x)(x+e_1) - J(x)(x), \qquad \text{the horizontal slope of } J^s(x), \tag{11}$$

$$b^s(x) := J^s(x)(x+e_2) - J(x)(x), \qquad \text{the vertical slope of } J^s(x). \tag{12}$$

We treat u^s, a^s, b^s as $(m \times n)$-matrices whose entries are given by $u^s(x), a^s(x), b^s(x), x \in \Omega$ and denote by $\|\cdot\|$ the Frobenius norm, i.e., $\|u\|^2 = \sum_{x\in\Omega} u(x)^2$. By using this notation the constrained problem (9) can be written as

$$\begin{aligned}&\operatorname*{argmin}_{J^1,\dots,J^S} \sum_{s=1}^{S} \gamma\omega_s \|\nabla_{d_s} J^s\|_0 + \frac{1}{S}\sum_{x\in\Omega} |u^s(x) - f(x)|^2,\\ &\text{subject to } u^s(x) = u^t(x),\ a^s(x) = a^t(x),\ b^s(x) = b^t(x),\ 1 \le s < t \le S, \text{ for all } x \in \Omega.\end{aligned} \tag{13}$$

We implicitly used the fact that two affine-linear polynomials are equal if and only if their slopes are equal and their evaluation coincides in one point.

We utilize ADMM to decompose the constrained problem (13) into coupled subproblems. ADMM performs iterative block-coordinate-wise minimization on the augmented Lagrangian associated with (13) and a gradient ascent on the Lagrange multipliers corresponding to the constraints. The augmented

Lagrangian of (13) is given by

$$\begin{aligned}&\mathcal{L}_{\mu,\nu}(\{J^s\},\{\lambda^{s,t}\},\{\tau^{s,t}\},\{\rho^{s,t}\})\\&=\sum_{s=1}^{S}\Bigg\{\omega_s\gamma\|\nabla_{d_s}J^s\|_0+\frac{1}{S}\|u^s-f\|^2+\sum_{t=s+1}^{S}\bigg(\tfrac{\mu}{2}\big\|u^s-(u^t-\tfrac{\lambda^{s,t}}{\mu})\big\|^2-\tfrac{1}{2\mu}\|\lambda^{s,t}\|^2\\&\qquad+\tfrac{\nu}{2}\big\|a^s-(a^t-\tfrac{\tau^{s,t}}{\nu})\big\|^2-\tfrac{1}{2\nu}\|\tau^{s,t}\|^2+\tfrac{\nu}{2}\big\|b^s-(b^t-\tfrac{\rho^{s,t}}{\nu})\big\|^2-\tfrac{1}{2\nu}\|\rho^{s,t}\|^2\bigg)\Bigg\}.\end{aligned}\tag{14}$$

The hard constraints in (13) are now part of the functional in the form of soft constraints, i.e., the squared Frobenius norms of the differences of the splitting variables. The corresponding Lagrange multipliers are denoted by $\lambda^{s,t},\rho^{s,t},\tau^{s,t}\in\mathbb{R}^{m\times n}$. The coupling parameters μ,ν determine how strong deviations between the splitting variables are penalized.

In each iteration, we minimize $\mathcal{L}_{\mu,\nu}$ w.r.t. each J^s and perform a gradient ascent on the Lagrange multipliers. Next, we consider the subproblem in the $(j+1)$-th iteration for a fixed s. By dropping all terms in $\mathcal{L}_{\mu,\nu}$ that do not depend on J^s (those terms are irrelevant when minimizing w.r.t. J^s), the subproblem for $(J^s)^{j+1}$ is given by

$$\begin{aligned}&(J^s)^{j+1}=\\&\operatorname*{argmin}_J\Big\{\omega_s\gamma\|\nabla_{d_s}J\|_0+\tfrac{1}{S}\|u-f\|^2\\&+\sum_{t=s+1}^{S}\Big(\tfrac{\mu}{2}\|u-\big((u^t)^j-\tfrac{(\lambda^{s,t})^j}{\mu_j}\big)\|^2+\tfrac{\nu_j}{2}\|a-\big((a^t)^j-\tfrac{(\tau^{s,t})^j}{\nu_j}\big)\|^2+\tfrac{\nu_j}{2}\|b-\big((b^t)^j-\tfrac{(\rho^{s,t})^j}{\nu_j}\big)\|^2\Big)\\&+\sum_{r=1}^{s-1}\Big(\tfrac{\mu_j}{2}\|u-\big((u^r)^{j+1}+\tfrac{(\lambda^{r,s})^{j+1}}{\mu_j}\big)\|^2+\tfrac{\nu_j}{2}\|a-\big((a^r)^{j+1}+\tfrac{(\tau^{r,s})^{j+1}}{\nu_j}\big)\|^2+\tfrac{\nu_j}{2}\|b-\big((b^r)^{j+1}+\tfrac{(\rho^{r,s})^{j+1}}{\nu_j}\big)\|^2\Big)\Big\}.\end{aligned}\tag{15}$$

Note that $J^1,\dots,J^{s-1}$ in (15) were already updated. The final subproblems in the ADMM scheme are derived from (15) by algebraic manipulations which make use of properties of the inner product. This was accomplished in [13]. For the reader's convenience we also provide this derivation in the appendix of this paper. Ultimately, we obtain the following iterative scheme

$$\begin{aligned}(J^s)^{j+1}&=\operatorname*{argmin}_J\tfrac{2\omega_s\gamma}{(S-1)\nu_j}\|\nabla_{d_s}J\|_0+\tfrac{2+\mu_jS(S-1)}{S(S-1)\nu_j}\|u-(\overline{u}^s)^j\|^2+\|a-(\overline{a}^s)^j\|^2+\|b-(\overline{b}^s)^j\|^2,\\&\quad\forall s=1,\dots,S\\(\lambda^{s,t})^{j+1}&=(\lambda^{s,t})^j+\mu_j((u^s)^{j+1}-(u^t)^{j+1})\quad\forall 1\le s<t\le S,\\(\tau^{s,t})^{j+1}&=(\tau^{s,t})^j+\nu_j((a^s)^{j+1}-(a^t)^{j+1})\quad\forall 1\le s<t\le S,\\(\rho^{s,t})^{j+1}&=(\rho^{s,t})^j+\nu_j((b^s)^{j+1}-(b^t)^{j+1})\quad\forall 1\le s<t\le S\end{aligned}\tag{16}$$

for the abbrevations given by

$$(\overline{u}^s)^j=\frac{2f+\mu_jS\Big(\sum_{t=s+1}^{S}\big((u^t)^j-\frac{(\lambda^{s,t})^j}{\mu_j}\big)+\sum_{r=1}^{s-1}\big((u^r)^{j+1}+\frac{(\lambda^{r,s})^{j+1}}{\mu_j}\big)\Big)}{2+\mu_jS(S-1)},\tag{17}$$

$$(\overline{a}^s)^j=\frac{1}{S-1}\sum_{t=s+1}^{S}\big((a^t)^j-\tfrac{(\tau^{s,t})^j}{\nu_j}\big)+\frac{1}{S-1}\sum_{r=1}^{s-1}\big((a^r)^{j+1}+\tfrac{(\tau^{r,s})^{j+1}}{\nu_j}\big),\tag{18}$$

$$(\overline{b}^s)^j=\frac{1}{S-1}\sum_{t=s+1}^{S}\big((b^t)^j-\tfrac{(\rho^{s,t})^j}{\nu_j}\big)+\frac{1}{S-1}\sum_{r=1}^{s-1}\big((b^r)^{j+1}+\tfrac{(\rho^{r,s})^{j+1}}{\nu_j}\big).\tag{19}$$

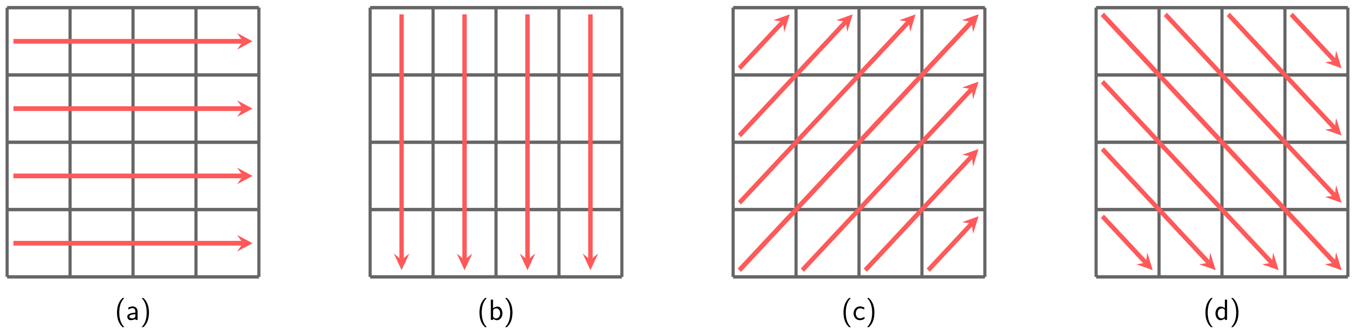


Figure 3: Decomposition of the ADMM subproblems into univariate segmented jet problems along pixel scanlines. The specific scanlines depend on the directions used for discretizing the gradient. (a)-(b): anisotropic discretization. (a)-(d): near-isotropic discretization.

As it is common when dealing with nonconvex problems, we increase the coupling parameters μ, ν after each iteration. Thus, the splitting variables can evolve rather independently in the beginning and become closer in the course of the iterations. We stop the iterations when the splitting variables become equal up to the set tolerance. Please note that we return the mean values to avoid accentuating a particular set of splitting variables. We provide a pseudocode for the algorithmic scheme (16) in Algorithm 1.

2.2 Dynamic Programming Approach to the Subproblems

The expensive part in the algorithmic scheme (16) consists in solving the S nonconvex optimization problems given by the first line of (16). The crucial observation is that by (7) these subproblems decompose into mutually independent one-dimensional partitioning problems along the paths in Ω induced by the direction d_s. We observe that they can further be solved in parallel. We give an illustration of these paths in Figure 3 for the compass and diagonal directions. In the following, we describe an exact and efficient solver for the one-dimensional partitioning problems.

We utilize a minimization strategy based on dynamic programming inspired by the approach for one-dimensional Mumford-Shah problems developed in [19]. We explain the computational scheme and adapt it to the present problems. We consider exemplarily the first subproblem in (16) and note that solving the others can be done analogously in principle. (Further details are elaborated in a subsequent paragraph.) Let $\overline{u}, \overline{a}, \overline{b} \in \mathbb{R}^n$ denote the data for the functional values and the data for the slopes, respectively. Then by using the jet notation (10)-(12), the univariate problems we have to solve have the generic form

$$J^* = \operatorname*{argmin}_{u,a,b \in \mathbb{R}^n} \gamma' \|\nabla J\|_0 + \eta^2 \|u - \overline{u}\|^2 + \|a - \overline{a}\|^2 + \|b - \overline{b}\|^2. \tag{20}$$

Recall that for the one-dimensional domain $\{1, \ldots, n\}$, we have $\|\nabla J\|_0 = |\{i \in 1, \ldots, n-1\} : J_i \neq J_{i+1}\}|$ and hence (20) amounts to a one-dimensional segmented least squares problem on $\{1, \ldots, n\}$. Solving it consists of two steps. First, we compute an optimal partition of $\{1, \ldots n\}$. After an optimal partition has been found, we obtain the minimizer J^* of (20) by solving the corresponding least squares problems for each segment.

Regarding the first step, we reformulate (20) in terms of partitions. A partition $\mathcal{I}$ of $\{1, \ldots, n\}$ is a set of pairwise disjoint discrete intervals of the form $I = \{l, l{+}1, \ldots, r\}$ such that $\{1, \ldots, n\} = \bigcup_{I \in \mathcal{I}} I$. For convenience we use the MATLAB type notation $I = l : r$. The formulation in terms of partitions is now given by

$$\mathcal{I}^* = \operatorname*{argmin}_{\mathcal{I} \text{ partition of } 1:n} \sum_{I \in \mathcal{I}} (\mathcal{E}^I + \gamma'), \tag{21}$$

Algorithm 1: ADMM strategy to the piecewise affine-linear Mumford-Shah model

```
Input: Image $f \in \mathbb{R}^{m\times n}$; edge penalty $\gamma > 0$; directions $\{d_s\}_{s=1}^S$
Output: Piecewise affine-linear approximation $u \in \mathbb{R}^{m\times n}$;
corresponding piecewise constant jet $a, b, c \in \mathbb{R}^{m\times n}$
/* Initialization */
1  $u^s \leftarrow u_0$, $a^s \leftarrow a_0, b^s \leftarrow b_0$ for all $s = 1, \dots, S$ /* splitting variables */
2  $\lambda^{s,t}, \tau^{s,t}, \rho^{s,t} \leftarrow 0$ for all $1 \le s < t \le S$ /* Lagrange multipliers */
3  $\mu \leftarrow \mu_0$, $\nu \leftarrow \nu_0$ /* initial coupling penalties */
/* ADMM iterations */
4  repeat
5      for $s = 1, \dots, S$ do
           /* Collect data for one-dimensional subproblems */
6          $\overline{u}^s \leftarrow \frac{2f + \mu S\left(\sum_{t=s+1}^{S}\left(u^t - \frac{\lambda^{s,t}}{\mu}\right) + \sum_{r=1}^{s-1}\left(u^r + \frac{\lambda^{r,s}}{\mu}\right)\right)}{2 + \mu S(S-1)}$;
7          $\overline{a}^s \leftarrow \frac{1}{S-1}\sum_{t=s+1}^{S}\left(a^t - \frac{\tau^{s,t}}{\nu}\right) + \frac{1}{S-1}\sum_{r=1}^{s-1}\left(a^r + \frac{\tau^{r,s}}{\nu}\right)$;
8          $\overline{b}^s \leftarrow \frac{1}{S-1}\sum_{t=s+1}^{S}\left(b^t - \frac{\rho^{s,t}}{\nu}\right) + \frac{1}{S-1}\sum_{r=1}^{s-1}\left(b^r + \frac{\rho^{r,s}}{\nu}\right)$;
           /* Solve one-dimensional subproblems along the paths induced by $d^s$ with Algorithm 2 */
9          $(u^s, a^s, b^s) \leftarrow \operatorname*{argmin}_J \frac{2\omega_s\gamma}{(S-1)\nu}\|\nabla_{d_s} J\|_0 + \frac{2+\mu S(S-1)}{S(S-1)\nu}\|u - \overline{u}^s\|^2 + \|a - \overline{a}^s\|^2 + \|b - \overline{b}^s\|^2$;
10     end
       /* Update Lagrange multipliers */
11     for $1 \le s < t \le S$ do
12         $\lambda^{s,t} \leftarrow \lambda^{s,t} + \mu(u^s - u^t)$;
13         $\tau^{s,t} \leftarrow \tau^{s,t} + \nu(a^s - a^t)$;
14         $\rho^{s,t} \leftarrow \rho^{s,t} + \nu(b^s - b^t)$;
15     end
       /* Update coupling penalties */
16     $\mu \leftarrow \varphi\mu$; $\nu \leftarrow \varphi\nu$;
17 until $\max_{i,j}\left\{\frac{|q^s(i,j) - q^{s+1}(i,j)|}{|q^s(i,j)| + |q^{s+1}(i,j)|}\right\} \le \eta_{\text{stop}}$ for all $s = 1, \dots, S-1$, $q \in \{u, a, b\}$
/* Take averages of splitting variables */
18 $u \leftarrow \frac{1}{S}\sum_{s=1}^S u^s$;  $a \leftarrow \frac{1}{S}\sum_{s=1}^S a^s$;  $b \leftarrow \frac{1}{S}\sum_{s=1}^S b^s$;
/* Compute offsets in origin $c$ from $u, a, b$ */
19 for $i = 1 \dots m$ do
20     for $j = 1, \dots, n$ do
21         $c(i,j) \leftarrow u(i,j) - ia(i,j) - jb(i,j)$;
22     end
23 end
24 return $u, a, b, c$;
```

where $\mathcal{E}^I$ denotes the (optimal) jet approximation error on the segment I in the sense of

$$\mathcal{E}^I = \min_{\alpha,\beta,\delta\in\mathbb{R}} \sum_{i\in I} \eta^2|\delta + i\alpha - \overline{u}_i|^2 + |\alpha - \overline{a}_i|^2 + |\beta - \overline{b}_i|^2. \tag{22}$$

Regarding the second step, the corresponding optimal jet J^* is recovered from the optimal partition by solving the least squares objective (22) w.r.t. u, a, b for each segment $I \in \mathcal{I}$ separately.

Coming back to the first step, the partitioning problem (21) can be solved efficiently by dynamic programming [10, 24] which we describe in the following. For this purpose, we denote the objective in (21) by B, i.e.,

$$B(\mathcal{I}) = \sum_{I\in\mathcal{I}} \Big(\mathcal{E}^I + \gamma'\Big). \tag{23}$$

On the reduced domain $1:r$, $r < n$ we denote its minimal value by

$$B_r^* = \min_{\mathcal{I}\text{ partition on }1:r} B(\mathcal{I}). \tag{24}$$

We observe that the minimal value B_r^* on the domain $1:r$ satisfies the Bellman equation

$$B_r^* = \min_{l=1,\ldots,r} \Big\{\mathcal{E}^{l:r} + \gamma' + B_{l-1}^*\Big\}, \tag{25}$$

where we set $B_0^* = -\gamma'$. Thus, the dynamic programming principle [2] allows us to compute $B_1^*, B_2^*, \ldots$ until we reach B_n^*. Since we are interested in an optimal partition rather than the optimal value, we keep track of an optimal partition $\mathcal{I}^*$ by storing at step r the minimizing argument l' of (25) as the value L_r so that L encodes the boundaries of an optimal partition of $1:n$ after B_n^* has been found.

In order to solve (21), we have to solve $\mathcal{O}(n^2)$ of the least squares problems (22). Each least squares problem has an approximation error $\mathcal{E}^I$. Using a standard least squares solver to compute $\mathcal{E}^I$ requires $\mathcal{O}(n)$ for each interval I which would sum up to an $\mathcal{O}(n^3)$ algorithm. (One reason for this is that after such a solver has computed optimal values $\alpha^*, \beta^*, \delta^*$ in (22), we still have to compute the optimal functional value from $\alpha^*, \beta^*, \delta^*$ which costs $\mathcal{O}(n)$.) Instead, we utilize an update strategy based on Givens rotations which was first developed in [19] for solving (higher order) one-dimensional Mumford-Shah models. In contrast to the problems considered in [19], the problems here are given in terms of discrete jets instead of discrete functional values only. However, as described in [13] a similar update strategy allows to compute the jet approximation errors $\mathcal{E}^I$ in $\mathcal{O}(1)$ which in turn yields an overall algorithm which has $\mathcal{O}(n^2)$ worst time complexity. In practice, the runtime is significantly sped-up by utilizing a pruning strategy which we describe in a subsequent paragraph.

We provide a derivation of the update scheme. To this end, we reformulate (22) to

$$\mathcal{E}^{l:r} = \min_{\alpha,\beta,\delta} \|A^q\big(\alpha,\beta,\delta\big)^T - g^{l:r}\|^2, \tag{26}$$

where $q = r - l + 1$ denotes the length of the interval $I = l:r$ and the system matrix $A^q \in \mathbb{R}^{3q\times 3}$ and the data vector $g^{l:r} \in \mathbb{R}^{3q}$ are given by

$$A^q = \begin{pmatrix} \eta & 0 & \eta \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ q\eta & 0 & \eta \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \text{and} \quad g^{l:r} = \begin{pmatrix} \eta\overline{u}_l \\ \overline{a}_l \\ \overline{b}_l \\ \vdots \\ \eta\overline{u}_r \\ \overline{a}_r \\ \overline{b}_r \end{pmatrix}, \tag{27}$$

respectively. Please note that by the definition of A^q we implicitly shifted the base point of the approximating univariate affine-linear polynomial given by $p(i) = \delta_i + i\alpha_i$ from 0 to the left boundary of I, i.e., to $l-1$. We point out that the approximation errors $\mathcal{E}^{l:r}$ in (22) and (26) are equal.

We now explain the adaption of the scheme in [19] necessary to compute the approximation errors in (26) in a fast and stable way. The approximation errors $\mathcal{E}^{l:r+1}$ are obtained from $\mathcal{E}^{l:r}$ by (precomputable) recurrence relations without having to compute the minimizer of the underlying least squares problem (26) explicitly. To keep things focused we consider the prototype interval $I = 1:r$. The approach for general left interval borders l is analogous. We use the symbols Q^r and R^r to denote the QR decomposition of the matrix A^r, that is, $A^r = Q^r \begin{pmatrix} R^r \\ 0 \end{pmatrix}$ for the orthogonal matrix Q^r and an upper triangular matrix R^r. We make the well-known basic but important observation that after applying $(Q^r)^T$ to (26), the approximation error $\mathcal{E}^{1:r}$ is given by

$$\mathcal{E}^{1:r} = \min_{\alpha,\beta,\delta} \left\| \begin{pmatrix} R^r \\ 0 \end{pmatrix} \begin{pmatrix} \alpha & \beta & \delta \end{pmatrix}^T - (Q^r)^T g^{1:r} \right\|^2 = \|((Q^r)^T g^{1:r})_{4:3r}\|^2, \tag{28}$$

where we used the invariance of the ℓ^2-norm to multiplication with orthogonal matrices and that the upper linear system induced by R^r can be solved exactly. Note that the approximation error $\mathcal{E}^{1:r}$ is explicitly given by the right-hand side of (28).

We now assume that we have computed the QR decomposition Q^r, R^r of A^r, and elaborate on how to obtain a QR decomposition for the system matrix corresponding to the extended interval $1 : r+1$, i.e., how to obtain a QR decomposition Q^{r+1}, R^{r+1} of A^{r+1} from Q^r, R^r. We begin by defining the auxiliary matrix W^r given by

$$W^r = \begin{pmatrix} R^r \\ V^r \end{pmatrix}, \quad \text{where} \quad V^r = \begin{pmatrix} (r+1)\eta & 0 & \eta \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{29}$$

Note that V^r is given by the rows of A^n corresponding to the $(r+1)$-th data points, i.e., $V^r = (A^n)_{3r+1:3r+3}$. We want to transform W^r into upper triangular form without altering zeros already present. To this end, we employ Givens rotations since a Givens rotation operates exclusively on two rows of W^r. Then, by the structure of W^r the present zeros are not destroyed. (Note that Householder transformations would not preserve the zeros.)

Recall that a Givens rotation for a rotation angle θ is represented by the matrix $G = G(j, m, \theta)$ which is an identity matrix with the 2×2 submatrix $(G_{jj}, G_{ji}; G_{ij}, G_{ii})$ replaced by a planar rotation

$$G(j,i,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & \sin(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -\sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}. \tag{30}$$

To eliminate the matrix entry A^n_{ij} by the pivot element A^n_{jj} we use the left multiplication by the Givens rotation $G(j, m, \Theta_{ij})$. The parameters of $G(j, i, \Theta_{ij})$ are given by

$$\cos(\Theta_{ij}) = A^n_{jj}/\rho, \quad \sin(\Theta_{ij}) = A^n_{ij}/\rho, \tag{31}$$

where $\rho = \operatorname{sign}(A^n_{ij})\sqrt{(A^n_{jj})^2 + (A^n_{ij})^2}$. We denote for $i = 1, \ldots, 3n,\ \ j = 1, 2, 3$ the terms in (31) by

$$\mathcal{C}_{ij} = \cos(\Theta_{ij}), \quad \mathcal{S}_{ij} = \sin(\Theta_{ij}). \tag{32}$$

γ	No pruning	With pruning	Factor
0.25	90.7 sec	37.3 sec	2.4
0.50	57.5 sec	29.0 sec	2.0
1.00	49.2 sec	27.0 sec	1.8

Table 1: Runtime improvements by pruning the subproblems. The runtimes correspond to the results of Figure 2. The pruning strategy accelerates the computations roughly by a factor 2.

By eliminating the last three rows of W^r (given by $V^r = (A^n)_{3r+1:3r+3}$) with Givens rotations we obtain R^{r+1}. Thus, we can recursively obtain R^{r+1} from R^r and the recursion coefficients are given by $\mathcal{C}, \mathcal{S}$ in (32). Consequently, in view of (28) and assuming that we have already computed $y^r = (Q^r)^T g^r$, we have the recurrence relation

$$y^{r+1} = G^{r+1} \begin{pmatrix} y^r & \eta \overline{u}_{r+1} & \overline{a}_{r+1} & \overline{b}_{r+1} \end{pmatrix}^T. \tag{33}$$

Here, G^{r+1} denotes the composition of the Givens rotations which eliminate the last three rows of W^r. As G^{r+1} only operates on the first three entries and the last three entries of the vector on the right hand side of (33), we obtain $\|y^{r+1}_{4:3r+3}\|^2 = \|y^r_{4:3r}\|^2 + \sum_{i=1}^3 (y^{r+1}_{3r+i})^2$, that is, the squared norm of the rotated data plus the squared norm of the last three entries of the new data points after applying the Givens rotations G^{r+1} encoded by the coefficients in $\mathcal{C}, \mathcal{S}$. Thus, in view of (28), the error update $\mathcal{E}^{1:r} \to \mathcal{E}^{1:r+1}$ is given by

$$\mathcal{E}^{1:r+1} = \mathcal{E}^{1:r} + \sum_{i=1}^{3} (y^{r+1}_{3r+i})^2, \tag{34}$$

that is, the error update consists of computing y^{r+1} by (33) and updating $\mathcal{E}^{1:r+1}$ by (34).

Recall that the application of G^{r+1} is given by the recursion coefficients $\mathcal{C}, \mathcal{S}$. We note that the recurrence coefficients do not depend on the data and we have to compute them only once for A^n and then use them for computing all $\mathcal{E}^{l:r}$. Also note that the Givens rotation matrices G^{r+1} never have to be computed explicitly. As a consequence, the error updates have $\mathcal{O}(1)$ computational costs; see also [19].

Acceleration by pruning strategy. To further speed up the computations we prune the search space with the strategy developed in [20]. The pruning is motivated by the observation that the approximation errors satisfy $\mathcal{E}^{l:r} \leq \mathcal{E}^{l':r}$ for all $l' \leq l$. Hence, if the current value B_r for B_r^* satisfies $B_r < \mathcal{E}^{l:r} + \gamma'$, we can skip all $l' < l$, i.e., we do not have to compute $\mathcal{E}^{l':r}$ and we set $B_r^* = B_r$. Besides the theoretical quadratic runtime guarantee which also applies to the worst case scenario (see [19]), we observe in our experiments, that the runtime is significantly reduced when additionally using the aforementioned pruning strategy. We sum up the approach to the univariate subproblems in Algorithm 2. Table 1 illustrates the acceleration effect on the overall algorithm which roughly halves the runtimes (Intel XeonE5- 2620v4, 2.10GHz, 16 cores, 256 GB RAM).

Adaption to other directions. We described a fast and stable approach to solve the first ADMM subproblem (corresponding to the direction d_1). Now we consider the other ADMM subproblems in (16), i.e., the subproblems for d_s, $s = 2, 3, 4$, and describe the necessary adaptions. For $s = 2$ the roles of a and b in (20) are simply interchanged and we can proceed as for $s = 1$. By linearity, for $s = 3$ the role of a in (21) is taken by $a(x) + b(x)$ and the role of b by $a(x) - b(x)$ (and vice-versa for $s = 4$). Thus, the input for the slope data $\overline{a}$, $\overline{b}$ in (20) and the computed slopes have to be

Algorithm 2: Solver for the univariate subproblems

```
Input: Offset and slope data $\overline{u}, \overline{a}, \overline{b} \in \mathbb{R}^n$, model parameter $\gamma' \geq 0$
Output: Optimal jet $J^* = (u^*, a^*, b^*)$ of the univariate segmented jet problem (20)
1  Row-wise transform the matrix $A^n$ from (27) to upper triangular form using successive Givens rotations;
2  Store the recurrence coefficients (32) in $\mathcal{C}, \mathcal{S}$
3  Compute $\mathcal{E}^{1:r}$ for all $r = 1, \dots, n$ with $\mathcal{C}, \mathcal{S}$
4  $L_1 \leftarrow 1$; $B_1^* \leftarrow 0$;
5  for $r = 2, \dots, n$ do
6      $L_r \leftarrow 1$; $B_r^* \leftarrow \mathcal{E}^{1:r}$;
7      for $l = r-1 \dots, 2$ do
8          Compute $\mathcal{E}^{l:r}$ from $\mathcal{E}^{l:r-1}$ with the recurrence (33)-(34) using $\mathcal{C}, \mathcal{S}$
9          $b \leftarrow B_{l-1}^* + \gamma' + \mathcal{E}^{l:r}$;
10         if $b \leq B_r^*$ then
11             $B_r^* \leftarrow b$; $L_r \leftarrow l$;
12         end
           /* Pruning */
13         if $\mathcal{E}^{l:r} + \gamma' > B_r^*$ then
14             break;
15         end
16     end
17 end
   /* Recover the optimal partition $\mathcal{I}^*$ from $L$ */
18 $r \leftarrow n$, $\mathcal{I}^* \leftarrow \emptyset$
19 while $r > 0$ do
20     $l \leftarrow L_r$, $\mathcal{I}^* \leftarrow \mathcal{I}^* \cup \{(l:r)\}$, $r \leftarrow l-1$;
21 end
   /* Recover the optimal jet $J^*$ on each segment $I \in \mathcal{I}^*$ */
22 for $I \in \mathcal{I}^*$ do
23     Solve $(u^*, a^*, b^*)_I = \operatorname{argmin}_{v,w,z} \|A^{|I|}\big(v_1, w_1, z_1, \dots, v_{|I|}, w_{|I|}, z_{|I|}\big)^T - g^I\|^2$ for $A, g$ from (27)
24 end
25 return $u^*, a^*, b^*$
```

transformed accordingly. As a consequence, the recurrence coefficients (32) have to be computed only once in each ADMM iteration and can be used for all S subproblems.

2.3 Multichannel Images

In this section, we extend the description in [13] of adapting the method to vector-valued images $f : \Omega \to \mathbb{R}^L$ with L channels. In particular, we illustrate its practical benefits compared with its channelwise counterpart. To this end, we consider the first order Taylor jet of a function $u : \Omega \to \mathbb{R}^L$ which is given by the Taylor jets of its component functions, i.e.,

$$\mathcal{J}u(x) = \Big(\mathcal{J}u_1(x), \dots, \mathcal{J}u_L(x)\Big)^T. \tag{35}$$

The multichannel version of the jet formulation of the piecewise affine-linear Mumford-Shah model (5) is then given by

$$J^* = \operatorname*{argmin}_{J \in \mathrm{PC}(\Omega, \Pi_1^L)} \gamma \|\nabla J\|_0 + \sum_{l=1}^{L} \int_\Omega |J_l(x)(x) - f_l(x)|^2 \, dx. \tag{36}$$

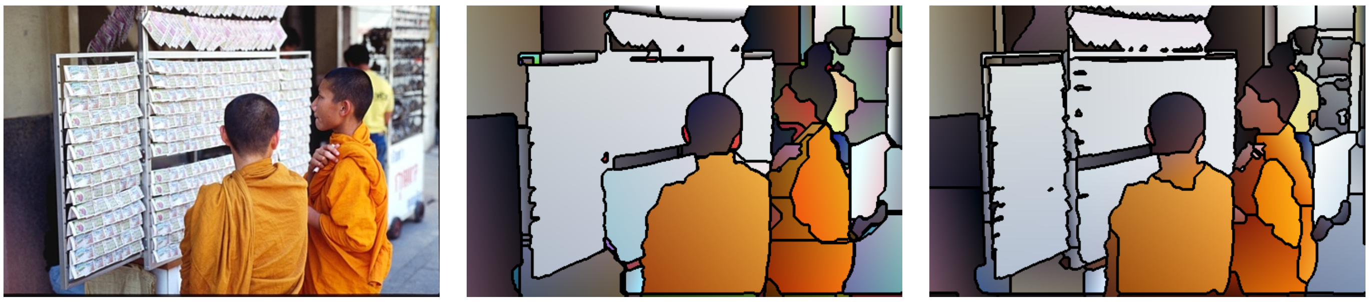

(a) Input image (b) Channelwise approach (68.0 sec) (c) Multichannel approach (26.9 sec)

Figure 4: Comparison of channelwise and multichannel approach for the input image (a). The channelwise approach (b) suffers from color artifacts (see, e.g., the face of the right boy or the ears of the left boy). (c) The multichannel approach yields a more reasonable image approximation and an improved partition. Furthermore, it needs less computation time.

Towards the discrete version of (36), the discrete length term reads

$$\|\nabla_{d_s} J\|_0 = \#\big\{x \in \Omega : J_l(x) \neq J_l(x+d_s) \text{ for at least one component } l = 1,\dots,L,\ x+d_s \in \Omega\big\}. \tag{37}$$

In particular, the segment boundaries are aligned across channels and a jump of J in one component has the same cost as a jump in all components. Altogether, the discrete problem reads

$$\operatorname*{argmin}_{J:\Omega\to\Pi_1^L} \sum_{s=1}^{S} \gamma\omega_s \|\nabla_{d_s} J\|_0 + \sum_{l=1}^{L} \sum_{x\in\Omega} |J_l(x)(x) - f_l(x)|^2. \tag{38}$$

After introducing the splitting jets $J^1,\dots,J^S$ we obtain

$$\begin{aligned} &\operatorname*{argmin}_{J^1,\dots,J^s} \sum_{s=1}^{S} \Big\{\gamma\omega_s \|\nabla_{d_s} J^s\|_0 + \frac{1}{S} \sum_{l=1}^{L} \sum_{x\in\Omega} |J_l^s(x)(x) - f(x)|^2\Big\}, \\ &\text{subject to} \quad J_l^1 = J_l^2 = \ldots = J_l^S \quad \text{for all } l = 1,\dots,L. \end{aligned} \tag{39}$$

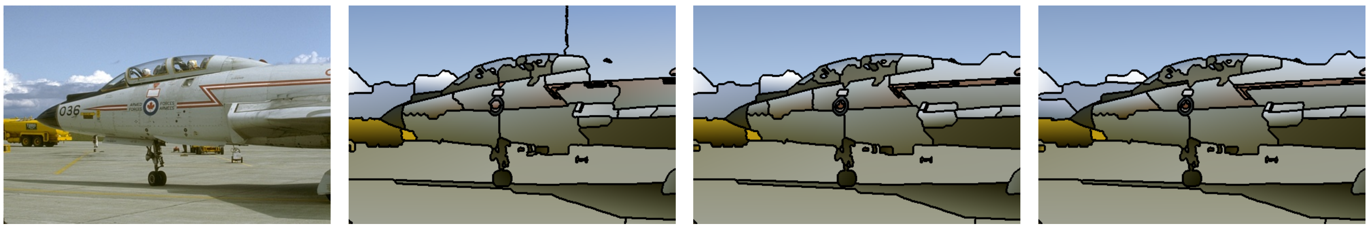
In view of the notation (10)-(12), we denote $u, a, b \in \mathbb{R}^{m\times n\times L}$ and the Lagrangian of (39) is understood w.r.t. the Lagrange multipliers $\lambda^{s,t}, \tau^{s,t}, \rho^{s,t} \in \mathbb{R}^{m\times n\times L}$. Then the derivation of the ADMM scheme is analogous to the single channel case.

Concerning the univariate subproblems, the jet approximation errors $\mathcal{E}^I$ in the dynamic programming scheme are given by the channelwise sum $\mathcal{E}^I = \sum_{l=1}^{L} \mathcal{E}_l^I$. Consequently, the error update strategy based on Givens rotations from Section 2.2 can be used by simply updating the approximation errors channelwise. For this we can use the same system matrix A^q from (27) for all channels and the recurrence coefficients (32) have to be computed only once for all channels.

In Figure 4 we illustrate the benefits of the multichannel approach compared to its channelwise counterpart.

3 Implementation Details

Our profiling of Algorithm 1 revealed that the bulk of computation time is spent on solving the univariate subproblems (more than 90%). As a consequence, we implemented this time critical part in the C++ programming language to provide high performance code. The remaining parts of Algorithm 1 –which are not time-critical– are implemented in the MATLAB programming language. Thus, PALMS calls its procedures in MATLAB for which the C++ routines are made accessible by



(a) Input image

(b) Fast progression, $\varphi = 1.7$, 37 iterations, $E = 2899.9$

(c) Medium progression, $\varphi = 1.3$, 65 iterations $E = 2821.7$

(d) Slow progression, $\varphi = 1.05$, 285 iterations $E = 2796.0$

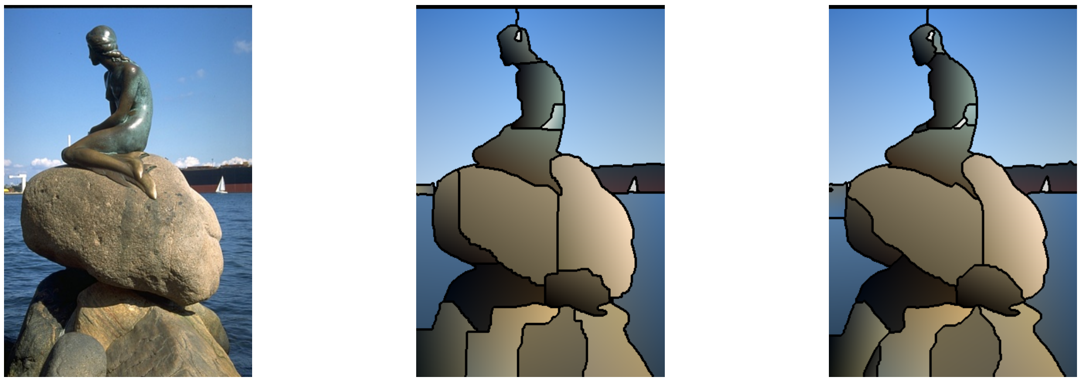
Figure 5: Effect of progression parameter. Small progression parameters φ yield improved partitions and lower energy values E, but more iterations are needed. As a good compromise of runtime and quality we find the medium choice $\varphi = 1.3$.

using MEX-files. The parts which are implemented in C++ use the Armadillo library [17, 18] –a common library for linear algebra– and the OpenMP library which is a standard choice for parallelization on multi-core CPUs. We will see that parallelization with OpenMP speeds up computations by a factor 8-9.

In the following, we discuss the different algorithmic parameters in Algorithm 1. In particular, we motivate the default values used in PALMS experimentally. Further, we illustrate the effect of parallelization. Afterwards, we describe how to obtain the partition corresponding to the computed piecewise constant polynomial field in an efficient way.

Progression choice. As pointed out in Section 2.1 we use the coupling parameters μ, ν to penalize deviations between the splitting variables and increase them after each iteration. More precisely, we initialize the values with small parameters μ_0, ν_0 and increase them by a factor $\varphi > 0$, that is, we employ the geometric progressions given by $\mu_j = \varphi^j \mu_0$, $\nu_j = \varphi^j \nu_0$. This strategy has turned out to work well in practice. Recall that we use different initial values μ_0 and ν_0 since offsets and slopes typically live on different scales. Naturally, if the progression parameter φ is chosen small, the coupling parameters grow slowly and more iterations are needed until the S splitting variables (u^s, a^s, b^s) become equal. However, a slower progression typically improves the result. In Figure 5 we compare different choices of φ. We obtain the best quality when using the small parameter $\varphi = 1.05$ and the least number of iterations for the large parameter $\varphi = 1.7$. The medium parameter $\varphi = 1.3$ yields satisfying results which are comparable to the results produced with the small parameter $\varphi = 1.05$, while needing considerably fewer iterations. (Typically the progression parameter $\varphi = 1.05$ needs 225-290 iterations and the progression parameter $\varphi = 1.3$ needs 55-70 iterations, respectively.) Hence, PALMS uses the progression parameter $\varphi = 1.3$ as default. On the one hand, towards faster runtimes the user can instead choose a larger value in PALMS. On the other hand, for higher quality the user can use a smaller value.

Neighborhood system. We compare the anisotropic discretization and the near-isotropic discretization of the length penalty term in (5). Recall that the anisotropic discretization corresponds to a 4-neighborhood system using the compass directions $d_1 = e_1, d_2 = e_2$ and the isotropic discretization corresponds to an 8-neighborhood system which also includes diagonal directions, that is, $d_1 = e_1, d_2 = e_2, d_3 = e_1 + e_2, d_4 = e_1 - e_2$. The results in Figure 6 illustrate the benefit of employing the 8-neighborhood system. It produces smoother boundaries with reduced anisotropy effects. The algorithm has to solve roughly twice as many univariate subproblems when using the 8-neighborhood system (see Figure 3). However, the practical runtimes increase only by roughly 40 % (Intel XeonE5-2620v4, 2.10GHz, 16 cores, 256 GB RAM) since less iterations are needed than for the anisotropic discretization. This might be attributed to the tighter coupling of splitting variables. (Recall that



(a) Input image

(b) Anisotropic discretization, (16.0 sec)

(c) Near-isotropic discretization, (24.9 sec)

Figure 6: The near-isotropic discretization produces smoother boundaries at the cost of about $50\,\%$ higher computation time since it needs to solve more univariate subproblems in each iteration; see Figure 3 and Algorithm 1.

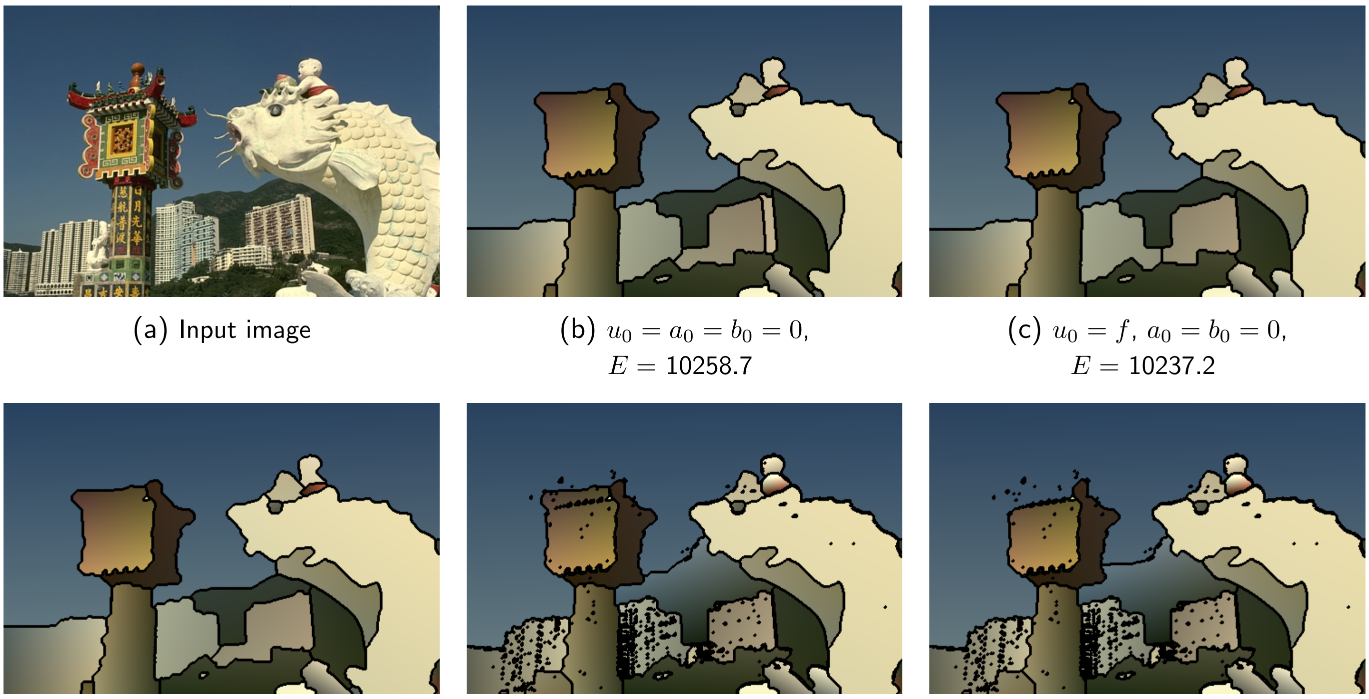
each splitting variable is coupled with all the others.) Because of the improved quality and moderate runtime increase, PALMS uses the 8-neighborhood system as default. If runtime is more critical than quality, PALMS allows the user to switch to the 4-neighborhood system instead.

Parallelization. We compare the runtimes of the single-core and the multi-core implementation of Algorithm 1. More precisely, we compare solving the univariate partitioning problems of the form (21) arising in the ADMM subproblems (16) sequentially and in parallel. (Note that up to the maximum number of image rows and columns $\max\{m, n\}$ problems can be approached in parallel.) To this end, we consider the runtimes for the input image and model parameters depicted in Figure 2. The respective runtimes are shown in Table 2. We observe that parallelization roughly accelerates the computation times by factor 8 on a 16 core CPU which illustrates the benefit of the multi-core implementation (Intel XeonE5- 2620v4, 2.10GHz, 16 cores, 256 GB RAM). Consequently, PALMS makes use of a multi-core CPU parallelization with OpenMP.

γ	Single-core	Multi-core	Factor
0.25	266.4 sec	37.6 sec	7.9
0.50	232.0 sec	29.1 sec	8.0
1.00	228.4 sec	27.2 sec	8.4

Table 2: Comparison of runtimes of single-core and multi-core implementation for the results of Figure 2. The multi-core implementation provides a significant speedup compared with its single-core counterpart.

Initialization. The splitting variables u^s, a^s, b^s of the ADMM splitting approach (16) are initialized by u_0, a_0, b_0 before the iterations start. The simplest choice amounts to an all-zero initialization. Alternatively, the initial image values could be chosen as the input image $u_0 = f$. Since the method produces piecewise affine-linear approximations, another reasonable choice amounts to the affine-linear least-squares approximation of the input image. Furthermore, the slopes a_0 and b_0 may be set to the horizontal and vertical forward differences $\nabla_1 f, \nabla_2 f$ of f, respectively. In Figure 7, we compare the different initializations. We observe that the results produced when using $u_0 = a_0 = b_0 = 0$, $u_0 = f$ or the affine-linear approximation are very similar and the respective energy values are very close. For the initialization with forward differences we see that the results show some clutter and



(a) Input image

(b) $u_0 = a_0 = b_0 = 0$, E = 10258.7

(c) $u_0 = f$, $a_0 = b_0 = 0$, E = 10237.2

(d) u_0, a_0, b_0 affine-linear approx. of f, E = 10250.0

(e) $u_0 = f$, $a_0 = \nabla_1 f, b_0 = \nabla_2 f$, E = 14208.7

(f) $u_0 = 0$, $a_0 = \nabla_1 f, b_0 = \nabla_2 f$, E = 14210.0

Figure 7: Results of different initializations for the input image (a) and $\gamma = 1.3$. We report the energy values E of the results below the images. Initializing with zeros (b), with the input image (c) and with the affine-linear approximation (d) give very similar results. Using the forward difference to initialize the slopes (e)-(f) produces clutter and yields larger energy values. Due to this and its slightly lower energy value, we use initialization (c).

the energy values become larger. Due to its simplicity and the slightly lower energy value, PALMS employs $u_0 = f, a_0 = b_0 = 0$ as the default initialization. Alternatively, the user can also employ different initializations.

Recovering the partition from the jet. The proposed algorithm computes a piecewise affine-linear approximation u of the input image together with the corresponding (piecewise constant) jet, i.e., the slopes a, b and the offsets in matrix origin c, respectively. Consequently, the jet encodes the associated partition of the input image in the sense that two neighboring pixels x and $x + d_s$ belong to the same segment if and only if the jet takes the same values in x and $x + d_s$. Algorithmically, we exploit this relation to obtain the adjacency matrix of the graph corresponding to the partition and compute its connected components to obtain the partition corresponding to a, b, c as a label image $L : \Omega \to \mathbb{N}$. The computation of the connected components is efficiently carried out by the MATLAB function `conncomp`.

Default parameters of the algorithm. In Table 3 we summarize the default parameters of the provided implementation. In particular, we give the standard values for initialization, progression and initial values of the penalty parameters, the threshold for the stopping criterion and the discretization of the gradient.

Parameter	Meaning	Default value
u_0, a_0, b_0	Initialization	$u_0 = f$, $a_0 = b_0 = 0$
μ_0	Initial offset progression value	10^{-3}
ν_0	Initial slopes progression value	$\min(450\gamma\mu_0, 1)$
φ	Progression parameter	1.3
η_{stop}	Stopping parameter	10^{-2}
$\{d_s\}$	Directions of discretized gradient	Figure 3 (a)-(d)

Table 3: Summary of default parameters. The results for other choices of u_0, a_0, b_0 are shown in Figure 7. The choices of φ and $\{d_s\}$ are justified in Figure 5 and Figure 6, respectively.

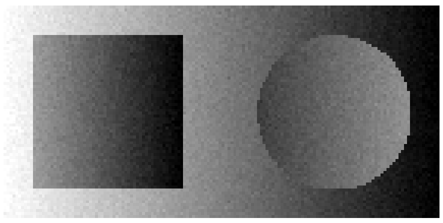
4 Experiments

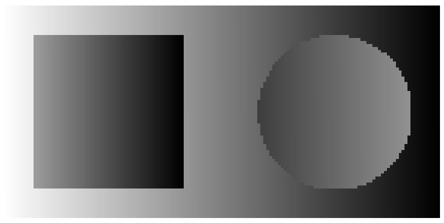
In this section, we first illustrate the outputs of Algorithm 1 by means of a synthetic image. To give an impression of the runtimes for different image sizes we then report the runtimes for the same image at different resolutions. Afterwards, we give a quantitative comparison of the proposed method with iterative application of graph cuts. All experiments were conducted on a workstation (Intel XeonE5-2620v4, 2.10GHz, 16 cores, 256 GB RAM).

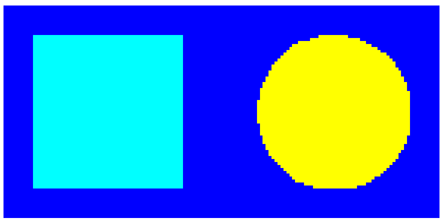
Illustration of the outputs. In addition to the piecewise affine-linear approximation u of the input image, Algorithm 1 computes the corresponding piecewise constant field of first order polynomials in terms of their coefficients a, b, c. In Figure 8, we apply the proposed algorithm to a synthetic image and illustrate the corresponding jet by 3D-plots. Indeed, while u is piecewise affine-linear, the corresponding field of first order polynomials is piecewise constant and encodes the partition associated with u.

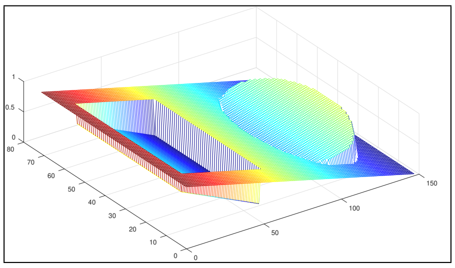
Runtime and image size. We study the dependency of the runtime of the provided implementation on the image size. To this end, we consider the same image at various resolutions and check the runtimes. In order to obtain comparable results for all resolutions, we adapt the model parameter γ accordingly, i.e., we chose $\gamma = 0.75$ for the lowest resolution and increased it proportionally w.r.t. the number of pixel rows. In Figure 9 we report the runtimes of the partitioning results in Figure 10. The considered image sizes range from 320×222 to 1024×712. (Please note that we obtained the image data for 400×279 and 480×334 by downsampling.) In particular, we observe that the runtime is about two minutes for image size 800×556.

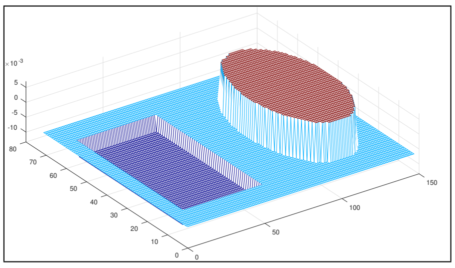
Acceleration with down-scaled images. In Figure 10, we observe that the results on the low-resolution images are remarkably close to the results at finer scales. This motivates the following basic acceleration strategy. As a first step, the input image is scaled down by a factor which can be chosen as 2 or 4 (surplus rows and columns are cut off). As a result, the total number of pixels is reduced by factor 4 and 16, respectively. (This step is accomplished by the MATLAB function `imresize`.) To account for the smaller image size, the algorithm is then run with the adapted jump penalty $\gamma/2$ and $\gamma/4$, respectively. Next, the resulting partition is scaled up to the original image size without any interpolation. (Here we use the MATLAB function `imresize` with the option 'nearest'.) In order to obtain smoother segment boundaries, we perform morphological opening on this up-scaled partition using a disk shaped structuring element of size 2 or 4, respectively. (We use the MATLAB function `imopen` with the option 'disk' and we favor larger segments in this step.) Then we obtain the corresponding jet $a', b'c'$ by repeating the values of a, b, c on the up-scaled partition. Finally, the corresponding piecewise affine-linear image of original image size u' is computed from a', b', c' so that

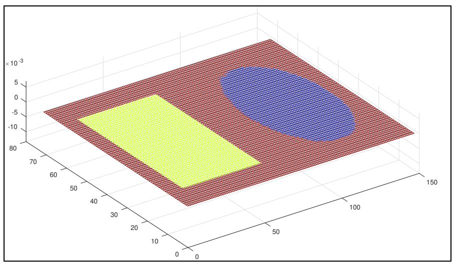

(a) Input image


(b) Piecewise affine-linear approximation u


(c) Partition corresponding to (b) (shown as label image)


(d) 3D-plot of (b)


(e) Horizontal slopes a


(f) Vertical slopes b

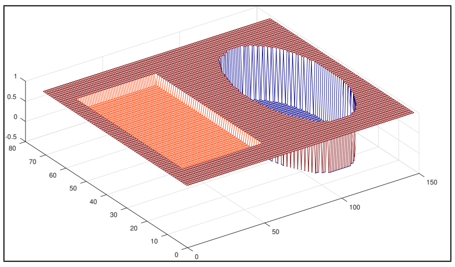

(g) Offsets c

Figure 8: Illustration of the computed outputs. While the reconstruction u is piecewise affine-linear (b),(d), the corresponding first order polynomial field (i.e., the affine-linear coefficients) (e)-(g) is piecewise constant and induces the associated partition (c).

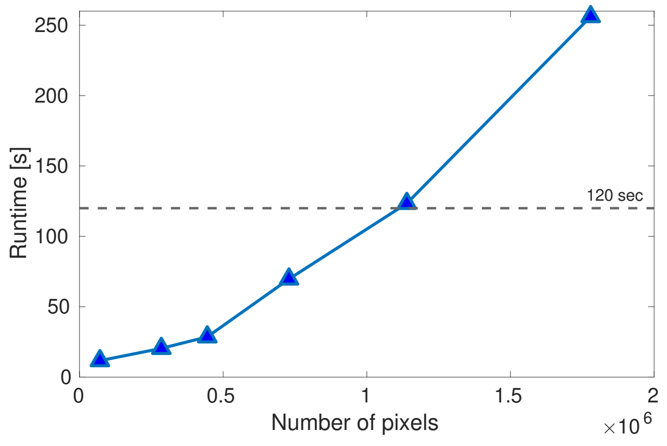
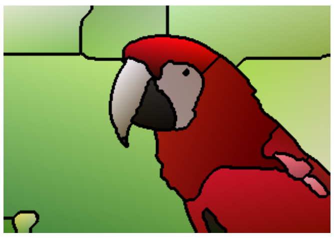


Image size	Runtime
320×222	11.7 sec.
400×279	20.4 sec.
480×334	28.6 sec.
640×445	69.7 sec.
800×556	123.4 sec.
1024×712	256.1 sec.

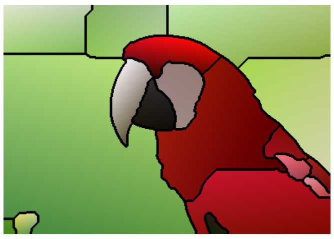
Figure 9: Plot and table of the runtimes for different resolutions of the parrot image depicted in Figure 10. In particular, for the image size 800×556 the runtime of the proposed implementation is roughly 2 minutes.

u' is not blurred over segment boundaries. In PALMS, the user can employ the described strategy optionally. In Figure 11, we compare the results for the unaltered input image with the results for the down-scaled input images (the latter were scaled up to the original image size as described above). We observe that applying this basic strategy reduces the runtimes at the expense of quality (the segment boundaries are less smooth); yet, the resulting partitions are reasonable and comparable

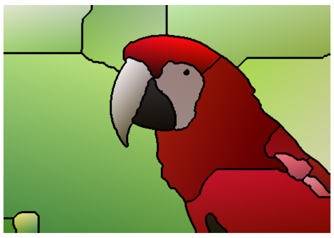
(a) Image size 320×222


(b) Comp. time 11.7 sec

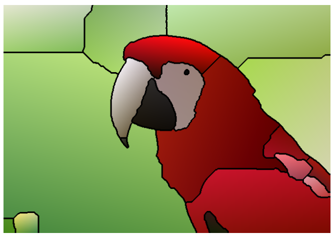
(c) Image size 400×279


(d) Comp. time 20.4 sec

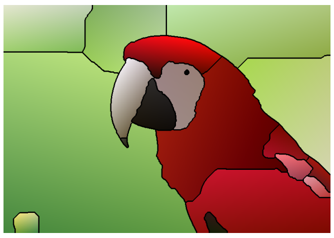
(e) Image size 480×334


(f) Comp. time 28.6 sec

(g) Image size 640×445


(h) Comp. time 69.7 sec

(i) Image size 800×556


(j) Comp. time 123.4 sec

(k) Image size 1024×712

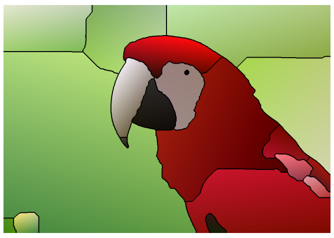
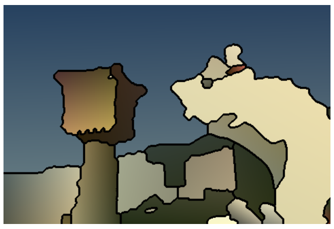
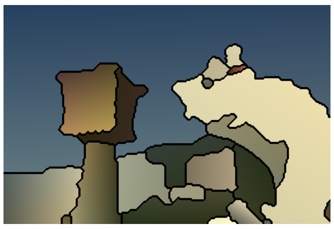

(l) Comp. time 256.1 sec

Figure 10: The results and runtimes of Table 9. The partitioning results are comparable throughout the increasing image sizes by adapting the model parameter accordingly. For image size 800×556 the proposed implementation requires about 2 minutes.

(a) Input image


(b) Result for original image size, 26.0 sec


(c) Result for half image size, 5.8 sec

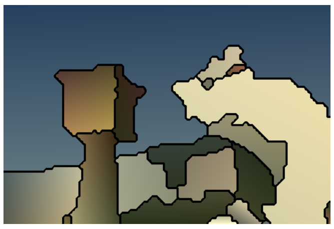

(d) Result for quarter image size, 1.3 sec

Figure 11: Acceleration by a basic down-scale strategy. The computation times for large input images can be improved for the cost of a slight quality loss (the boundaries of the partition become less smooth). The images displayed in (c) and (d) are obtained by applying the proposed method on a down-scaled image and the results were scaled up afterwards (using the jet representation as described in Section 4).

with the result for the original image size.

Comparison with iterative graph cuts. We compare the proposed algorithm to an iterative label-based repartitioning scheme based on graph cuts which we will call GC in the following. We used the α-expansion algorithm of the toolbox GCO v3.0 [5, 4, 14]. The scheme is given as follows:

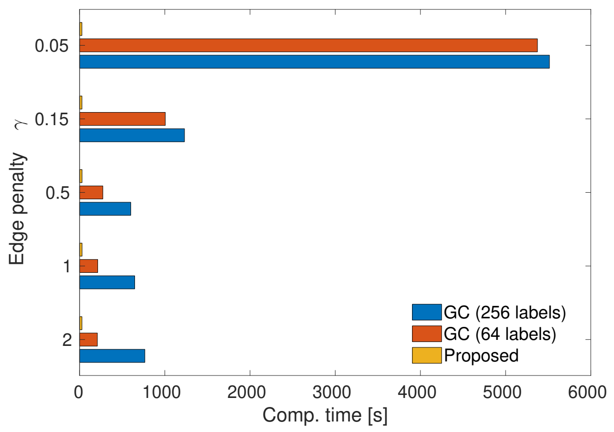
1. Choose an initial partition $\mathcal{P}^0$.
2. Iterate until convergence:
   a. for all $P \in \mathcal{P}^t$, and $l = 1, \ldots, L$, compute the optimal affine coefficients,

$$(a^*, b^*, c^*) = \underset{a,b,c}{\operatorname{argmin}} \sum_{(i,j)\in P} (ai + bj + c - f_L(i,j))^2;$$

   b. compute a partition $\mathcal{P}^{t+1}$ which is (approximately) optimal for the determined set of affine coefficients with the α-expansion algorithm.

The initial partition $\mathcal{P}^0$ is obtained by a piecewise constant partitioning of the input image using a subset of K color values as labels. We proceeded with the iterative repartitioning until the relative improvement of the energy was less than 10^{-3} or after a maximum number of 10 iterations was reached.

We compare Algorithm 1 to GC using 64 and 256 initial labels for the 200 test images (481×321 RGB) of the Berkeley segmentation data set [1]. We report the average model energies and the average runtimes in Figure 12. We observe that the proposed scheme achieves lower mean energies and faster computation times than GC which illustrates the effectiveness of the algorithm. In particular for small model parameter γ, the proposed scheme is notably faster. In Figure 13 we give a comparison of the proposed method with GC (with 256 initial labels) for example images from the Berkeley test set. Further examples of the proposed method for fixed model parameter are provided in Figure 14.



γ	GC (64 labels)	GC (256 labels)	Proposed
0.05	1802.2	1802.1	**1772.3**
0.15	3180.3	3178.2	**3146.0**
0.50	5484.8	5480.7	**5418.2**
1.00	7324.7	7317.5	**7202.7**
2.00	9551.5	9526.4	**9356.7**

Figure 12: *Left:* Average runtimes for the Berkeley test set. The proposed method has significantly lower mean computation times than the iterative graph cuts method ($\approx$28 sec. for any γ). *Right:* Mean energies over the Berkeley test set and the parameters γ in the left column. The proposed algorithm gives the lowest mean energies.

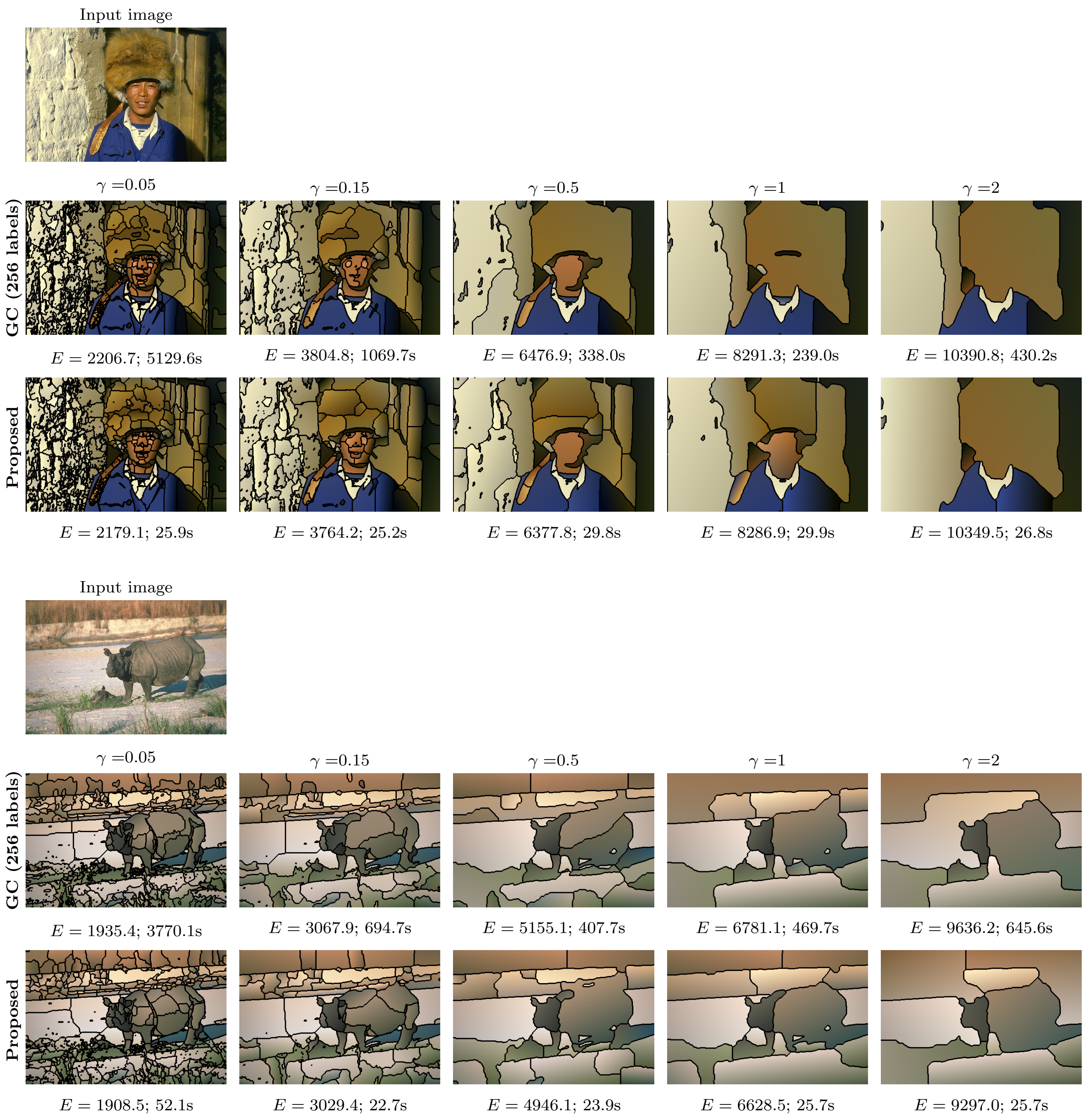


Figure 13: Comparison to GC for examples from Berkeley test images. As a tendency, the proposed algorithm produces slightly more segments than GC (see, e.g., the man's hat in the first image for $\gamma = 0.5$ or the background of the second image for $\gamma = 1$). Further, the proposed algorithm gives lower energies E and faster runtimes.

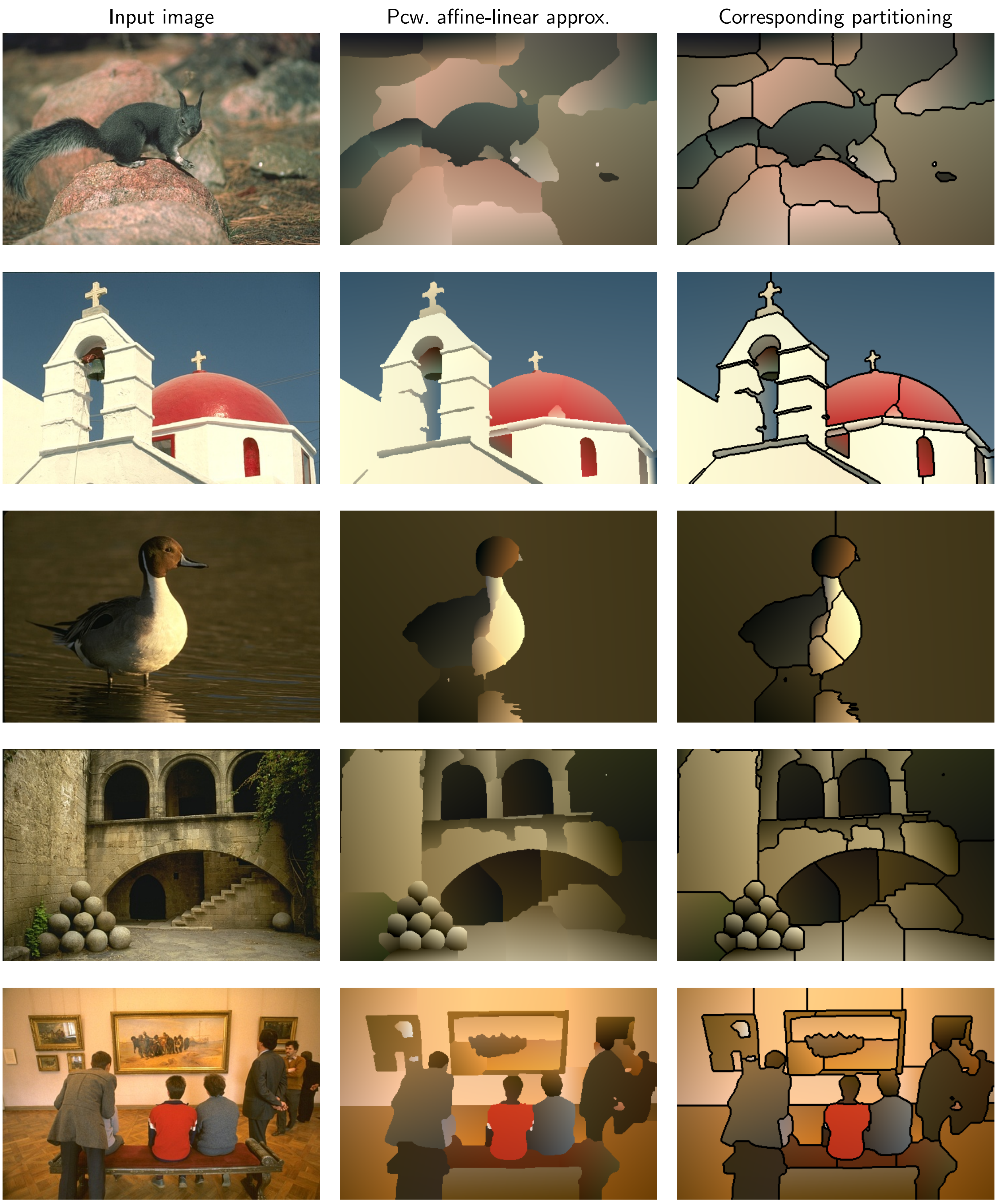


Figure 14: Example results for the Berkeley data set. *From left to right:* Input image, piecewise affine-linear approximation and the corresponding partitioning. The model parameter is $\gamma = 0.5$.

5 Conclusion

In this paper, we detailed the PALMS image partitioning which implements an algorithm for the piecewise affine-linear Mumford-Shah model, which is a variational approach to image partitioning. The algorithm is based on the authors' ADMM approach in [13]. In particular, we gave an in-depth description of the exact and efficient solver for the arising subproblems. These subproblems correspond to univariate partitioning problems and the solver is based on dynamic programming. We accelerated these computations by incorporating a pruning strategy. We illustrated the advantages of using the multichannel approach compared to its channelwise counterpart. Furthermore, we discussed the effect of different choices of penalty progressions. Likewise, we illustrated the benefits of employing an 8-neighborhood discretization compared with a 4-neighborhood discretization. In particular, despite the fact that twice as many subproblems had to be solved, the runtime increased by only 50 %. The subproblems can be approached in parallel. We studied the speedup when using a parallelized implementation compared with a sequential implementation and observed a speedup by factor 8 for a multi-core CPU with 16 cores. We further compared different choices for the initialization. As a result of these improvements and studies, we obtain a further optimized algorithm. Subsequently, we experimentally tested the image sizes our implementation can efficiently handle and found that images of size 800 × 556 are processed in roughly two minutes. Finally, we compared the algorithm to iterative application of graph cuts using α-expansion moves on the Berkeley segmentation test set. This showed the efficiency of the implementation which yields lower mean energies and faster runtimes.

A Derivation of the ADMM Subproblems

We here derive the jet subproblems (16) from (15). To this end, we will repeatedly use the fact that

$$\sum\nolimits_{i=1}^{N} x_i(p-t_i)^2 = \big(\sum\nolimits_{i=1}^{N} x_i\big)\big(p - \tfrac{\sum_{i=1}^{N} t_i x_i}{\sum_{i=1}^{N} x_i}\big)^2 + C \tag{40}$$

holds for $p, t_1, \ldots, t_N \in \mathbb{R}$ and $x_1, \ldots, x_N > 0$ and a constant C that does not depend on p. Initially, this allows us to rewrite the summands in (15) and we get

$$\begin{aligned}\operatorname*{argmin}_J \Big\{ & \omega_s \gamma \|\nabla_{d_s} J\|_0 + \tfrac{1}{S}\|u-f\|^2 \\ & + \tfrac{(S-s)\mu}{2}\Big\| u - \frac{\sum_{t=s+1}^{S}(u^t - \frac{\lambda^{s,t}}{\mu})}{(S-s)}\Big\|^2 + \tfrac{(s-1)\mu}{2}\Big\| u - \frac{\sum_{r=1}^{s-1}(u^r + \frac{\lambda^{r,s}}{\mu})}{(s-1)}\Big\|^2 \\ & + \tfrac{(S-s)\nu}{2}\Big\| a - \frac{\sum_{t=s+1}^{S}(a^t - \frac{\tau^{s,t}}{\nu})}{(S-s)}\Big\|^2 + \tfrac{(s-1)\nu}{2}\Big\| a - \frac{\sum_{r=1}^{s-1}(a^r + \frac{\tau^{r,s}}{\nu})}{(s-1)}\Big\|^2 \\ & + \tfrac{(S-s)\nu}{2}\Big\| b - \frac{\sum_{t=s+1}^{S}(b^t - \frac{\rho^{s,t}}{\nu})}{(S-s)}\Big\|^2 + \tfrac{(s-1)\nu}{2}\Big\| b - \frac{\sum_{r=1}^{s-1}(b^r + \frac{\rho^{r,s}}{\nu})}{(s-1)}\Big\|^2 \Big\}.\end{aligned} \tag{41}$$

We dropped terms that do not depend on J. For readability we use abbreviations for the sums in (41) which are given by

$$\begin{aligned} \Lambda &= \sum\nolimits_{t=s+1}^{S}(u^t - \lambda^{s,t}/\mu), & \Psi &= \sum\nolimits_{t=s+1}^{S}(a^t - \tau^{s,t}/\nu), & \Delta &= \sum\nolimits_{t=s+1}^{S}(b^t - \rho^{s,t}/\nu), \\ \Gamma &= \sum\nolimits_{r=1}^{s-1}(u^r + \lambda^{r,s}/\mu), & \Phi &= \sum\nolimits_{r=1}^{s-1}(a^r + \tau^{r,s}/\nu), & \Theta &= \sum\nolimits_{r=1}^{s-1}(b^r + \rho^{r,s}/\nu). \end{aligned}$$

After applying (40) to all but the first line of (41) we obtain

$$\operatorname*{argmin}_{J} \omega_s\gamma\|\nabla_{d_s}J\|_0 + \tfrac{1}{S}\|u-f\|^2 + \tfrac{(S-1)\mu}{2}\left\|u-\tfrac{\Lambda+\Gamma}{S-1}\right\|^2 + \tfrac{(S-1)\nu}{2}\left\|a-\tfrac{\Psi+\Phi}{S-1}\right\|^2 + \tfrac{(S-1)\nu}{2}\left\|b-\tfrac{\Delta+\Theta}{S-1}\right\|^2. \tag{42}$$

A final application of (40) to both remaining terms depending on u in (42) leads to

$$\operatorname*{argmin}_{J} \omega_s\gamma\|\nabla_{d_s}J\|_0 + \tfrac{2+\mu S(S-1)}{2S}\left\|u-\tfrac{2f+\mu S\left(\Lambda+\Gamma\right)}{2+\mu S(S-1)}\right\|^2 + \tfrac{(S-1)\nu}{2}\left\|a-\tfrac{\Psi+\Phi}{S-1}\right\|^2 + \tfrac{(S-1)\nu}{2}\left\|b-\tfrac{\Delta+\Theta}{S-1}\right\|^2. \tag{43}$$

Finally, multiplying (43) by $\frac{2}{(S-1)\nu}$, we obtain

$$\operatorname*{argmin}_{J} \tfrac{2\omega_s\gamma}{(S-1)\nu}\|\nabla_{d_s}J\|_0 + \tfrac{2+\mu S(S-1)}{S(S-1)\nu}\left\|u-\tfrac{2f+\mu S\left(\Lambda+\Gamma\right)}{2+\mu S(S-1)}\right\|^2 + \left\|a-\tfrac{\Psi+\Phi}{S-1}\right\|^2 + \left\|b-\tfrac{\Delta+\Theta}{S-1}\right\|^2 \tag{44}$$

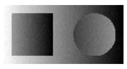
which yields (16).

Acknowledgments

This work was supported by the German Research Foundation (DFG STO1126/2-1 & WE5886/4-1).

Image Credits

by Berkeley [1].


provided by the authors.

Tuxyso/Wikimedia Commons/CC BY-SA 3.0.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, *Contour detection and hierarchical image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 33 (2011), pp. 898–916. https://doi.org/10.1109/TPAMI.2010.161.

[2] R. Bellman, *The theory of dynamic programming*, Bulletin of the American Mathematical Society, 60 (1954), pp. 503–516.

[3] Y. Boykov and V. Kolmogorov, *Computing geodesics and minimal surfaces via graph cuts*, in IEEE International Conference on Computer Vision, 2003, pp. 26–33. https://doi.org/10.1109/ICCV.2003.1238310.

[4] Y. Boykov and V. Kolmogorov, *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 1124–1137. https://doi.org/10.1109/TPAMI.2004.60.

[5] Y. Boykov, O. Veksler, and R. Zabih, *Fast approximate energy minimization via graph cuts*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 23 (2001), pp. 1222–1239. https://doi.org/10.1109/34.969114.

[6] A. Chambolle, *Finite-differences discretizations of the Mumford-Shah functional*, ESAIM: Mathematical Modelling and Numerical Analysis, 33 (1999), pp. 261–288. https://doi.org/10.1051/m2an:1999115.

[7] T.F. Chan and L.A. Vese, *Active contours without edges*, IEEE Transactions on Image Processing, 10 (2001), pp. 266–277. https://doi.org/10.1109/83.902291.

[8] R. Chartrand and B. Wohlberg, *A nonconvex ADMM algorithm for group sparsity with sparse groups*, in IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 6009–6013. https://doi.org/10.1109/ICASSP.2013.6638818.

[9] D. Cremers, M. Rousson, and R. Deriche, *A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape*, International Journal of Computer Vision, 72 (2007), pp. 195–215. https://doi.org/10.1007/s11263-006-8711-1.

[10] F. Friedrich, A. Kempe, V. Liebscher, and G. Winkler, *Complexity penalized M-estimation*, Journal of Computational and Graphical Statistics, 17 (2008), pp. 201–224. https://www.jstor.org/stable/27594299.

[11] K.-S. Fu and J.K. Mui, *A survey on image segmentation*, Pattern Recognition, 13 (1981), pp. 3–16. https://doi.org/10.1016/0031-3203(81)90028-5.

[12] R.M. Haralick and L.G. Shapiro, *Image segmentation techniques*, Computer Vision, Graphics, and Image Processing, 29 (1985), pp. 100–132. https://doi.org/10.1016/S0734-189X(85)90153-7.

[13] L. Kiefer, M. Storath, and A. Weinmann, *An efficient algorithm for the piecewise affine-linear Mumford-Shah model based on a Taylor jet splitting*, IEEE Transactions on Image Processing, 29 (2020), pp. 921–933. https://doi.org/10.1109/TIP.2019.2937040.

[14] V. Kolmogorov and R. Zabih, *What energy functions can be minimized via graph cuts?*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 147–159. https://doi.org/10.1109/TPAMI.2004.1262177.

[15] J-M. Morel and S. Solimini, *Variational methods in image segmentation: with seven image processing experiments*, vol. 14, Springer Science & Business Media, 2012. ISBN 978-1-4684-0567-5.

[16] R. Ramlau and W. Ring, *A Mumford–Shah level-set approach for the inversion and segmentation of X-ray tomography data*, Journal of Computational Physics, 221 (2007), pp. 539–557. http://dx.doi.org/10.3934/ipi.2011.5.137.

[17] C. Sanderson and R. Curtin, *Armadillo: A template-based C++ library for linear algebra*, Journal of Open Source Software, 1 (2016), p. 26. https://doi.org/10.21105/joss.00026.

[18] ——, *Practical sparse matrices in C++ with hybrid storage and template-based expression optimisation*, Mathematical and Computational Applications, 24 (2019), p. 70. https://doi.org/10.3390/mca24030070.

[19] M. Storath, L. Kiefer, and A. Weinmann, *Smoothing for signals with discontinuities using higher order Mumford–Shah models*, Numerische Mathematik, 143 (2019), pp. 423–460. https://doi.org/10.1007/s00211-019-01052-8.

[20] M. Storath and A. Weinmann, *Fast partitioning of vector-valued images*, SIAM Journal on Imaging Sciences, 7 (2014), pp. 1826–1852. https://doi.org/10.1137/130950367.

[21] M. Storath, A. Weinmann, J. Frikel, and M. Unser, *Joint image reconstruction and segmentation using the Potts model*, Inverse Problems, 31 (2015), p. 025003. https://doi.org/10.1088%2F0266-5611%2F31%2F2%2F025003.

[22] O. Veksler, *Efficient graph-based energy minimization methods in computer vision*, PhD thesis, Cornell University, 1999.

[23] Y. Wang, W. Yin, and J. Zeng, *Global convergence of ADMM in nonconvex nonsmooth optimization*, Journal of Scientific Computing, 78 (2019), pp. 29–63. https://doi.org/10.1007/s10915-018-0757-z.

[24] G. Winkler and V. Liebscher, *Smoothers for discontinuous signals*, Journal of Nonparametric Statistics, 14 (2002), pp. 203–222. https://doi.org/10.1080/10485250211388.

[25] Z. Xu, S. De, M. Figueiredo, C. Studer, and T. Goldstein, *An empirical study of ADMM for nonconvex problems*, arXiv:1612.03349, (2016). https://arxiv.org/abs/1612.03349.