Efficient Multichannel Image Partitioning: Theory and Application

Tetyana Ivanovska

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in

Computer Science

Dissertation Committee:
Prof. Dr.-Ing. Lars Linsen (supervisor)
Prof. Dr.-Ing. Horst K. Hahn
Prof. Dr. Jos B.T.M. Roerdink

October 1, 2009
School of Engineering and Science
Jacobs University, Bremen, Germany
To A.C.K.
Acknowledgements

I would like to thank all people who have helped and inspired me during my doctoral study.

First, I sincerely thank my advisor Dr. Lars Linsen, for his continuous support, encouragement and inspiration. His perpetual energy and enthusiasm in research motivated all his advisees, including me. In addition, he was always accessible and willing to help his students with their research. As a result, research life became smooth and rewarding for me.

I am also immensely indebted to my second advisor, Dr. Horst K. Hahn. He opened the exciting and challenging world of medical image processing for me, and taught me how to approach a research problem and express my ideas.

I wish to thank Dr. Jos B.T.M. Roerdink who kindly agreed upon reviewing my work.

I am grateful to Andrea Schenk and André Homeyer from Fraunhofer MEVIS for our numerous and fruitful discussions, for sharing ideas, for friendly environment and collaborative support.

A special gratitude goes also to Dr. Uta Dahmen for her hospitality during my visits to Essen, and for making the world of histology so close and interesting to me.

I want to thank Alexander Köhn from Fraunhofer MEVIS for giving me insights and useful comments on GPU programming.

The words of gratitude go to Anja Hennemuth for introducing me to the project on segmentation of atherosclerotic plaques and being helpful and friendly.

I acknowledge Wolf Spindler for organizing such useful and entertaining “MeVisLab School” days and teaching me how to make the first steps in programming under MeVisLab platform.
I would like to thank MeVisLab-Forum administrators and participants for answering my numerous and sometimes misleading questions.

Many thanks to my colleagues and groupmates, especially, to Paul Rosenthal and Steffen Hauth, for being friendly, helping and collaborative.

I would like to show my gratitude to Jacobs University in Bremen for facilitating this Ph.D. project and offering so nice study conditions.

Finally, I want to thank all my friends and family and, especially, my dearest Sasha.
Abstract

Segmentation is an important tool in image processing. Some of its practical applications are medical and biomedical imaging, traffic control systems, satellite imaging and many others. The principal goal of the segmentation process is to separate the structures of interest from the background and from each other.

The focus of the current work is to explore and develop a range of efficient segmentation approaches for multichannel image data. As main examples of application, we use synthetic color (RGB) images as well as medical (cryosections, MRI) and biomedical (histological) image data.

In our studies the objective is to divide the entire image into subregions. Such a procedure is often called partitioning. By partitioning we basically understand the separation of the whole image domain into non-overlapping regions that are homogeneous with respect to one or more characteristics or features.

We start from a generalization of grayscale segmentation techniques to multichannel data by converting it to a scalar field in a preprocessing step. We propose a procedure that converts color to scalar data while preserving as many salient features as possible which is important for segmentation purposes. We apply standard segmentation techniques to the converted data and discuss the advantages and limitations of such conversion.

Apart from applying the standard segmentation techniques for grayscale data to color data, we propose an approach that allows for the direct segmentation and surface extraction mechanisms for color data. Our approach consists of several automated steps and an intuitive mechanism for user-guided feature selection. We demonstrate the obtained results and discuss the approach’s limitations.

For certain cases operating only in the feature space is not fully adequate, and more sophisticated algorithms which operate both in feature and object space are preferable. We analyze several global multichannel image partitioning approaches using minimum description length (MDL). We review existing single-channel approaches, analyze and discuss their behavior and parameters from a theoretical and an experimental aspect, and develop a novel extension of the approach to multichannel image partitioning.

In certain highly specialized segmentation tasks, prior knowledge, e. g., feature and shape characteristics, about the areas to be extracted might be useful. For such problems a relatively simple partitioning can be used in combination with several
post-processing steps. We apply such approach for evaluation of certain cell types, namely, hepatocytes in the histological images of rat livers. A processing pipeline for automatic counting of hepatocytes from images of histological sections is presented. We discuss the results and limitations of the individual processing steps as well as of the overall automatic quantification approach.
Contents

1 Introduction 1

2 Color-to-Scalar Conversion for Partitioning Purposes 8
   2.1 Conversion Procedure ............................................. 10
      2.1.1 Color Space Conversion ..................................... 11
      2.1.2 Color2Gray with Prior Clustering ............................. 12
   2.2 Results and Discussion ............................................. 16

3 Semi-automated Feature Space Partitioning 26
   3.1 General Approach ................................................. 28
   3.2 Interactive Feature-space Segmentation ........................... 30
      3.2.1 Color clustering .............................................. 30
      3.2.2 User Interaction for Cluster Selection ....................... 34
      3.2.3 Feature-space Segmentation ................................ 36
   3.3 Object-space Segmentation and Surface Extraction ............... 37
   3.4 Results and Discussion ............................................. 40

4 Combined Feature-Object Space Partitioning 47
   4.1 Theoretical background ............................................ 49
      4.1.1 Mumford-Shah Functional .................................... 49
      4.1.2 Maximum A Posteriori Probability - Markov Random Fields Framework ........................................ 51
      4.1.3 Minimum Description Length ................................ 54
   4.2 Related Work ..................................................... 56
## Contents

4.3 MDL-Based Energy Functional ........................................... 57
4.4 Energy Minimization ..................................................... 61
   4.4.1 Expansion move Graph cut algorithm ............................. 61
   4.4.2 GNC-like Relaxation .............................................. 62
   4.4.3 Implementation Details: GPU-based version ................. 69
4.5 Results and Discussion .................................................. 79
   4.5.1 Minimization Scheme comparison ................................ 79
   4.5.2 Algorithm Behavior ............................................. 82
   4.5.3 Parameter Selection ............................................. 85
   4.5.4 General Results ............................................... 90

5 Partitioning With Prior Assumptions ................................. 100
   5.1 Medical Background ............................................... 100
      5.1.1 Liver Anatomy Basics ....................................... 100
      5.1.2 Hepatocyte Quantification .................................... 103
      5.1.3 Current Study Description .................................... 104
      5.1.4 Material ..................................................... 105
   5.2 Related Work ....................................................... 106
   5.3 Processing Pipeline ................................................ 108
      5.3.1 Smoothing .................................................... 109
      5.3.2 Thresholding ................................................ 112
      5.3.3 Connected Components Processing ......................... 116
      5.3.4 Hepatocyte detection ....................................... 117
   5.4 Software GUI and functionality description .................... 118
      5.4.1 Hepatocyte Quantification .................................... 118
      5.4.2 Recognition Rates Detection ................................ 120
   5.5 Results and Discussion ............................................. 120
      5.5.1 Smoothing filter comparison ................................ 122
      5.5.2 Thresholding analysis ....................................... 129
      5.5.3 Vein exclusion analysis ...................................... 132
## Contents

5.5.4 Hepatocyte detection analysis ........................................ 133
5.5.5 General analysis .................................................. 134

6 Conclusions .......................................................... 139

References .......................................................... 142
Chapter 1

Introduction

*Science cannot solve the ultimate mystery of nature. And that is because, in the last analysis, we ourselves are a part of the mystery that we are trying to solve.*

Max Planck

Image segmentation is one of the most important topics in image processing and the first step in visualization pipeline. The main goal of segmentation is separation of structures of interest from the background and from each other.

Let $C \subset \mathbb{R}^n$ denote the intensity domain, where $n$ is the number of image channels, and let $\Omega \subset \mathbb{R}^m$ be an image domain, where $m$ is the number of image dimensions. Then, $I : \Omega \rightarrow C$ denotes the image function. The segmentation problem consists in computing a decomposition of the image domain into a finite set of non-overlapping regions $R_i, i = 1, \ldots, S$, with

$$I = \bigcup_{i=1}^{S} R_i, \quad \text{and} \quad R_i \cap R_j = \emptyset, \quad i \neq j. \quad (1.1)$$

In the last decades, numerous segmentation algorithms have been developed in the field of image processing (see surveys in [HS85, PP93, RTT95, LM01, F+02]) for different practical applications. Until very recently, attention has been focused on segmentation of gray-level images since these have been the only kind of visual information that acquisition devices were able to take and computer resources to handle. Nowadays, color imagery has definitely supplanted monochromatic information and computation power is no longer a limitation in processing large volumes of data. The attention has accordingly been focused in recent years on algorithms for segmentation of color images and various techniques, often borrowed from the
Chapter 1. Introduction

background of gray-level image segmentation, have been proposed. We study several directions in general-purpose multichannel image segmentation and apply our methods to multichannel medical and biomedical data.

There exist many ways to classify the segmentation methods. We use the following main features: the level of user interaction, the type of processed data, and the type of extracted result.

If segmentation is performed independently by a computer, it is called “automatic” or “unsupervised” process, if the user input is required, then a segmentation procedure is “semi-automatic”. When the expert needs to define the segments only with computer assistance, this process is called “manual segmentation”. Manual segmentation is usually used to validate the results of semi-automatic or automatic procedure with the expert-detected ground truth, and appears to be very tedious and time-consuming, if much data is processed.

In terms of the type of processed data, segmentation techniques can be classified into single channel and multichannel categories. Traditionally, most segmentation techniques use one type of image (one intensity channel). For example, such medical imaging methods as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), Positron Emission Tomography (PET) produce single channel data as output. Numerous segmentation techniques have been developed for such data. However, sometimes much more information can be extracted while processing multichannel image data, for instance, different modalities of MRI (T1 and T2 relaxation times and the proton density). Some scanning techniques, like cryosections, generate multichannel data in form of RGB color images as their output. For such types of data one channel segmentation methods are generalized or multispectral segmentation methods are developed.

In terms of “extracted result”, we would like to classify segmentation methods into two broad and in certain cases overlapping categories.

The first category can be defined as “region separation” methods, where one or several specific structures must be extracted from the whole image, for example, anatomical structures, such as bones, muscles, blood vessels, or pathological regions, such as tumors and lesions [B+97, B+00].

One of the basic methods from this category is the region growing approach, that expands regions from initial seed points until a stopping criterion is met [AB94]. The stopping criteria can be formulated differently. For instance, the simplest way is to define the maximum difference in color space between the average region color and
Chapter 1. Introduction

candidate pixels (voxels) to be added to it. In more sophisticated approaches, for example, in the region competition [ZY96] method, an energy functional is formulated and regions are merged until the energy value stops decreasing.

There exist implementations of the region growing approach both for grayscale and multichannel data [SHB08]. Numerous methods that specifically address region-growing segmentation of color images are presented by Moghaddamzadeh and Bourbakis [MB97], Shih [SC05], Schettini [Sch93], Tremeau and Borel [TB97], Ikonomakis et al. [IPV98].

Two deformable contour/surface models, namely, parametric models (active contours) and geometric deformable models (level sets) also belong to the first category. Both approaches are based on energy minimization. The active contour model, or snake, is defined as an energy-minimizing spline. The snake’s energy depends on the shape and location within the image [KWT88a]. Active contours methods have been introduced for grayscale data. In the literature, some methods for color data are presented [NJ96]. Classical snakes can not handle topological changes and the approaches that overcome these difficulties have been developed. For example, an active contour-like method based on charged particle model (CPM) that can handle topological changes and is not that sensitive to the initialization step has been presented [JWR04]. Another direction is an extension of the classical snakes model based on level sets [CV01, CSV00, VC02]. Geometric deformable models perform by starting with an initial curve and evolving its shape. During the evolution process, curvature deformation is used and the speed of curve evolution is locally dependent on the image data [OP03]. The traditional level set methods use single-channel data for processing. However, their extensions for color data have been proposed recently [JJFZm08, VC02]. The main difference of the level set model from the snakes is that it is using only geometric computation, independent of any parametrization [SHB08].

The second category (we call it “partitioning”, “classification” or “clustering”) of the methods classifies segment membership of an image element (pixel or voxel) according to its or its closest neighbors’ color. As a result, the whole image is divided into a set of “classes”. Though in medical imaging algorithms that belong to the first category are more often required, the partitioning of the whole image is also of relevance. For example, tissue classification algorithms have been useful in the segmentation of MR brain or atherosclerotic plaque images, where the tissue contrast allows for volumetric quantification of the different tissue parts, e. g., white matter, gray matter, and the cerebrospinal fluid spaces of the brain [P⁺00a].
Histogram thresholding is a common segmentation method for gray-level images [LM01] that belongs to this category. In this technique a threshold is selected, and an image is divided into groups of pixels having values less than the threshold and groups of pixels with values greater or equal to the threshold. There are several thresholding methods: global methods based on gray-level histograms, global methods based on local properties, local threshold selection, and dynamic thresholding [Gla93, SHB08].

In the case of color images, histogram thresholding is more complicated, as there either histograms of each channel are processed or an $ND$-histogram is partitioned. For example, Celenk and de Haag propose to compute thresholds for the histogram of each channel independently and, then, to combine the result [CH98]. Several approaches have in common the partition of histograms in HSV (hue, saturation, intensity) color space into chromatic and achromatic regions [TC94, PA94].

Watershed segmentation is a standard approach utilizing grayscale image morphology. Image data is interpreted as a topographic surface where the gradient image gray-levels represent altitudes, and region edges correspond to high watersheds and low-gradient region interiors correspond to catchment basins. Catchment basins of the topographic surface are homogeneous in the sense that all pixels belonging to the same catchment basin are connected with the basin’s region of minimum altitude (gray-level) by a simple path of pixels that have monotonically decreasing altitude along the path. Such catchment basins then represent the regions of the segmented image [VS91]. In medical imaging, watershed algorithms have been successfully applied to segmentation of neuroanatomic structures and bone segmentation [HP03], cell segmentation [K+08] etc. Watershed approaches for multiscale data are addressed by Vanhamel et al. [V+06], Lezoray and Cardot [LC02], Shafarenko et al. [S+97b].

Clustering algorithms achieve region segmentation by partitioning the image into sets or clusters of pixels that have strong similarity in the feature space [DHS00]. The basic operation is to examine each pixel and assign it to the cluster that best represents the value of its characteristic vector of features of interest. Such methods as K-means [Sch97, Mac67], ISODATA [MMN+06], Gaussian clustering via the Expectation Maximization (EM) [Mac02], fuzzy clustering (e. g., fuzzy C-means [Bez81]). Unlike K-means and ISODATA algorithms, the latter two can produce “soft” segmentations, that can better model partial volume effect (PVE), which occurs due to limited spatial resolution, when information of at least two structures is mixed.
within a single image element. These methods deal in a uniform manner with data with any number of channels.

In order to make the approaches more robust to noise, a smoothness constraint is typically employed within the segmentation model. In statistical classifiers, modeling of spatial interactions between neighbouring or nearby pixels is frequently used. Popular directions are Markov Random Field (MRF) and Minimum Description Length (MDL) techniques [Li01], which are based on energy minimization procedure. Originally, these methods have been applied to single-channel data, however, nowadays, methods for color data are also referred, e.g., in [Muk02, CH92, TRS97, LHZ04, LK06, BVZ98, FL97, K+94].

The above-mentioned methods are considered to be general-purpose fundamental approaches, which can be applied to any images. Their specializations, i.e., special extensions, tricks and method combinations are used in many application fields when some prior information about the image appearance and structures of interest is available. The medical applications include, for instance, surgical planning, measurement of tumor volume, lesion quantification, automated classification of blood cells, tumor detection, etc. [Ban08].

Despite the great variety of the image segmentation techniques, there is no standard solution that can produce perfect results for all applications.

In this work, we focus on multichannel image segmentation with different levels of user interaction. We mainly use color images as examples, namely, artificially created RGB images, images of cryosections, images combined from several MRI modalities, and images of histological sections. However, most of the concepts described here can be applied to multichannel image data with any number of channels.

The main objective of this work is to find efficient partitioning algorithms for multichannel data and define the area of their applicability, i.e., to identify, for which problems and data certain methods are more preferable comparing to the others. To fulfil it, we investigate different tasks, namely,

- we explore the general purpose multispectral segmentation methods based on partitioning, compare several existing single-channel techniques, develop extensions for the multichannel case, and

- we develop an efficient solution for extraction and quantification of a certain cell type from the histological sections of a rat liver.
In the scope of first task, we would like to adapt the existing single channel segmentation techniques to multichannel (color) data via adequate color-to-scalar conversion. For this purpose, we explore existing conversion techniques and extend the recently presented non-linear conversion scheme [IL07a]. Our approach is comprehensively described in Chapter 2. The applied color-to-scalar conversion method preserves salient features which is crucial for the subsequent segmentation. The method is automatic and does not require user interaction. Thereafter, the converted data can be segmented with any single channel technique. Its selection must be justified by the correspondent segmentation task. To optimize the computations, we reduce the feature space, by applying a prior clustering step. However, it becomes responsible for the outcome and if certain important features have been misclassified in the clustering step, any segmentation technique will not be able to detect them appropriately even after a conversion preserving salient colors.

Clusterization separates data directly in the feature space, so, we have decided to investigate clustering techniques operating in the feature space and use this information for direct surface extraction. While dealing with 3D color imaging data, the user needs a convenient and fast way of extracting a ROI for further visualization. We present a user-friendly tool for segmentation and feature extraction from color volume data [IL07b]. Our method consists of several automated steps and an intuitive mechanism for user-guided feature selection. It is presented in Chapter 3.

For many cases operating only in the feature space is not fully adequate, and more sophisticated algorithms which operate both in feature and object space are preferable. We explore image partitioning methods that incorporate also spatial distribution during segmentation, and focus on the global approaches that are based on energy minimization and do not require from the user any knowledge about the number of clusters. We analyze several multichannel image partitioning approaches using minimum description length (MDL), review existing single-channel approaches, analyze and discuss their behavior and parameters from a theoretical and an experimental aspect, and develop a novel extension of the approach to multichannel image partitioning [IHL09]. We discuss several minimization techniques and develop a strategy for processing real medical datasets. The proposed method is stable to noise and allows for simultaneous denoising and partitioning the image into a set of non-overlapping regions. Our observations are described in Chapter 4.

When the prior assumptions about the color, shape, and size of the regions to be extracted are available, a relatively simple segmentation technique combined with
several postprocessing steps can lead to pleasing results. In the scope of the second task, the specialized application problem has been solved, see Chapter 5. We propose a pipeline for automatic hepatocyte quantification from the histological images of a proliferating rat liver \cite{ISD08}, \cite{IS09}. We use smoothing as an effective preprocessing step. Different smoothing filters have been compared and the most suitable ones for this task are selected. Our pipeline is robust while processing images with different color characteristics within the applied staining. The proposed approach is fast and gives results with high sensitivity and specificity compared to the expert-detected ground truth.
Chapter 2

Color-to-Scalar Conversion for Partitioning Purposes

These exist many segmentation algorithms operating on 3D scalar fields, that are nowadays considered standard and have shown their efficacy (see surveys in [HS85, PP93, F+02]). Since different approaches are more or less suitable for different segmentation purposes, one would like to have the whole range of all segmentation algorithms generalized to multichannel (color) data. A viable way to achieve this goal is to convert such data into an appropriate scalar field without losing the property of being able to segment distinguishable regions.

We consider the conversion to be optimal for further segmentation purposes, if the conversion method can preserve salient features (i.e., as many features as possible), so that the subsequent segmentation (automatic or semi-automatic) would be able to find the boundaries. Such a method should take into account not only local color differences, but also include the global information about the colors, so that the converted grayscale output would correspond to user expectations. This is important for semi-automatic segmentation, where the user is involved. Moreover, the global information is important, as the method must not map different colors located in different positions in the image to the same gray values. For example, in Figure 2.1 a test synthetic image is shown. This image consists of two rows of isoluminant colors. The upper row represents the colors with luminance equal 80 (which means quite light) and the lower row colors have the luminance equal 40 (pretty dark). Each column of colors is an isochrominant pair. An optimal conversion method should distinguish between all colors and preserve the global difference (light-dark).

One of the simplest and standard color-to-grayscale approaches is to treat each color
channel independently. Such an approach is often used in practical applications, where the structures to be extracted are clearly presented in a separate channel. Another similar method would be to mix the channels to get a grayscale output. This method is used for standard conversion in such software packages as GIMP (GNU Image Manipulation Program http://www.gimp.org/). Here, an RGB image is converted to grayscale using the following channel mix expression: $G = 30\%R + 59\%G + 11\%B$. Another standard mapping for a color-to-grayscale transformation is the projection of the colors of a multispectral image to the gray axis in the respective color space. Thus, color is mapped to its luminance. Such a method is inadequate for the pixels with same luminance but different chrominance. Several methods have been proposed for solving the general problem of reducing an $n$-dimensional set of data to $m$ dimensions, where $m < n$. Since all standard color spaces have three dimensions, $n = 3$ and $m = 1$ in our case. Principal Component Analysis (PCA) is one of such methods [Jol02]. PCA can be used for computation of an ellipsoid in color space (principal components). Color values in the image can then be projected on a (luminance) axis defined by the primary axis of this ellipsoid, which represents the axis of the highest variability. The efficacy of this method depends on the color space. However, PCA can also suffer from the problem of projecting colors of different chrominance to the same position on the axis.

Another way of generating grayscale images out of color images is to match local contrast in color images [SW02]. The contrast is regarded as a gradient and then the grayscale is recovered by solving a Poisson equation. This method has difficulties with certain classes of images as the global contrast influence is avoided. Moreover, local approaches do not work for our purposes, as one color may be mapped to different scalar values in different regions.

---

Figure 2.1: Test synthetic image with two isoluminant rows and three isochrominant columns. Optimal conversion method should be able to distinguish all colors.
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

Another recently presented method maintained the proportionality between perceived color difference and perceived luminance difference, but ignores spatial arrangement of pixels [RGW05].

The goals of the listed approaches are indeed diverse, but none of them is targeted towards a subsequent segmentation step. Our approach instead aims for the necessary properties, which are to distinguish all significant colors and to maintain luminance order for colors with comparable chrominance values. We use a generalization [IL07a] of the recently developed two-dimensional Color2Gray algorithm [GOTG05] coupled with a clustering procedure, similar to proposed in [RCWN06]. We extend the idea in the sense that we use a subsequent segmentation procedure and analyze and compare different clustering methods [IL07a], whereas only popularity algorithm is applied in [RCWN06]. Our approach is comprehensively described in Section 2.1.

To prove the concept we apply standard segmentation methods, such as thresholding and region growing. We present our results and discuss the limitations of our approach in Section 2.2.

2.1 Conversion Procedure

We present a procedure for converting color data to a scalar field in a way that is amenable for subsequent segmentation of the volume. Any segmentation technique may be applied to the resulting scalar field.

Our conversion approach consists of three main steps. Assuming that the color data is given in form of RGB data, we first convert the RGB values to a color representation in the $L^*a^*b^*$ color space. Secondly, we apply a clustering (quantization) step that reduces the number of used colors in the given color data from their original number to a typically much smaller number of colors in the same color space. The reduced amount of colors is chosen with respect to the number of distinguishable output values in the to be generated scalar field. When considering a scalar field that allows us to store 1 byte information per sample point, we would set the reduced number of colors for clustering to 256. The quantization step must assure that all important colors can still be distinguished after quantization, so that the result would not perturb the user perception. The quantization step also generates a unique mapping of each color of the original color set to one color of the reduced color set. Finally, the reduced set of colors is mapped to scalar values in a way that luminance order is preserved for colors with similar chrominance values.
2.1.1 Color Space Conversion

Operating in RGB space is inadequate for our purposes, since it does not distinguish between luminance and chrominance. Several color models such as HLS or HSV have this property and are widely used for this particular reason. However, we decided to use the L*a*b* color space, because its Euclidean norm closely correspond to perceptual dissimilarity [Pas03]. CIE L*a*b* is the most complete color model used conventionally to describe all the colors visible to the human eye. It was developed for this specific purpose by the International Commission on Illumination or Commission Internationale d’Eclairage (CIE). The three parameters in the model represent the luminance \( L \) of the color, where \( L = 0 \) yields black and \( L = 100 \) indicates white, its position \( a \) between red and green, where negative values indicate green while positive values indicate red, and its position \( b \) between yellow and blue, where negative values indicate blue and positive values indicate yellow. The L*a*b* color model has been created to serve as a device-independent, absolute model that can be used as a reference. Therefore, it is crucial to realize that the visual representations of the full spectrum of colors in this model are never accurate. They just help in understanding the concept, but they are inherently inaccurate. However, a useful feature of the model is that the first parameter is extremely intuitive: changing its value is like changing the brightness setting in a TV set. Therefore, only a few representations of some horizontal “slices” in the model are enough to conceptually visualize the whole spectrum, assuming that the luminance would be represented on the vertical axis. Figure 2.2 shows three such slices for \( L = 25 \), \( L = 50 \), and \( L = 75 \).

![Figure 2.2: Slices through the L*a*b* color space for L = 25, L = 50, and L = 75.](image)

Assuming that the original data is given in RGB color space, we need to convert the colors to a representation in the L*a*b* color space. To do so, we first transform the RGB data to the CIE XYZ color space and, afterwards, convert the XYZ values to L*a*b* colors. Note that the matrix of transformation from RGB data to
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

$XYZ$ depends on the chosen $RGB$ standard. We consider the $R709$ $RGB$ standard. Hence, the three channels of the $L^*a^*b^*$ colors are computed by

\[
L^* = \begin{cases} 
116 \cdot \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} - 16, & \text{if } \frac{Y}{Y_n} > 0.008856 \\
903.3 \cdot \frac{Y}{Y_n}, & \text{otherwise}
\end{cases},
\]

\[
a^* = 500 \cdot \left( \left( \frac{X}{X_n} \right)^{\frac{1}{3}} - \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} \right),
\]

\[
b^* = 200 \cdot \left( \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} - \left( \frac{Z}{Z_n} \right)^{\frac{1}{3}} \right),
\]

where $X_n, Y_n,$ and $Z_n$ are the values of $X, Y,$ and $Z$, respectively, for a specified reference of the white, i.e., illuminant, color, and $X, Y,$ and $Z$ are computed by

\[
\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}.
\]

2.1.2 Color2Gray with Prior Clustering

The most common way of converting $RGB$ images to grayscale in terms of image processing or converting an $RGB$ color data set to a scalar field in terms of visualization, is to operate on the luminance. First, the image colors are converted from $RGB$ to a color space with a luminance channel, for example, to the $L^*a^*b^*$ color space. Then, luminance values are taken as resulting scalar values. More sophisticated techniques for mapping the 3D color space to one particular axis have been developed, but all these methods are ineffective at preserving different colors orthogonal to the chosen axis. Often the axis is oriented close to the luminance axis such that isochromatic colors are projected to similar regions on the axis.

**Color2Gray** The Color2Gray algorithm allows for taking into account both luminance and chrominance differences in a source image and constructing an appropriate grayscale image. It was introduced for the conversion of 2D images, but could be generalized to volume data.

The user can influence the output of the Color2Gray algorithm using three simple and intuitive parameters. The first parameter $\theta$ controls whether chromatic differences are mapped to increases or decreases in luminance value. The second
parameter $\alpha$ determines how much chromatic variation is allowed to change the source luminance value. The third parameter $\mu$ sets the neighbourhood size used for chrominance estimation and luminance gradients.

The color differences between pixels in the color image are expressed as a set of signed scalar values. The differences are measured in the various channels of the $L^*a^*b^*$ color space. Thus, both luminance and chrominance differences are computed. The generation of the output in form of a grayscale version of the image is based on these differences. For each pixel $i$ and each neighbor pixel $j$, the signed distance scalar $\delta_{ij}$ based on luminance and chrominance differences is computed by

$$\delta_{ij}(\alpha, \theta, \mu) = \begin{cases} 
\Delta L_{ij}, & \text{if } \| \Delta L_{ij} \| > \text{crunch}(\| \Delta \tilde{C}_{ij} \|) \\
\text{crunch}(\| \Delta \tilde{C}_{ij} \|), & \text{if } \text{crunch}(\| \Delta \tilde{C}_{ij} \|) \cdot \hat{V}_\theta \geq 0 \\
\text{crunch}(- \| \Delta \tilde{C}_{ij} \|), & \text{otherwise}
\end{cases}$$

where $L_i$ is the luminance of $i$th pixel, $\Delta L_{ij} = L_i - L_j$, $\| \Delta \tilde{C}_{ij} \|$ is the Euclidean norm of the vector $\Delta \tilde{C}_{ij} = (\Delta A_{ij}, \Delta B_{ij})$ with $\Delta A_{ij}$ and $\Delta B_{ij}$ being the differences between pixels $i$ and $j$ in the chrominance channels $a^*$ and $b^*$, respectively, $\hat{V}_\theta = (\cos \theta, \sin \theta)$ is a normalized vector defined by $\theta$, and $\text{crunch}(y) = \alpha \cdot \tanh \left( \frac{y}{\alpha} \right)$.

Given a set of signed differences $\delta_{ij}$ for the values stored at pixels $i, j$ of a totally ordered set $S_\mu$. A scalar field $x$ is computed such that $x$ minimizes a target function $f(x)$. The target function $f(x)$ is given by

$$f(x) = \sum_{(i,j) \in S_\mu} ((x_i - x_j) - \delta_{ij})^2.$$ 

Hence, the minimization problem can be written in the form of

$$\min(f(x)) = \min \left( \frac{1}{2} \sum_{(i,j) \in S_\mu} (x_i - x_j - \delta_{ij})^2 \right). \quad (2.1)$$

This is a least-squares problem of the form

$$\min \left( \frac{1}{2} (Ax - b)^T (Ax - b) \right),$$

with $A$ being a $N \times N$ matrix, where $N$ is the number of pixels.

This expression can be rearranged to

$$\min \left( \frac{1}{2} x^T A^T Ax - (A^T b)^T x + \frac{1}{2} b^T b \right).$$
This equation is quadratic with a symmetric, positive semi-definite Hessian. Therefore, minimizing it is equivalent to satisfying the linear equation:

\[ A^T Ax = A^T b. \]

Deriving the terms \( d_k \) on the right-hand side of the equation, we obtain

\[ d_k = \left[ A^T b \right]_k = \sum_{j \geq k} \delta_{kj} - \sum_{i < k} \delta_{ik}, \]

where \( k, i, j = 1, \ldots, N \). Because of the regular form of the Hessian, \( A^T Ax = A^T b \) expands to

\[ (N - 1) \cdot x_k - \sum_{l \neq k} x_l = d_k. \]

For any two indices \( i \) and \( j \), we obtain

\[ d_i - d_j = ((N - 1) \cdot x_i - x_j) - ((N - 1) \cdot x_j - x_i) \]

leading to

\[ x_i = \frac{d_i - d_j + N \cdot x_j}{N}. \tag{2.2} \]

For any \( x_i = c \) there is exactly one solution to Equation 2.1, which may be obtained by taking any known minimal vector \( x' \), and shifting all of its elements by \( x_i - x'_i \).

So, the problem can be solved by setting \( x_0 = 0 \) and getting all other \( x_i \) from Equation 2.2. Then, the found grayscale values \( x \) are shifted to be as close to the source luminances as possible:

\[ x_i = x_i - \frac{\sum_{0}^{N-1} (x_i - l_i)}{N}, \]

where \( x \) is the found scalar field, \( l \) is the luminance field, \( N \) is the number of pixels in the image.

**Color2Gray on Clusters** The bottleneck of the algorithm is the calculation of the coefficients \( d_k \). The cost of the calculations is \( O(N^6) \) for a \( N \times N \times N \) volume image. Using local variants of the algorithm by adjusting parameter \( \mu \) is inappropriate for our purposes, as one color value should always be assigned to the same grayscale value, which is only assured by using a global version of the algorithm. Gooch et al. [GOTG05] have shown that the conversion with small neighborhood can lead to inadequate conversion results (see Figure 2.3).
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

Figure 2.3: Image with isoluminant color transition from blue to gray (left). Conversion result for $\mu = 9$ (middle) contains some artifacts. Conversion result with total neighborhood (right) preserves the horizontal fade with respect to the background.

However, if we do the prior clusterization of the image, and calculate the coefficients $d_k$ on the clustered data, the calculations can be reduced dramatically. After the quantization of the image, we are left with arrays of length $K$, where $K$ is the number of generated clusters. Any clustering method can be used for this purpose. We have investigated the existing methods and invented a novel approach for validation, as it is described in Chapter 3. If the number of unique colors in the image is less than the available output values, the clustering step can be skipped. Then, each color defines its own cluster. However, for real data we are dealing with, this case does never occur in practice when using byte-sized output values.

Let the output generated by the cluster be given in form of the color values of each cluster’s center stored in array $Centers$, the number of occurrences of colors from each cluster stored in array $Occurs$, and the indices that assign to each pixel of the color the appropriate cluster stored in array $Indices$.

For each cluster $k = 1, \ldots, K$ we can calculate (on the analogy of $d_l = [A^Tb]_l$, computed for $l = 1, \ldots, N$) $d'_k$ by

$$d'_k = \sum_{j \geq k} \delta_{kj} \cdot Occurs[j] - \sum_{i < k} \delta_{ik} \cdot Occurs[i] \quad (2.3)$$

The $\delta_{kj}$ and $\delta_{ik}$ are computed on the cluster colors only using the information stored in array $Centers$. The cost of this calculation is $O(K^2)$. Thus, it only depends on the typically small number of clusters and is independent of the number of pixels/voxels in the image. In particular, it does not matter, of which dimension the original image is. Our approach scales to arbitrary dimensions, as it only operates on the clusters and their centers.

Finally, for each pixel or voxel $i$ of the original 2D or 3D image, we get the desired value $d_i$ by determining the cluster it has been assigned to and using the respective value $d'_k$. Thus, we retrieve $d_i = D[Indices[i]]$, where $D$ is the array that stores the
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

d_k we computed for all clusters k. These values d_i are used, as before, to compute the x_i from Equation 2.2.

By bringing down the computational costs for the bottleneck computation from \(O(N^6)\) to \(O(K^2)\), we significantly speed up the procedure, which makes it applicable to larger 3D data sets.

2.2 Results and Discussion

To show that the Color2Gray method corresponds the requirements described above, we apply the it to the synthetic image shown in Figure 2.1 and compare to the standard conversion techniques, namely, to the PCA and channel mix (GIMP) conversion. In Figure 2.4 the conversion results are presented. As the rows of colors are isoluminant, the GIMP conversion fails to distinguish between the colors within the row. The PCA has problems with two isoluminant colors in the bottom row and does not preserve the global information, as the colors in the upper row appear to be darker than the colors in the lower one.

![Cropped Figure 2.4 showing conversion results](image)

(a) Channel mix-based conversion has problems in distinguishing isoluminant colors. (b) PCA conversion does not distinguish between two isoluminant colors in the bottom row and reverts the contrast. (c) Color2Gray preserves both color information and spatial arrangement.

Figure 2.4: Conversion results for Figure 2.1. Color2Gray corresponds to the “optimal” conversion requirements.

In the following part, we will show several examples of our method results obtained for real datasets. Since the improvement of our approach over luminance-based conversion models can be documented best by looking at 2D images, we first want to give some examples, where we convert individual slices through a 3D color data set.
Figure 2.5 shows a horizontal slice through the Visible Female data set\(^1\). The data set is obtained by taking cryosections of an entire female human body. We encircle a particular region, where we would like to concentrate on. The region shows a red organ surrounded by a yellowish tissue. The luminance of the surrounding tissue varies, but the chrominance values of the surrounding yellow region are clearly distinguishable from the chrominance value of the red organ.

![Image of a horizontal slice through the Visible Female data set with a red organ encircled](image)

Figure 2.5: Horizontal slice through Visible Female data set. Region of interest is encircled.

Figure 2.6 shows a conversion of the slice to a grayscale image using a luminance-based approach on the left-hand side and our approach on the right-hand side. Using the luminance-based approach, the boundary of the initially red region to the left gets lost. Using our approach, the initially red region is still clearly distinguishable from the surrounding tissue.

For generating the results, we used the Median Cut and K-Means clustering methods with the number of clusters \(K = 256\). As we used byte-size output structures, i.e., the data is mapped into 256 scalar values, we reduce the number of unique colors in our example images to 256 with clustering.

We produced best results for our purposes by choosing the following Color2Gray parameters: \(\alpha = 40\) and \(\theta = \frac{\pi}{4}\) or \(\theta = \frac{3\pi}{2}\), as it was recommended by Gooch et al. [GOTG05]. Parameter \(\mu\) always was chosen such that the entire image is considered as a neighborhood to obtain a global approach.

For generation of Figure 2.7 we applied a segmentation algorithm (thresholding) to the middle and right images of Figure 2.6. In Figure 2.7, the contours are shown in blue. The figure illustrates that the algorithm was not able to segment the initially

---

\(^1\)Data set courtesy of the National Institute of Health.
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

Figure 2.6: Comparison of converted images using luminance-based approach (middle) and our approach (right) when applied to the region of interest (left) of Figure 2.5. Using the luminance-based approach, the boundary of the initially red region to the left gets lost. Using our approach, the initially red region is clearly distinguishable from the surrounding tissue.

red region when applied to the image converted by the channel mix-based approach on the left-hand side. When applied to the image generated with our conversion algorithm, the region can be segmented and the boundary between the red and yellowish regions is appropriately detected.

Figure 2.7: Comparison of segmentation of the converted images from Figure 2.6. For the image generated by a luminance-based conversion, the segmentation of the initially red region fails (left). For the image generated with our conversion algorithm, the region can be separated from the surrounding tissue (right).

In Figure 2.8, we show a part of a slice of a cryosection of a Macaque monkey brain.² The brain slices have been digitized using high-resolution digital photography that allows to scan even very small structures up to single neurons. We pick a region of interest with different tissues. The encircled tissue of interest has a purple color and is surrounded by tissue with brownish color shades.

²Data set courtesy of Edward G. Jones, Center for Neuroscience, University of California, Davis, USA
In Figure 2.9, we show the results when applying a segmentation algorithm (thresholding) to separate the brownish and purple tissues. Segmentation after luminance-based conversion only leaves us with a segmentation of the black spots. The purple regions cannot be segmented. After the conversion with Color2Gray, the tissue of interest has more contrast comparing to the background. However, the final contour separating the tissue has not been built appropriately and some more complicated segmentation technique should be applied after the conversion.

For generation of Figures 2.10 we used a data set of a cancer cell that has been scanned using fluorescence microscopy\(^3\). In fluorescent microscopy, fluorescent dyes are used to treat purified antibodies. An antigen (antibody generator) is a substance that prompts the generation of antibodies and can cause an immune system response. Antibodies are used by the immune system to identify and neutralize foreign objects. Each antibody binds to a specific antigen. The fluorescent dye-antibody complex is then added to a cell, whereupon it binds to the corresponding antigens. Then,

\(^3\)Data set courtesy of the Department of Medicine, University of California, San Diego, USA.
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

Figure 2.9: Comparison of segmentation of the converted images from Figure 2.8. Luminance-based conversion cannot detect the formerly purple region (left), while conversion with our approach gives a result with much higher contrast (right).

Figure 2.10: Slice of a fluorescence microscopy data set of a cancer cell (left). Luminance-based conversion makes it impossible to distinguish the yellow from the green and red parts (middle), while our conversion approach allows for a segmentation of the yellow regions (right).

the cell is placed within a microscope and illuminated. The light causes the dyed antibodies to fluoresce. Since the chosen antibody complex binds only to specific proteins in the cell, the fluorescence acts like a structural marker within a cell. In Figure 2.10, two dyed antibodies (red and green) are added to the cancer cell. Yellow regions indicate portions of the cell that contains both red and green antibodies. We convert a slice shown in Figure 2.10. When converting the original slice (left) using a luminance-based approach (middle), the green, yellow, and red parts cannot be distinguished anymore. When converting the slice using our approach (right), the
colors can be separated. In the Figure 2.10, we segmented the yellow parts with thresholding.

**Limitations** Despite the decent results, presented above, our approach has certain limitations. They concern two aspects, namely, the color characteristics of the data and the algorithm execution speed.

The application of our approach is beneficial, when an image has a certain variety isoluminant colors. However, when most of the colors in the image have the same chrominance, but a different luminance, i.e., gradations of the same color, the result obtained with Color2Gray approach will be similar (with some negligible improvement) to the result, obtained with PCA or simple luminance-based conversion.

In Figure 2.11 conversion results for a slide from a human brain dataset (cryosection)\(^4\) are presented. Here, the Color2Gray approach has the result with a little higher contrast between the background and the tissue comparing to other conversion techniques.

In Figure 2.12 conversion results for a slice from Mouse brain dataset (cryosection)\(^5\) are presented. Due to the lack of the variety in chrominance values in the image, the Color2Gray-based approach gives a similar result, comparing to the result of the PCA method. Visually the result of Color2Gray-based approach gives some higher contrast, comparing to the result of the PCA and luminance-based methods. However, the boundary in the region of interest, shown in Figure 2.13 can be easily distinguished by a thresholding method in all three cases.

Another major issue of the Color2Gray-based approach is its computational costs. To our knowledge, several conversion methods also preserving salient colors, but being much less computationally expensive have been published simultaneously (cf. [Z+08, N+07]). We followed the approach, similar to the one, described in [RCWN06], using the prior clustering. In such a case, the selected clustering scheme is responsible for the quality of the result and the computational costs. If some colors, that belong to different tissues, have been assigned to the same cluster, and the boundary between them has been lost, obviously, the subsequent Color2Gray with any segmentation method will not be able to reconstruct this boundary again.

For instance, in Figure 2.15 the initial image was quantized to 256 colors with K-Means scheme. One can observe that due to the clustering, some blue cells can not be

---

\(^4\)Data set courtesy of A. Toga, University of California, Los Angeles, USA.

\(^5\)Data set courtesy of Center of Neuroscience, University of California, Davis, USA.
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

(a) A slide from a human brain dataset (cryosection).  
(b) Conversion result for Color2Gray-based approach.  
(c) Conversion result for PCA algorithm.  
(d) Conversion result for channel mix approach.

Figure 2.11: For images that have a low chrominance variety Color2Gray results with negligible improvement comparing to standard conversion techniques.

...distinguished from the background anymore. This happens due to the fact, that the smooth color transitions inside the cell regions and on the boundaries of cells consist after the clustering of regions of constant intensities, which do not always coincide with the cell boundaries, as it is shown in Figure 2.14. Increasing the number of clusters can improve the situation. However, it will also increase the computational costs of Color2Gray conversion, and the choice of number of clusters is not obvious for the user.

22
Chapter 2.  Color-to-Scalar Conversion for Partitioning Purposes

(a) A slide from mouse brain dataset (cryosection).

(b) Conversion result for Color2Gray-based approach.

(c) Conversion result for PCA algorithm.

(d) Conversion result for channel mix-based approach.

Figure 2.12: The result of Color2Gray method is similar to the result of the standard conversion methods, when the colors in the image are mostly isochromatic.

Clustering algorithms reduce the number of colors in the image, without changing the number of channels, i.e., clusterization separates the data directly in the feature space. We decided to use this information for partitioning purposes and make the choice of the number of clusters to be more intuitive.
Figure 2.13: Close-up view of the region, where Color2Gray method (left) gives higher contrast comparing to PCA method (center) and luminance-based method (right). The boundary between grey and white tissues can be detected in all three cases.

Figure 2.14: Close-up view of a region from a histological image of a rat liver. The initial image (left), result of conversion without clustering (center), result of conversion with clustering (right) regions are shown. The blue cells whose boundaries have been broken after clustering are enclosed in a red rectangle. These cells have not been detected on the image converted with prior clustering.
Chapter 2. Color-to-Scalar Conversion for Partitioning Purposes

(a) A histological image of a rat liver.  (b) Image quantized to 256 colors with KMeans method.

(c) Conversion result for Color2Gray approach for the initial image.  (d) Conversion result for Color2Gray approach for the quantized image.

(e) Detected liver cells for the Color2Gray image without clustering.  (f) Detected liver cells for the quantized image. Many blue cells have been lost.

Figure 2.15: Clustering method is responsible for the quality of the result.
Chapter 3

Semi-automated Feature Space Partitioning

For many clinical observations it is often needed to have an explicit surface representation of the organs/tumors boundary at hand. It is also of relevance to relate the clinical measurements to higher-resolution ex-vivo measurements such as cryosections. Cryosections typically lead to 3D images of color RGB data. Hence, one would like to have a tool that allows segmentation and surface extraction mechanisms for color data, similar to the techniques existing for scalar data. The selection mechanisms in 3D feature space, i.e., in color space, are obviously much more complex than the analogous ones in 1D feature space are. Still, all selection mechanisms have to be intuitive not only to computer science specialists but in particular to medical people. Complex operations in feature space should not be visible to the user, but should be replaced by some intuitive selection steps. Moreover, user interaction should be reduced to a minimum.

To ease the feature selection step for the user, one can utilize certain automatic segmentation algorithms for color data. Such algorithms cluster similar colors to some kind of “average” color. These automated clustering procedures easily generalize to color volume data. However, when dealing with volume data, occlusion prohibits the visualization of the entire segmentation output. But selecting a region of interest (ROI) allows for careful observations. For ROI extraction further processing steps are required in addition to the clustering, which can chiefly be grouped into a selection mechanism and a subsequent surface extraction and rendering step. The selection step comprises the methods to determine regions of interests or “features”. A final step extracts the features and displays them in a three-dimensional setting.
While the extraction and rendering step can typically be performed fully automated, there needs to be a certain portion of user interaction or, better, user guidance during the feature selection step. The user needs to communicate to the system, what parts of the data set he/she would like to visually explore. Ideally, the user interaction is limited to a minimal amount of intuitive selection choices.

The outcome of the entire feature extraction processing pipeline could be a volume-rendered image or a surface representation. In many applications and especially in medical ones when dealing with cryosection data, one favors surface representations, as one can precisely reveal the geometric structure of the underlying data field distribution. In particular, surface representations allow for quantitative analysis of the segmented objects. To extract surfaces from scalar volume data, intensive research effort has been undertaken in past decades since the pioneer work presenting the Marching Cubes algorithm for isosurface extraction [LC87]. Several improvements have been made leading to robust, fast, and flexible algorithms. To date, many approaches for surface extraction from scalar data exist. Unfortunately, they do not scale to color data in a straightforward manner.

Several prominent approaches in the area of segmentation and surface rendering from color volume data were presented by Schiemann [S+97a] and Pommert [P+00b]. These are the approaches most closely related to our work. In both approaches object classification is based on ellipsoidal regions in $\text{RGB}$ space. Schiemann proposed a semi-automated method for segmentation that was applied in both approaches. The user outlines a typical region of the object and, then, an ellipsoid in feature space is built. The procedure is based on thresholding followed by binary mathematical morphology and connected component labeling. The surface location is computed using the intersection with the classification ellipsoid. The subsequent rendering step, however, is a mere volume rendering approach. No explicit surface representation is generated. We followed a different mechanism of feature selection, operating in $L*a*b*$ color space, which should provide more accuracy. One of the approaches, such as [O+02], deals with the cryosections dataset and uses a hardware assisted volume renderer for visualization. There are also several approaches for direct volume rendering of photographic volumes [KKH02, EMRY02], where the authors present sophisticated transfer function generations, but extracting surfaces from the color volume data has hardly been addressed.

We analyze several clustering algorithms, validate their results with a our own approach based on a genetic algorithm, and present a user-friendly tool for segmentation and surface extraction of color volume data. Our approach consists of several
automated steps and an intuitive mechanism for user-guided feature selection, which is described in detail in Section 3.1. For clustering purposes we make use of and compare three of the most commonly used clusterization methods. Their output is presented to the user in a transparent fashion using a sorted color table. Simple and, thus, intuitive selection mechanisms are provided to select the desired region of interest based on the clusters. The details are given in Section 3.2. The selection by the user is processed in feature space using geometric operations. The final surface extraction step iterates through all the cells in object space, while computing points on the surface using linear interpolation in feature space, see Section 3.3. All these processing steps executed subsequent to the user selection are fully automated, as a user should not be faced with any incomprehensible user interaction in feature space.

### 3.1 General Approach

Our processing pipeline for segmentation and surface extraction from color volume data comprises several automated steps and a user-guided feature selection mechanism. The entire pipeline reaching from reading the RGB color data to the rendering of the extracted surfaces is shown in Figure 3.1.

Since standard distance metrics applied to RGB color space are biased by the selection of the three coordinates of the color space, we start with converting the data into L*a*b* color space, where Euclidean distance is correlated with human perception, see Section 2.1.1 for more details. Afterwards, we apply the clustering algorithms, thus, operating in L*a*b* color space. For clusterization, we investigate the performance of several algorithms, introduce a novel clustering approach based on a genetic algorithm for validation of results, and select three standard clustering algorithms, namely the Median Cut, the K-Means and the C-Means algorithm. Details of our investigations are given in Section 3.2.1. These approaches are fast, yet produce decent results, hence, they are suitable for a user-guided approach. If desired, they could be replaced by any other, possibly more sophisticated and more time-consuming, clustering technique. The output of the clustering algorithm is a set of clusters in L*a*b* color space each associated with a representative color.

The generated clusters are presented to the user by displaying their representative in a color list that is sorted by the frequency of occurrences of the initial colors belonging to the respective cluster. Some high-level selection mechanisms are provided to the user to pick the set of clusters representing the ROI. With the selection of
clusters in the color table the respective 2D contours in object space are visualized overlaid with a 2D color slice. The user can go through all the original 2D color slices. If the user is not satisfied with the results of the clusterization, the result can be refined by simply selecting the representative colors of the desired clusters and applying a reclustering of this reduced set of colors into a larger number of clusters. This process can be repeated until the user gets the desired result. A few selections and iterations typically suffice. The output of the selection procedure is the collection of clusters, which correspond to a point cloud in feature space of all colors that belong to the clusters.

Given the point cloud in feature space, we would like to define a continuous region in feature space that covers all the selected points but excludes all not selected colors. Although this region does not have to be convex, in general, we obtain an approximate representation by computing the convex hull of the point cloud. Dealing with a convex region significantly speeds up computations.
Chapter 3. Semi-automated Feature Space Partitioning

This segmentation is fed into the surface-extraction method to compute a triangular mesh. A marching method is used for surface extraction, where the marching cubes table can be applied to determine the topology of the resulting surface. The points on the surface are computed by linear interpolation in feature space, which involves the computations of intersections of the convex hull with feature-space edges. Special care has to be taken in regions, where the convexity assumption violates the inside/outside-property.

The final step of the pipeline is the surface rendering step, which makes use of the actual color values given at the points on the surface.

The details on the individual processing steps are given in the subsequent sections.

3.2 Interactive Feature-space Segmentation

3.2.1 Color clustering

Exploiting the color distribution in the $L^*a^*b^*$ space for a given data set, we cluster the colors into $K$ regions, where $K$ is the predefined cardinal number of the range of the resulting scalar field. For the quantization step, we first choose the number $K$ of clusters to be generated.

We have investigated the following existing quantization algorithms: static color look-up algorithm using look-up tables [Hec82], popularity algorithm [Hec82], median cut algorithm [Hec82], $K$-Means algorithm [Mac67], and fuzzy $C$-Means algorithm [Bez81]. The details of each algorithm are given below.

**Look-up table Algorithm**  The idea of static color look-up table algorithms is to divide the color cube into $K$ equally thick slices in each dimension. The crossproduct of these color levels can be used as the entities of the color look-up table. A significant drawback of this method are artifacts in form of edges in the resulting image.

**Popularity algorithm**  The main idea of popularity algorithms is to build a colormap by finding the $K$ most frequently occurring colors in the original image. The colors are stored in a histogram. These $K$ most frequently occurring colors are extracted and used as entries in the color table. The image is quantized with respect to that table. The question that remains is how to map the colors that appear in the original image but are not stored in the color table. Each pixel has to be tested
Chapter 3. Semi-automated Feature Space Partitioning

to find the shortest distance to one of the $K$ most frequently used color values. The main drawback of this method is that some important but “unpopular” image colors could be lost.

**Median Cut** The Median Cut [Hec82] method partitions the color space with the goal to balance the number of pixels in each color cell. The idea behind the Median Cut algorithm is to generate a synthesized color look-up table with each color covering an equal number of pixels of the original image. The algorithm partitions the color space iteratively into subspaces of decreasing size. It starts off with an axes-aligned bounding box in feature space that encloses all the different color values present in the original image. The box is given by the minimum and maximum color component in each of the three coordinate directions (color channels). For splitting the box one determines the dimension in which the box will be (further) subdivided. The splitting is executed by sorting the points by increasing values in the dimension where the current box has its largest edge and by partitioning the box into two sub-boxes at the position of the median. Approximately equal numbers of points are generated on each side of the cutting plane. Splitting is applied iteratively and continued until $K$ boxes are generated. The number $K$ may be chosen to be the maximum number of color entries in the color map used for the output. The color assigned to each of the $K$ boxes is calculated by averaging the colors of each box. The Median cut method performs well for pixels/voxels, whose colors lie in a high-density region of the color space, where repeated divisions result in cells of small size and, hence, small color errors. However, colors that fall in low-density regions of the color space are within large cells, where occurrences of large color errors may be observed.

**K-Means** The main idea of the $K$-Means algorithm [Mac67] is to define $K$ centroids, one for each cluster. These centroids should be placed as far from each other as possible. Typically, this is approximated by using randomly distributed centroids. Each point from the initial data set is associated with the nearest centroid. When all the points have been assigned a centroid, the $K$ centroids are recalculated as the average centers of each cluster.

$$C^{t+1}_i = \frac{\sum_{x_j \in S_{C_i}^t} x_j}{|S_{C_i}^t|}$$  \hspace{1cm} (3.1)
where \( C_{t+1}^{(i)} \) is the \( i \)-th centroid of the \( t + 1 \)st iteration, and \( Sc_i \) is the cluster (set of data points) associated with the centroid \( C_i \), and \( x_j \) are the data points that belong to \( Sc_i \) cluster. Thereafter, the assignment of colors to centroids is updated using the original color points but new centroids. Again, one assigns to each color the nearest centroid. This procedure is iterated until the assignments stabilize. The main advantages of this algorithm are its simplicity and speed which allows it to run on large datasets. Its drawback is its sensitivity to the initial choice of the centroids, i.e., that it does not yield the same result with each run, since the resulting clusters depend on the initial random assignments. The \( K \)-Means algorithm maximizes inter-cluster (or minimizes intra-cluster) variance, but does not ensure that the computed solution actually represents a global optimum.

**C-Means** The idea of the fuzzy \( C \)-Means method \([Bez81]\) is similar to the \( K \)-Means approach, but it allows one data point to belong to two or more clusters. Fuzzy partitioning is carried out through an iterative minimization of the functional \( J \) and the optimization of the data points membership in the clusters and the corresponding update of the cluster centers. The iteration terminates when no further changes with respect to the membership results occur with some given tolerance.

\[
J = \sum_{j=1}^{N} \sum_{i=1}^{c} u_{ij}^m \|x_j - v_i\|^2 \tag{3.2}
\]

\[
u_{ij} = \frac{1}{\sum_{k=1}^{c} \left( \frac{\|x_i - v_i\|}{\|x_j - v_k\|} \right)^{m-1}} \tag{3.3}
\]

\[
v_i = \frac{\sum_{j=1}^{N} u_{ij}^m x_j}{\sum_{j=1}^{N} u_{ij}^m} \tag{3.4}
\]

where \( N \) is the number of data points, \( c \) is the number of clusters, \( x_j \) is the \( j \)-th data point, \( v_i \) is the \( i \)-th centroid, \( m \) is the parameter that controls fuzziness, and \( u \) is the membership matrix.

The algorithm minimizes intra-cluster variance as well, but has the same problems as \( K \)-Means: The minimum is a local minimum, and the results depend on the initial choice of centroids.
As the results of the $K$-Means and $C$-Means approaches happen to be rather sensitive to the choice of the initial centroids, we thought of a good seeding strategy for our task: We take a random data point as the first initial centroid and then iteratively find other centroids among the rest of the points by taking points that are located as far as possible from the already chosen ones.

**Genetic Algorithm** We also introduced our own genetic algorithm to compare its results with the results for the existing clusterization schemes. A genetic algorithm (GA) is a global search heuristics technique inspired by evolutionary biology processes such as inheritance, mutation, selection, and crossover. In broad overview, such methods proceed as follows. First, a population, consisting of individuals, is created. Each individual varies somewhat from the others. Each individual is scored according to a fitness function. The best ones are retained. Then, we stochastically alter the individuals to produce the next generation. Some offspring individuals will have higher fitness values and the best ones are retained, randomly altered to give yet another generation and so on. Each generation has, on average, a slightly higher fitness value than the previous one. The process is halted when either a number of generations has exceeded some maximal value or the single best individual in a generation has a fitness value that exceeds a desired criterion value. The presence of random variations implies that evolutionary methods can find good individuals (lying close to global optimum) even in extremely complex discontinuous spaces that are hard to address by techniques such as gradient descent. The proposed algorithm generates high-quality results and we have used its results as a standard for other clusterization algorithms for the images where the ground truth is unknown.

For the genetic algorithm, we have to define the genetic material, its initialization, the update rules that are iteratively applied, and the fitness function. The genetic material of the individual is stored in a chromosome made up of basic genes which define the physical features of the individual. A previously presented genetic clusterization algorithms [TAE98] takes the mapping of all sample colors to the $K$ palette colors as a genetic chromosome. This choice leads to an extremely high memory consumption, as each chromosome consists of a number of genes equal to the number of unique colors in the picture. In our algorithm, we take the centers of all cluster, i.e., points in the $L^*a^*b^*$ color space, as a gene. Thus, a chromosome consists of $K$ genes only, where $K$ is the number of clusters. For the initial population generation we build the median cut tree, take the cubes lying on the depth equal to $\log_2(K)$, where $K$ is the number of clusters, and determine which unique colors of the image
belong to which cube. Then, we find the average centers of each cluster as it is done in the median-cut algorithm. The initial population is formed by the centers of each cube.

We define an overall fitness function $F_l$ that characterizes each individual $l$ by

$$F_l = \sum_{m=1}^{M} \left\{ D(\overline{c}_m, \tilde{c}_m) + \max_{i,j \in P_m} (D(i,j)) \right\},$$

where $\overline{c}_m$ is the average center of the image colors, which belong to cluster $m$, $\tilde{c}_m$ is the cluster center chosen by the genetic algorithm, $D$ is the Euclidean distance function in the $L^*a^*b^*$ color space, and $P_m$ is the set of the colors that belong to cluster $m$. We minimize this function for the individuals by generating new populations using reproductions (mutations, crossovers) and comparing the fitness functions.

On the test examples the genetic algorithm achieved a clustering that distinguishes all important colors and generates an equal distribution among the remaining colors. Unfortunately, the computation times for the genetic algorithm tend to be rather high (as for all approaches that use stochastic search), such that it is not practical to apply it to larger 3D data sets. Therefore, we use this algorithm only as a standard, to which we compare the results generated by faster clustering algorithms. The most suitable for our purposes according to the validation were K-Means, C-Means and Median Cut algorithms.

### 3.2.2 User Interaction for Cluster Selection

To cluster the colors a user is supposed to choose the appropriate clustering algorithm and to choose the number of clusters. Figure 3.2 shows the user interface. The resulting clusters are shown in the sorted cluster color table. Each cluster is represented by its average color, see Figure 3.2. The clusters are sorted in descending order according to the number of voxels that are covered by the colors that belong to the respective cluster.

Ideally, the generated clusters represent the desired regions of interest and the user can just pick the desired feature (such as an organ, part of an organ, or a tumor) by a single mouse click on the cluster’s representative. Obviously, the clustering algorithms do not always generate perfect solutions. The desired feature may be represented by several clusters or it may be contained in a cluster together with undesired features. We provide mechanisms to quickly and intuitively resolve such situations.
Figure 3.2: For the clusterization step, the user can choose between three clustering algorithms and can set the number of generated clusters (left). The clusterization result is shown as a sorted cluster list, which is drawn using cluster representatives (right). Clusters can be selected by clicking at the representatives and increased by dragging the slider. Any selection can be adaptively refined by reclustering. The segmentation result is shown in a preview in object space by overlaying a 2D contour (pink) with any 2D slice (middle).

If a feature is represented by several clusters, the user may just click on the clusters’ representatives in the cluster list and they are combined to larger clusters. If many clusters are to be united, a semi-automated method can be used by clicking at one of them and moving a slider until all the desired clusters are included. When moving the slider, the next closest clusters are subsequently included with respect to the Euclidean metric in \(L^*a^*b^*\) color space. As mentioned above, this distance is related to the color difference that a human eye can perceive.

If one or several clusters contain both desired and undesired features, user can select these clusters as before and apply a refinement of the clustering. For the refinement, only the selected clusters are taken into account. The user can, again, choose the clustering method and the desired number of new clusters.

These interactions can be iterated. Typically, a few iteration steps suffice to retrieve the desired features. This iterative procedure with few clusters at a time is not only intuitive for the user due to a low complexity of the displayed items, it also reduces the computation times during clusterization.

While selecting, combining, and refining clusters, the user is provided with a “preview” of the current selection in object space. The preview contains a 2D contour rendered on top of a 2D slice. Any of the original 2D color slices can be chosen. In Figure 3.2, the pink contour encloses the white matter of the cryosection slice through a human brain. When selecting other or additional clusters, the contour is immediately updated. A slice-based preview is desirable, as the user can simultaneously observe the contour, the interior, and the exterior of the current selection, which allows for a good judgment of the accuracy of the feature selection.
Once the user is satisfied with the selection, the result is handed to the automatic feature-space segmentation algorithm.

### 3.2.3 Feature-space Segmentation

The feature space is represented by an $L^*a^*b^*$ color space, where the colors of the initial dataset are shown as points using a scatter plot, see Figure 3.3. The user selection left us with a combination of certain clusters that define the region of interest. These clusters contain a set of colors, which in feature space represent a point cloud that is a subset of the points shown in Figure 3.3. Illuminated scatterplots [SW09] would be a great improvement while displaying the point cloud. We need to build a hull around the point cloud in feature space. The hull must include all the points selected by the user and exclude all the other points. In general, the hull’s shape is not convex. As the subsequent surface extraction algorithm can be designed significantly more efficiently when assuming a convex shape, we approximate it by constructing the convex hull over the selected points, i.e., the smallest convex set that contains the point cloud.

![Feature space segments](image)

Figure 3.3: The feature space with the colors of the initial dataset presented as a scatter plot (left). The convex hull (pink) of the selected point cloud (blue) in feature space (middle). Feature selection in feature space using an axes-aligned cuboid (right).

For the computation of the convex hull we chose the Quick Hull algorithm [B+96]. We used the implementation presented in the Computational Geometry Algorithms library (CGAL) [http://www.cgal.org/](http://www.cgal.org/).

The outline of the algorithm can be described as follows:

- Create an initial hull of linearly independent points
- Partition the remaining points into the initial hull’s outside sets
  - for each facet with a non-empty outside set do
    - Select the furthest point of the set
Find the horizon (visible facet boundary) and other visible facets for the point
Make a cone of new facets from the point to the horizon
Partition the outside sets into the cone

end for

Return Convex Hull

The process of a hull computation is automated and does not require user interaction. Figure 3.3 shows the result for a selected point cloud, where the point cloud is highlighted in blue and the convex hull is shown in pink. We compare our results with the results one would obtain using an axes-aligned cuboid for selection of colors in the feature space. The cuboid selection as shown in Figure 3.3 does not require any clusterization and allows for a fast surface extraction, but obviously it is not capable of extracting features as precisely as we can do with our method.

3.3 Object-space Segmentation and Surface Extraction

Having segmented the feature space, we can segment the object space and extract the boundary surface of the desired feature. Since we are dealing with a stack of registered 2D color slices, we operate on a structured rectilinear grid. Thus, we can apply a marching technique that steps through all the cells and extracts the surface components within each cell independently. We use a method for the isosurface extraction, namely Marching cubes (squares). It has $O(L)$ time complexity, where $L$ is the number of all cells which can be created from given volume data. Inputs are the threshold value and the structured scalar volume data. Output is a set of triangles which is an approximation of the isosurface. The method sequentially processes all cells which can be constructed from the volume data. Main steps of the MC algorithm are (for one cell):

- For each vertex define its “inside-outside” property, comparing the intensity value in it with the threshold value
- Compute the index according to “inside-outside” values of 8 vertices (8-bit)
- Use the index to find all edges intersected by the isosurface
- Linearly interpolate isosurface function values at all intersected edges using the threshold value
A cell has a cuboid shape and is easily created from the structured volume data. Then an 8-bit binary index is created. Each cell has 8 vertices and each vertex corresponds to one bit in index. The bit is set to 0 if the value in corresponding vertex is lower than threshold value (the vertex is outside) and vice versa. If the index is equal to 0 or 255, it means that either all vertices are “outside” or “inside” the surface correspondingly, and the cell is not intersecting the isosurface. There are 256 possibilities how an isosurface can intersect a cell. All 256 possibilities can be reduced into 15 basic cases because of rotation of a cell or “inside-outside” symmetry of the intersection case (Figure 3.5). Using the index to define the topology of the surface, the surface intersection along the edge can be interpolated. The linear interpolation is usually used.

For each cell we observe the colors at the eight corners. Since we know exactly, which colors belong to the desired feature, we do not have to perform any inside/outside-test but can just look up the inside/outside-the-hull-property from the feature-space collection. Based on the inside/outside-property we can determine the topology of the surface within the cell. This is the same decision that is made when operating on scalar data. Thus, we can just use the standard marching cubes look-up table, see Section 2.2.

For each edge of the cell that is intersected by the surface, we have to compute the exact intersection point. Let \( p_1 \) and \( p_2 \) be the two endpoints of the edge. Then, we look up the colors \( p_1.color \) and \( p_2.color \) at these vertices. In feature space, the edge between \( p_1.color \) and \( p_2.color \) cuts the convex hull, as shown in Figure 3.6.

We compute the coordinates of all points that belong to the line segment \( p_1.colorp_2.color \) and have intersection with the convex hull planes. Then, using

![Figure 3.4: A cell in the volume data](image)
the convexity of the generated hull, we detect the intersection point belonging to
the hull face by selecting the point that is closest to the segment point lying inside
the hull.

Having found the intersection point \( p_1.color + \lambda \cdot p_2.color \) in feature space, the
surface point in object space is given by \( p_1 + \lambda \cdot p_2 \), similarly, as it is done is the
Marching cube method for single channel data. Figure 3.7 shows the correspondence
between feature and object space.

As the convex hull is only an approximation of a “real” hull of the point cloud given
by the user selection, some colors may lie within the convex hull, although they do
not belong to the chosen point cloud. Since we know which colors are supposed to
be inside and outside, respectively, we always choose the correct marching cubes
case. The only problem is the computation of the surface point, as our intersection
point computation would fail. For this case we just pick a surface point close to the
endpoint of the edge labeled as being inside. This simple solution, of course, inserts

Figure 3.5: The basic 15 cases of surface-cell intersection.
some inaccuracy, but for the sake of a major speed-up in computation we have chosen to go this way. This choice was motivated by the fact that we are typically dealing with high-resolution data, where speed becomes an issue and the introduced inaccuracy become negligible.

If the data sets are of very high resolution, we even propose to use an alternative solution for “previewing” the segmentation results. This alternative solution circumvents the computation of the intersection point in feature space by always picking the center point of the cells’ edges as surface point. We refer to this method as constant division.

### 3.4 Results and Discussion

First, to analyze the results for different clustering algorithms, we have used some synthetic examples. For instance, in Figure 3.8, the clustering results for the genetic algorithm, K-Means, and Median Cut algorithms are shown. The initial image (see Figure 3.8a) has 512 unique colors. We quantized it to 256 colors. As it can be observed in Figure 3.8b, the genetic algorithm produces the smoothest result, both the gradient from black to yellow and the three horizontal stripes of different colors are preserved. K-Means algorithm introduces stripes on the gradient from black to yellow (Figure 3.8c). Median Cut preserves the gradient, but the color of the upper horizontal stripe has been shifted and the color of two bottom stripes has been
merged into one (Figure 3.8d). However, the genetic algorithm was too slow to be applied to real datasets, so we used it for analysis purposes on synthetic examples.

Then, we have applied our methods to several data sets. The first data set presented in this paper is a cryosection data set of a human brain. The resolution of the data set is $512 \times 512 \times 275$. The second data set we present is a fluorescence microscopy data set of a cancer cell. The resolution of the data set is $100 \times 100 \times 40$. To the data we have applied all of the three introduced clustering algorithms and the segmentation and surface extraction methods using hull intersection, constant division, and even the application of the axes-aligned cuboid.

Concerning the clustering algorithms we observed that all of them produced decent results but none of them perfect ones. Which clustering algorithm to pick depends on the properties of the data set. Algorithm choice is a trial-and-error process for each dataset. However, we observed that misclassification during clustering can easily be fixed using our reclustering option. The shown pictures are produced using $K$-Means and Median Cut clustering.

---

1. Data set courtesy of A. Toga, University of California, Los Angeles, USA.
2. Data set courtesy of the Department of Medicine, University of California, San Diego, USA.
Chapter 3. Semi-automated Feature Space Partitioning

(a) Initial synthetic image with a gradient from black to yellow and 3 horizontal stripes of different colors.

(b) Result for the genetic algorithm. Both gradient and horizontal stripes are preserved. Error rate when compared to Figure 3.8a = 1.5812.

(c) Result for K-Means algorithm. On the place of the gradient stripes are introduced. Error rate when compared to Figure 3.8a = 13.3494.

(d) Result for Median Cut algorithm. The colors of two horizontal stripes are merged into one. Error rate when compared to Figure 3.8a = 34.0164.

Figure 3.8: We quantized a synthetic image (512 colors) to 256 colors and compared the result for different clustering algorithms.

Concerning the segmentation and surface extraction methods, the hull intersection method produced the best results, as expected. For high-resolution data set, the difference to the constant division method decreases. Obviously, the error is in the subvoxel range. The axes-aligned cuboid method proved to be hardly applicable for real-world examples. The cuboid was just not flexible enough to extract the features precisely. The shown pictures are produced using hull intersection and constant division.

Figures 3.9 and 3.10 show the application of our algorithms to the cryosection data set. For the generation of the results shown in Figure 3.9 we applied a few iterations of clusterization with K-Means to separate all brain tissue from the surrounding background. In Figure 3.10, we show the result when extracting the white matter of the brain. In particular, the white matter needed to be separated from the gray
matter. In both pictures, we also depict the segmentation and surface extraction applied to a few consecutive slices such that the contours can be compared to the shown 2D slice. In Figure 3.11, the result when applying our techniques to a down-sampled version of the entire brain data set can be observed. The rendering of the brain cortex, obviously, does not exhibit as crisp contours anymore.

Figure 3.9: Segmentation and surface extraction from cryosection data representing a human brain. We separated the brain tissues from the surrounding background for part of the brain (left). We also show one of the original slices (upper right) and a band of the surface when applying our algorithm to a few consecutive slices (lower right).

Figure 3.12 shows that when applying our techniques to the fluorescence microscopy data set, red regions in the cancer cell are extracted. Here, Median Cut clustering is used.

Limitations  We showed that our tool can fulfill the demanded tasks with few user interaction. The user interaction is intuitive, which makes our tool appropriate for handing it to people with a medical background. However, the clusterization in feature space can sometimes lead to undesired results, especially, in presence of noise or partial volume effects (PVE). Even after a thorough selection of clusters,
Chapter 3. Semi-automated Feature Space Partitioning

Figure 3.10: Segmentation and surface extraction to extract white matter of part of the brain (right). We also show a band of the surface when applying our algorithm to a few consecutive slices (left).

Figure 3.11: Segmentation and surface extraction applied to a downsampled version of the entire brain. The gray matter of the brain is extracted.

Figure 3.12: Segmentation and surface extraction from fluorescence microscopy data representing a cancer cell. We extracted the regions that show up as red (right) and compare the result to the dataset shown using direct volume rendering (left).
representing the ROI, the user often ends up with a number of inhomogeneous non-
simply-connected regions.

For example, in Figure 3.13, the clusterization results of white matter of the human
brain are presented. As it can be observed, the extracted region contains numerous
holes, and small regions around the white matter tissue have been also detected.

![Figure 3.13: Operating only in feature space is not fully adequate, as the clustering process often ends up with numerous non-simply connected regions instead of one desired ROI.](a) A slide from a human brain dataset (cryosection) with extracted white matter. (b) A point cloud of extracted white matter surface.](a)

In Figure 3.14, the extraction results of the purple tissue in a Macaque monkey
brain\(^3\) are presented. Due to the color inhomogeneities, it is practically impossible
to extract the purple tissue using only feature space clustering methods. In fact, the
individual cells were segmented, instead of the tissue itself. As a solution for this
particular problem, texture segmentation methods [SHB08] can be applied.

![Figure 3.14: Extraction results of the purple tissue in a Macaque monkey brain.](3) Data set courtesy of Edward G. Jones, Center for Neuroscience, University of California, Davis, USA
Methods that include some smoothing prior constraint and take into account not only the voxel’s color itself, but also its spatial location and the colors of the neighbouring voxels, may give much more adequate partitioning result. Simple image pre-smoothing might be sufficient to reduce the noise and improve the clustering results. But in general, it affects the data, so that the boundaries of tissues are dislocated.
Chapter 4

Combined Feature-Object Space Partitioning

While exploring the existing clustering methods, we observed that operating only in the feature space can not always fulfil the segmentation task appropriately, especially in presence of noise. Hence, we investigated multichannel approaches that take into account not only feature-space characteristics but also spatial arrangement and have a smoothing constraint.

We would like to detect and extract all regions of an image which can be distinguished with respect to certain image characteristics. The resulting areas are non-overlapping and cover the entire image domain.

If the underlying image is supposed to be piecewise constant, image partitioning is equivalent to the restoration of that image, which is often obtained using denoising algorithms. Although there exist plenty of methods for solving image partitioning, segmentation, and restoration problems, many of them are task-specific or require intensive user interaction and triggering of a large number of parameters.

We restrict ourselves to generic solutions using automatic methods based on a global optimization procedure. Global optimization in this context is based on the minimization of some functional. Hence, two tasks need to be tackled: First, one needs to formulate an energy functional using appropriate assumptions and, second, one needs to find an appropriate and computationally feasible minimization algorithm.

In many approaches, starting from the seminal paper of Mumford and Shah [MS85], the energy functional has been formulated by analogy to the energy in the Ising model with an external magnetic field. Leclerc [Lec89] formulated the energy functional using basic concepts from information theory, namely the theory of Minimum
Chapter 4. Combined Feature-Object Space Partitioning

Description Length (MDL). The main idea is that the image partitioning can be considered as an “efficient coding” problem. The most “compact”, i.e. shortest, encoding corresponds to the best partitioning.

We follow the same approach as such a description of an image is natural and gives a certain degree of freedom, allowing (in theory) to describe any class of images, including the images with smooth color transitions within regions. Moreover, the description technique allows to tackle the images in presence of noise, and allows for an appropriate description of the images with different noise distributions.

The energy functional constructed with the MDL theory is closely related to the ones based on the Markov Random Field theory and the Mumford-Shah approach. In Sections 4.1.1, 4.1.2, 4.1.3 we will explain the theoretical basics of these methods and describe how these techniques are applied to image partitioning. The overview of segmentation methods based on energy functionals is given in Section 4.2. Formulating the MDL-based energy, we extend the idea to multichannel image data. The formulated MDL functional includes a noise prior based on a selected model. We assume that the image was subject to white Gaussian noise and consider three different noise models, namely known noise variance, unknown spatially uniform noise variance, and unknown spatially varying noise variance. The description of our approach including all three noise models and its consistency with the MDL principle can be found in Section 4.3. In Section 4.4 we give a detailed description of two minimization approaches: first, the Expansion Move Graph Cut method and, second, the Relaxation scheme with several modifications using Jacobi iteration or Steepest Descent. The implementation details and influence of the parameter selection on the algorithm behavior are discussed in Section 4.5.3. In Section 4.5.4, we focus on the suitability of the minimization approaches to the problem, investigate the behavior of MDL-based minimization, document the feasibility of our multichannel approach, and present results for synthetic and real-world images. The main motivation behind our work was its application to medical multichannel images. We mainly use color images as examples, as they document the algorithmic behavior in an intuitive manner. However, all steps work for any multichannel image data of any number of channels.
4.1 Theoretical background

4.1.1 Mumford-Shah Functional

Mumford and Shah in [MS85, MS89] introduced and studied the most basic properties of three variational problems which were suggested by applications to computer vision. In the current subsection, we will describe shortly two of them. The third functional concerns only the boundaries and is not of interest for our region-based approach.

Let \( z \) be an input signal \( z : S \rightarrow \mathbb{R} \), where \( S \) is either an open region or a lattice in Euclidean space \( \mathbb{R}^d \). The signal \( z \) may take either any real value or be restricted to a finite set of values. When \( d = 2 \) and \( S \) is a pixel lattice, then \( z \) is an image itself.

In the case of the **piecewise smooth model**, the segmentation problem consists of computing a decomposition

\[
R = R_1 \cup \cdots \cup R_n
\]

of the image \( z \) such that there is a smooth and slow color transition within each \( R_i \), and the image \( z \) varies rapidly across most of the boundary \( \Gamma \) between different \( R_i \).

Now, we define functionals \( E \). In these functionals, the following notation is used: the \( R_i \) will be disjoint connected open subsets of a planar domain \( R \), each one with a piecewise smooth boundary. And \( \Gamma \) is the union of the part of the boundaries of the \( R_i \) inside \( R \), so that

\[
R = R_1 \cup \cdots \cup R_n \cup \Gamma,
\]

where \( \cup \) denotes the disjoint union which is a modified union operation which indexes the elements according to which set they originated in. This notation is used to separate the boundaries \( \Gamma \) of the regions \( R_i \) from the regions themselves. For the functional \( E \), let \( u \) be a differentiable function on \( \cup R_i \), which is allowed to be discontinuous across \( \Gamma \). Let

\[
E(u, \Gamma) = \mu^2 \int \int_R (u - z)^2 \, dxdy + \int \int_{R - \Gamma} \| \nabla u \|^2 \, dxdy + \nu |\Gamma|,
\]

where \( \mu, \nu \) are constants and \( |\Gamma| \) is the total length of all boundaries. The smaller \( E \), the better \((u, \Gamma)\) segments \( z \):

- the first term assures closeness (fidelity) \( u \) to \( z \),
- the second term asks that \( u \) and, hence, \( z \), does not vary much within each \( R_i \).
Chapter 4. Combined Feature-Object Space Partitioning

- The third term asks that the boundaries \( \Gamma \) be as short as possible.

\( f, \Gamma \) is an idealization of a true-life complicated image by an ideal “cartoon” image. The second functional \( E_0 \) is the restriction of \( E \) to \textit{piecewise constant} functions \( u \): i.e., \( u = a_i \) on each \( R_i \). In this case, the term \( \int \int_{R_i \setminus \Gamma} \| \nabla u \|^2 dx dy = 0 \), as there is no color transition within each region. And the term \( \int \int_R (u - z)^2 \ dx dy \) can be written as a sum over the regions \( R_i \). Then, we obtain

\[
E_0(\Gamma) = \sum_i \int \int_{R_i} (a_i - z)^2 \ dxdy + \nu_0 |\Gamma|,
\]

where \( \nu_0 = \frac{\mu}{\nu^2} \), \( a_i \) is the mean on \( R_i \) region. \( E_0 \) is closely related to the energy functional in the Ising model: Assume \( u \) can take on only two values: +1 and −1, and \( z \) and \( u \) are functions on lattice. In this setting, \( \Gamma \) is the path made up of lines between all pairs of adjacent lattice points on which \( u \) changes sign (see Figure 4.1). So, the length of the boundary can be expressed as

\[
|\Gamma| = \sum_{k,l \in \text{Neigh}} (u_k - u_l)^2, \tag{4.4}
\]

where \( \text{Neigh} \) denotes the neighborhood, so that \( k \) and \( l \) are neighbors.

![Figure 4.1: Discrete segmentation according to the Ising model.](image)

Then, \( E_0 \) can be rewritten as

\[
E_0(u) = \sum_{i \in S} (u_i - z_i)^2 + \nu_0 \sum_{k,l \in \text{Neigh}} (u_k - u_l)^2, \tag{4.5}
\]

which is the Ising model energy and \( \nu_0 \) includes all additional constant factors.
4.1.2 Maximum A Posteriori Probability - Markov Random Fields Framework

Let $S$ index a discrete set of $m$ sites

$$S = \{1, ..., m\}$$  \hspace{1cm} (4.6)

in which $1, ..., m$ are indices. In 2D case $S$ is a lattice.

The sites in $S$ are related to one another via a neighbourhood system, which is defined as $N = \{N_i | \forall i \in S\}$, where $N_i$ is the set of sites neighboring $i$.

A (pair-wise) clique $c$ for $S$ with the neighborhood system $N$ is defined as a pair of neighboring sites in $S$. The interaction energy inside a clique $c$ is described with a clique interaction potential $V_c = V_{i,j}$, where $i$ and $j$ are neighboring sites.

A label is an event that may happen to a site. Let $\mathcal{L}$ be a set of labels (discrete or continuous). The labeling problem is to assign a label from the label set $\mathcal{L}$ to each of the sites in $S$. The set

$$u = \{u_1, ..., u_m\}$$  \hspace{1cm} (4.7)

is called a labeling of the sites in $S$ in terms of the labels in $\mathcal{L}$, with $u_i \in \mathcal{L}$. In vision, a labeling can correspond to an image (image partitioning), an edge map (edge detection), and so on.

For the image partitioning and restoration problem, the following model for the labeling problem is assumed: an observed labeling $z = \{z_1, ..., z_m\}$ is an array of pixel values. Every pixel takes a value $z_i$ in a set $\mathcal{L}$. An observation $z$ is considered as a transformed and degraded version of some ideal labeling (image) $u$. In the simplest observation model independent additive Gaussian noise is assumed. Each observed pixel is assumed to be the sum of the true (underlying) value $u_i$ and independent Gaussian noise $e_i \sim N(0, \sigma_i^2)$

$$z_i = u_i + e_i.$$  \hspace{1cm} (4.8)

Having the observed labeling and making the assumptions about the noise distribution, one needs to find the “best” underlying labeling, according to some criterion, in this case the maximum a posteriori (MAP) solution is considered. Using the Bayes rule [DHS00], the posterior probability can be computed by using the formulation

$$p(u|z) = \frac{p(z|u)P(u)}{p(z)},$$  \hspace{1cm} (4.9)
where $P(u)$ is the prior probability of the labelings $u$, $p(z|u)$ is the conditional probability density function of the observation $z$ for $u$ fixed, and $p(z)$ is the density of $z$, which is a constant when $z$ is given.

If $p(z|u)$ is considered as a function of $u$ for $z$ fixed, it is called the likelihood function of $u$

$$p(z|u) = L(u|z). \quad (4.10)$$

So, the posterior probability which we define as energy of $u$ can be re-written as

$$E(u) = P(u|z) \propto P(u, z) = p(z|u)P(u). \quad (4.11)$$

Then, the MAP estimate is equivalently found by

$$u^\ast = \arg \max_{u \in \mathcal{F}} \{p(z|u)P(u)\} = \arg \max_{u \in \mathcal{F}} E(u), \quad (4.12)$$

where $\mathcal{F}$ is the whole family of possible labelings in $\mathcal{L}$.

To maximize the energy $E(u)$, we need to define its parts appropriately.

**Definition of $p(z|u)$**  $p(z|u)$ is defined according to the assumptions about the noise distribution. In the case of independent Gaussian noise $N(0, \sigma^2_i)$ in each site $i$ it is

$$p(z|u) = \frac{1}{\prod_i^{m} \sqrt{2\pi \sigma^2_i}} \exp(-U(z|u)), \quad (4.13)$$

where

$$U(z|u) = \sum_{i \in S} \frac{(z_i - u_i)^2}{2\sigma^2_i} \quad (4.14)$$

is the likelihood energy.

**Definition of $P(u)$** For definition of $P(u)$, the Markov random fields (MRF) theory is used. Let $F = \{F_1, ..., F_m\}$ be a family of random variables (pixels) defined on the set $S$ in which each random variable $F_i$ takes a value $u_i$ in a set of labels (colors) $\mathcal{L}$. The family $F$ is called a random field. We use the notation $F_i = u_i$ to denote the event that $F_i$ takes the value $u_i$ and the notation $(F_1 = u_1, ..., F_m = u_m)$ to denote the joint event. For simplicity, a joint event is abbreviated as $F = u$, where $u = \{u_1, \ldots, u_m\}$ is a configuration of $F$ corresponding to a realization of the field. For a discrete label set $\mathcal{L}$, the probability that random variable $F_i$ takes the value $u_i$ is denoted $P(u_i) = P(F_i = u_i)$, and the joint probability is denoted $P(u) = P(F = u) = P(F_1 = u_1, \ldots, F_m = u_m)$. For a continuous $\mathcal{L}$, we have the probability density functions $p(u_i) = p(F_i = u_i)$ and $p(u) = p(F = u)$.
Chapter 4. Combined Feature-Object Space Partitioning

\[ F \] is called Markov random field on \( S \) with a neighborhood system \( N \) if and only if the following two conditions are satisfied:

1. Positivity: \( P(u) > 0 \), \( \forall u \in F \),

2. Markovianity (Locality): \( P(u_i|u_{S-\{i\}}) = P(u_i|u_{N_i}) \),

where \( S-\{i\} \) is the set difference, \( u_{S-\{i\}} \) denotes the set of labels at the sites \( S-\{i\} \), and \( u_{N_i} = \{ u_{i'} | i' \in N_i \} \) stands for the set of labels at the sites neighbouring \( i \).

A set of random variables \( F \) is said to be a Gibbs random field (GRF) on \( S \) with neighbourhood system \( N \) if and only if its configurations obey a Gibbs distribution, which takes the form:

\[
P(u) = Z^{-1} \exp \left( -\frac{1}{T} U(u) \right), \tag{4.15}\]

where

\[
Z = \sum_{u \in F} \exp \left( -\frac{1}{T} U(u) \right) \tag{4.16}
\]

is a normalizing constant called the partition function, \( T \) is a temperature constant, which is assumed to be 1 unless otherwise stated, and \( U(u) \) is the energy function. The energy

\[
U(u) = \sum_{c \in C} V_c(u) \tag{4.17}
\]

is a sum of clique potentials \( V_c(u) \) over all possible cliques \( C \). An MRF is characterized by its local property (Markovianity) whereas a GRF is characterized by its global property (the Gibbs distribution).

The Hammersley-Clifford theorem [Li01] establishes the equivalence of these two types of properties. It states that \( F \) is an MRF on \( S \) with \( N \) if and only if \( F \) is a GRF on \( S \) with \( N \). The theorem provides a simple way of specifying the joint probability \( P(u) \). The clique potential functions \( V_c(u) \) are specified according to the Markovian property and the defined neighborhood system \( N \), and only pair-site cliques are considered, hence, Equation 4.17 can be re-written as

\[
U(u) = \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(u_i, u_j). \tag{4.18}
\]

The exact form of the interaction potential \( V_{i,j} \) can be defined differently. For example, it can defined as \((u_i - u_j)^2\) or as \(1 - \delta(u_i - u_j)\).
Maximizing the energy $E(u)$ in Equation 4.12 for finding the MAP estimate $u^*$ is equivalent to maximizing the natural logarithm of it, Equation 4.12 can be re-written as:

$$
\arg \max_{u \in \mathbb{F}} \{ E(u) \} = \arg \max_{u \in \mathbb{F}} \{ \log E(u) \}
$$

$$
\max_{u \in \mathbb{F}} \{ \log E(u) \} = \max_{u \in \mathbb{F}} \{ \log p(z|u) + \log P(u) \} = \min_{u \in \mathbb{F}} \{ - (\log p(z|u) + \log P(u)) \} = \min_{u \in \mathbb{F}} \{ \sum_{i \in S} \frac{(u_i - z_i)^2}{2\sigma^2} + \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(u_i, u_j) + C \} = \min\{ E_1(u) \},
$$

where $C$ includes all additive constants and can be omitted. If we consider the uniform noise, then $\sigma_i$ is also a constant, and multiplying energy term $E_1(u)$ on $\lambda = 2\sigma^2$, the final energy term that must be minimized can be written as

$$
E(u) = \sum_{i \in S} (u_i - z_i)^2 + \lambda \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(u_i, u_j).
$$

### 4.1.3 Minimum Description Length

The fundamental idea behind the Minimum Description Length (MDL) [Ris87] principle is that any regularity in the given data can be used to compress the data. The image partitioning problem with respect to the MDL principle can be formulated as follows: Using a specified descriptive language, construct the description of an image that is simplest in the sense of being shortest [Lec89].

Let $L(M)$ denote the language for describing a model $M$ and $L(D|M)$ the language for describing data $D$ given model $M$. Moreover, let $|.|$ denote the number of bits in the description. The goal is to find the model $M$ that minimizes the code length

$$
C_I = |L(M)| + |L(D|M)|.
$$

If the a priori probabilities $P(M)$ of the described models are known, then the number of bits in the description equals the negative base-two logarithm of the probability of the described models [Ris87]:

$$
|L(M)| = - \log_2 P(M).
$$

(4.22)

For example, when considering a model $M$ with $n$ options that are equally likely, we obtain $P(M_i) = \frac{1}{n}$ and the corresponding number of bits needed to encode such data
Chapter 4. Combined Feature-Object Space Partitioning

is \(|L(M)| = \log_2(n)|. This observation relates to the entropy encoding according to Shannon’s source coding theorem [Sha48].

In terms of image partitioning and restoration the code length can be written as

\[ C_I = |L(u)| + |L(z - u)|, \quad (4.23) \]

where the model we are looking for is the underlying image representation (or partitioning) \(u\) that minimizes the code length. The term \(z\) describes the initial (or given) image, and the difference \((z - u)\) between the given image \(z\) and the partitioning \(u\) corresponds to the noise in the image. The noise describes the data with respect to model \(u\).

A simple implementation of the MDL principle for image partitioning was proposed by Leclerc [Lec89]. He assumed a piecewise constant model and derived the functional (or energy term)

\[ C_I = \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} (1 - \delta(u_i - u_j)) + a \sum_{i \in I} \left(\frac{z_i - u_i}{\sigma}\right)^2, \quad (4.24) \]

where \(u\) denotes the underlying image, \(z\) the given image, and \(\sigma^2\) the noise variance. Moreover, \(\delta(x)\) denotes the Kronecker delta, \(S\) denotes the lattice of the image, and \(N_i\) is the neighbourhood of the \(i\)th pixel. Coefficients \(a\) and \(b\) are constants. To compute constant \(b\), we add the number of bits needed to encode each boundary element of a region, the number of bits required to encode a starting point of the region’s boundary, and the number of bits required to encode the constant intensity within the region. To obtain \(b\) that sum is divided by the average region boundary length.

The first term in Equation 4.24 encodes the boundaries of the regions. The boundary length is obtained by detecting all those neighboring pixels that belong to different regions, i.e., the ones that have different color values in the underlying image. Since this calculation counts each boundary twice, the number of boundary pixels is obtained by

\[ \frac{1}{2} \sum_{i \in S} \sum_{j \in N_i} (1 - \delta(u_i - u_j)) , \quad (4.25) \]

where

\[ \delta(u_i - u_j) = \begin{cases} 1 & (u_i = u_j) \\ 0 & (u_i \neq u_j) \end{cases} . \]

The second term in Equation 4.24 encodes the noise in form of uncorrelated white Gaussian noise.
If the interaction potential \( V_{i,j} \) is defined as \( 1 - \delta(u_i - u_j) \), one observes that the functional in Equation 4.24 totally coincide with the form of the energy considered in the MAP-MRF approach (cf. 4.21) and corresponds to piecewise constant version of Mumford-Shah functional (cf. 4.3), however, it is formulated in a more natural way (from the information theory point of view), and appears to be more intuitive when prior probabilities are not precisely known.

### 4.2 Related Work

Global energy minimization approaches originate from such seminal works as the ones by Mumford and Shah [MS85] and Blake and Zisserman [BZ87]. The Markov Random Fields (MRF) framework, initially introduced by Geman and Geman [GG90], is a stochastic branch of the energy minimization approaches. In the last years a great breakthrough has been done in the direction of Markov random fields methods [Li01]. Such methods as Graph Cuts [BVZ01] are mathematically well described and allow for finding a solution that lies close to the global optimum. We will discuss the applicability of such methods.

The MDL-based approach [Lec89] uses basic considerations from the information theory [Mac02] in the formulation of the energy term. Leclerc supposed the best image partitioning to be equivalent to obtaining the minimum description length of the image with respect to some specific description language. The formulation of the minimum description length depends on the description, which introduces some degree of freedom. Kerfoot and Bresler [KB99] formulated a full MDL-based criteria for piecewise constant image partitioning and restoration, but the class of images that are dealt with is limited to the class of images only with simply-connected regions. In this work, we formulate a reduced MDL-based criterion for partitioning of multichannel images that is applied to piecewise constant color images without limitations on the image class. For our criterion, we consider three different noise models.

There exist methods that use the MDL formulated functional as a cost function [ZY96, K+94, LK06] to define the “goodness” of the current partitioning. The modification of the configuration is, typically, based on local changes such as region growing, merging, or splitting. The iteration stops when no further energy decrease can be achieved by the local modifications. These methods sometimes allow
to achieve a good approximation of the optimal solution, but require some pre-
processing for the initial partitioning and the results depend heavily on this initial
partitioning.

4.3 MDL-Based Energy Functional

We present an approach for multichannel image partitioning based on the minimum
description length that generalizes the concept presented in Section 4.1.3 to mul-
tichannel data. The goal is to obtain a partition of the image with respect to all
channels. Applying the above-described single-channel image partitioning to each
channel individually would lead to several unrelated partitions. In Figure 4.2, we
give an example for a three-channel case, where the three channels represent RGB
channels in an RGB color representation. The three found segments shown in red,
blue, and green color do not coincide. In fact, the overlap is small. If we want to
obtain a partitioning of a multichannel image, we have to bring the areas in the
individual channels in accordance. Hence, we want to derive a functional whose
minimization leads to a maximization of the overlapping regions or a minimization
of the non-overlapping regions.

Figure 4.2: A multichannel region is formed by overlapping single-channel regions. The image shows
the overlap of single-channel regions of a three-channel RGB color image.

Following Equation (4.23) we need to derive the multichannel image description
length. For encoding the model, i.e., deriving $L(u)$, we have to encode the boundaries
of the regions. To do so, we calculate the number of pixels that contribute to the
boundary. In the single-channel case, the number of boundary pixels is computed
using Equation (4.25). In the multichannel case, the number of boundary pixels in
all channels is estimated by

$$
\frac{1}{2} \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta (u_i^k - u_j^k) \right),
$$

57
where index \( k \) denotes the channel, \( Ch \) denotes the range of channels (e.g., RGB in the three-channel color case), and other notations are as above. This expression computes all pixels that belong to non-overlapping regions in all image channels. The shortest description (practically, approaching to zero) will be reached when the area of the regions coinciding in all channels is maximized.

Hence, the code length for the boundary and region colors encoding is given by

\[
|L(u)| = \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta(u^k_i - u^k_j) \right),
\]

where \( b \) is the same constant as in Equation (4.24). This formula is a generalization of the first term in Equation (4.24) to the multichannel case.

To encode the data that do not fit the model, i.e., the noise, we derive \( L(z - u) \) assuming that the values in each channel are subject to white Gaussian noise with parameters \((0, (\sigma^k)^2)\), where index \( k \) denotes the channel. This assumption implies that the noise between channels is not correlated.

The probability density function of a normal distribution is described by

\[
f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x)^2}{2\sigma^2}\right)
\]

with \( x \in \mathbb{R} \). However, the color space is discrete and the noise values that are added to the color values of pixels must be quantized. The definite Riemann integral is written as

\[
\int_{a}^{b} f(x)dx \approx \sum_{i=1}^{n} f(\xi_i) \Delta x_{i-1}, \tag{4.26}
\]

where \([a, b]\) is the integration interval which is decomposed into \( n \) subintervals by the choice of \( n - 1 \) arbitrary points \( x_1, \ldots, x_{n-1} \), and a point \( \xi_i \) is chosen inside of the subinterval \([x_{i-1}, x_i]\), and \( \Delta x_{i-1} = x_i - x_{i-1} \) is the interval length. When we quantize \( f(x) \) to the color space, we can write that

\[
\int_{x_{i-1}}^{x_i} f(x) \approx f(\xi_i)(x_i - x_{i-1}), \tag{4.27}
\]

where \( \xi_i \) is a exact color value (0...255) and \( x_i - x_{i-1} \) is the length of the interval (distance between the colors). This adds a precision parameter \( q = x_i - x_{i-1} \) to the probability of the noise. Therefore, the probability of the noise in pixel \( i \) and channel \( k \) is estimated by

\[
P(r^k_i) \approx \frac{q}{\sqrt{2\pi(\sigma^k)^2}} \exp\left(-\frac{(r^k_i)^2}{2(\sigma^k)^2}\right), \tag{4.28}
\]
where \( q \) is the precision of the colors and \( r^k_i = |u^k_i - z^k_i| \) is the difference between the underlying image \((u)\) and the initial one \((z)\). Following Equation (4.22), we derive the codelength of the noise as

\[
|L(z - u)| = -\log_2 P(r) = -\log_2 \prod_{i \in S} \prod_{k \in Ch} P(r^k_i)
\]

\[
= - \sum_{i \in S} \sum_{k \in Ch} \log_2 P(r^k_i)
\]

\[
= - \sum_{i \in S} \sum_{k \in Ch} \log_2 \left( \frac{q}{\sqrt{2\pi}(\sigma^k)^2} \exp \left( -\frac{(r^k_i)^2}{2(\sigma^k)^2} \right) \right)
\]

\[
= - \sum_{i \in S} \sum_{k \in Ch} \frac{1}{\ln 2} \ln \left( \frac{q}{\sqrt{2\pi}(\sigma^k)^2} \exp \left( -\frac{(r^k_i)^2}{2(\sigma^k)^2} \right) \right)
\]

\[
= \frac{1}{2\ln 2} \sum_{i \in S} \sum_{k \in Ch} \left( \frac{r^k_i}{\sigma^k} \right)^2 + \frac{N}{\ln 2} \left( \sum_{k \in Ch} (\ln \sigma^k) - \ln q + \ln \frac{2\pi}{2} \right),
\]

where \( N \) denotes the total number of pixels in the image. When applying our method to multichannel images, we set pixel precision \( q \) to 1, as 1 is the smallest distance between any two values in color space when operating with natural numbers.

To determine the noise variance \((\sigma^k)^2\) we consider three noise models:

A. Known noise,

B. Unknown spatially uniform noise, and

C. Unknown spatially varying noise.

For the simplest model with known noise we consider that \( \sigma^k \) is the same constant for each channel \( k \), i.e., for RGB images \( \sigma = \sigma^R = \sigma^G = \sigma^B \), and this parameter is to be selected manually. The spatially uniform noise model supposes that the entire image is affected uniformly by an unknown noise factor with constant variance, i.e., for RGB images the parameters \( \sigma^R, \sigma^G, \sigma^B \) affecting the red, green, and blue channels, correspondingly, are estimated. The spatially varying noise model supposes that each region is differently affected by noise, i.e., parameter \( \sigma \) must be estimated separately for each region. Thus, the noise region boundaries coincide with the segment boundaries of the underlying image. The motivation for the last model is that, for real images, the residuals are due not only to sensor noise but also to small-scale texturing of the objects [Lec89]. We compare three models to each other and discuss their applicability.
Chapter 4. Combined Feature-Object Space Partitioning

The simplest approach is the one where the noise variance is a known value. In this case, any but the first term in the derived expression for $|L(z - u)|$ is constant. As additive constants do not affect the minimization process, they can be discarded. The resulting codelength functional becomes

$$\begin{align*}
C_l &= \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta(u^k_i - u^k_j) \right) \\
&\quad + \frac{1}{2 \ln 2} \sum_{i \in S} \sum_{k \in Ch} \left( \frac{r^k_i}{\sigma^k} \right)^2,
\end{align*}$$

(4.29)

where $\sigma^k$ represents the known variance of the noise in channel $k$. As it is a non-zero constant for all channels it can be taken out of the sum and the energy term can be rewritten as

$$\begin{align*}
C_l = B \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta(u^k_i - u^k_j) \right) + \sum_{i \in S} \sum_{k \in Ch} (r^k_i)^2,
\end{align*}$$

(4.30)

where $B = b \ln 2(\sigma^k)^2$.

For the model of unknown spatially uniform noise, $\sigma^k$ needs to be included in optimization procedure. Hence, the codelength functional can be written as

$$\begin{align*}
C_l &= \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta(u^k_i - u^k_j) \right) \\
&\quad + \frac{1}{2 \ln 2} \sum_{i \in S} \sum_{k \in Ch} \left( \frac{r^k_i}{\sigma^k} \right)^2 \\
&\quad + \frac{N}{\ln 2} \left( \sum_{k \in Ch} (\ln \sigma^k) \right).
\end{align*}$$

(4.31)

Finally, for the unknown spatially varying noise model, we have to bring into accordance the boundaries of the noise regions and the segments of the underlying image. We calculate the number of pixels that contribute to the noise variance boundaries and, at the same time, do not contribute to the boundaries of the underlying image. Minimizing this additional term makes the boundaries coincide. Therefore, the
code\text{length functional when considering spatially varying noise is given by}

\begin{equation}
C_l = \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} \left( 1 - \prod_{k \in Ch} \delta (u_i^k - u_j^k) \right) \tag{4.32}
+ \frac{1}{2 \ln 2} \sum_{i \in S} \sum_{k \in Ch} \left( \frac{r_k^i}{\sigma_k^i} \right)^2 \\
+ \frac{1}{\ln 2} \left( \sum_{i \in S} \sum_{k \in Ch} \ln (\sigma_k^i) \right) \\
+ \frac{g}{2} \sum_{i \in S} \sum_{j \in N_i} \prod_{k \in Ch} \delta (u_i^k - u_j^k) \left( 1 - \prod_{l \in Ch} \delta (\sigma_l^i - \sigma_l^j) \right),
\end{equation}

where constant $g$ is the analogon to constant $b$. Constants $b$ and $g$ are used as weighting coefficients for the contribution of the terms for boundaries of the underlying image and the noise variance, respectively.

4.4 Energy Minimization

Having formulated the energy functionals, one needs to minimize them for a given image in order to compute the image partitioning. As it has been shown that computing the global optimum even of the simplest functional is an NP-hard problem \cite{BVZ01}, in practice one has to look for efficient approximations for it.

We will deal with two approaches for energy minimization, namely, with the $\alpha$-expansion Graph Cut algorithm, one of the state-of-the-art approaches used for energy functional minimization, introduced by Boykov \cite{BVZ01}, and the GNC-like approach, initially introduced for such a task by Leclerc \cite{Lec89}, with several modifications, which have been done to check the convergence.

4.4.1 Expansion move Graph cut algorithm

Graph cuts \cite{BVZ01} is an efficient minimizing technique that allows for finding a local minimum within a known factor of the global one. This technique can be applied to any energy of the form given in Equation (4.21), where the interaction potential $V_{i,j}$ is a metric. One of the examples of the metric is the Potts model $V_{i,j} = 1 - \delta (u_i - u_j)$, where $i, j$ are pixels and $u_i, u_j$ are their colors. This corresponds to the simplest known noise model of Equation (4.30). Let $S$ and $\mathcal{L}$ denote image pixels (lattice) and the palette (set of all possible colors), correspondingly. The
labeling \( u \) is described as \( \{ S_l | l \in L \} \), where \( S_l = \{ p \in S | u_p = l \} \) is a subset of pixels with assigned color \( l \). Given a label \( \alpha \) a move from a labeling \( u \) to a new labeling \( u' \) is called an \( \alpha \)-expansion if \( S_\alpha \subset S'_\alpha \) and \( S'_l \subset S_l \) for any label \( l \neq \alpha \). In other words, an \( \alpha \)-expansion move allows any set of image pixels to change their labels to \( \alpha \) [BVZ01].

The outline of the algorithm is as follows:

Start with an arbitrary partitioning \( u \)

repeat

for all \( \alpha \in L \) do

Find \( \hat{u} = \arg \min E(u') \) among \( u' \) within one \( \alpha \)-expansion of \( u \)

if \( E(\hat{u}) < E(u) \) then

\( u \leftarrow \hat{u} \)

\( success \leftarrow 1 \)

end if

end for

until \( success \neq 0 \)

Return \( u \)

The minimum of the energy \( E \) for each label \( \alpha \) is found by constructing a graph and finding the minimum cut for it. It is efficiently done by the algorithm developed by Boykov and Kolmogorov [BK04].

### 4.4.2 GNC-like Relaxation

A Relaxation method using ideas of Graduated Non Convexity (GNC) by Blake and Zisserman [BZ87] was proposed by Leclerc [Lec89]. The basic concept of the minimization procedure is to replace the non-convex code-length functional \( C_l \) by an embedding in a family of continuous functions \( C_l(u, s) \) that converge towards the target functional \( C_l \) when \( s \) goes to zero. For the starting value of \( s \), the functional \( C_l(u, s) \) is convex such that standard convex minimization procedures can compute the single minimum. When \( s \) approaches zero, number and positions of the local minima of \( C_l(u, s) \) become those of \( C_l \). The minimization procedure iterates over \( s \), which steadily decreases, and minimizes \( C_l(u, s) \) for the respective value of \( s \) in each iteration step.

To obtain a continuous embedding, the discontinuous parts in functional \( C_l \) need to be replaced by a continuous approximation. The discontinuity of \( C_l \) is due to the
use of the function $\delta$. Hence, function $\delta$ is replaced by a continuous approximation that converges to $\delta$ when $s$ goes to zero [Lec89]. We use the approximation

$$\delta (u_i^k - u_j^k) \approx \exp \left( -\frac{(u_i^k - u_j^k)^2}{(s\sigma^k)^2} \right)$$

(4.33)

for the first two noise models with known noise and unknown spatially uniform noise. For the model with unknown spatially varying noise, function $\delta$ is used twice and we use the approximations

$$\begin{align*}
\delta (u_i^k - u_j^k) &\approx \exp \left( -\frac{4 (u_i^k - u_j^k)^2}{(s (\sigma_i^k + \sigma_j^k))^2} \right), \\
\delta (\sigma_i^k - \sigma_j^k) &\approx \exp \left( -\frac{4 (\sigma_i^k - \sigma_j^k)^2}{(s (\sigma_i^k + \sigma_j^k))^2} \right).
\end{align*}$$

(4.34)

The minimization iteration starts with a sufficiently large value $s^0$ and computes the (global) minimum $u^0$ of the convex functional $C_l(u, s^0)$. In each iteration step $T + 1$, we set $s^{T+1} = rs^T$, where $0 < r < 1$, and compute the local minimum of $C_l(u, s^{T+1})$ starting from minimum $u^T$ of the previous iteration step. The iteration is repeated until $s$ is sufficiently small, i.e., until $s < \epsilon$ with $\epsilon$ being a small positive threshold.

To compute the local minimum $u$ on each iteration two techniques have been applied:

1) Jacobi iteration,

2) Gradient Descent.

Jacobi iteration To compute the local minimum for each functional $C_l(u, s)$ Leclerc used the property of the convex function to have only one local and global minimum. He solved the linear system, obtained by $\frac{\partial C_l(u, s^T)}{\partial u_i^k} = 0$ for all $i$.

In the following, we give the details of the instructions that are executed in each iteration step for all three noise models.

Known noise model For the noise model with constant known noise variance, we have derived the functional $C_l$ in Equation (4.29). For its minimization, we apply the embedding using Equation (4.33) and calculate the local minimum of the functional $C_l(u, s)$. The functional $C_l(u, s)$ is convex, if we choose a value for $s$ that satisfies $(x_i^k - x_j^k)^2 \leq 0.5 (s\sigma^k)^2$ for all pixels $i$ and $j$ with $i \neq j$ and all channels $k$. Hence, when this condition is met, the local minimum must be a global one. The condition needs to be fulfilled for the starting value $s = s^0$. 

63
To make the following description more concise, we use the notation

\[ e_{ij}^t = \exp\left(-\frac{(u_i^t - u_j^t)^2}{(s^T\sigma^t)^2}\right). \]

Then, the condition \( \frac{\partial C_u(u, s^T)}{\partial u_i} = 0 \) for the local minimum at iteration step \( T \) becomes

\[ \frac{2a}{(\sigma^k)^2} \left(z_i^k + \frac{b}{2} \sum_{j \in N_i} \frac{2 (u_i^k - u_j^k)}{(s^T\sigma^k)^2} \prod_{l \in Ch} e_{ij}^l \right) = 0, \quad (4.35) \]

where constant \( a \) is given by

\[ a = \frac{1}{2 \ln 2}. \]

As Equation (4.35) cannot be solved explicitly, we use an iterative approach, where at each iteration step \( t + 1 \) we compute

\[ u_i^{k,t+1} = \frac{z_i^k + \frac{b}{2} \sum_{j \in N_i} u_j^{k,t} \prod_{l \in Ch} e_{ij}^{l,t}}{1 + \frac{b}{2a} \sum_{j \in N_i} \prod_{l \in Ch} e_{ij}^{l,t}} \quad (4.36) \]

using the notation

\[ e_{ij}^{l,t} = \exp\left(-\frac{(u_i^{l,t} - u_j^{l,t})^2}{(s^T\sigma^l)^2}\right). \]

In total, we have two nested iterations. The outer iteration denoted by \( T \) iterates over \( s^T \). The outer iteration stops when \( s < \epsilon \) for some small \( \epsilon > 0 \). The inner iteration denoted by \( t \) iterates over \( u_i^{k,t+1} \). Considering the behavior of the exponential function, the termination criterion for the inner loop is given by

\[ \left| u_i^{k,t+1} - u_i^{k,t} \right| < s^T\sigma^k, \quad \forall i \in S. \]

Starting with \( u = z \), the minimization procedure can be summarized by the following pseudo-code:

\begin{verbatim}
while s ≥ \epsilon do
    start with local minimum for u found in previous iteration
    while termination criterion for u is not met do
        recalculate u using Equation (4.36)
    end while
    update s
end while
\end{verbatim}
Comparison to Anisotropic Diffusion. Interestingly, the derived iterative scheme for computing $u_{t+1}^{i}$ in the single-channel case, cf. [Lec89], is similar to the iteration in the well-known anisotropic diffusion approach. The continuous form of the Perona-Malik equation [PM90] for anisotropic diffusion is given by

$$\frac{\partial I}{\partial t} = \text{div} (g (\|\nabla I\|) \cdot \nabla I), \quad (4.37)$$

where $I$ denotes the image and function $g$ is defined by

$$g (\|\nabla I\|) = \exp \left( - \left( \frac{\|\nabla I\|}{K} \right)^2 \right)$$

with flow constant $K$. The discrete version of Equation (4.37) is given by

$$I_{t+1}^{i} - I_{t}^{i} + \lambda \sum_{j \in N_{i}} \left[ (I_{t}^{i} - I_{j}^{i}) \exp \left( - \left( \frac{(I_{t}^{i} - I_{j}^{i})}{K} \right)^2 \right) \right] = 0,$$

where $\lambda$ is a normalization factor. The MDL-based energy $C_{l}(u, s)$ for the single-channel case is given by

$$C_{l}(u, s) = a \sum_{i \in S} \left( \frac{z_{i} - u_{i}}{\sigma} \right)^2 + \frac{b}{2} \sum_{i \in S} \sum_{j \in N_{i}} \left( 1 - \exp \left( - \left( \frac{u_{i} - u_{j}}{s\sigma} \right)^2 \right) \right)$$

Finding its local minimum (for a fixed $s$), i.e., solving $\frac{\partial C_{l}(u, s)}{\partial u_{i}} = 0$, leads to

$$(u_{i} - z_{i}) + \alpha \sum_{j \in N_{i}} \left[ (u_{i} - u_{j}) \exp \left( - \left( \frac{u_{i} - u_{j}}{s\sigma} \right)^2 \right) \right] = 0, \quad (4.38)$$

where $\alpha = \frac{b}{as^2}$. The continuous version of Equation 4.38 can be written as

$$\int_{0}^{t_{\text{end}}} \frac{\partial u}{\partial t} dt = \text{div} (g (\|\nabla u\|) \cdot \nabla u), \quad (4.39)$$

where $u_{0} = z$ and $u_{t_{\text{end}}}$ describes the image values at the current step $t_{\text{end}}$, the values of $u$ in the right side of equation are also taken on the step $t_{\text{end}}$. Comparing the continuous form of the Perona-Malik equation (4.37) with the continuous form of the MDL-based equation (4.39), one can immediately observe the similarity. The main difference is the integral on the left-hand side of Equation (4.39). The integral represents the changes between the current image and the initial image. Thus, this version of the MDL-based minimization algorithm can be considered as an “anisotropic diffusion algorithm with memory”.

65
Unknown spatially uniform noise model For the noise model with unknown spatially uniform noise variance, we have derived the functional $C_l$ in Equation (4.31). For its minimization, we apply the embedding using Equation (4.33) and calculate the local minimum of the functional $C_l(u, s)$. For large $s$, the functional $C_l(u, s)$ is convex, if we consider the noise variance $(\sigma^k)^2$ to be constant. Hence, we propose to proceed as follows: Starting with an initial estimate for $\sigma^k$ given by

$$\sigma^k = \frac{1}{|S|} \sum_{i \in S} \sqrt{\frac{\sum_{j \in N_i} (z^k_i - z^k_j)^2}{|N_i|}},$$

i.e., estimating $\sigma^k$ by deviations within local neighborhoods of the initial image, we apply the minimization scheme as for the known noise model. Afterwards, we recalculate $\sigma^k$ according to

$$\sigma^k = \sqrt{\frac{1}{|S|} \sum_{i \in S} (u^k_i - z^k_i)^2}, \quad (4.40)$$

i.e., we estimate $\sigma^k$ by the root-mean-square deviation of the current values of $u^k_i$ from the initial values $z^k_i$, and repeat the minimization procedure. This process is iterated until $\sigma^k$ does not change significantly anymore, i.e., until $\sigma^k$ fulfills the criterion

$$|\Delta \sigma^k| < \varepsilon, \quad \forall k \in Ch \quad \text{for some small } \varepsilon > 0.$$

The iterations over $s$ must be stopped when it reaches some small $\varepsilon > 0$. Starting with $u = z$, the minimization procedure can be summarized by the following pseudo-code:

```plaintext
calculate the initial estimates of $\sigma^k$

while $|\Delta \sigma^k| \geq \varepsilon$ for any $k \in Ch$ do
    while $s \geq \varepsilon$ do
        start with local minimum for $u$ found in previous iteration
        while termination criterion for $u$ is not met do
            recalculate $u$ using Equation (4.36)
        end while
        update $s$
    end while
    update $\sigma^k$ for all $k \in Ch$ using Equation (4.40)
end while
```

66
**Unknown spatially varying noise model** For the noise model with unknown spatially varying noise variance, we have derived the functional $C_i$ in Equation (4.32). For its minimization, we apply the embedding using Equation (4.34). Moreover, we linearize the logarithmic term $\ln \sigma_i^k$ by replacing it with a term that converges to the average of $\sigma_r^k$ within each region $r$ [Lec89]. As we do not know the regions yet, we estimate the $\sigma$s in each pixel $i$ as follows. In a region $R_r$ the average region color and variance are defined as

$$u_r = \frac{1}{N_r} \sum_{i \in R_r} z_i$$

$$\sigma_r^2 = \frac{1}{N_r} \sum_{i \in R_r} (z_i - u_r)^2 \approx \frac{1}{N_r} \sum_{i \in R_r} \tilde{\sigma}_i^2,$$

where

$$\tilde{\sigma}_i^2 = \frac{\sum_{j \in N_i \cap R_r} (u_r - z_j)^2}{\sum_{j \in N_i \cap R_r} 1} = \frac{\sum_{j \in N_i} \delta(u_i - u_j)(u_i - z_j)^2}{\sum_{j \in N_i} \delta(u_i - u_j)}.$$

Hence, at iteration $T$ of our non-convex function minimization procedure, the approximation is defined by

$$\ln \sigma_i^k \approx \left( \frac{\sigma_i^k - \hat{\sigma}_i^{k,T-1}}{\sigma_i^{k,T-1}} \right)^2,$$

where $\sigma_i^{k,T-1}$ is the square root of the local minimum of the noise variance found in the iteration $T - 1$ and $\hat{\sigma}_i^{k,T-1}$ is the square root of an estimate of noise variance within the regions found in iteration $T - 1$. The square root of the regional noise variance estimate $\hat{\sigma}_i^{k,T-1}$ is computed by

$$\hat{\sigma}_i^{k,T-1} = \max \left\{ q, \sqrt{\frac{\sum_{j \in N_i} e_{i,j}^{k,T-1} (u_i^{k,T-1} - z_j)^2}{\sum_{j \in N_i} e_{i,j}^{k,T-1}}} \right\} \quad (4.41)$$

with $q$ being the precision of the pixel colors from Equation (4.28) and using the short notation

$$e_{i,j}^{k,T-1} = \exp \left( -\frac{4 \left( u_i^{k,T-1} - u_j^{k,T-1} \right)^2}{s^T \left( \sigma_i^{k,T-1} + \sigma_j^{k,T-1} \right)^2} \right).$$
Chapter 4. Combined Feature-Object Space Partitioning

The different noise variance estimates are initialized by

$$\sigma^0_i = \sigma^{k,0}_i = \hat{\sigma}^{k,0}_i = \max \left\{ q, \sqrt{\frac{1}{|N_i|} \sum_{j \in N_i} (z^k_i - z^k_j)^2} \right\}.$$  

These substitutions lead to the following functional

$$C_l(u, s, \sigma) = \frac{b}{2} \sum_{i \in S} \sum_{j \in N_i} \left(1 - \prod_{k \in Ch} e_{u_{ij}}^k \right)$$  

$$+ a \sum_{i \in S} \sum_{k \in Ch} \left( \frac{u^k_i - z^k_i}{\sigma_i^{k,T-1}} \right)^2$$  

$$+ 2a \sum_{i \in S} \sum_{k \in Ch} \left( \frac{\sigma_i^k - \hat{\sigma}_i^{k,T-1}}{\sigma_i^{k,T-1}} \right)^2$$  

$$+ \frac{g}{2} \sum_{i \in S} \sum_{j \in N_i} \left( \prod_{k \in Ch} e_{\sigma_{ij}}^{k,T-1} \right) \left(1 - \prod_{k \in Ch} e_{\sigma_{ij}}^k \right)$$  

using the short notations

$$e_{u_{ij}}^k = \exp \left( -\frac{4 \left( u^k_i - u^k_j \right)^2}{s^T \left( \sigma_i^{k,T-1} + \sigma_j^{k,T-1} \right)^2} \right),$$  

$$e_{\sigma_{ij}}^k = \exp \left( -\frac{4 \left( \sigma_i^k - \sigma_j^k \right)^2}{s^T \left( \sigma_i^{k,T-1} + \sigma_j^{k,T-1} \right)^2} \right).$$

For large $s$, the derived functional is convex in $u$ and $\sigma$, respectively, when the other is considered constant. For estimating the local minima for $u$ and $\sigma$ we consider the equations $\frac{\partial C_l(u, s, \sigma)}{\partial u^k_i} = 0$ and $\frac{\partial C_l(u, s, \sigma)}{\partial \sigma^k_i} = 0$, which are, again, solved in a descending algorithm using the assignments

$$u^{k,t+1}_i = \frac{z^k_i + b\alpha \sum_{j \in N_i} \left( \beta_{ij} u^k_j \prod_{l \in Ch} e_{u_{ij}}^l \right)}{1 + b\alpha \sum_{j \in N_i} \left( \beta_{ij} \prod_{l \in Ch} e_{u_{ij}}^l \right)}$$  

(4.43)

and

$$\sigma^{k,t+1}_i = \frac{\hat{\sigma}_i^{k,T-1} + g\alpha \sum_{j \in N_i} \left( \beta_{ij} \sigma_j^{k,T-1} \prod_{l \in Ch} e_{\sigma_{ij}}^{l,T-1} e_{\sigma_{ij}}^l \right)}{1 + g\alpha \sum_{j \in N_i} \left( \beta_{ij} \prod_{l \in Ch} e_{\sigma_{ij}}^{l,T-1} e_{\sigma_{ij}}^l \right)}$$  

(4.44)
Chapter 4. Combined Feature-Object Space Partitioning

with $\alpha = 4 \left( \sigma_i^{k,T-1} \right)^2 \ln(2) \left( s^T \right)^{-2}$ and $\beta_{ij} = \left( \sigma_i^{k,T-1} + \sigma_j^{k,T-1} \right)^{-2}$.

Starting with $u = z$, the minimization procedure can be summarized by the following pseudo-code:

- calculate the initial estimates of $\sigma^k$
- while $s \geq \epsilon$
  - while termination criterion for $u$ is not met do
    - recalculate $u$ using Equation (4.43)
  - end while
  - recalculate $\hat{\sigma}$ (Equation (4.41))
  - while termination criterion for $\sigma$ is not met do
    - recalculate $\sigma$ using Equation (4.44)
  - end while
- update $s$
- end while

The inner loops can be terminated when changes are small. Here, we omit the indexes $i$ and $k$, which denote lattice index and color channel index, considering that the stopping criteria must be true for all $i$ in each $k$. We stop when

$$\frac{u}{s^T \sigma^{T-1}} \approx \frac{u + \Delta u}{s^T \sigma^{T-1}} \quad \text{and} \quad \frac{\sigma}{s^T \sigma^{T-1}} \approx \frac{\sigma + \Delta \sigma}{s^T \sigma^{T-1}},$$

respectively, where $T$ denotes the current iteration of the outer loop, $\sigma^{T-1}$ denotes the square root of the local minimum for noise variance found in iteration $T - 1$, and $\Delta u$ and $\Delta \sigma$ denote the differences in $u$ and $\sigma$ when comparing the current values (iteration $t$) to the ones of the previous inner iteration $t - 1$. We obtain the stopping criteria $|\Delta u| \ll s^T \sigma^{T-1}$ and $|\Delta \sigma| \ll s^T \sigma^{T-1}$, respectively.

**Gradient descent** In general, the assumptions used for the above-described approach do not hold anymore, when $s$ is small, $C_l(u,s)$ is non-convex and this method can give an inadequate result. To check this we implemented the steepest descent minimization scheme [PTVF02] and compared the results.

### 4.4.3 Implementation Details: GPU-based version

Nowadays, Graphics Processing Units (GPUs) are high-performance many-core processors capable of very high computation and data throughput. Once specially designed for computer graphics and difficult to program, today’s GPUs are general-
Chapter 4. Combined Feature-Object Space Partitioning

purpose parallel processors with support for accessible programming interfaces and industry-standard languages such as C.

In order to understand how the graphics hardware works, one needs to know, which steps are required to display some data. The rendering pipeline or graphics pipeline in computer graphics is a conceptual representation of the stages through which a certain representation of a 2D or 3D scene goes through to produce as output the final 2D raster image which is displayed. Each graphics package has its own rendering pipeline in terms of 2D or 3D primitives representation and sequence of processes applied to them. Graphics rendering pipelines are commonly represented as state-diagrams, where each state refers to the sequence of transformations applied to the input data [O+07].

![OpenGL2.0 rendering pipeline](image)

Figure 4.3: OpenGL2.0 rendering pipeline. The vertex and fragment processor stages are both programmable by the user.

In Figure 4.3 the OpenGL graphics pipeline is shown (Image is from [Ros06]).

The input to the pipeline is a list of geometry, expressed as vertices in object coordinates; the output is an image in a framebuffer. The first stage of the pipeline, the geometry stage, transforms each vertex from object space into screen space, assembles the vertices into triangles, and traditionally performs lighting calculations on each vertex. The output of the geometry stage is triangles in screen space. The next stage, rasterization, both determines the screen positions covered by each triangle and interpolates per-vertex parameters across the triangle. The result of the rasterization stage is a fragment for each pixel location covered by a triangle. The third
stage, the fragment stage, computes the color for each fragment, using the interpolated values from the geometry stage. This computation can use values from global memory in the form of textures; typically the fragment stage generates addresses into texture memory, fetches their associated texture values, and uses them to compute the fragment color. In the final stage, composition, fragments are assembled into an image of pixels, usually by choosing the closest fragment to the camera at each pixel location [O+07].

In old graphics card each stage of the pipeline was implemented as fixed function, i.e. the algorithms (shaders) used to perform the transformations were decided at the card manufacturing stage and could not be altered afterwards. The novelty introduced by graphics accelerators was to give the programmer access to two of the graphics pipeline stages: vertex transformation stage and fragment transformation stage. The rendering pipeline has then become programmable, piece of code used to apply transformation effects on the input data would be called shaders. There exists a number of APIs (such as GLSL, Cg, Sh, Brook, CUDA) to ease the process of GPU programming.

The vertex processor is a programmable unit that operates on incoming vertices and their associated data. Vertex shaders are set of functions used to transform each abstract vertex position in world space to the 2D coordinate system at which it appears on the screen. Vertex shaders run on the vertex processor [Ros06].

The fragment processor is a programmable unit that operates on incoming fragments and their associated data. Fragment shaders are set of functions used to transform each abstract vertex attribute such as colour, normal, texture value to the display color system. Fragment shaders are typically used to mimic effects such as scene lighting and colour toning on a per pixel base. Fragment shaders run on the fragment processor [Ros06].

The programmable units of the GPU follow a single-instruction multiple-data (SIMD) programming model. For efficiency, the GPU processes many elements (vertices or fragments) in parallel using the same program. Each element is independent from the other elements, and elements can not communicate with each other. All GPU programs must be structured in this way: many parallel elements, each processed in parallel by a single program [OHL+08].

GLSL basics For our implementation, we have used OpenGL Shading Language (GLSL http://www.opengl.org/documentation/glsl/).
As GLSL relies on the graphics vendor to provide a compiler directly to GPU machine code, no “intermediate” compilers for shaders are needed. The general structure of a simple GPGPU GLSL program is shown in Figure 4.4.

![Figure 4.4: The structure of a simple general-purpose GLSL program.](image)

One must first do all the initializations, then create the needed textures, create the GPU program, attach the output textures to the Framebuffer Object, enable the GPU program, bind the variables (including scalars and input textures), set the render destination (the output texture), and execute the rendering pass, i.e. execute the shaders and write to the output texture. Then, the output texture must be transferred to the CPU side. One must have in mind, that the textures can be either readable or writable, but not both at the same time. The process of a GPU program creation is schematically shown in Figure 4.5.

For multiple rendering passes a standard technique, called “ping pong”, is used. It means to alternately use the output of a given rendering pass as input in the next one.

For more information, we refer to the tutorials for general purpose GPU programming (GPGPU) kindly provided by Dominik Goeddeke (http://www.mathematik.uni-dortmund.de/~goeddeke/gpgpu/tutorial.html).

**Relaxation scheme implementation** The Relaxation scheme with Jacobi iterations fits perfectly to the SIMD model. In Figure 4.6 the CPU-GPU interaction for
the Relaxation scheme is schematically shown.

The CPU-GPU interaction consists of several steps. First, on the CPU side we create and compile the GLSL program using the OpenGL-like API. Then, the image data and the parameters are bound with GPU side and the data values are sent to GPU. Thereafter, the main relaxation loop starts, see pseudocode descriptions in Section 4.4.2. For each loop iteration, which computes the next \( u \), the GPU program (rendering passes) is executed until the stop criterion is met. GPU-part is based on the “ping pong” technique. Afterwards, the parameter values correspondent to the main loop are updated on the GPU side and the GPU part is executed again. Finally, the resulting data is transferred back from GPU to CPU. This step is the most expensive in the whole CPU-GPU computation pipeline.

The implementation of the Relaxation scheme with constant known noise consists of two programs that are run subsequently. One program computes the next \( u \) and the other checks the stop criterion. In the following part we present the code listings with detailed comments.

Let us first describe the shaders of the first “calculating \( u \)” program. It consists of two shaders: one vertex, one fragment. In the vertex shader the positions of the neighbouring texels are calculated.
Figure 4.6: CPU-GPU interaction for Jacobi iteration Relaxation Scheme.

```c
/* calc. vert — vertex shader,  
** calculates the 8 neighbor positions 
** for the current texel */

// image size
uniform float imgWidth;
7 uniform float imgHeight;

// 8 neighbour coordinates to pass to fragment shader
9 // each texel has 2 coordinates
    varying vec4 n_w_n_o;
11 varying vec4 n_n_n_s;
    varying vec4 n_w_n_n_on;
13 varying vec4 n_w_s_n_os;
   // current coordinate to pass to fragment shader
```
Chapter 4. Combined Feature-Object Space Partitioning

15 varying vec2 texCoords;

17 void main(void) {
    // obtain the transformed (by the modelview and projection matrices) incoming texel
19     gl_Position = ftransform();
    // current texel coords
21     texCoords = gl_MultiTexCoord0.st;
    // offset in width and height to texel neighbors within texture
23     float ofst_w = 1.0/imgWidth;
     float ofst_h = 1.0/ imgHeight;
25     // get coordinates of 8 texel neighbors using current texel coordinates and offsets
    vec2 n_w = vec2(texCoords.x-ofst_w, texCoords.y);
    vec2 n_o = vec2(texCoords.x+ofst_w, texCoords.y);
    vec2 n_n = vec2(texCoords.x, texCoords.y-ofst_h);
    vec2 n_s = vec2(texCoords.x,texCoords.y+ofst_h);
    vec2 n_wn = vec2(texCoords.x-ofst_w,texCoords.y-ofst_h);
    vec2 n_on = vec2(texCoords.x+ofst_w,texCoords.y-ofst_h);
    vec2 n_ws = vec2(texCoords.x-ofst_w,texCoords.y+ofst_h);
    vec2 n_os = vec2(texCoords.x+ofst_w,texCoords.y+ofst_h);
    // pack 8 neighbors into 4 vec4 to pass to fragment shader
35     n_w_n_o = vec4(n_w, n_o);
     n_n_n_s = vec4(n_n, n_s);
37     n_wn_n_on = vec4(n_wn, n_on);
     n_ws_n_os = vec4(n_ws, n_os);
39 }

In the fragment shader, the calculation of \( u \) (cf. Equation 4.36) for a three-channel image is performed.

/* calc.frag – fragment shader,
 ** next u
 */
5 #version 110
    //current u
7 uniform sampler2D textureU;
    // initial image
9 uniform sampler2D textureZ;
    // img size
11 uniform float imgWidth;
    uniform float imgHeight;
13 // some constants
    uniform float mult_s_sigma_sqrs;
15 uniform float b_a_sq;

75
Chapter 4. Combined Feature-Object Space Partitioning

// texel coordinate
17 varying vec2 texCoords;
// neighbor coordinates
19 varying vec4 n_w_n_o;
    varying vec4 n_n_n_s;
21 varying vec4 n_wn_n_on;
    varying vec4 n_ws_n_os;

23 void main (void) {
25 // result
    vec3 res_ue;
    res_ue.r = 0.0;
    res_ue.g = 0.0;
    res_ue.b = 0.0;
// current texel coordinates
31 vec2 texCoord = texCoords;
    // initial image color in texel
33 vec3 z = texture2D (textureZ, texCoord).rgb;
    // underlying image color in texel
35 vec3 u = texture2D (textureU, texCoord).rgb;
        float res_e = 0.0;
37 // calculate (accumulate values) for 1 neighbor
    // res_ue += un*exp((u-un)^2/(s*sigma)^2)
39 // res_e += exp((u-un)^2/(s*sigma)^2)
    vec2 neighCoord = vec2(n_w_n_o.r, n_w_n_o.g);
41 vec3 u_n = texture2D (textureU, neighCoord).rgb;
    vec3 diff = (u-u_n);
43 bvec3 eq = equal(vec3(0.0,0.0,0.0), u_n);
        bool b = all (eq);
45 float st = step(0.5,1.0 - float (b));
        float dist = dot (diff,diff);
47 float exp_dist = exp(-dist*mult_s_sigma_sqrs);
    res_e += st*exp_dist;
49 res_ue += st*u_n*exp_dist;
    // the same calculation for other 7 neighbors
51 .........."
This implementation suffers from the inflexibility, as it is suited for 3-channel images only.

To stop the loop, we have considered two possibilities. The first one is to use some fixed number of iterations, but this solution is inefficient for real computations. The second one is to discard the texels, for which the difference between \( u_{\text{prev}} \) and \( u_{\text{next}} \) is bigger some \( \epsilon \), and then calculate the number of texels that have passed. In the next listings, we show the fragment shader and a part of C-code, which computes the next local minimum of \( u \) We show in detail the part that computes the number of texels that were not discarded.

```c
/* stop.frag - fragment shader, */
2 ** that discards texels where \( |u_{\text{prev}}-u_{\text{next}}|<\epsilon \)
3 */
4 // previous u
5 uniform sampler2D textureU_prev;
6 // next u
7 uniform sampler2D textureU_cur;
8 // threshold
9 uniform float s_sigma_thresh;
10 void main(void){
11    // color in previous u texel
12    vec3 u_prev = texture2D(textureU_prev, gl_TexCoord[0].st).rgb;
13    // color in current u texel
14    vec3 u_cur = texture2D(textureU_cur, gl_TexCoord[0].st).rgb;
15    // check the distance, if it's threshold ----> discard texel
16    if (distance(u_prev, u_cur)<s_sigma_thresh)
17       { gl_FragColor = vec4(0.0,0.0,0.0,1.0); }
18    else
19       { discard; }
20 } }
```
Chapter 4. Combined Feature-Object Space Partitioning

10 // call the stop criterion program
glUseProgram(stopProgID);
12 // draw to "no texture"
glDrawBuffer(GL_NONE);
14 glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
   glPolygonMode(GL_FRONT, GL_FILL);
16 // start the test on the number of samples (texels) that were not discarded
   glBeginQueryARB(GL_SAMPLES_PASSED_ARB, query);
18 // execute the shader (do the rendering)
   glBegin(GL_QUADS);
   TexCoord2f(0, 0); glVertex3f(-1,-1,-0.5f);
   TexCoord2f(1, 0); glVertex3f(1,-1,-0.5f);
   TexCoord2f(1, 1); glVertex3f(1,1,-0.5f);
   TexCoord2f(0, 1); glVertex3f(-1,1,-0.5f);
24 glEnd();
   glEndQueryARB(GL_SAMPLES_PASSED_ARB);
26 // stop the test
28 glGetQueryObjectuivARB(query, GL_QUERY_RESULT_ARB, &sampleCount);
   // if all texels passed the test or maxIter is reached --> stop
30 if (sampleCount==imWidth*imHeight || iter >= maxIter)
   break;
32 . . .
34 . . .

The implementation for the locally varying noise model is similar with additional shaders and rendering passes for σ computations.

**Speed up Results** We tested our GPU implementation on two graphics cards, namely NVidia Quadro FX 350m (4 Fragment and 3 Vertex Processors) and NVidia Quadro FX 4500 (24 Fragment and 8 Vertex Processors). The CPU implementation results have been obtained on Intel(R) Xeon(R) CPU 5150 @ 2,66GHz 2.66 Ghz, 3.25 Gb. To check the speed up, we fixed the number of iterations and executed both rendering passes.

In Tables 4.1, 4.2 the comparison for 596 iterations for different image sizes is shown. As it can be observed, we achieve almost 200x and 150x speed ups for NVidia Quadro FX 4500 for simple known noise model and for unknown spatially varying noise, respectively.

However, the current implementation suffers from several disadvantages. First, it
Table 4.1: Timing comparison for the CPU and GPU implementations of the simple known noise model (Jacobi iterations).

<table>
<thead>
<tr>
<th>Image</th>
<th>CPU</th>
<th>GPU: 350m</th>
<th>GPU: 4500</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 × 16</td>
<td>1.85</td>
<td>0.2</td>
<td>0.14</td>
</tr>
<tr>
<td>32 × 32</td>
<td>7.18</td>
<td>0.2</td>
<td>0.135</td>
</tr>
<tr>
<td>64 × 64</td>
<td>28.21</td>
<td>0.73</td>
<td>0.36</td>
</tr>
<tr>
<td>128 × 128</td>
<td>113.54</td>
<td>1.23</td>
<td>0.94</td>
</tr>
<tr>
<td>256 × 256</td>
<td>458.41</td>
<td>4.67</td>
<td>2.74</td>
</tr>
<tr>
<td>512 × 512</td>
<td>1833.64</td>
<td>17.11</td>
<td>10.31</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>7334.56</td>
<td>65.6</td>
<td>38.6</td>
</tr>
</tbody>
</table>

Table 4.2: Timing comparison for the CPU and GPU implementations of the unknown spatially varying noise model (Jacobi iterations).

<table>
<thead>
<tr>
<th>Image</th>
<th>CPU</th>
<th>GPU: 350m</th>
<th>GPU: 4500</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 × 16</td>
<td>2.92</td>
<td>0.61</td>
<td>0.38</td>
</tr>
<tr>
<td>32 × 32</td>
<td>12.19</td>
<td>1.14</td>
<td>0.38</td>
</tr>
<tr>
<td>64 × 64</td>
<td>49.84</td>
<td>1.57</td>
<td>0.74</td>
</tr>
<tr>
<td>128 × 128</td>
<td>204.69</td>
<td>5.32</td>
<td>1.34</td>
</tr>
<tr>
<td>256 × 256</td>
<td>869.08</td>
<td>18.8</td>
<td>4.6</td>
</tr>
<tr>
<td>512 × 512</td>
<td>3476.32</td>
<td>72.42</td>
<td>17.7</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>13905.28</td>
<td>287.8</td>
<td>69.28</td>
</tr>
</tbody>
</table>

is not flexible with the number of image channels. The shader codes are suited for three channels only. Second, to the author’s knowledge, there is no possibility in GLSL to check, how much of the texture memory is available for allocation. For the above-mentioned graphics cards, we could not attach a texture of a size 2048 × 2048 and more to the FBO.

4.5 Results and Discussion

4.5.1 Minimization Scheme comparison

We investigated the results for a synthetic color image with added noise with parameters \( N(0, 70^2) \) on three algorithms, namely Graph Cuts and Relaxation with
Chapter 4. Combined Feature-Object Space Partitioning

Jacobi iteration and Steepest Descent for the known noise model. Figure 4.7 illustrates the high quality results that were obtained with the Graph Cuts algorithm. It allows us to reconstruct the exact regions of the initial image with few misclassifications. However, when the size of the palette is large, the efficient precomputation or direct "on-the-fly" computation is problematic and the computational costs increase dramatically. In our experiments we have reduced the space of labels, which for RGB images is equal to 256³, cf. [BVZ98], and took only unique colors of the image and their closest neighbors as labels. The computation for a 100 × 100 color image with 50,000 labels took around 10 minutes per iteration. As usually several iterations are needed, this approach appears to be infeasible for bigger datasets. Several workarounds can be imagined. First, one can reduce the number of color channels using PCA or some non-linear conversion method [IL07a]. However, such a workaround definitely leads to a loss of information. Second, each channel can be treated separately and then one combines the results into a multichannel image again. However, such a workaround leads usually to the artifacts that are suppressed when the “real” multichannel approach is used. Third, one may set the number of labels manually and use, for example, a combined Graph cuts and Expectation Maximization method, cf. [ZK04], or look for a more sophisticated scheme for defining the appropriate size of the palette. However, the reduction of the palette size can lead to inadequate results. If the important colors of the palette have been mistakenly reduced, the algorithm will not be able to converge to the appropriate local minimum.

The relaxation scheme with the Gradient Descent method for defining the local minimum also gives high quality results and reconstructs the initial image (Figure 4.7). However, the computational costs are high, for example, for a 100 × 100 color image the costs are in the range of several hours, which makes this method inapplicable for real datasets.

The relaxation technique with Jacobi iterations generally does not reproduce the initial regions completely. In Figure 4.7d the restoration result obtained with Jacobi iterations is shown. The resulting image consists of 49 regions. Those ones that form one region from the initial image (Figure 4.7a) have very close colors and the “weak” region boundaries can not be distinguished by a human eye. Due to such color closeness, these regions can be merged into one using a simple region-merging procedure. Starting at any pixel (seed point), the algorithm subsequently adds neighbouring pixels to the region if the distance in color space between them and the region color is lower than a certain threshold. The region color is the average
Figure 4.7: Comparison of the results produced by Graph Cuts, Relaxation with Steepest Descent, and Relaxations with Jacobi iteration on a synthetic image with 8 regions. The first row shows the restoration results. The second row shows the piecewise constant regions of the corresponding result images shown on the first row now marked with distinguishable colors.

color of the pixels that belong to it. The algorithm stops when all the pixels belong to some regions. Computational costs of the region merging procedure are negligible when compared to the ones of the Relaxation method.

Though Graph Cuts and Steepest Descent approaches produce high quality results, due to their computational costs they are hardly applicable to the real world datasets. Jacobi iteration scheme is relatively fast (cf. Tables 4.2, 4.1), but does not fully reconstruct the regions. However, a simple post-processing procedure, namely, region merging, can improve the results with negligible computational costs. Hence,
relaxation with Jacobi iterations in conjunction with subsequent region merging can be used for real applications.

4.5.2 Algorithm Behavior

Here, we discuss the behavior of the relaxation scheme.

Figure 4.9 documents the general behavior of the Relaxation scheme for unknown spatially varying noise model. Definitely, the similar behavior “rules” hold for two other models.

Starting with a synthetic 100 \times 100 image with manually added noise, see Figure 4.8d, we apply the Jacobi iteration minimization procedure. The three rows in Figure 4.9 show (intermediate) results at iteration \( T = 4 \), \( T = 600 \), and \( T = 1484 \), respectively. The first column shows the currently detected underlying image and the second column the currently removed noise, i.e., the difference between the initial image (Figure 4.8d) and the currently detected underlying image (first column of Figure 4.9). For the generation of the images in columns three and four, we picked one row of the image, i.e., a horizontal cut through the image, and show the current values of \( u \) (third column) and \( \sigma \) (fourth column) for that single row. The third and fourth column in Figure 4.9 document nicely, how individual values (first row) start assimilating and grouping together (second row) to eventually form a piecewise constant representation (third row). This is true for both the underlying image and the noise variance. We also observe that positions of jumps in the representations shown in the third and fourth column start to coincide, which means that the regions of the segmented image and the respective noise regions coincide.

The first and the second column in Figure 4.9 document that the minimization procedure traverses different phases. In fact, we observed three phases. In a first phase, the noise gets removed, which leads to a blurring of the image. This can be seen when comparing Figure 4.9e and 4.9f to Figure 4.9a and 4.9b. Figure 4.9e shows the blurring effect and Figure 4.9f the denoising. In a second phase, sharp boundaries are formed. This is possible due to the “initial image memory” (cf. Section 4.4.2, Comparison to Anisotropic diffusion). Finally, in a third phase, adjacent segments of similar value merge to form larger segments. The effect of the last two phases can be seen when comparing Figure 4.9i to Figure 4.9e. How the third phase affects the image can hardly be observed visually, as the difference in value between adjacent segments before the merging are subtle. For example, Figure 4.9i seems to consist of three colors only, while Figure 4.9k exhibits that the selected row consists of seven
different colors. Clearly, one can observe the similarity of some of these seven colors, which are candidates for further merging in subsequent iterations.

For synthetic images with manually added uniformly distributed Gaussian noise, the results for all three noise models are similar, which is documented in Figure 4.10.

Figure 4.8: Synthetic images we used for analysis purposes.
(a) Iteration $T = 4$: Currently detected underlying image.
(b) Iteration $T = 4$: Currently removed noise.
(c) Iteration $T = 4$: Current image values $u$ of one row.
(d) Iteration $T = 4$: Current values $\sqrt{\sigma}$ of one row.

(e) Iteration $T = 600$: Currently detected underlying image.
(f) Iteration $T = 600$: Currently removed noise.
(g) Iteration $T = 600$: Current image values $u$ of one row.
(h) Iteration $T = 600$: Current values $\sqrt{\sigma}$ of one row.

(i) Iteration $T = 1484$: Currently detected underlying image.
(j) Iteration $T = 1484$: Currently removed noise.
(k) Iteration $T = 1484$: Current image values $u$ of one row.
(l) Iteration $T = 1484$: Current values $\sqrt{\sigma}$ of one row.

Figure 4.9: Different phases of MDL minimization.
Figure 4.10: MDL-based partitioning results for the three noise models for image in Figure 4.8f. All three models find a partitioning that is equivalent to the original synthetic image before introducing noise.

4.5.3 Parameter Selection

In this section we discuss the parameters to be selected for the MDL-based functional with the unknown spatially varying noise model in the Relaxation scheme implementation. For the other two models the same parameters must be selected similarly.

Minimization parameters The first group of parameters to discuss are the parameters \( s_{\text{start}} \), \( s_{\text{end}} \), and \( r \), which have been introduced for the minimization procedure. It is intuitively clear that the choices for \( s_{\text{end}} \) and \( r \) are a tradeoff between accuracy and speed. The closer \( s_{\text{end}} \) is to 0 and the closer \( r \) is to 1, the more iterations we have to apply but the more accurate the result will be. The choice of \( s_{\text{start}} \) is not intuitively clear. We just know that it is supposed to be “large enough”.

Parameter \( s_{\text{start}} \) is responsible for the initial approximation of the Kronecker delta. With decreasing \( s \) the approximation becomes more precise. In a first experiment, we take \( s_{\text{start}} = 1 \) and apply the MDL minimization algorithm to a synthetic 100 × 100 image with different signal-to-noise ratio as shown in Figure 4.8i, 4.8j, and 4.8k. Figure 4.11 shows the results, where the other parameters are set to \( s_{\text{end}} = 0.0001 \), \( r = 0.99 \), \( b = 4 \), and \( g = 40 \). The variances of the manually added noise are \( \sigma = \{20, 70, 130\} \). The first row shows the segmentation result, while the second row shows the minimization procedure by plotting the average change of the image \( |u^T - u^{T-1}| \) over iteration \( T \), where \( u^T \) and \( u^{T-1} \) are the minima found in iterations \( T \) and \( T - 1 \), respectively. The plots (and all subsequent plots) only show the results for ...
one channel of the multichannel image, as the behavior in all channels is qualitatively equivalent. We observed that a higher signal-to-noise ratio requires a larger start value $s_{\text{start}}$. Underestimating $s_{\text{start}}$ leads to undesired segmentation results, see first row of Figure 4.11, and to bad convergence results, see second row of Figure 4.11.

Figure 4.11: Resulting images and convergence plots for MDL minimization using $s_{\text{start}} = 1$ for the images in Figure 4.8i, 4.8j, and 4.8k.

In a second experiment, we tried to explore what a good initial value $s_{\text{start}}$ would be. To do so, we took one of the images, namely the one with $\sigma = 70$ shown in Figure 4.8j, and apply the MDL minimization starting with $s_{\text{start}} = 1, 10, 100,$ and $1,000$. The plot of $|u^T - u^{T-1}|$ over the iteration $T$ has been shown in Figure 4.8j for $s_{\text{start}} = 1$. Figure 4.12a shows the same plot for $s_{\text{start}} = 10, 100,$ and $1,000$ in red, green, and blue, respectively. This depiction clearly shows that the shape of the curve is the same for all starting values, just that they are shifted. The curve consists of an ascending part followed by a descending part with several peaks. In Figure 4.11, the ascending part is cut off, which led to non-optimal results. The results for $s_{\text{start}} = 10, 100,$ and $1,000$ were comparable. Thus, for the given example, there is no need to choose $s_{\text{start}} > 10$, as it would only prolong the minimization
process. We can conclude that an ideal value for \( s_{\text{start}} \) is one where the ascending part is small but still existing. For the (synthetic and real-world) examples we considered, \( s_{\text{start}} \in [10, 100] \) turned out to be a proper choice even for images with low signal-to-noise ratio.

![Plots](image)

**Figure 4.12:** Plots of \( |u^T - u^{T-1}| \) for synthetic image with \( \sigma = 70 \) shown in Figure 4.8j with different values for \( s_{\text{start}} \) and \( s_{\text{end}} \).

Parameter \( s_{\text{end}} \) controls the stopping criterion of the iteration. The smaller \( s_{\text{end}} \), the more iterations we perform. To examine what would be a good choice for \( s_{\text{end}} \), we use the image in Figure 4.8j again as our input and apply MDL minimization for varying values of \( s_{\text{end}} \), where \( s_{\text{start}} = 100 \) and \( r = 0.99 \). The plots of \( |u^T - u^{T-1}| \) for \( s_{\text{end}} = 0.5 \) (red), 0.1 (green), 0.0001 (blue) are presented in Figure 4.12b. The graphs partially coincide such that the green and the blue graph are only partially visible. The resulting images of the minimization procedure are shown in Figure 4.13 and seem to be identical, which is due to the fact that for all of them the blurring phase and the boundary sharpening phase are completed. However, there are slight differences, as the merging phase has not been completed. Thus, the image in Figure 4.13a contains significantly more segments than the image in Figure 4.13b and even more than the image in Figure 4.13c.

To analyse the impact of parameter \( r \) that controls the step size of \( s \), we take our input image (Figure 4.8c) and apply the minimization procedure for different values of \( r \), where \( s_{\text{start}} = 100 \) and \( s_{\text{end}} = 0.0001 \) are fix. In Figure 4.14, we plot the relative number of changing pixels in each iteration \( T \), i.e., the number of changing pixels divided by the total number of pixels, for \( r = 0.7, 0.8, 0.9, 0.95, \) and 0.99. We observe that the resulting plot becomes less bumpy when increasing \( r \). In fact, only for \( r = 0.99 \) large peaks can be avoided, which leads to a stable minimum. Hence, choosing a value for \( r \) that is smaller than 0.99 may lead to undesirable results. The resulting plot for \( r = 0.99 \) in Figure 4.14e exhibits a similar behavior as the plots of
Chapter 4. Combined Feature-Object Space Partitioning

(a) Result for $s_{end} = 0.5$. Number of segments is 7743. 
(b) Result for $s_{end} = 0.1$. Number of segments is 424. 
(c) Result for $s_{end} = 0.0001$. Number of segments is 24.

Figure 4.13: Resulting images for MDL minimization using different values $s_{end}$ for the image in Figure 4.8j.

$|u^T - u^{T-1}|$ in Figure 4.12a. To speed up the algorithm and reduce the computational time, it is senseful to stop the execution after the boundary sharpening phase and leave merging to a region growing (merging) procedure.

Figure 4.14: Plots of the relative number of changing pixels in each iteration $T$ for different values of parameter $r$, which controls the step size of $s$. Input was the image in Figure 4.8j.

**MDL parameters** The second group of parameters to be discussed are the parameters $b$ and $g$ that have been introduced for the MDL description. The parameters are weights that control the influence of the boundary length of the multichannel...
Chapter 4. Combined Feature-Object Space Partitioning

segments and the noise regions, respectively, see Equation (4.42). The impact of the choice of \( b \) and \( g \) is documented in Figure 4.15, where the image in Figure 4.8g is chosen as the input image and we applied the minimization procedure for various combinations of \( b \) and \( g \). The two parameters are not independent of each other.

For Figure 4.15a, 4.15b, and 4.15c, we kept the ratio of \( b \) and \( g \) fix and increased the value of \( b \). It can be observed that the boundary segments start vanishing at some points, as they are penalized too much, i.e., removing boundaries is much more important than sticking closely to the colors of the input image. On the other hand, one can observe that when choosing \( b \) too small, individual pixels that are affected much by the noise error cannot be merged with its neighbor regions. Hence, if \( b \) is too small noise cannot be removed, whereas if \( b \) is too large features are considered as noise. For our examples, \( b \) had to be larger than 1, but should not be significantly larger. For example, for the images in the top row of Figure 4.8 a good choice would be \( b \in [3.5, 4] \) and for the images in the bottom row one should choose \( b \in [4, 8] \).

For arbitrary images, we take \( b \in [3, 5] \) and we observed that further parameter adjustment is rarely needed.

For Figure 4.15b, 4.15d, and 4.15e, we took an appropriate value of \( b \) and varied \( g \). If the ratio of \( b \) versus \( g \) is not chosen correctly, the boundaries of the segments and of the noise regions are not brought in accordance, which leads to some vanishing regions.

To further examine the observed behavior, we plot for all of the studied cases of Figure 4.15 the change in \( u \) and \( \sigma \) for one channel between two subsequent iterations \( T - 1 \) and \( T \). Figure 4.16 shows the plots of \(|u^T - u^{T-1}| \) (red) and \(|\sigma^T - \sigma^{T-1}| \) (green). The two curves exhibit a similar behavior. For a proper ratio of \( b \) versus \( g \), the process shown by the green curve starts slightly before the process shown by the red curve. If \( g \) is too small in comparison to \( b \), the process of forming noise regions is governed by the formation of the multichannel regions as shown in Figure 4.16d. If \( g \) is too large in comparison to \( b \), the process of forming noise regions has priority and reaches a stable state before the formation of the multichannel regions has started (see Figure 4.16e). To obtain an equilibrium between the two processes of region formation, a behavior like the one shown in Figure 4.16b is desirable. In our experiments, best results were achieved by selecting \( g = k \cdot b \) with \( k \in [10, 100] \). Note that if \( b \) is not chosen appropriately, the choice of \( g \) cannot compensate for this.

For the model with known noise \( \sigma \) is also to be selected. For the synthetic images we take the “actual” noise variance or, sometimes, a slightly lower value. Here, the
values of $b$ and $\sigma$ should be selected to “balance” each other, i. e., both terms in the energy functional are weighted appropriately.

### 4.5.4 General Results

Due to the fact that the global MDL-based approach incorporates not only feature, but also object space information, and includes the assumption about noise, it allows for more accurate segmentation, compared to the results obtained with the methods that operate only in the feature space (see Chapter 3).

**Experimental comparison to other techniques** In Section 4.3 we have shown the similarity of our approach to the Anisotropic diffusion [PM90] proposed by Perona and Malik. However, the presented approach has a “memory”, as it always refers to the initial image instead of iterating from the result obtained on the previous time step. Such a behavior should guarantee a better boundary preservation if the parameters are properly chosen (cf. Section 4.5.3) when compared to the Anisotropic diffusion method. For instance, in Figure 4.17, the result for a synthetic image with artificially added Gaussian noise $(0, 70^2)$ (cf. Figure 4.8c) is shown. As it can be observed, Anisotropic diffusion either does not erase the noise completely or
oversmooths the boundaries (even after a thorough parameter tuning), whereas our approach appears to be much more stable to noise and, at the same time, preserves object boundaries.

In Section 4.1, we have shown the coincidence of the MDL-based functional and Mumford-Shah functional for the piecewise constant case. One of the well known approaches to minimization of the Mumford-Shah functional is an iterative region merging. We have compared to the algorithm proposed by Koepfler, Lorez and Morel [KLM94]. However, such an iterative technique, generally, does not lead to a stable local minimum and gives much coarser results, when compared to the relaxation approach. In Figure 4.18, the results for Figure 4.8c with different $\nu_0$ (cf.
Chapter 4. Combined Feature-Object Space Partitioning

(a) Result for Figure 4.8c for Anisotropic diffusion with parameters: $\sigma = 1.4$, $stSize = 5$, $numStep = 10$, $edge = 3.5$. Error rate when compared to the initial underlying image = 37.85.

(b) Result for Figure 4.8c for Anisotropic diffusion with parameters: $\sigma = 1.4$, $stSize = 10$, $numStep = 10$, $edge = 3.5$. Error rate when compared to the initial underlying image = 55.703.

(c) Result for for Figure 4.8c for our approach. The model with known noise with parameters $\sigma = 45$, $b = 4$ has been applied. Error rate when compared to the initial underlying image = 32.29.

Figure 4.17: When compared to Anisotropic diffusion (Perona-Malik method), our method outperforms in boundary preservation.

Equation 4.5) are shown. The error rate was computed as a square root of the mean distance from the initial (not noisy) image. As it can be observed, the iterative region merging produces results with some misclassified boundaries and not completely erased noise when compared to Figure 4.17c.

In our approach, we consider a simplified MDL-based functional that requires a user to manually adjust several parameters, e.g., in the model with unknown spatially varying noise the parameters $b$ and $g$ need to be adjusted. In several approaches [LK06], [Lee00], an extended version of the functional that includes the number of regions is used and all the user-defined parameters are eliminated. These approaches also utilize a region growing approach, as the minimization of the extended functional is infeasible [LK06]. The initial region selection has a strong impact on the efficiency and effectiveness of the region merging procedure [LK06]. There different approaches have been employed. For example, Luo and Khoshgoftaar [LK06] propose to use the well known Mean-Shift approach [CM02a] to obtain the initial segmentation and start the MDL-based region merging procedure from it. The Mean-Shift algorithm takes both multichannel and spatial arrangement into account. Similarly to the bilateral filter [PD06], it has two user-defined kernel bandwidths in color ($h_c$) and spatial ($h_s$) domains. We have applied the Mean-Shift method to our synthetic examples, having selected the bandwidths carefully. We demonstrate in
Chapter 4. Combined Feature-Object Space Partitioning

(a) Result for Figure 4.8c for Iterative region merging with $\nu_0 = 1000$. Noise is not erased completely. Error rate when compared to the initial underlying image = 84.2912.

(b) Result for Figure 4.8c for Iterative region merging with $\nu_0 = 10^6$. Noise is still not erased completely. Further increasing the parameter $\nu_0$ does not improve the result. Error rate when compared to the initial underlying image = 36.442.

(c) Result for Figure 4.8c for the pipeline consisting of the Anisotropic diffusion and region merging based on the Mumford-Shah functional with initialization with the Mean-Shift method. Error rate when compared to the initial underlying image = 35.0945.

Figure 4.18: An iterative region growing does not converge to a stable local minimum. However, using a pipeline of methods including region-growing can give reasonable results.

Figure 4.19, that the Mean Shift algorithm has problems in presence of noise and misclassifies the region boundaries.

(a) Result for Figure 4.8d for the Mean-Shift algorithm with $h_s = 7$, $h_r = 60$. Error rate when compared to the initial underlying image = 64.8605.

(b) Result for Figure 4.7e for the Mean-Shift algorithm with $h_s = 7$, $h_r = 50$.

(c) Result for Figure 4.19a for the region merging procedure with $\nu_0 = 10^6$. Error rate when compared to the initial underlying image = 65.0781.

Figure 4.19: The Mean-Shift algorithm misclassifies the boundaries in presence of noise and the subsequent region merging can not correct the mistakes.
Thereafter, we have applied the iterative region merging procedure to the result (Figure 4.19a) for the Mean-Shift method. Figure 4.19c shows that if the initial boundaries have been misclassified, the region merging can not restore them properly, independent from the functional that is minimized with it.

In Section 4.5.4, we describe the general behavior of the algorithm and show that it can be roughly split into three phases: blurring, boundaries sharpening, and region merging. In this way, the relaxation approach can be approximated with a pipeline that utilizes the above-mentioned methods. This resembles the approach proposed by Petrovic and Vanderheyndt [P+03], however, there the alternating steps of smoothing and region merging are used, whereas we apply the methods subsequently. The result shown in Figure 4.18c is obtained by the subsequent application of the Anisotropic diffusion and the region growing methods. As it can be observed, the pipeline produces the best result in terms of boundary preservation and noise elimination when compared to the independent application of these methods (cf. Figure 4.17a and Figure 4.18b). However, in general, each method in the pipeline requires additional adjustment of the parameters for each image.

We also compared our results to the results obtained with the Active contours without edges [CV01, CSV00, VC02] method introduced by Chan and Vese. This is a variational approach based on energy minimization. This approach unlike classical snakes allows handles the topological changes in the images and allows for obtaining the image partitioning. The energy is formulated using the Mumford-Shah functional for piecewise constant case. It looks as follows:

$$E(c_1, c_2, C) = \mu \text{Length}(C) + \nu \text{Area}(C) +$$
$$\lambda_1 \int_{\text{inside}(C)} |u_0(x, y) - c_1| dxdy + \lambda_2 \int_{\text{outside}(C)} |u_0(x, y) - c_2| dxdy,$$

where $u_0$ is the initial image, $c_1$ and $c_2$ are the means inside and outside the curve $C$. For minimization, the curve representation is rewritten using the level sets formulation. Initially, the method was introduced for binary segmentation [CV01]. Thereafter, the extensions for multichannel case [VC02] and multiphase (multi-object) segmentation have been proposed [CSV00, VC02]. For our tests, we applied the 4-phase version of the algorithm for color piecewise constant images with the parameters for each phase: $\lambda_{1,2} = 1$, $\nu = 0$, as it is recommended by the authors. We experimented with different values of parameter $\mu$ and initial contour locations. We executed the algorithm with max. 2000 iterations with the timestep $= 0.5$. We
observed that the evolution process (and the speed of convergence) is heavily dependent on the initialization and it is problematic to obtain decent results for images with low signal-to-noise ratio for the reasonable amount of time.

In Figure 4.20 the segmentation results for image in Figures 4.8b and 4.8c with different initial contours and $\mu = 0.09$ are presented.

![Figure 4.20](image)

(a) Result for Figure 4.8b.

(b) Initial contours.

(c) Result for Figure 4.8c.

(d) Initial contours.

The resulting contours are still noisy.

Figure 4.20: With the appropriate initialization Chan-Vese method can partition the image appropriately if the noise level is not high.

**Application to the real-world problems** The choice of the appropriate noise model for a real dataset is dependent on many factors, e.g., the way, the data was acquired, the size and shape of the objects to be detected etc. In general, one should try with the simplest model, as it is the fastest one and if it is not possible to improve the partitioning results, then switch to a more complicated one. However, choosing the simplest model, one should find an appropriate value of $\sigma$.

We have applied the Relaxation scheme to the real-world datasets to demonstrate the behavior of the of different noise models. In Figure 4.21, the segmentation result
of the gray matter of a human brain\(^1\) is shown. When compared to the results, obtained in Chapter 3, the MDL-based method with subsequent region merging produces much more accurate segmentation results. To perform the selection of the ROI (gray matter) from the partitioned image, we have applied color thresholding and extracted the region boundaries. In Figure 4.21c, the boundaries of the gray matter are depicted in an overlayed manner on the initial image. In Figure 4.22, we apply our methods to a histology dataset of a rat liver\(^2\). The goal is to automatically detect all red and blue cells in the image. Figure 4.22 shows that the spatially uniform

\(^1\)Data set courtesy of A. Toga, University of California, Los Angeles, USA.
\(^2\)Data set courtesy of Dr. Uta Dahmen and Dr. Olaf Dirsch, Experimental Surgery Group, Department of General, Visceral, and Transplantation Surgery, Essen, Germany.
noise model leads to undersegmentation, i.e., some low contrast-to-noise regions are not detected as segments but contribute to the background. The spatially varying noise model, however, can deal with these low contrast-to-noise regions and leads to the result containing finer structures. This observation supports that the assumption of having coinciding regions of noise with regions in the underlying image is valid.

![Figure 4.22: Comparing noise models for a histological image.](image)

(a) Initial image.

(b) Initial image with manually detected groundtruth.

(c) Result for model with known noise.

(d) Result for model with unknown spatially uniform noise. Areas where the cells with low contrast can be missed are marked with red.

(e) Result for model with unknown spatially varying noise. The cells have higher contrast and clear defined boundaries compared to the unknown spatially uniform noise result.

Figure 4.22: Comparing noise models for a histological image. The spatially uniform model exhibits undersegmentation, as several cells with low contrast contribute to the background, while the spatially varying model leads to the desired result.

In Figure 4.23 one slide from the plaque dataset is shown. The data is obtained from

3Data set courtsey of Dr. med. Andreas Harloff, Dept. of Neurology and Clinical Neurophys-
patients who underwent carotid endarterectomy for ICA stenosis > 70%. Plaques were fixed in 4% formalin for ≥ 24 hours after surgical resection, transferred into tubes containing free oil (1,1,2-Trichlorotrifluoroethane) and evaluated at a high field 9.4T MRI system (Bruker BioSpin 94/20, Ettlingen, Germany). The MRI protocol provided 3D imaging with a spatial resolution of 100µm³ including T1-fatsat-GRE, T2*-GRE, T2-RARE, and PD-RARE imaging. The research task is to detect and study the structure of the plaque, i.e., find the meaningful regions, for example, calcification, fat etc.

Figure 4.23 documents that even the simplest known noise model allows to achieve a well-structured result comparing to the unknown spatially varying noise model. This is due to the fact, that, seemingly, the noise is homogeneous for the whole image. The application of the simplest model here is favourable in terms of computational costs. However, to give more exact qualitative description of the results, one needs a direct comparison with the manually detected ground truth.

![Figure 4.23: A slice from Plaque dataset, T1, T2, T2* weights are combined (left). Both models with known noise (middle) and unknown spatially varying noise (right) give well structured results.](image)

**Limitations**  Although our approach gives promising results when compared to other techniques and has a certain merit, it has a number of limitations that can be divided into the following groups:

- model limitations,
- minimization scheme limitations,
- performance limitations.
**Model Limitations** In the current research, we considered only piecewise constant model that can be insufficient for real-life images. In medical datasets, there is usually a certain color transition within regions, and piecewise constant model will suffer from oversegmentation and the region-merging postprocessing becomes necessary. For instance, in Figure 4.21b a slide from human brain dataset is shown. As it can be observed, the image obtained is the MDL-based segmentation approach suffers from oversegmentation and needs postprocessing. However, a simple region merging procedure usually suffices. Another limitation of the model is that only white Gaussian noise is assumed, which will lead to inadequate results if this assumption does not hold.

**Minimization scheme limitations** The minimization scheme based on Jacobi iterations generally does not converge to the global minimum. With this scheme, we only approximate a solution in a critical point (the first derivative equals 0 there), i.e., it may be any local minimum or maximum. The obtained solution will lie relatively close to the local minimum obtained with the gradient descent-based as well as graph cut methods (cf. Section 4.5.1). However, it depends heavily on the maximum number of iterations and precision parameters. In practice, this leads to oversegmentation and needs postprocessing.

**Performance limitations** There is a need in an efficient and rather fast implementation of the minimization scheme, that will lead to (at least) a strong local minimum convergence. We have analysed CPU implementations of the relaxation scheme with Jacobi iterations and gradient descent method, and the graph cut method. It is obvious that current CPU implementations can not be applied to large real-world datasets due to the speed limitations (cf. Section 4.5.1).

We have implemented the relaxation scheme with Jacobi iterations on GPU using GLSL, the details are given in Section 4.4.3. We achieved around 200x speed up for the scheme with known noise on NVidia Quadro FX 4500 graphics card. However, due to the limited memory on graphical unit, we could process only images with $1024 \times 1024$ at a time. Processing of 3D data will be also possible only for small datasets.

An implementation on a powerful GPU station, for example, NVidia Tesla (http://en.wikipedia.org/wiki/NVIDIA_Tesla), may give some interesting results and open a further application perspective for these methods.
Chapter 5

Partitioning With Prior Assumptions

In the previous chapters, we discussed several general-purpose partitioning strategies for multichannel data. In the current chapter, we present an approach, where the segmentation target, i.e., its shape, size, and color, is known in advance. This knowledge should be exploited for a more efficient segmentation algorithm.

In the current study, the task of segmentation of the histological sections and further specific cell type, namely, hepatocyte quantification is to be solved. We propose an automatic pipeline for the hepatocyte quantification. We describe the medical background of the problem in Section 5.1. In Section 5.2 methods similar and/or related to our pipeline are discussed. The general pipeline is presented in Section 5.3. The software GUI and functionality details are given in Section 5.4. The obtained results are presented in Section 5.5.

5.1 Medical Background

5.1.1 Liver Anatomy Basics

The liver is one of the largest, most important organs in the body. It has a diversity of functions that includes formation and secretion of bile, storage of glycogen, buffer for blood glucose, synthesis of urea, metabolism of cholesterol and fat, synthesis and endocrine secretion of many plasma proteins including clotting factors, detoxification of many drugs and other poisons, cleansing of bacteria from blood, processing of several steroid hormones and vitamin D, volume reservoir for blood, and catabolism
of hemoglobin from worn-out red blood cells. The liver is necessary for survival; there is currently no way to compensate for the absence of liver function. Figure 5.1 shows the internal vascular structure of a human liver\(^1\). The liver structure and anatomical placement reflect its function as the site for removal of impurities and for conditioning of the blood. The liver receives a dual vascular supply. The hepatic portal vein, marked blue in Figure 5.1, brings to the liver all of the blood which has previously passed through the intestine and spleen. It usually contains various contaminants (drugs, toxins from food, bacteria, byproducts of blood-cell recycling). The hepatic artery marked red in Figure 5.1, brings fresh, oxygenated blood from the aorta. Blood from both portal vein and hepatic artery mixes together in the hepatic sinusoids (i.e., vascular spaces) and then passes out of the liver through the three-part hepatic vein (right, left, and middle), marked blue in Figure 5.1.

The liver is organized into lobules which take the shape of polygonal prisms, see Figure 5.2. Each lobule is typically hexagonal in cross section. Figure 5.3 shows a lobule on the sample taken from the liver of a pig\(^2\).

Each lobule is supplied by blood from two sources: a branch of the hepatic artery and a branch of the portal vein. These enter the lobule at its corners and form two parts of the portal canal or portal triad. Blood flowing into the lobule is drained out via the central vein, which marks the beginning of the drainage circulation via

---
\(^1\)Image is from Lippincott Williams and Wilkins Atlas of Anatomy, 1st Edition, 2008

\(^2\)Image courtesy of Southern Illinois University, School of Medicine.
Chapter 5. Partitioning With Prior Assumptions

Figure 5.2: Liver is organized into lobules. Each hepatic lobule is shaped as a polygonal prism with portal triads (portal vein, hepatic artery, bile duct) at vertices. Blood goes through branches of the portal vein and hepatic artery and flows out through the central vein.

Figure 5.3: A lobule on a sample from a pig liver.

the hepatic vein. Central veins from adjacent lobules merge to form larger vessels, and eventually all of the drainage leaves the liver via the hepatic vein. Thus, blood flow in the lobule is from the periphery to the center, as it is schematically shown in Figure 5.2. The third part of the triad is the intralobular bile ductule. The bile ductules carry bile produced in the hepatocytes. Small ones join with others to form larger and larger vessels.

Hepatocytes are the functional parenchymal cells of the liver, that form 70-80% of its mass. They are arranged into cords, separated by vascular sinusoids, where the incoming blood mixes, as it is depicted in Figure 5.4. Hepatic cords look like intricately branching and interconnecting strings of cells which radiate out from the lobule center. An example of cords and sinusoids is shown in Figure 5.5.

The cells lying along side or draped across the liver sinusoids are called Kupffer cells,
Chapter 5. Partitioning With Prior Assumptions

Figure 5.4: Hepatocytes are arranged in cords, radiated out from the lobule center. Cords are separated by vascular sinusoids.

Figure 5.5: Example showing cords and sinusoids on a liver lobule. Kupffer cells lie along side or across the sinusoids.

see Figure 5.5\(^3\).

5.1.2 Hepatocyte Quantification

A liver has the unique ability to regenerate in response to injury or loss of liver mass. One well accepted way to describe the kinetics of this process is to quantify the relative proportion of dividing hepatocytes at different time points after the liver injury.

Dividing hepatocytes can be identified by special immunohistochemical staining techniques of the hepatocyte nuclei such as the BrdU-Staining (5-Bromo-2-deoxyuridine). A sample of the liver, about 0.5-1 cm\(^3\) in size, has to be formalinfixed and paraffinembedded. After cutting sections of about 4 – 6 µm thickness they are subjected to a special immunohistochemical staining procedure. Nuclei of dividing cells, hepatocytes, but also other non-parenchymal stromal cells, are marked in one color, e.g. red, whereas the nuclei of the non-dividing cells are marked with a counterstain, e.g. blue. The proliferation index can be calculated after determining the

\(^3\)Image courtesy of Southern Illinois University, School of Medicine.
number of proliferating hepatocytes with a red nucleus and the total number of 
hepatocytes with either a red or a blue nucleus.

5.1.3 Current Study Description

In the past, the proliferation index was determined by simple counting of proliferat-
ing and non-proliferating cells using a sample size of 1000 to 3000 hepatocytes. This 
is a time-consuming procedure requiring an experienced observer, who is trained to 
discriminate hepatocytes from the other cells types in the liver.

Recently, with the availability of digital photography the computer-assisted 
cell counting has gained popularity. The observer marks each cell to be in- 
cluded using image analysis tools, e.g. GIMP (http://www.gimp.org/) or Im-
age Tool (http://ddsdx.uthscsa.edu/dig/itdesc.html), and the marked events 
are enumerated by some software. The image overlaid with marked target 
cells is saved for documentation. There exist also semi-automatic and auto-
matic solutions based on image analysis systems used in clinical routine. For 
example, there has been recently presented a macro based on a commer-
cially available software (http://industrial-microscope.olympus-global.com/
en/ga/product/analysisfive/) [DKH+09].

Such solutions based on the analysis of small 2D samples from a large 3D object suffer 
from the sampling bias problem. Analysis of small 2D samples is only valid, if target 
events are distributed homogeneously in the whole 3D object. This assumption does 
not hold generally for liver regeneration, as this process is subject to local regulation. 
Spatial distribution of proliferating hepatocytes within the smallest functional liver 
unit, the lobules, depends on the hepatic zone and can vary substantially throughout 
the liver. Hence, the entire 3D object needs to be looked at, which is, again, a tedious 
and time-consuming effort when keeping the user in the loop.

The ultimate solution to this problem and our overall project goal is to subject 
serial sections of the whole sample to an automatic quantitative assessment. The 
first step towards this full automatization is to detect the proliferation index, i. e., 
the ratio of the number of proliferating cells and the overall number of cells, in whole 
sections of the rat liver; an example image is shown in Figure 5.6. To accomplish 
this goal a series of tasks needs to be tackled. First, the zones of interest containing 
hepatocyte information must be defined. Second, due to the sample size it has to be 
divided into smaller parts. Third, the parts containing hepatocyte information has
Chapter 5. Partitioning With Prior Assumptions

to be processed, i.e., the nuclei must be detected in each image. Fourth, the nuclei quantification information must be accumulated for the whole section.

In this work, we address the hepatocyte quantification task. The specific aim of this task was to develop an automatic approach, that is fast, robust to different image appearances, and allows to analyze batches of images without additional user interaction.

5.1.4 Material

In this study, liver samples from rats subjected to 70% liver resection have been used. Digitized images of the stained sections have been taken at a 200-fold magnification. An example of such an image is shown in Figure 5.6.

The rat liver consists of parenchymal cells (hepatocytes) and non-parenchymal cells (e.g.: bile duct cells, Kupffer cells, sinusoidal endothelial cells, lymphocytes). According to the portal blood flow, the hepatic parenchyma is divided into 3 zones. Zone 1, also called portal zone, is surrounding the portal tract (PT), a complex histological structure consisting of several vascular components such as a portal vein, a hepatic artery and several bile ducts embedded in histiocytic cells and connective tissue. Zone 2 is surrounding the central vein (CV), which is draining the smallest functional unit of the liver, the hepatic lobule. Zone 3 is the midzonal area between zone 1 and zone 2. Despite their anatomical and functional differences, for excluding areas with non-parenchymal structures from further proliferation analysis portal...
tract and central vein are considered to be similar structures and are referred to as “veinous structures” throughout the text.

ROIs with a resolution of $2576 \times 1932$ pixels are selected from these three liver zones. We have made a series of tests on images from eight different datasets. Each dataset represents regions of interest (ROIs) from the liver samples of one animal. These datasets belong to two groups that had different contact time of the section with staining solutions, which resulted in variations in image contrast. The datasets within each group are also subject to variations in color properties, which occur due to some differences in the histological processing of the tissue sample and may also occur during image acquisition (camera settings).

We keep the following naming convention: each image name has a number that denotes the dataset. For example, D1 denotes the image from liver zone 3 of dataset 1. PT and ZV in the names belong to the images that were taken from the liver zones 1 and 2. Liver zones 1 and 2 contain vessel structures.

## 5.2 Related Work

Although a lot of automatic image processing approaches for histological sections have been developed [LL03], it is difficult to compare them to each other due to the difference of the staining methods applied to the data and the related image analysis problems.

For example, one of the popular directions is the fuzzy clustering. Begelman et al. [BG+04] propose to use a semi-automatically trained fuzzy logic engine that uses color and shape features for segmentation. Kernel-based clustering combined with genetic algorithm allows to segment the cell boundaries, however, severe cell occlusions can be problematic for such an approach [YJ01]. Methods using basic fuzzy C-means clustering [LSP03] or adaptive thresholding [P+06] seem to have difficulties when dealing with images that have large variability in the color properties.

A more sophisticated approach based on active contour models [BL98] seems to be less sensitive to staining variability and produces decent results as long as the nuclei are non-overlapping. Overlapping nuclei are not handled appropriately. A level-set based method has been proposed in [NY09]. It is suitable for cell images with high degree of cell overlapping, as it can detect not only the nuclei of isolated cells but also nuclei in cell clusters with high sensitivity [NY09]. However, the geometric
deformable models appear to be rather computationally expensive, so practically inapplicable for our goals, where speed of processing was a mandatory requirement.

Another popular approach is the morphological watershed-based segmentation, to deal with overlapping nuclei [SHB08]. Its task consists in extracting objects from a gray-level image as precisely as possible [VS91]. The initial watershed method applied to gradient of a gray-level image suffers from the problem of oversegmentation. The correct way to use watersheds for grayscale image segmentation consists in first detecting markers of the objects to be extracted. The design of robust marker detection techniques involves the use of knowledge specific to the series of images under study. Not only object markers, but also background markers need to be extracted [VS91, Beu91, BM92]. For example, in [CB08], the authors propose to use the intensity properties in the overlapping areas to locate the initial watersheds markers, which generally does not hold for our type of data. For our case, the task of segmentation has been narrowed to counting cells, so the appropriate marker definition practically solves the problem.

Naik et al. [NDA+08] proposed to integrate several levels of image information to segment gland and nuclei in breast and prostate cancer images. For nuclei segmentation the template matching scheme has been used. The templates have been selected according to the size and the elliptical shape of the nuclei, which correlates in some sense to the last step of our processing pipeline.

Datar et al. [DPC08] proposed to use hierarchical self-organizing maps to segment four types of tissue, namely glands, epithelia, stroma, and nuclei, but it is not suitable for separating the individual cells in order to determine their quantity.

A compact Hough transform-based approach [MRB98] has been successfully utilized for nuclei segmentation. The authors used thresholding to reduce the search space, applied Hough transformation to detect the nuclei centroids and maximized the likelihood function to correct the boundary detection.

Though the variety of the proposed methods is huge, most of them are aimed to detect the boundaries of the nuclei cells as precisely as possible, which is actually not needed for our purposes. The complexity and computational costs of these methods are not necessary and not justified in our case. In addition, the methods have either problems with overlapping nuclei or are strongly dependent on the data staining. Our task was to develop an approach that is fast, robust to different data appearances within the staining specific to our project, and whose aim is not to detect cell boundaries but rather to evaluate the number of cells, in particular, that it can deal with overlapping cells appropriately.
5.3 Processing Pipeline

To solve the task of automatic processing of the ROIs of histological sections, we have developed an algorithm and created a tool which can assess the total number of events (total hepatocyte nuclei) as well as the number of positive events (proliferating red-stained hepatocyte nuclei). It calculates the relative proportion of the positive events, namely the ratio between proliferating and total hepatocytes (BrdU-LI - 5- Bromo-2-deoxyuridine labeling index, also referred as “proliferation index”) automatically in one “processing run”. To do so, it eliminates morphological structures impairing the analysis (central vein or portal tracts). It creates a batch calculation allowing to analyze several images without user interaction, which forms the basis for evaluating a whole section. Moreover, it facilitate the validation by creating a tool for determining statistical measures of quantification quality, namely sensitivity and specificity.

In Figure 5.7, the processing pipeline of our algorithm is depicted.

The initial images appear to be quite noisy and hinder a satisfying direct hepatocyte detection. We propose to process images first in such a way that the amount of all other structures besides the hepatocytes is reduced, while the resulting image still contains all important information about them. For the applied staining of the hepatocytes, it can be observed that all of them are visible in the red color channel, while the proliferating hepatocytes are easily distinguishable in the blue color channel. We apply the smoothing filters to the corresponding channels of the image. Furthermore, we compare five methods for the image smoothing which are described in detail in Section 5.3.1 and select the most suitable for the current task methods.

The second part of our algorithm consists of a sequence of processing steps that are applied to the smoothed images. We automatically distinguish between the hepatocytes, the vascular structure, and the background by utilizing automatic thresholding, see Section 5.3.2.

Then we use morphological operations and size and roundness filters to exclude the non-hepatocyte regions, see Section 5.3.3.

In order to estimate the number of different hepatocytes in an image, we still need to handle overlapping hepatocyte regions appropriately. These problems are dealt with by a Hough transform, see Section 5.3.4.
Figure 5.7: Flowchart for automatic hepatocyte quantification. Our processing pipeline consists of four main steps (left). The detection of total HC-nuclei in the red image channel (middle) and proliferating HC-nuclei in the blue image channel (right) is shown side-by-side. The vein region is smoothed and eliminated. For proliferating HC-nuclei detection, the vein mask is excluded on the thresholding step.

As a result, we obtain circles that depict the position and size of the detected hepatocytes. The circles are counted, their ratio is determined, and this number is the resulting output.

5.3.1 Smoothing

We have chosen to apply and compare the following techniques for this preprocessing step: Gaussian smoothing, non-linear Gaussian (Bilateral) smoothing, median...
filtering, anisotropic diffusion, and minimum description length (MDL) partitioning (see Section 4) which allows for simultaneous denoising and segmentation. Our choice has been motivated by the following considerations. The Gaussian filtering technique is a simple standard approach to image denoising. Median, anisotropic diffusion, and non-linear Gaussian filters are non-linear, denoising, and edge-preserving approaches. In addition, we decided to use an MDL segmentation technique with a piecewise constant image model to check, whether a local spatial segmentation algorithm can give some advantages in this task. In the following, we describe the methods in more detail. The MDL-based partitioning is described in Section 4.

**Gaussian Filter**  The Gaussian smoothing operator is a 2D convolution operator. It removes detail and noise and thus blurs images. The idea of Gaussian smoothing is to use this 2D distribution as a “point-spread” function, which is achieved by convolution. It uses a kernel that represents the shape of a Gaussian (“bell-shaped”) hump. A 2D Gaussian has the form:

\[
G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),
\]

where \((x, y)\) are spatial positions and \(\sigma^2\) is the (constant) noise variance. Using the property of separability of the Gaussian filter, we subsequently apply a 1D Gaussian filter in each dimension. The width of the kernel window is taken proportional to \(\sigma\).

So, the discrete convolution with a kernel window \(G_m\) of size \(s_1 \times s_2\) for the pixel with coordinates \((x, y)\) looks as follows:

\[
(I \ast G_m)[x, y] = \sum_{i}^{s_1} g_1[i] \sum_{j}^{s_2} g_2[j] I[x - j, y - i],
\]

where \(g_1\) and \(g_2\) are the correspondent normalized 1D kernels.

**Nonlinear Gaussian Filter**  The basic idea of the algorithm is that the concept of Gaussian smoothing is extended to the intensity domain by weighting the filter coefficients with their corresponding relative pixel intensities [TM98]. Pixels that are very different in intensity from the central one have smaller weights even though they can lie spatially close to the central pixel. It is applied as two Gaussian filters at a localized pixel neighbourhood, one in the spatial domain (called the domain filter) and one in the intensity domain (the range filter). Let \(f\) be the original image intensity function, then the bilateral filtering of it is described as:

\[
G_{\sigma_d,\sigma_r} f(x) = k^{-1} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\xi) g_{\sigma_d}(\xi - x) g_{\sigma_r}(f(\xi) - f(x)) d\xi,
\]
where \( g_{\sigma_d} \) denotes the Gaussian convolution in the spatial domain, and \( g_{\sigma_r} \) denotes the Gaussian convolution in the range domain, and \( k \) is the normalization, calculated as:

\[
    k(x) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g_{\sigma_d}(\xi - x) g_{\sigma_r}(f(\xi) - f(x)) \, d\xi
\]

(5.4)

There are different implementations of Bilateral filter in the literature [AMG98, TM98, PD06]. We propose to use two different variations for it. One is the bilateral filter chain as implemented by Aurich and Weule [AMG98]. They proposed to use several filter steps with different parameters. It allows to get rid of fine details while preserving the edges of the coarser structures. The first filter step serves mainly for reducing the contrast of the fine details. The following steps continue this reduction, but sharpen at the same time the edges of the coarser structures with may have been blurred by the first step. The parameter \( \sigma_d \) of the first step should be as small as possible in order to avoid unnecessary blurring of coarser structures, but big enough to blur the fine structures. \( \sigma_r \) of the first step determines approximately the maximal contrast the fine structures may have in order to be finally eliminated. In the next steps, \( \sigma_d \) must be increased and \( \sigma_r \) must be decreased[AMG98]. The authors suggested to double \( \sigma_d \) and to halve \( \sigma_r \). However, the computational costs of this implementation are quite high, and, therefore, we also applied another implementation, namely, the fast approximation of the bilateral filter introduced by Paris et al. [PD06]. The fast approximation is based on downsampling the image, applying the convolution and upsampling it back again. The average (box) downsampling and linear upsampling are used. Such a technique is favourable in terms of speed.

**Median Filter** The idea of median filtering is to examine a sample of the input and to decide whether it is a good representative for the signal. To do so, a window consisting of an odd number of samples is used, whose center lies at the currently examined pixel. For each pixel in the image, the values in the window are sorted numerically and the median value, i.e., the value of the sample located in the center of the window after sorting, is selected as the output.

**Anisotropic Diffusion** Anisotropic diffusion filter is a non-linear smoothing filter that encourages intraregion smoothing while inhibiting interregion smoothing. It was initially formulated by Perona and Malik [PM90]. The continuous form for anisotropic diffusion is given by

\[
    \frac{\partial I}{\partial t} = \text{div} \left( g(\|\nabla I\|) \cdot \nabla I \right),
\]

(5.5)
where $I$ denotes the image, $t$ is an artificially introduced time parameter, which shows the evolution of the image with time and allows to solve the partial differential equation, $\text{div}$ denotes the divergence, and function $g$ is defined by

$$g(\|\nabla I\|) = \exp \left(-\left(\frac{\|\nabla I\|}{K}\right)^2\right)$$

with flow constant $K$. The discrete version of Equation (5.5) is given by

$$I_{t+1}^i - I_t^i + \lambda \sum_{j \in N_i} \left[ (I_{t}^i - I_{t}^j) \exp \left(-\left(\frac{(I_{t}^i - I_{t}^j)}{K}\right)^2\right) \right] = 0,$$

where $\lambda$ is a normalization factor. The discrete version is used to iteratively compute the image values $I_{t+1}^i$ at iteration step $t + 1$ from the image $I_t^i$ at iteration step $t$, where $I_t^0$ describes the original image values. The Perona-Malik equation is ill-posed and needs regularization for good results [CLMC92].

### 5.3.2 Thresholding

The straightforward approach is simple or interval thresholding, where the user must select a threshold or threshold interval, and all the data is divided into two classes: below (or equal to) and above the threshold. Such methods need threshold adjustment for each image and can not be considered for an automatic pipeline.

We applied automatic approaches for the threshold detection, namely, Otsu [Ots79] and Expectation Maximization [Mac02] methods.

**Otsu method** The original Otsu method performs histogram-based image thresholding. It assumes that the image contains two classes of pixels (background and foreground) and calculates the optimum threshold, separating the classes, so that their within-class variance is minimized. The within-class variance is a weighted sum of two class variances:

$$\sigma_{\text{within}}^2(k) = P_i(k)\sigma_i^2(k) + P_2(k)\sigma_2^2(k), \quad (5.6)$$

where $k$ is the threshold and $P_i$ and $\sigma_i^2$ are the probability and the variance of class $i$, respectively. Otsu showed that minimizing the within-class variance is the same as maximizing the between-class variance, which is defined as

$$\sigma_{\text{between}}^2 = \sigma^2 - \sigma_{\text{within}}^2(k) = P_1(k)(\mu_1(k) - \mu_T)^2 + P_2(k)(\mu_2(k) - \mu_T)^2 = P_1(k)P_2(k)(\mu_1(k) - \mu_2(k))^2, \quad (5.7)$$
where \( \mu_i \) is the mean of class \( i \), \( \mu_T \) is the total mean of the image, and \( \sigma \) is the total variance of the image. The original binary thresholding is not sufficient for processing images containing vascular structures [ISD+08], as there are not only two, but three classes in such images, namely nucleus, vein, and background. For such a purpose, the multi-level Otsu thresholding can be used [LCC01]. In such a case, the maximization problem becomes more complicated. Assuming that there are \( M - 1 \) thresholds, \( k_1, ..., k_{M-1} \), which divide the image into \( M \) classes, the optimal thresholds are chosen by maximizing \( \sigma^2_{\text{between}} \) as follows:

\[
\sigma^2_{\text{between}} = \sum_{i=1}^{M} P_i(k_i)(\mu_i(k_i) - \mu_T)^2 = \sum_{i=1}^{M} P_i(k_i)\mu^2_i(k_i) - \mu^2_T.
\] (5.8)

**Expectation Maximization**  Our experiments showed that applying multi-level Otsu thresholding to our segmentation task has some severe drawbacks (see Section 5.5). Thus, we decided to replace it by an automatic thresholding step based on expectation maximization (EM) [Mac02].

Let \( y \) denote incomplete data consisting of observable variables, and let \( z \) denote the missing data. \( z \) can be either missing measurements or a hidden variable that would make the problem easier if its values were known. \( y \) and \( z \) together form the complete data. Let \( p(y, z|\theta) \) denote the joint probability density function of the complete data with parameters given by vector \( \theta \). The conditional distribution of the missing data is expressed as

\[
p(z|y, \theta) = \frac{p(y, z|\theta)}{p(y|\theta)} = \frac{p(y|z, \theta)p(z|\theta)}{\int p(y|\hat{z}, \theta)p(\hat{z}|\theta)d\hat{z}}.
\] (5.9)

The log-likelihood of the complete data is

\[
L(\theta) = \log p(y, z|\theta).
\] (5.10)

As we do not know the values of \( z \), we can find a posteriori estimates of the probabilities for various values of the unknown data. For each set of \( z \) there is a likelihood value for \( \theta \), and we can calculate an expected value of the likelihood with the given values of \( y \) and a previously assumed value of \( \theta_t \). The expected value of the log-likelihood at timestep \( t \) is

\[
Q(\theta, \theta_t) = \sum_z p(z|y, \theta_t) \log p(y, z|\theta),
\] (5.11)
where \( \theta_t \) denotes the fixed values from timestep \( t \) and \( \theta \) denotes a parameter. The EM algorithm iteratively improves the value of \( Q(\theta) \) by computing the estimate of \( \theta_{t+1} \) from \( \theta_t \) as

\[
\theta_{t+1} = \arg \max_{\theta} Q(\theta, \theta_t).
\]

(5.12)

The value of \( \theta_{t+1} \) is the value that maximizes the conditional expectation of the complete data log-likelihood given the observed values under the previous parameter values. The EM consists basically of two steps: expectation and maximization. In the expectation step, the expected log-likelihood is calculated given the parameters from the previous step, and in the maximization step, the next parameter values that maximize the log-likelihood are calculated.

In our case, we consider the Gaussian mixture model for the EM algorithm. It assumes that the image pixel intensities \( y_1, \ldots, y_m \in \mathbb{R} \) are samples of independent observations from \( n \) Gaussian distributions with unknown parameters. Let \( z_j \in \{1, 2, \ldots, n\} \) denote the index number of the Gaussian distribution that \( y_j \) has been drawn from.

The probability density function of the \( i \)-th one-dimensional Gaussian distribution is

\[
p(x|z = i, \theta) = N(\mu_i, \sigma_i) = (2\pi\sigma_i)^{-1/2} \exp \left( -\frac{(x - \mu_i)^2}{2\sigma_i} \right),
\]

(5.13)

where \( \theta = \{\mu_1, \ldots, \mu_n, \sigma_1, \ldots, \sigma_n, p(z = 1), \ldots, p(z = n)\} \) with \( \mu_i \) and \( \sigma_i \) being the mean and the variance of the \( i \)-th Gaussian distribution, and \( p(z = i) \) being the \( i \)-th class membership probability (or proportion). The class proportion is the probability for each Gaussian distribution being drawn from all observations.

The log-likelihood of the joint event that is to be maximized can be written as

\[
Q(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} p(z_j = i|y_j, \theta_t) \ln(p(z_j = i, y_j|\theta)).
\]

(5.14)

Before the first iteration, one must find initial estimates of \( \mu, \sigma \), and the class proportions. To compute the initial \( \mu \), the histogram is divided into \( n \) equal parts and \( \mu_i \) is taken as a mean of each part. The initial \( \sigma_i \) are taken to be equal to the maximum value in the image. The initial class proportions are equal to \( \frac{1}{n} \). Then, the main goal is to identify the unknown distribution parameters \( \theta \).

In the expectation step, the conditional distribution with respect to the current unknown parameter estimates is computed by

\[
p(z_j = i|y_j, \theta_t) = \frac{p(y_j|z_j = i, \theta_t)p(z_j = i|\theta_t)}{\sum_{k=1}^{n} p(y_j|z_j = k, \theta_t)p(z_j = k|\theta_t)},
\]

(5.15)
where \( \theta_t \) is the estimation of the parameters on iteration \( t \), \( p(y_j|z_j=i, \theta_t) = N(\mu_i, \sigma_i) \) is the Gaussian probability density function at \( y_j \), and \( p(z_j=i|\theta_t) \) is the probability of the class \( i \) for \( y_j \). The values of \( \mu, \sigma, \) and \( p \) are taken from the previous maximization step.

In the maximization step, the values of \( \mu, \sigma, \) and \( p \) which maximize the log-likelihood are re-estimated:

\[
\mu_i = \frac{\sum_{j=1}^{m} p(z_j = i|y_j, \theta_t)y_j}{\sum_{j=1}^{m} p(z_j = i|y_j, \theta_t)}, \quad (5.16)
\]

\[
\sigma_i = \frac{\sum_{j=1}^{m} p(z_j = i|y_j, \theta_t)(y_j - \mu_j)^2}{\sum_{j=1}^{m} p(z_j = i|y_j, \theta_t)}, \quad (5.17)
\]

\[
p(z = i|\theta) = \frac{\sum_{j=1}^{m} p(z_j = i|y_j, \theta_t)}{\sum_{k=1}^{n} \sum_{j=1}^{m} p(z_j = k|y_j, \theta_t)}. \quad (5.18)
\]

These estimates now become \( \theta_{t+1} \) and are used in the next estimation step.

To speed up the computations, we assume not the image pixels, but the image histogram values to be \( y \). Thus, the corresponding probabilities for \( y_j \) are taken into account in each expectation step when computing \( p(y_j|z_j = i, \theta_t) \).

When the difference between the log-likelihood values \( Q(\theta_t) \) and \( Q(\theta_{t+1}) \) found in two iterations is lower than a certain accuracy value, the algorithm stops. The algorithm execution time varies depending on the selected accuracy. For our application, an accuracy value of \( 10^{-4} \) was enough to obtain the desired results in a short amount of time (less than a second for a typical ROI image).

In the current implementation, the choice of the number of classes in the image is performed by the user and indicated to our tool using a filename convention. The filenames must indicate, which files contain vein structures. After the automatic thresholding is completed, we obtain the proper mask that consists either of two or three classes. We place each class in a separate image channel and combine the RGB mask image (see Figure 5.7).
5.3.3 Connected Components Processing

To discriminate the candidates for hepatocytes from all other structures of the mask image, we apply some standard algorithms, namely morphological operations [Soi99] and size and roundness filters.

First, the Fill Holes filter [Soi99] is applied to erase potentially existing holes in regions and, hence, make the regions simply-connected. Second, the morphological operation of erosion [Soi99] shrinks the regions to separate “weakly connected” regions, i.e., regions with a thin and fragile connectedness. Then, the dilation operation [Soi99] can be applied to expand the regions and restore the original region sizes. Third, we use connected-component labeling in the image [SHB08] to analyze each connected component according to its area, i.e., we threshold the connected components according to their area. In Figure 5.8, the impact of each processing step is shown.

Figure 5.8: Impact of each connected component processing step.
Finally, we eliminate the non-round regions by excluding the connected components that have a form factor lower than a certain threshold. The form factor $F$ given by

$$F = \frac{4\pi A}{P^2},$$

is equal to 1 for circles and approaches 0 for elongated regions, where $A$ is the area of the connected component and $P$ is its perimeter. Perimeter $P$ is computed by summing the distances of all neighboring pairs of pixels along the border of the connected component.

In case there are vein structures present in the image, those are being eliminated using the vein structure mask obtained by the thresholding step. We compute the ratio between the area of the largest vein component and the whole area covered by vein, then smooth the vein region according to that ratio, and subtract the resulting mask from the hepatocyte channel mask. Subtracting the mask means that we set all those pixels to black in the hepatocyte mask that are white in the vein mask, see Figure 5.7. Smoothing is needed to connect disjoint vein regions and to exclude inner part of the vein. To smooth the vein mask we, again, apply a bilateral filter.

Such a vein exclusion technique can handle vein structures that consist of several disconnected regions lying close together. If the vein region consists of one major connected component, smoothing is not necessary and can be omitted. However, such a connected vein regions may contain some blood, which appears as holes in the vein structure mask. A Fill Holes filter [Soi99] takes care removing the blood. In the current implementation, the type of the vein exclusion is defined by a parameter in the pipeline settings.

The vein exclusion is applied to the red image channel during the detection of the whole number of hepatocytes. In the blue image channel the vein mask that was found in the red channel is discarded in the thresholding step, see Figure 5.7.

### 5.3.4 Hepatocyte detection

To calculate the number of circular, possibly overlapping regions in the image, we utilize the Hough transformation [Hou59], looking for circles within a pre-defined radius interval. The algorithm operates in the parameter space of the circles. The parametric equation of a circle with radius $r$ and center $(x_0, y_0)$ is given by

$$x = x_0 + r \cos \theta$$

$$y = y_0 + r \sin \theta.$$
Knowing the \((x, y)\) pairs, one need to obtain the position of the circle centers \((x_0, y_0)\). This is done by filling a 2D voting matrix \(acc\). The voting matrix is initialized with zeros. Then, for each non-black pixel \((x, y)\), pairs \((x_0, y_0)\) are calculated and the entries \(acc[x_0, y_0]\) in the voting matrix are increased by one. To speed up the procedure, we fill the vote entries only for those \((x, y)\) that lie at the boundary of a region in the direction of the gradient, such that the potential locations of the circle centers \((x_0, y_0)\) coincide with the hepatocyte mask, i.e., pixel values at \((x_0, y_0)\) are not equal to zero. The local maxima in the accumulator correspond to the centers of the circular structures in the input image. The voting matrix is smoothed (using Gaussian smoothing) for more adequate results. The search for circles stops when the height of the currently detected maximum in the accumulator matrix is smaller than a certain percentage of the first detected maximum. We took 50% for all our examples.

In a post-processing step, we analyze the obtained circle list. We exclude all those circles whose center lies outside the image or does not lie inside the hepatocyte region mask. Moreover, from a group of circles lying close together we exclude the ones having less overlap with the hepatocyte region mask. The closeness of the circles is defined as

\[
dist(c_1, c_2) \leq F_{cl} \max(r_1, r_2)^2,
\]

where \(c_1, c_2\) denote the centers of the circles, \(r_1, r_2\) denote the circles’ radii, and \(F_{cl}\) is a user-defined parameter. We took \(F_{cl} = 0.9\) for all our examples.

### 5.4 Software GUI and functionality description

The presented system consists of two parts, the main application for hepatocyte quantification, and the extension for creating the groundtruth masks, calculating sensitivity and specificity measures obtained by overlaying groundtruth mask and hepatocyte quantification results. The system has been implemented, using MeVisLab, Software for Medical Image Processing and Visualization (see [http://www.mevislab.de](http://www.mevislab.de)).

#### 5.4.1 Hepatocyte Quantification

The main application consists of two display panels, showing the initial image and the current processing result, and the parameter panel (see Figure 5.9).
Chapter 5. Partitioning With Prior Assumptions

Figure 5.9: Main application GUI. The main application panel consists of two viewers, showing initial image and intermediate result and several functional panels for each processing step.

In the parameter panel there are “Initial Settings” panel, four functional panels, namely, “Smoothing”, “Thresholding”, “Hepatocyte Detection”, and “Results” panel. In the initial panel, the user selects the mode and initial data for processing. There are two modes for the application: basic and extended. The extended mode is used for the step-by-step one file processing and to obtain either the total number of hepatocytes or the number of proliferating hepatocytes. The user selects a file and switches from one panel to another, setting the parameters (or using the ones set by default) and getting the result in each functional panel. The functional panels correspond to the algorithmic solution pipeline described above. The result of each panel can be stored into a png file. This semi-automatic approach is usually useful for selecting the optimal parameters and testing on which step of the pipeline there are some problems. After the parameters of the algorithms and algorithm chains are adjusted, they can be stored in the settings file and used for the future processing. The settings on each processing step are saved into a log file. On
the last processing step (“Results” panel) the binary mask of detected hepatocytes and the overlay with the initial image can be stored.

In the basic mode, a batch of files can be processed. The user must select the settings (load the correspondent settings file or use the pre-defined ones), load the data and press the “Process Batch” button. The log directory is created for each processed file, and the results are put into it. Each folder includes the name of the processed file and the date and time of the processing. In this mode, the user can select what type of cells or proliferation index to be detected. If the detection of the proliferation index is chosen, after two runs detecting the total number of nucleus and the number of proliferating nucleus, the list of total hepatocytes is supplemented with the proliferating cells that were found in the second run. In the result folder, a log file and three subfolders are created, namely ‘all’, ‘proliferating’, and ‘all_corrected’. The results (overlays, binary masks, xml-lists) for total hepatocytes, proliferating hepatocytes, and supplementing results are stored in there.

The results of the calculations (total number of hepatocytes, number of proliferating hepatocytes and the proliferation index) are stored in the “comma-separated-value” file, created in the batch root folder.

### 5.4.2 Recognition Rates Detection

For validation of the results calculated with the Hepatocyte detection part, we developed an extension that allows to create manual detection mask and store it (see Figure 5.10). When the two masks (manual and automatic detection results) have been computed, one can overlay them and obtain the sensitivity and specificity rates automatically. The overlay result can be also stored into a png file.

### 5.5 Results and Discussion

Images of stained sections are subject to variations in the color properties due to some differences in the histological processing of the tissue sample, consisting of several non-automatic steps. Variations may include thickness of the section and contact time of the section with different staining solutions, but may also occur during image acquisition (camera settings) [ISD+08].

We have made a series of tests on images from 8 different datasets. Each dataset represents ROIs from the liver samples of one animal. The evaluated images have
Figure 5.10: The extension tool allows to create manual groundtruth masks and compute the statistical measures.

differences in color properties. We tested our processing pipeline on the detection of the number of total HC and the number of proliferating HC. The proliferation index detection takes on an Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz computer with 2GB DDR2 for one ROI image with a resolution of 2576 × 1932 on average 135 seconds. The Hough transformation step appears to be the most time-consuming.

Our evaluations are presented in Table 5.1, 5.2, and 5.3. The following notations for the headings are used: “Detected” means the number of circles found by Hough transform; “TP” is the number of True Positive hepatocytes, which is the result of
the overlay of detected circles and the user expectations; “FN” denotes the number of False Negative hepatocytes, which is the difference between the Ground Truth Positives and the True Positives; “FP” stands for the number of False Positive hepatocytes, which is the difference between “Detected” and “TP”; “TN” is the number of True Negative hepatocytes, which is calculated as the difference between FP and the number of Ground Truth Negatives; and User P and N are the number of Positive and Negative hepatocytes manually specified by the expert. The most important numbers are the computed sensitivities and specificities. Sensitivity is defined by \( TP / (TP + FN) \) and measures the proportion of actual positives, while specificity is defined by \( TN / (TN + FP) \) and measures the proportion of true negatives.

### 5.5.1 Smoothing filter comparison

To discuss the results we obtain using our approach with the different smoothing methods, we focus on the counting of the total number of hepatocytes, which appears to be a challenging quantification task, as the hepatocytes stained with blue are often very hard to visually distinguish from the background. For the tests we use four images that contain no vein. An example image is given in Figure 5.16. For the tests with different smoothing techniques we always chose exactly the same parameters for all the other steps of our pipeline.

While testing the smoothing filters with different parameter settings, we intended to find a filter that will give results with high sensitivity and specificity for images with different quality using pre-defined settings, so that the user does not need to adjust the parameters manually for each image. For the smoothing algorithms we empirically defined such parameter intervals for our data, so that choosing the exact parameter values within these intervals the user will obtain acceptable results. Our strategy for parameter selection is schematically shown in Figure 5.11.

We present our observations in Figure 5.12 on an subimage from image D3. Image D3 is of low quality but still accepted by medical experts.

Since our smoothing step is followed by an automatic thresholding step, the smoothing must be sufficient, otherwise the thresholding will most likely fail. First, we selected rather low parameter values and applied thresholding to the smoothed image. In Figure 5.13, the smoothing result for Median filter with a \( 5 \times 5 \) window and the correspondent thresholding result which documents our observations are shown. Thereafter, we selected quite high parameter values, for which the object boundaries were oversmoothed and applied the thresholding again. When the filter
Chapter 5. Partitioning With Prior Assumptions

Figure 5.11: While selecting the parameter settings we tried to define the appropriate parameter intervals.

parameters were too high, the cells lying close were merged together and detected as one connected component, that could be either rejected on the connected component analysis step as a too big or too non-round object (this would cause a number of false negatives) or the Hough transformation would detect there a number of false positives.

After defining (approximately) the value intervals for the filter parameters, we tested the filters slightly changing the parameters and observed that it does not affect much the thresholding result. While selecting the smoothing parameters, we also observed that even though the smoothing results visually look slightly different for the boundary preserving filters, the thresholding result is very similar. It is documented in Figure 5.15.

Finally, we selected the parameters that allow to achieve close to the best result. We took the following parameters for the smoothing filters. For Gaussian smoothing
Chapter 5. Partitioning With Prior Assumptions

Figure 5.13: When smoothing is not sufficient, thresholding will most likely fail.

(a) Smoothing result for red channel of subimage in Figure 5.12. Median filter with $5 \times 5$ window was applied.

(b) Thresholding result for the smoothed image.

Figure 5.14: When an image is oversmoothed, the cells lying close together will merge into a single connected component, which will be most likely rejected on the connected component analysis step.

(a) Smoothing result for red channel of subimage in Figure 5.12. Gaussian filter with $\sigma = 15$ was applied.

(b) Thresholding result for the smoothed image.

$\sigma = 4$; for Median filtering the kernel is $14 \times 14$; for Perona-Malik anisotropic diffusion the time step size is taken 7.4, the number of steps is 4, the edge parameter of the diffusivity function is 3.8, in each diffusion step the image is processed with Gaussian
Chapter 5. Partitioning With Prior Assumptions

(a) Smoothing result for red channel of subimage in Figure 5.12 for the anisotropic diffusion filter.

(b) Smoothing result for red channel of subimage in Figure 5.12 for the non-linear Gaussian filter.

(c) Smoothing result for red channel of subimage in Figure 5.12 for the MDL filter.

(d) Thresholding result for Figure 5.15a.

(e) Thresholding result for Figure 5.15b.

(f) Thresholding result for Figure 5.15c.

Figure 5.15: Smoothing and thresholding results for three boundary preserving filters.

smoothing with parameter, equal 1.54; for bilateral filtering start values for $\sigma_x$ and $\sigma_z$ are 10 and 100 respectively and the chain of several filtering steps is used, where $\sigma_x$ is increased and $\sigma_z$ is decreased. In addition, we compared our results to the results that can be obtained without preprocessing.

In the case of no pre-smoothing of the images, we observed that automatic thresholding using Otsu method does not work. We had to replace it with manual thresholding for each image. Thus, it was not possible to keep the entire pipeline fully automated.

In fact, it turned out that the selection of the manual threshold was rather cumbersome, as we needed to tune the threshold for each image individually and the tuning was not as intuitive as expected.

For the MDL segmentation algorithm we had to cut regions of $1024 \times 1024$ out of the full-sized images and to evaluate the true and false positive rates on the smaller images, since the MDL-based method was implemented on the GPU which was
limited for an NVidia Quadro FX 4500 video card to this image size. The results of our evaluation are presented in Table 5.1.

In Figure 5.17, several examples of our results and the manually defined ground truth images are depicted in an overlaid fashion. The hepatocytes marked with green dots represent the ground truth, while the red circles are the output of the Hough transform after applying our entire processing pipeline. The results for MDL segmentation methods are presented in Figure 5.18b. The images visually document the findings in Table 5.1 by showing the results when applying different smoothing methods side by side.

As can be observed from Table 5.1, appropriate smoothing as a preprocessing step allows for reasonable quantification results. In general, such boundary preserving filters, as anisotropic diffusion and non-linear Gaussian (bilateral) smoothing appear to be the most suitable methods for the given type of data. The MDL-based approach allows to smooth the image and reduce the number of colors (see Figure 5.18a) while preserving the important details, which is necessary to achieve a good segmentation. However, for the given quantification task, the MDL method does not give better results, as the overall goal is just the number of hepatocytes and not their perfect boundaries. Hence, the computational costs for the minimization procedure in MDL segmentation is not justified for our purposes.

Median filtering is rather computationally expensive especially for bigger window sizes, as it requires sorting of the window elements to detect the median.

We decided to use the non-linear Gaussian (bilateral) filter as an essential part for the hepatocyte quantification tool. The faster implementation by Paris et al. [PD06] is favorable in terms of speed, when compared to the implementation by Aurich et al. [AMG98], but introduces some distortions in the image for the slight smoothing, which can affect the subsequent automatic thresholding step negatively and may
<table>
<thead>
<tr>
<th>Image</th>
<th>Filter</th>
<th>Detected</th>
<th>TP</th>
<th>FN</th>
<th>FP</th>
<th>TN</th>
<th>Sens.</th>
<th>Spec.</th>
<th>User P/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>None</td>
<td>224</td>
<td>211</td>
<td>32</td>
<td>13</td>
<td>150</td>
<td>0.87</td>
<td>0.92</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1</td>
<td>Gaussian</td>
<td>237</td>
<td>220</td>
<td>23</td>
<td>17</td>
<td>146</td>
<td>0.91</td>
<td>0.9</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1</td>
<td>Bilateral</td>
<td>234</td>
<td>222</td>
<td>21</td>
<td>12</td>
<td>151</td>
<td>0.91</td>
<td>0.93</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1</td>
<td>Median</td>
<td>227</td>
<td>219</td>
<td>24</td>
<td>8</td>
<td>155</td>
<td>0.9</td>
<td>0.95</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1</td>
<td>An. Diff.</td>
<td>242</td>
<td>224</td>
<td>19</td>
<td>18</td>
<td>145</td>
<td>0.92</td>
<td>0.89</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1*</td>
<td>MDL</td>
<td>55</td>
<td>49</td>
<td>5</td>
<td>6</td>
<td>21</td>
<td>0.91</td>
<td>0.78</td>
<td>54 / 27</td>
</tr>
<tr>
<td>D2</td>
<td>None</td>
<td>218</td>
<td>193</td>
<td>22</td>
<td>25</td>
<td>133</td>
<td>0.9</td>
<td>0.84</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2</td>
<td>Gaussian</td>
<td>236</td>
<td>202</td>
<td>13</td>
<td>34</td>
<td>124</td>
<td>0.94</td>
<td>0.78</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2</td>
<td>Bilateral</td>
<td>221</td>
<td>197</td>
<td>18</td>
<td>24</td>
<td>134</td>
<td>0.92</td>
<td>0.85</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2</td>
<td>Median</td>
<td>199</td>
<td>184</td>
<td>31</td>
<td>15</td>
<td>143</td>
<td>0.86</td>
<td>0.91</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2</td>
<td>An. Diff.</td>
<td>228</td>
<td>199</td>
<td>16</td>
<td>29</td>
<td>129</td>
<td>0.93</td>
<td>0.82</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2*</td>
<td>MDL</td>
<td>43</td>
<td>36</td>
<td>5</td>
<td>7</td>
<td>29</td>
<td>0.88</td>
<td>0.81</td>
<td>41 / 36</td>
</tr>
<tr>
<td>D3</td>
<td>None</td>
<td>191</td>
<td>149</td>
<td>91</td>
<td>42</td>
<td>82</td>
<td>0.62</td>
<td>0.66</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3</td>
<td>Gaussian</td>
<td>277</td>
<td>220</td>
<td>57</td>
<td>20</td>
<td>67</td>
<td>0.91</td>
<td>0.54</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3</td>
<td>Bilateral</td>
<td>299</td>
<td>230</td>
<td>69</td>
<td>10</td>
<td>55</td>
<td>0.96</td>
<td>0.52</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3</td>
<td>Median</td>
<td>277</td>
<td>225</td>
<td>52</td>
<td>15</td>
<td>72</td>
<td>0.93</td>
<td>0.58</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3</td>
<td>An. Diff.</td>
<td>280</td>
<td>226</td>
<td>54</td>
<td>14</td>
<td>70</td>
<td>0.92</td>
<td>0.54</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3*</td>
<td>MDL</td>
<td>57</td>
<td>48</td>
<td>7</td>
<td>9</td>
<td>21</td>
<td>0.87</td>
<td>0.7</td>
<td>55 / 30</td>
</tr>
<tr>
<td>D4</td>
<td>None</td>
<td>225</td>
<td>198</td>
<td>47</td>
<td>27</td>
<td>116</td>
<td>0.81</td>
<td>0.81</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4</td>
<td>Gaussian</td>
<td>244</td>
<td>211</td>
<td>34</td>
<td>33</td>
<td>110</td>
<td>0.86</td>
<td>0.77</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4</td>
<td>Bilateral</td>
<td>252</td>
<td>222</td>
<td>23</td>
<td>30</td>
<td>113</td>
<td>0.91</td>
<td>0.79</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4</td>
<td>Median</td>
<td>246</td>
<td>219</td>
<td>26</td>
<td>27</td>
<td>116</td>
<td>0.89</td>
<td>0.81</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4</td>
<td>An. Diff.</td>
<td>258</td>
<td>222</td>
<td>23</td>
<td>36</td>
<td>107</td>
<td>0.91</td>
<td>0.75</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4*</td>
<td>MDL</td>
<td>49</td>
<td>44</td>
<td>8</td>
<td>5</td>
<td>21</td>
<td>0.85</td>
<td>0.81</td>
<td>52 / 26</td>
</tr>
</tbody>
</table>

Table 5.1: Hepatocyte quantification results for four different data sets comparing smoothing filters.

hinder the appropriate nucleus detection. However, we observed that the distortion effect is “corrected” by the connected components post-processing and the detection results for the two implementations are similar. In Figure 5.19, smoothing results for both bilateral filter implementations are presented. The bilateral filter chain approach produces much smoother and “round” result, while the fast bilateral filter introduces some distortions due to its implementation with downsampling. The bilateral filter chain detects 200 HC out of user-marked 205, while the fast bilateral filter due to the introduced distortions detects only 192 HC out of 205. However, it does not significantly worsen the detection rates: The bilateral filter chain produces
Figure 5.17: Overlaying the output of the Hough transform (red circles) with the manually specified output (green dots) for image D2.

results with sensitivity of 97% and specificity of 82%, while the fast bilateral filter produces results with sensitivity of 94% and specificity of 88%.
Chapter 5. Partitioning With Prior Assumptions

(a) Result of MDL-based segmentation algorithm (before applying automatic thresholding and Hough transform). The cell boundaries are well-preserved.

(b) Overlaying the output of Hough transform (red circles) with the manually specified output (green dots).

Figure 5.18: A chunk from image D2. MDL segmentation algorithm is taken as a preprocessing step.

Figure 5.19: Comparison of the smoothing results obtained with different implementations of the bilateral filter for image D8: Bilateral filter chain (left), fast bilateral filter (right).

5.5.2 Thresholding analysis

In general, Otsu thresholding and expectation maximization produce quite similar results with threshold values lying close to each other. For instance, in Figure 5.20, the initial image and results of both methods are shown. In Figure 5.21, the threshold values obtained with Otsu thresholding and EM are presented being overlaid with the respective histogram. Otsu thresholding delivers two threshold values, namely 118 and 185, which separates the data range into three regions. The EM method delivers three Gaussian curves, which represent the likelihood of belonging to the three classes, i. e., the class with the highest value is the most likely one. The
classification changes where the curves intersect, i.e., intersection points deliver the
treshold locations. They are close to the Otsu thresholds (marked with bars).

(a) Red image channel. (b) Result of Otsu method. (c) Result of EM method.

Figure 5.20: Otsu thresholding and EM thresholding give very similar results if the classes are representative.

However, when one of the classes in images is not representative, i.e. it is negligible
compared to the other classes, the nuclei class can be misclassified. For example, for
an image with low proliferation rate, depicted in Figure 5.22, the results for Otsu
and EM are shown in Figure 5.23. Otsu separates the vein from the background,
but the information about the cell class is lost. EM assigns the vein and the cells to
same Gaussian curve, which can lead to some additional false positives detection.

Figure 5.21: Results of Otsu and EM method overlaid with image histogram for Figure 5.20.
Chapter 5. Partitioning With Prior Assumptions

Figure 5.22: Close-up view for the smoothed blue image channel and vein mask for Figure 5.7. The cell class is not representative in the image.

Figure 5.23: Results for Otsu and EM methods for the image in Figure 5.22 without vein exclusion. Otsu totally misclassifies the nuclei. EM assigns both nuclei (the darkest spots) and vein (the brightest spots) to the same Gaussian distribution. After connected component processing, it is possible to extract the cells from EM result, however, a number of false positives (on the vein) is also detected.

Figure 5.24: When the vascular structure is not excluded, Otsu totally misclassifies the cell class. With vein exclusion, Otsu method is still not able to classify the cells as the variations inside the background class are more noticeable. EM assigns in both cases all non-representative classes (cells and vein) to the same Gaussian curve.
For more adequate detection of the proliferating hepatocytes, we discard the vein region, detected in the red channel, excluding the vein mask from the image histogram. In Figures 5.23 and 5.25 a comparison of the result obtained with and without vein are presented. In Figure 5.24 the resulting threshold values obtained with Otsu and EM are presented in a histogram-overlaid manner. As the cell class is negligible compared to either vein and background classes or to the intensity variations inside the background class (see Figure 5.24), Otsu method gives an incorrect result. Due to vein exclusion, the result of the EM method corresponds to the cell class in the initial image.

![Figure 5.25](image1.png)  
(a) Result of Otsu method for Figure 5.22 without vascular structure.  
(b) Result of EM method for Figure 5.22 without vascular structure.

Figure 5.25: Otsu separates the background variations into two classes, instead of separating the cell and background classes, as the cell class is negligible. EM assigns all non-representative classes to one Gaussian curve. Due to vein elimination, the cell class is correctly extracted.

### 5.5.3 Vein exclusion analysis

The preceding subsection showed that vein structures in the images make the classification task much tougher, but it also showed that vein exclusion can solve the problem. Vein elimination is needed, if there is the presence of blood in the vein, whose components have a similar appearance as non-proliferating (blue) hepatocytes.

If the vein region is represented by one connected component that encloses all the blood components, the Fill Holes filter removes the blood components and the resulting vein mask can be used to remove the vein structures from the image channels. A close-up view of such a vein region in image “D4-ZV1” is shown in Figure 5.26.

If the vein region is represented by several components, the Fill Holes filter fails to remove all the blood in the vein, which may lead to false positives. In Figure 5.27
Figure 5.26: Close-up view of a vein region in image D5-ZV1 (left). As the region is represented by a connected component, the Fill Holes filter suffices to generate the mask for vein exclusion. Detection result after vein exclusion (right).

(left), one can observe the close-up view of such a vein region from Figure 5.7. In such a case, we apply the vein smoothing step to build the vein mask. Figure 5.27 (middle) shows the correctly detected cells after vein smoothing and elimination in comparison to the result obtained without prior vein handling shown in Figure 5.27 (right). It is documented that if no vein exclusion is applied, the detection results in a number of additional false positives and false negatives. The false negatives appear due to the fact that the hepatocytes lying close to the vein have been merged together with the inner and boundary parts of the vein and excluded as a too big and non-round connected component.

Figure 5.27: Close-up view of the vein region of the image in Figure 5.7 (left), cell detection result after vein smoothing and elimination (middle), and cell detection result without prior vein handling (right). As the blood areas separate the vein into several components, the Fill Holes filter fails to remove all the blood, see the areas marked with arrows (left). Vein smoothing and elimination allows for a correct detection (middle). If no vein exclusion is applied, the detection results in a number of additional false positives and false negatives (right).

5.5.4 Hepatocyte detection analysis

One of the possible strategies of detecting overlapping cells is to use the watershed segmentation (see [Beu91], http://www.mathworks.com/company/newsletters/news_notes/win02/watershed.html). The preprocessing steps that are aimed to
Chapter 5. Partitioning With Prior Assumptions

obtain a binary image of overlapping cell objects are similar to the first steps of our processing pipeline shown in Figure 5.7. Then, for the proper marker selection, the ultimate erosion operator $[\text{Soi99}]$ is applied to the image that is computed by building the distance transformation $[\text{SHB08}]$ on the original binary image. However, Vincent has shown that the ultimate erosion does not quite yield perfect markers $[\text{VS91}]$, as some objects could contain several markers instead of one. The method of correction relies upon the fact that two components of the ultimate erosion marking the same cell are pretty much on the same maximal zone of the distance function. In fact, it is possible to go from one to the other on the distance function by going down no more than one level $[\text{VS91}]$. Thus, if we subtract 1 from the distance function at the location of all the components of the ultimate erosion, we obtain a modified distance function whose maxima are exactly the desired object markers. Thereafter, the watershed method is applied to the corrected distance transformation image.

In Figure 5.28a, we have shown results for a cut for image $D2$ in an overlaid manner. For the hepatocyte detection, we have applied the watershed algorithm (see Figure 5.28c) and Hough transformation (see Figure 5.28b).

As it can be observed in Figure 5.28c, the result obtained with the watershed algorithm needs additional postprocessing, which is required to eliminate some non-round regions that have been found. Moreover, for certain cells the markers have not been found correctly and only one region was detected instead of two.

For the Hough transformation case, the algorithm can also find one bigger circle instead of two having smaller size, as it is shown in Figure 5.28b. However, such misclassification appears quite rare and can be neglected. The advantage of the Hough transformation algorithm consists in using the information about the cell size and shape, and the problem with detecting small elongated regions separated from bigger connected components generally does not appear.

5.5.5 General analysis

After having analysed each step of the pipeline, we proceed with the full run for detection of the proliferation index on batches of files with different color characteristics.

As stated in Tables 5.2 and 5.3, the average sensitivity for the detection of all hepatocytes is 93% and for the detection of proliferating hepatocytes is 91%. The specificity is 82% for the detection of all cells and 99% for the detection of the proliferating cells. All the results were obtained using the same set of parameters.
Chapter 5. Partitioning With Prior Assumptions

Figure 5.28: Hough transformation outperforms watershed algorithm, as it uses the information about the object shape and size.

for all the datasets. Although the datasets exhibit significant variations in color properties, no additional parameter adjustment was necessary.

Although the selected default algorithm settings work well for a wide range of images, we observed that some improvement of the results can still be obtained for certain images by fine-tuning the parameters. In general, to produce best results for a series of images with specific color properties, the user should process one “typical” ROI image from the sample, select the appropriate parameters for each step, and then apply the selected settings to the whole sample.

In Figure 5.29, images from three datasets with differences in color properties are shown. The results for the detection of all hepatocytes are presented in Figure 5.30 in an overlaid fashion. The circles around the cells represent the result obtained with our pipeline and correspond to the results shown in Table 1. Our approach is robust enough to deal with images with different color properties and gives results with high sensitivity (above 90%) and specificity (above 80%).

**Approach Limitations** In the current implementation, the choice of the number of classes, indicating presence of vascular structures in the image, is performed by the user and indicated to our tool using a filename convention.

In our tests, we have not considered images with no proliferating cells or images with some complicated liver pathology.
Table 5.2: Total hepatocyte quantification results for eight different data sets.

<table>
<thead>
<tr>
<th>Image</th>
<th>Detected</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>TN</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>User P/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>230</td>
<td>221</td>
<td>9</td>
<td>22</td>
<td>154</td>
<td>0.91</td>
<td>0.94</td>
<td>243 / 163</td>
</tr>
<tr>
<td>D1-PT1</td>
<td>210</td>
<td>188</td>
<td>22</td>
<td>20</td>
<td>160</td>
<td>0.90</td>
<td>0.88</td>
<td>208 / 182</td>
</tr>
<tr>
<td>D1-ZV1</td>
<td>302</td>
<td>269</td>
<td>33</td>
<td>38</td>
<td>171</td>
<td>0.88</td>
<td>0.84</td>
<td>307 / 204</td>
</tr>
<tr>
<td>D2</td>
<td>223</td>
<td>201</td>
<td>22</td>
<td>14</td>
<td>136</td>
<td>0.93</td>
<td>0.86</td>
<td>215 / 158</td>
</tr>
<tr>
<td>D2-PT1</td>
<td>212</td>
<td>166</td>
<td>46</td>
<td>9</td>
<td>203</td>
<td>0.95</td>
<td>0.82</td>
<td>175 / 249</td>
</tr>
<tr>
<td>D2-ZV1</td>
<td>230</td>
<td>200</td>
<td>30</td>
<td>17</td>
<td>149</td>
<td>0.92</td>
<td>0.83</td>
<td>217 / 179</td>
</tr>
<tr>
<td>D3</td>
<td>299</td>
<td>230</td>
<td>69</td>
<td>10</td>
<td>55</td>
<td>0.96</td>
<td>0.52</td>
<td>240 / 124</td>
</tr>
<tr>
<td>D3-PT1</td>
<td>268</td>
<td>206</td>
<td>62</td>
<td>13</td>
<td>129</td>
<td>0.94</td>
<td>0.67</td>
<td>219 / 191</td>
</tr>
<tr>
<td>D3-ZV1</td>
<td>269</td>
<td>216</td>
<td>53</td>
<td>13</td>
<td>156</td>
<td>0.94</td>
<td>0.75</td>
<td>229 / 209</td>
</tr>
<tr>
<td>D4</td>
<td>229</td>
<td>212</td>
<td>17</td>
<td>33</td>
<td>126</td>
<td>0.87</td>
<td>0.88</td>
<td>245 / 143</td>
</tr>
<tr>
<td>D4-PT1</td>
<td>216</td>
<td>198</td>
<td>18</td>
<td>15</td>
<td>138</td>
<td>0.93</td>
<td>0.89</td>
<td>213 / 156</td>
</tr>
<tr>
<td>D4-ZV1</td>
<td>237</td>
<td>207</td>
<td>30</td>
<td>21</td>
<td>131</td>
<td>0.90</td>
<td>0.81</td>
<td>228 / 161</td>
</tr>
<tr>
<td>D5</td>
<td>226</td>
<td>189</td>
<td>49</td>
<td>12</td>
<td>208</td>
<td>0.94</td>
<td>0.80</td>
<td>201 / 257</td>
</tr>
<tr>
<td>D5-PT1</td>
<td>217</td>
<td>210</td>
<td>7</td>
<td>15</td>
<td>265</td>
<td>0.93</td>
<td>0.97</td>
<td>225 / 272</td>
</tr>
<tr>
<td>D5-ZV1</td>
<td>237</td>
<td>197</td>
<td>40</td>
<td>5</td>
<td>174</td>
<td>0.97</td>
<td>0.81</td>
<td>202 / 214</td>
</tr>
<tr>
<td>D6</td>
<td>216</td>
<td>186</td>
<td>30</td>
<td>11</td>
<td>148</td>
<td>0.94</td>
<td>0.83</td>
<td>197 / 178</td>
</tr>
<tr>
<td>D6-PT1</td>
<td>224</td>
<td>186</td>
<td>38</td>
<td>14</td>
<td>170</td>
<td>0.93</td>
<td>0.81</td>
<td>200 / 208</td>
</tr>
<tr>
<td>D6-ZV1</td>
<td>224</td>
<td>186</td>
<td>38</td>
<td>14</td>
<td>170</td>
<td>0.93</td>
<td>0.81</td>
<td>200 / 208</td>
</tr>
<tr>
<td>D7</td>
<td>230</td>
<td>191</td>
<td>39</td>
<td>11</td>
<td>152</td>
<td>0.94</td>
<td>0.78</td>
<td>202 / 191</td>
</tr>
<tr>
<td>D7-PT1</td>
<td>236</td>
<td>190</td>
<td>46</td>
<td>3</td>
<td>205</td>
<td>0.98</td>
<td>0.81</td>
<td>193 / 251</td>
</tr>
<tr>
<td>D7-ZV1</td>
<td>218</td>
<td>204</td>
<td>14</td>
<td>8</td>
<td>200</td>
<td>0.96</td>
<td>0.93</td>
<td>212 / 214</td>
</tr>
<tr>
<td>D8</td>
<td>229</td>
<td>197</td>
<td>32</td>
<td>6</td>
<td>149</td>
<td>0.97</td>
<td>0.82</td>
<td>203 / 181</td>
</tr>
<tr>
<td>D8-PT1</td>
<td>214</td>
<td>177</td>
<td>83</td>
<td>10</td>
<td>236</td>
<td>0.95</td>
<td>0.74</td>
<td>187 / 319</td>
</tr>
<tr>
<td>D8-ZV1</td>
<td>208</td>
<td>174</td>
<td>34</td>
<td>16</td>
<td>173</td>
<td>0.91</td>
<td>0.83</td>
<td>190 / 207</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.93</td>
<td>0.82</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 5. Partitioning With Prior Assumptions

<table>
<thead>
<tr>
<th>Image</th>
<th>Detected</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>TN</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>User P/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>405</td>
<td>1.00</td>
<td>1.00</td>
<td>3 / 405</td>
</tr>
<tr>
<td>D1-PT1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>405</td>
<td>0.75</td>
<td>1.00</td>
<td>4 / 386</td>
</tr>
<tr>
<td>D1-ZV2</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>245</td>
<td>1.00</td>
<td>1.00</td>
<td>4 / 245</td>
</tr>
<tr>
<td>D2</td>
<td>22</td>
<td>18</td>
<td>4</td>
<td>1</td>
<td>351</td>
<td>0.94</td>
<td>0.98</td>
<td>19 / 355</td>
</tr>
<tr>
<td>D2-PT1</td>
<td>27</td>
<td>23</td>
<td>4</td>
<td>1</td>
<td>396</td>
<td>0.96</td>
<td>0.99</td>
<td>24 / 400</td>
</tr>
<tr>
<td>D2-ZV1</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>385</td>
<td>1.00</td>
<td>0.99</td>
<td>8 / 388</td>
</tr>
<tr>
<td>D3</td>
<td>27</td>
<td>24</td>
<td>3</td>
<td>1</td>
<td>356</td>
<td>0.96</td>
<td>0.99</td>
<td>25 / 359</td>
</tr>
<tr>
<td>D3-PT1</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>391</td>
<td>0.61</td>
<td>0.99</td>
<td>18 / 392</td>
</tr>
<tr>
<td>D3-ZV1</td>
<td>41</td>
<td>39</td>
<td>2</td>
<td>1</td>
<td>396</td>
<td>0.97</td>
<td>0.99</td>
<td>40 / 398</td>
</tr>
<tr>
<td>D4</td>
<td>98</td>
<td>96</td>
<td>2</td>
<td>27</td>
<td>263</td>
<td>0.78</td>
<td>0.99</td>
<td>123 / 265</td>
</tr>
<tr>
<td>D4-PT1</td>
<td>105</td>
<td>102</td>
<td>3</td>
<td>14</td>
<td>250</td>
<td>0.87</td>
<td>0.98</td>
<td>116 / 253</td>
</tr>
<tr>
<td>D4-ZV1</td>
<td>58</td>
<td>57</td>
<td>1</td>
<td>7</td>
<td>324</td>
<td>0.89</td>
<td>0.99</td>
<td>64 / 325</td>
</tr>
<tr>
<td>D5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>390</td>
<td>1.00</td>
<td>0.99</td>
<td>2 / 391</td>
</tr>
<tr>
<td>D5-PT1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>444</td>
<td>0.5</td>
<td>1.00</td>
<td>2 / 445</td>
</tr>
<tr>
<td>D5-ZV1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>375</td>
<td>1.00</td>
<td>1.00</td>
<td>1 / 375</td>
</tr>
<tr>
<td>D6</td>
<td>90</td>
<td>77</td>
<td>13</td>
<td>3</td>
<td>256</td>
<td>0.96</td>
<td>0.95</td>
<td>80 / 269</td>
</tr>
<tr>
<td>D6-ZV1</td>
<td>65</td>
<td>61</td>
<td>4</td>
<td>5</td>
<td>301</td>
<td>0.92</td>
<td>0.98</td>
<td>66 / 305</td>
</tr>
<tr>
<td>D7</td>
<td>40</td>
<td>37</td>
<td>3</td>
<td>2</td>
<td>332</td>
<td>0.94</td>
<td>0.99</td>
<td>39 / 335</td>
</tr>
<tr>
<td>D7-PT1</td>
<td>26</td>
<td>25</td>
<td>1</td>
<td>0</td>
<td>369</td>
<td>1.00</td>
<td>0.99</td>
<td>25 / 370</td>
</tr>
<tr>
<td>D7-ZV1</td>
<td>24</td>
<td>20</td>
<td>4</td>
<td>1</td>
<td>376</td>
<td>0.95</td>
<td>0.98</td>
<td>21 / 380</td>
</tr>
<tr>
<td>D8</td>
<td>138</td>
<td>120</td>
<td>18</td>
<td>2</td>
<td>224</td>
<td>0.98</td>
<td>0.92</td>
<td>122 / 242</td>
</tr>
<tr>
<td>D8-PT1</td>
<td>93</td>
<td>72</td>
<td>21</td>
<td>3</td>
<td>367</td>
<td>0.96</td>
<td>0.94</td>
<td>75 / 388</td>
</tr>
<tr>
<td>D8-ZV1</td>
<td>55</td>
<td>50</td>
<td>5</td>
<td>1</td>
<td>393</td>
<td>0.98</td>
<td>0.98</td>
<td>51 / 398</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td>0.99</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Proliferating hepatocyte quantification results for eight different data sets.
Chapter 5. Partitioning With Prior Assumptions

Figure 5.29: Images of stained sections are subject to variations in the color properties due to some differences in the histological processing of the tissue sample. Three images from three datasets are presented here, namely, D3-ZV1.jpg, D7-ZV1.jpg, D8-ZV1.jpg.

Figure 5.30: The resulting statistical measures in form of sensitivity/specificity are D3-ZV1: 0.94/0.75, D7-ZV1: 0.93/0.96, and D8-ZV1: 0.9/0.83. Our pipeline is robust enough to allow accurate analysis of images with different color properties without the need for individual adjustments.
Chapter 6

Conclusions

The presented work was concerned with research and development of efficient segmentation algorithms for multichannel medical images. Despite the variety of the segmentation techniques proposed in the recent years, there is no “standard solution” that will work perfectly for all applications. In the present work, we considered several approaches for color segmentation.

As there exist a whole range of well-investigated segmentation approaches for single-channel data, it is desirable to convert multichannel (color) data to scalar field to make these methods applicable to it. We generalized the recently presented color-to-scalar conversion method, that preserves salient features, for this purpose.

We demonstrated that this method can convert datasets with neighbored isoluminant areas, such that these areas stay naturally distinguishable in the grayscale image. This allows us to segment these areas using such simple and fast segmentation techniques as the global thresholding.

However, the conversion method appears to be rather computationally demanding. To overcome this limitation, we used a prior clustering. We applied several clustering techniques and validated the results with our own clustering method based on a genetic algorithm. We found out that the clustering result strongly affects the result of color-to-scalar conversion.

Clustering techniques partition the data in the feature space, so we used this information for the direct segmentation in the object space. We presented a tool that allows a user to partition an image iteratively in an intuitive manner. In order to visualize the results, we utilized a Marching-like method that extracts the boundaries of the surfaces from the color data.
Chapter 6. Conclusions

We applied the described techniques to several color datasets, and drew a conclusion, that clustering techniques such as K-Means and C-Means produce the most appropriate results. However, when the images are subject to noise and partial volume effect, such a scheme does not produce fully adequate results. For example, instead of detecting a clear boundary separating two regions, these methods end up with an area where small regions that belong to different classes are mixed with each other.

Techniques that consider not only a pixel’s (voxel’s) color, but also the colors of its neighbors and its spatial arrangement, are favorable for most practical purposes. We investigated a branch of automatic partitioning techniques that are based on energy minimization and include a smoothing prior and spatial arrangement. We formulated three energy functionals for piecewise constant partitioning for multichannel data that are based on different assumptions about the noise in the data. We considered models with constant, piecewise constant and spatially varying white Gaussian noise. We studied the algorithm’s behavior on artificial examples and made a thorough analysis of the algorithm’s parameters.

To minimize the functional, we applied a scheme based on Relaxation method and compared it to the Graph Cut minimization for the model with constant noise. We demonstrated that the relaxation scheme based on gradient descent method can compete with the Graph Cut technique in terms of quality. Although a simpler procedure based on Jacobi iterations does not allow to achieve the same convergence results as the other two methods, its results combined with post-region-merging can approach the same quality as Graph Cuts and Relaxation scheme with gradient descent method. The major advantage of the Jacobi iteration technique is that it is faster and, furthermore, can be parallelized in a straightforward manner. We implemented a GPU-based version of the algorithm, that gave a 200x speed up and applied it to several real-world datasets.

Apart from studying general-purpose segmentation techniques, we presented a solution, using the prior assumptions about the regions of interest to be segmented. We developed an automatic procedure for quantitative analysis of histological images of rat livers. The task was to calculate a relative proportion of dividing hepatocytes of different liver parts and develop an efficient algorithm that can be further used for the analysis of whole stained sections. The prior assumptions about the shape, color and size of hepatocytes, available from the medical experts, were used in the quantification pipeline. The results were compared to the manually defined ground-truth. The proposed algorithm appeared to be effective (90% sensitivity and
85% specificity) while dealing with images with significant variations in the color properties.

The selection of the preferable segmentation technique depends on many factors, and the appropriate choice must be justified by the correspondent application area. For instance, single channel approaches can be applied to multichannel data, if the effective transformation to a single representative channel is applicable or the structures of interest are already presented in one channel. Often, a completely automated segmentation is not possible and a user interaction must be included into the pipeline. Tools that guide the user through the data and help him to identify the region of interest are of big value. Such tools should be intuitive, the user should not be overburdened with complicated operations, hence, the tedious and effort-demanding part must be automated. The automatic segmentation part must provide accurate solutions. In general, fast yet robust and stable to noise approaches that consider both feature and object space information are desirable for effective segmentation.
References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Details</th>
</tr>
</thead>
</table>
References


References


## References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Authors</th>
</tr>
</thead>
</table>
References


References


References


References


References


References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Journal or Conference</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>Authors</td>
<td>Title</td>
<td>Journal</td>
<td>Volume</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
</tbody>
</table>
Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

Date, Signature