Graz University of Technology

Institute for Computer Graphics and Vision

Dissertation

Interactive Multi-Label Segmentation

Jakob Santner

Graz, Austria, October 2010

Thesis supervisors

Prof. Dr. Horst Bischof

Dr. Antonio Criminisi

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' but 'That's funny'...

Isaac Asimov

Abstract

Interactive image segmentation deals with partitioning an image into multiple pairwise-disjoint regions based on input provided by a human operator. Being interactive means, that an algorithm has to quickly react on user input, which limits the computational complexity of the employed algorithms drastically. Therefore, many interactive segmentation methods represent these regions with simple models based on low-dimensional feature spaces, which in turn introduces a limitation in terms of expressibility of these models and thus segmentation quality. Furthermore, most methods can only handle the two-label case, i.e. the segmentation of an image into foreground and background.

In this work, we investigate the incorporation of arbitrary high-dimensional features in an interactive multi-label segmentation framework. With such high-dimensional features, not only color and grayvalue information, but also complex textural properties of a region can be modeled. In order to not violate the runtime constraints, we carefully select the building blocks of our framework according to their ability of being implemented on parallel architectures: We employ Haralick texture features and Local Binary Patterns to represent local image structure, as well as Random Forests as learning algorithm. Finally, we employ a multi-label Potts regularizer in order to obtain spatially compact image segments. All these parts are implemented on the GPU or multi-core CPUs in order to achieve runtimes that allow for convenient user interaction.

We furthermore address the problem of comparatively evaluating interactive multi-label segmentation algorithms and introduce a large novel benchmark dataset. With this dataset, we perform detailed experiments in order to evaluate the performance and runtime of the building blocks of our framework. We show the benefit of incorporating texture and color features over employing color features alone. Finally, we compare our framework to the state-of-the-art Power Watersheds method and highlight advantages and drawbacks of both approaches.

Kurzfassung

In der interaktiven Bildsegmentierung wird ein digitales Bild anhand von Benutzereingaben in mehrere nichtüberlappende Segmente zerteilt. Um interaktiv sein zu können muss die verwendete Methode sehr schnell auf diese Benutzereingaben reagieren, was die rechnerische Komplexität der verwendeten Algorithmen stark limitiert. Aus diesem Grund verwenden viele interaktive Segmentiermethoden relativ einfache Modelle und niedrigdimensionale Bildrepräsentationen um die Segmente zu beschreiben. Die Aussagekraft dieser Modelle und folglich auch die Qualität der Bildsegmentierung ist dementsprechend limitiert. Weiters können viele interaktive Segmentiermethoden das Bild nur in zwei Segmente zerteilen (Segmentierung in Objekt und Hintergrund).

In dieser Arbeit beschäftigen wir uns mit der Verwendung von hochdimensionalen Bildrepresentationen in einem interaktiven Framework, das die Teilung in mehr als zwei Segmente erlaubt. Mit hochdimensionalen Bildrepresentationen können im Vergleich zu anderen Methoden nicht nur Grau- und Farbinformationen, sondern auch komplexe Textureigenschaften modelliert werden. Um die Interaktivität unseres Frameworks zu gewährleisten, werden die verwendeten Algorithmen sorgfältig in Bezug auf ihre Parallelisierbarkeit ausgewählt: Wir verwenden Haralick Features und Local Binary Patterns um Textureigenschaften darzustellen, und Random Forests um die Segmenteigenschaften zu lernen. Mit einer Potts-Regularisierung wird die räumliche Kohärenz der Segmente sichergestellt. Diese Algorithmen sind allesamt für Mehrkernprozessoren oder Grafikprozessoren optimiert um schnell genug für Benutzerinteraktion zu sein.

Zur vergleichbaren quantitativen Evaluierung von interaktiven Segmentieralgorith-

men stellen wir einen neuen Testdatensatz vor. Diesen Datensatz verwenden wir um die einzelnen Bauteile unseres Frameworks bezüglich Laufzeit und Einfluss auf die Qualität der Segmentierung zu testen. Wir können zeigen, dass die Verwendung von hochdimensionalen Bildrepresentationen Vorteile gegenüber einfachen Farbrepresentationen hat. Weiters vergleichen wir unser Framework mit dem Power Watersheds Algorithmus und zeigen Unterschiede zwischen diesen Methoden auf.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgments

Writing a thesis in order to earn a Doctor of Philosophy degree constitutes the end of an educational process that takes more than twenty years to finish. Throughout this time, I had the luck to experience the guidance, mentoring, friendship, assistance, criticism and love of many great people. It is their commitment that brought me in the position of being able to perform scientific research.

I want to thank my supervisor Horst Bischof, who already accompanied me through undergraduate studies and encouraged me to pursue a doctorate degree in the intriguing field of computer vision. He not only guided my research with his experience and gift to always ask the uncomfortable questions, he also liberally paid for conference journeys and barbecues. I would also like to thank Antonio Criminisi for his effort in being my second supervisor.

Moreover, I would like to give my most sincere thanks to my dear colleagues at ICG, especially Amir Saffari, Arno Knapitsch, Christian Leistner, Manuel Werlberger, Martin Lenz, Markus Heber, Markus Storer, Markus Unger, Matthias Ruether, Michael Maurer, Thomas Mauthner, Thomas Pock and Werner Trobin. I greatly appreciate the fruitful intra- and extra-faculty discussions, the collaborative institute cooking as well as the countless soccer games played and watched together. You have made the last four years an enjoyable and prolific time.

Finally, I feel greatly indebted to my family for their love and support: Mama, Papa, Hanna, Niki, Kathi, Opa, Oma, Manfred, Christa and Kati. Tempus fugit, amor manet.

This work is supported by the Austrian Research Promotion Agency (FFG) under the Federal Ministry for Transport, Innovation and Technology (BMVIT) through the project VM-GPU (813396).

Contents

1 Introduction **1**

1.1 Definition of Segmentation . . . 3
1.2 Ambiguity . . . 3
1.2.1 Ambiguity at Object Level . . . 4
1.2.2 Ambiguity at Pixel Level . . . 4
1.3 The Role of Supervision . . . 5
1.3.1 Unsupervised Segmentation . . . 5
1.3.1.1 Thresholding . . . 6
1.3.1.2 Edge-based Segmentation . . . 6
1.3.1.3 Region-based Segmentation . . . 8
1.3.1.4 Advanced Techniques . . . 10
1.3.2 Supervised Segmentation . . . 12
1.4 Interactive Segmentation . . . 13
1.4.1 Conceptual Differences . . . 14
1.4.2 Interaction . . . 14
1.4.3 Related Work . . . 15
1.4.3.1 Manual Drawing Tools . . . 15
1.4.3.2 Intelligent Scissors . . . 16
1.4.3.3 Seeded Region Growing . . . 19
1.4.3.4 Magic Wand . . . 20
1.4.3.5 Snakes / Active Contours . . . 20
1.4.3.6 Graph Cut Segmentation . . . 23
1.4.3.7 Random Walker . . . 28
1.4.3.8 Weighted Total Variation . . . 30
1.4.3.9 Geodesic Segmentation . . . 31
1.4.3.10 Power Watersheds . . . 33

1.4.3.11 Summary . . . 34
1.5 Parallel Computing . . . 40
1.5.1 CPU - Central Processing Units . . . 40
1.5.2 GPU - Graphics Processing Units . . . 41
1.6 Proposed Framework . . . 45
1.6.1 Contributions of this Thesis . . . 45
1.6.2 Organization . . . 46

2 Image Features **49**
2.1 Color Models . . . 49
2.1.1 HSV . . . 51
2.1.2 CIELAB . . . 51
2.1.3 Runtime . . . 52
2.2 Texture Description . . . 53
2.2.1 Image Patches . . . 54
2.2.2 Haralick Features . . . 54
2.2.2.1 Graylevel-Cooccurrence Matrix . . . 55
2.2.2.2 Statistical Measures . . . 56
2.2.2.3 Runtime . . . 58
2.2.3 Filter Banks . . . 59
2.2.4 Local Binary Patterns . . . 63
2.3 Conclusion . . . 65

3 Region Model **67**
3.1 Decision Trees . . . 69
3.1.1 Evaluation . . . 69
3.1.2 Training . . . 70
3.2 Random Forests . . . 71
3.2.1 Multi-Core Implementation . . . 72
3.2.1.1 Training . . . 73
3.2.1.2 Evaluation . . . 74
3.3 Performance Evaluation . . . 74
3.3.1 K-Nearest Neighbors . . . 75
3.3.2 Support Vector Machines . . . 75
3.3.3 Runtime . . . 75
3.3.4 Model Quality . . . 76

4 Segmentation **83**
4.1 Regularization Terms . . . 85
4.1.1 Properties . . . 87
4.1.1.1 Convexity . . . 87
4.1.1.2 Discontinuity Handling . . . 87

4.1.2 Applicability . . . 88
4.2 Binary Segmentation . . . 89
4.2.1 Discrete . . . 89
4.2.2 Continuous . . . 89
4.3 Convex Multi-label Functionals . . . 90
4.4 Sequential Methods . . . 90
4.4.1 1-vs-All . . . 91
4.4.2 Expansion / Swap Moves . . . 92
4.5 Convex Approximations . . . 93
4.6 Incorporation into the Framework . . . 96
4.7 Conclusion . . . 97

5 Benchmark **99**
5.1 Related Image Segmentation Benchmarks . . . 100
5.1.1 LabelMe Database . . . 100
5.1.2 PASCAL VOC Database . . . 101
5.1.3 GrabCut Database . . . 101
5.1.4 Berkeley Segmentation Dataset and Benchmark . . . 103
5.2 IcgBench . . . 105
5.2.1 Structure . . . 106
5.2.2 Tooltip . . . 110
5.2.3 Evaluation Multi-label Case . . . 110
5.2.4 Evaluation Binary Case . . . 112

6 Experiments **113**
6.1 Graphical User Interface . . . 113
6.2 Segmentation Parameters . . . 115
6.2.1 Number of Iterations . . . 115
6.2.2 Regularization and Edge Weight . . . 116
6.3 Features . . . 118
6.3.1 Color Model . . . 118
6.3.2 Textural Features . . . 119
6.3.2.1 Grayscale Patches . . . 119
6.3.2.2 Haralick Textural Features . . . 120
6.3.2.3 Local Binary Patterns . . . 120
6.3.3 Feature Combinations . . . 121
6.3.3.1 Color Features . . . 122
6.3.3.2 Color and Texture Features . . . 122
6.4 Learning Algorithm . . . 125
6.5 Tooltip . . . 126
6.6 Repeatability . . . 127
6.7 Runtime . . . 128

6.8 Comparison to Power Watersheds 130
6.9 Conclusion 132

7 Conclusion **139**
7.1 Summary 139
7.2 Future Work 140
7.2.1 Semi-Supervised Learning 140
7.2.2 Computational Effort 141
7.2.3 Evaluation 141
7.2.4 3D / Video 142

A List of Publications **143**
A.1 2010 143
A.2 2009 145

Bibliography **147**

List of Figures

1.1 A Basic Image Segmentation Problem 1
1.2 Image Segmentation Applications: Image Editing 2
1.3 Image Segmentation Applications: Medical and Industrial 2
1.4 Ambiguity in Image Segmentation at Object Level 4
1.5 Ambiguity in Image Segmentation at Pixel Level 5
1.6 Segmentation by Thresholding 7
1.7 Image Gradients as Segmentation Border Cues 7
1.8 Canny Edge Detection 8
1.9 Watershed Segmentation 9
1.10 Watershed Segmentation II 10
1.11 Clustering in the RGB-Color Space 11
1.12 The Elementary Interactive Image Segmentation Process 15
1.13 Boundary Marking Methods 18
1.14 The Magic Wand Tool of Adobe Photoshop 21
1.15 Segmentation with the Active Contour Model 22
1.16 Graph Cut Image Segmentation 25
1.17 The GrabCut Segmentation Tool 27
1.18 The TVSeg Segmentation Tool 32
1.19 Computational Performance CPU vs. GPU 42
1.20 Workflow of the Segmentation Framework 47

2.1 The RGB Color Space 50
2.2 The HSV Color Space 51
2.3 Texture as Segmentation Cue 54
2.4 Neighborhoods for Graylevel Co-Occurrence Matrices 55
2.5 Haralick Texture Features 59

2.6 Runtime Comparison for Haralick Features . . . 60
2.7 Filter Bank Examples . . . 62
2.8 Sampling Strategy for Local Binary Patterns . . . 63
2.9 Examples for Local Binary Patterns . . . 64
2.10 Runtime Comparison Local Binary Patterns . . . 66

3.1 The Class Average Segmentation Model . . . 68
3.2 A Basic Decision Tree . . . 70
3.3 An Artificial Segmentation Dataset . . . 76
3.4 Region Models I . . . 79
3.5 Region Models II . . . 80
3.6 Region Models III . . . 81

4.1 Noise in Posterior Probabilities . . . 83
4.2 Maximum Posterior Probability Labeling . . . 84
4.3 Signals for Evaluating the Discontinuity Handling . . . 87
4.4 1-vs-All Multi-label Segmentation . . . 91
4.5 Unassigned Pixels . . . 92
4.6 Swap Moves . . . 93
4.7 Expansion Moves . . . 93
4.8 Problems with Sequential Binary Segmentations . . . 94

5.1 LabelMe Database Example . . . 100
5.2 PASCAL VOC Database Examples . . . 102
5.3 GrabCut Database Examples . . . 103
5.4 BSDS300 Dataset Examples . . . 104
5.5 IcgBench Annotation Tool . . . 106
5.6 IcgBench Examples I . . . 108
5.7 IcgBench Examples II . . . 109
5.8 Different Scribble Interaction Tooltips . . . 110
5.9 Evaluation Score of our Benchmark . . . 112

6.1 Graphical User Interface . . . 114
6.2 Visual Convergence . . . 116
6.3 Visual Effect of the Regularization Weight . . . 117
6.4 Results for the Regularization and Edge Weight . . . 117
6.5 Haralick Feature Performance . . . 120
6.6 Local Binary Pattern Performance . . . 121
6.7 Results for CIELAB and Haralick Features . . . 123
6.8 Results for CIELAB and Local Binary Patterns . . . 124
6.9 Influence of different Tooltips . . . 126
6.10 Repeatability of the Results . . . 128
6.11 Runtime Evaluation . . . 130

6.12 Power Watersheds: Worst Results 133
6.13 Power Watersheds: Best Results 134
6.14 Worst Scoring Results within IcgBench 136
6.15 Best Scoring Results within IcgBench 137

List of Tables

1.1 A Unified Segmentation Framework . . . 33
1.2 Comparison of Related Work . . . 39

2.1 Runtime Comparisons for Color Space Conversions . . . 53
2.2 Filter Bank Runtime Comparisons . . . 62

3.1 Random Forest Data Structure . . . 73
3.2 Performance Comparisons of Learning Algorithms . . . 77

4.1 Regularization Terms for Energy Minimization Functionals . . . 86
4.2 Energies for Different Regularization Term Penalty Functions . . . 88
4.3 Labeling Algorithm Overview . . . 97

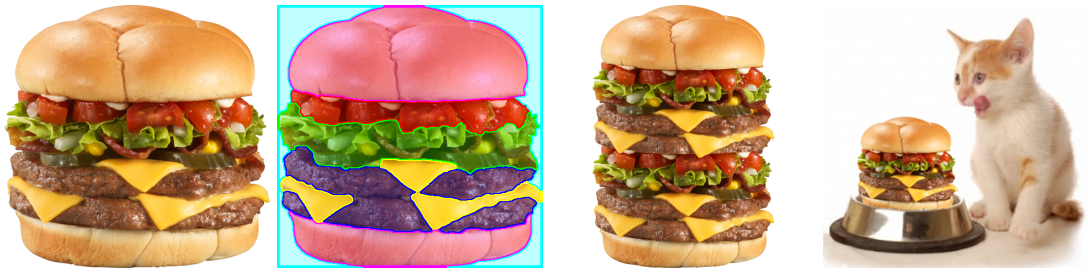
5.1 Benchmark: Number of Annotations per Image . . . 107
5.2 Benchmark: Number of Annotations per User . . . 107

6.1 Benchmark: Segmentation Algorithm Iterations . . . 115
6.2 Benchmark: Color . . . 118
6.3 Benchmark: Grayscale Patch Size . . . 119
6.4 Benchmark: Single Features . . . 121
6.5 Benchmark: Color Feature Combinations . . . 122
6.6 Benchmark: CIELAB and Haralick Features . . . 124
6.7 Benchmark: Learning Algorithms . . . 125
6.8 Benchmark: Tooltips . . . 127
6.9 Benchmark: Runtime . . . 129
6.10 Benchmark: Runtime Segmentation Algorithm . . . 129
6.11 Benchmark: Power Watersheds . . . 131

1 Introduction

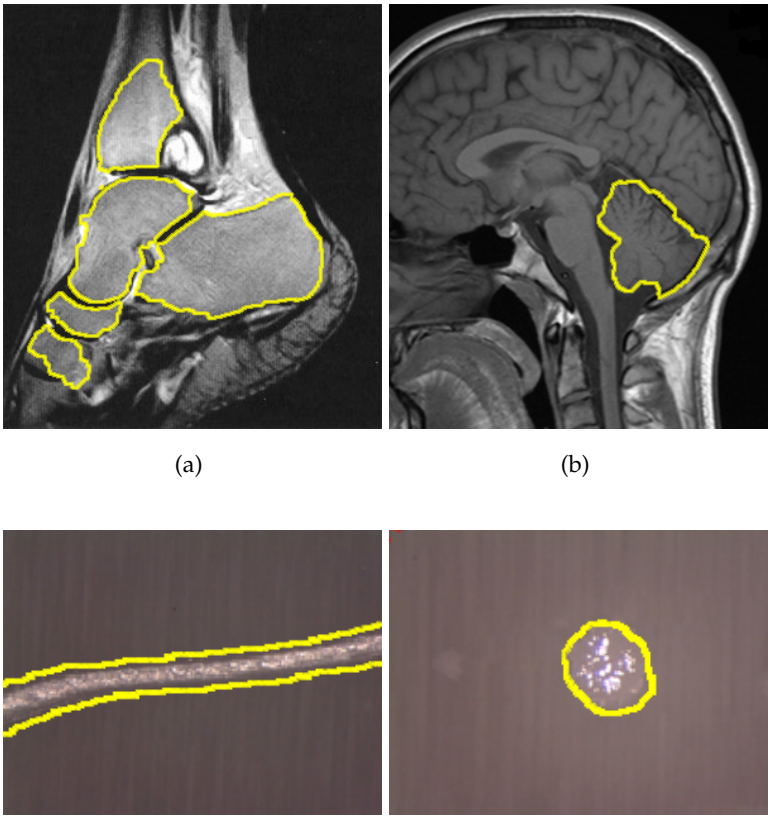
Image segmentation is one of the cardinal problems in computer vision. It describes the task of dividing an image into two or more disjoint regions, such that every pixel is part of exactly one region (Figure 1.1). Applications of image segmentation are found in the fields of medical image analysis, digital image manipulation, object recognition, industrial computer vision, aerial image processing and many more (Figure 1.2 and 1.3). The problem of image segmentation has been studied thoroughly during the last decades, and despite the existence of many seminal works to this research area, it is far from being solved.

Figure 1.1: The problem of image segmentation deals with partitioning an image into two or more disjoint regions. The yellow line shows a possible segmentation of this image, dividing it into two regions: Bears and background.



(a) (b) (c) (d)

Figure 1.2: Digital image editing is a major field of application for interactive image segmentation methods: Given an image (a) and an accurate segmentation (b), new images can be created by manipulation and composition (c,d).



(a) (b)

(c) (d)

Figure 1.3: Image segmentation is also a commonly used tool in medical applications, e.g. for MRI or CT data analysis (a,b), as well as industrial applications (c,d), e.g. quality inspection.

1.1 Definition of Segmentation

In image segmentation, the image domain $\Omega \in \mathbb{R}^2$ is partitioned into k sets E_l with $l \in \{0, 1, \ldots, k-1\}$

$$\bigcup_{l=0}^{k-1} E_l = \Omega, \tag{1.1}$$

such that every pixel is part of exactly one set

$$E_i \cap E_j = \emptyset \; \forall \; i \neq j. \tag{1.2}$$

A different definition of the segmentation problem is used in probabilistic / soft segmentation: There, a pixel might be part of more than one set E_i according to a specific probability, i.e. especially at the border between segments, a pixel typically belongs to both segments up to a certain extent. Moreover, there also exist higher dimensional segmentation problems ($\Omega \in \mathbb{R}^n$), such as e.g. the segmentation of 3D volumes in medical applications or in videos in spatial-temporal representation. However, in this thesis, we focus on the definitions (1.1,1.2), hence the hard segmentation of two-dimensional images.

From a theoretical point of view, there are k^N possible splits for an image with N pixels. However, from a practical point of view, we are only interested in grouping pixels together that are homogeneous in a certain respect. This homogeneity can be of various kinds, e.g. we could group pixels with similar color or spatially close position in the image, pixels that belong to the same object in the image, pixels that are of the same object class, pixels that do not belong to a specific object or object class etc. Adding this kind of higher level knowledge to a segmentation problem reduces the amount of possible solutions greatly and makes the segmentation problem tractable. However, note that the definition of homogeneity is very loose, i.e. there is still a vast amount of splits yielding regions which are homogeneous in a certain respect.

1.2 Ambiguity

A typical segmentation problem exhibits many possible solutions, which are not necessarily similar to each other, they can be completely different: E.g. at object level, it depends on the interpretation of the image which parts should be grouped into one region and which parts should form their own segment. At pixel level, it is often difficult to decide to which of the adjacent segments a pixel belongs to.

1.2.1 Ambiguity at Object Level

Figure 1.1 shows two bears sitting on an earth bank, segmented such that the bears are considered as one region, and everything else as the other region. However, there are many other possible solutions: Figure 1.4 shows several objects, which can be considered as a separate region. Neither one is correct nor wrong, the correctness of a segmentation solely depends on the interpretation of the image.

Figure 1.4: Some meaningful image segments based on the many possible interpretations of the image shown in Figure 1.1.

1.2.2 Ambiguity at Pixel Level

Besides the many possible interpretations of an image in terms of regions, there is also lots of ambiguity at pixel level. Assuming that there was only one interpretation of the image shown in Figure 1.1 (e.g. four regions: two separate bears, the earth bank and the background), there would still be many possible segmentations: Figure 1.5 shows detail crops with two marked pixels each. While in image 1.5(a), a human can easily tell which segments the marked pixels belong to, in image 1.5(b) and 1.5(c) even a human has difficulties to assign the marked pixels to either the left or the right bear (note that this ambiguity can be modeled well with probabilistic segmentation).

(a) (b) (c)

Figure 1.5: While the marked pixels in (a) can easily be related to a region, the marked pixels in (b) and (c) are even for a human difficult to assign to either the left or the right bear.

1.3 The Role of Supervision

One can divide the research field of image segmentation according to the (non-)existence and role of a supervisor: In unsupervised segmentation, the goal is to divide an image into regions completely automatic without any previously learned models or user interaction. Supervised segmentation is a two-step procedure, where in the first step models are trained for every expected region from a training database. In the second step, every pixel is assigned to one of these regions according to the trained models. In between lies the discipline of interactive (a.k.a. semi-supervised) segmentation, where the image is partitioned based on some kind of user-provided input. Despite the fact that the focus of research is quite different in these subfields, there are many similarities and common concepts between them. Therefore, although we focus on semi-supervised segmentation in this thesis, we want to first give a short overview over basic concepts of unsupervised and fully supervised segmentation.

1.3.1 Unsupervised Segmentation

In unsupervised segmentation, the goal is to partition an image into regions with the only prior assumption that pixels belonging to the same segment are homogeneous in a certain respect. Early approaches of unsupervised segmentation can be divided into three families: Segmentation by thresholding, edge-based segmentation and region-based segmentation. In order to develop basic ideas of image segmentation methods, in the following we give a short overview of these three families. For a detailed description of basic segmentation techniques, refer to (Sonka et al., 2007).

1.3.1.1 Thresholding

Let $I(x,y)$ represent a digital grayscale image as a two-dimensional function defined on the rectangular domain Ω, where x and y denote spatial coordinates. The aim is to find a division of the image into two regions represented by a labeling function $u(x,y) \in \{0,1\}$. Segmentation by thresholding finds this labeling function by employing a scalar threshold θ such that

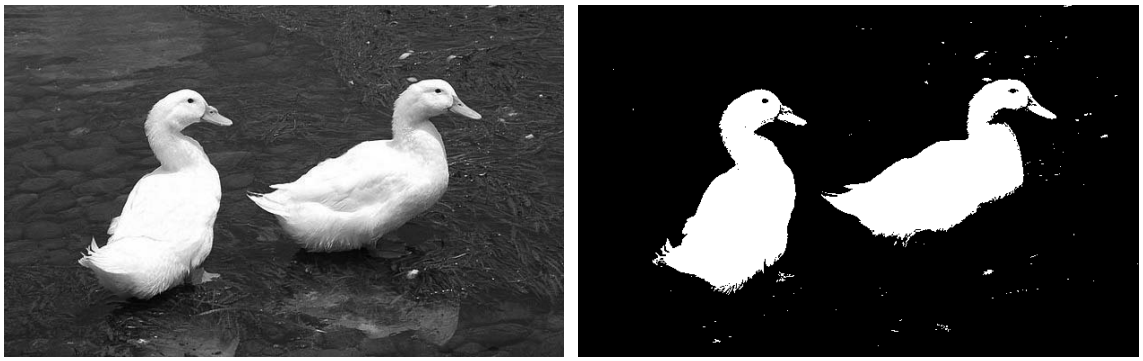
$$u(x,y) = \begin{cases} 1: & \text{if } I(x,y) \geq \theta \\ 0: & \text{else.} \end{cases} \tag{1.3}$$

This elementary segmentation algorithm (see Figure 1.6 for an example) is extremely fast to compute and yields good results when there is a significant difference between the grayscale distributions of the image foreground and background. The key part of the method is the search for a suitable threshold θ: Most threshold selection methods analyze the graylevel histogram of the image in order to find θ. E.g., the popular algorithm of Otsu (1979) works by searching greedily for the threshold θ that minimizes the combined intra-class variance for the foreground and background region. Extensions to thresholding include the incorporation of multiple thresholds to perform segmentation with multiple labels ($k \geq 2$) or the usage of different thresholds for different image regions. A good overview over thresholding techniques is presented in (Sahoo et al., 1988). Drawbacks of thresholding methods are, that in practice it is often hard to find thresholds that yield suitable results. Furthermore, the resulting segmentations are typically noisy and spatially incoherent.

1.3.1.2 Edge-based Segmentation

As stated before, an elementary assumption for image segmentation is that regions contain pixels which are homogeneous in a certain respect. This assumption can also be interpreted such that there has to be a change of the homogeneity criterion at the border between two adjacent image regions. A large group of early image segmentation methods therefore searched for grayscale value changes in order to find borders between segments. These intensity changes can be detected e.g. by using image derivatives: The gradient of a 2D image is defined as

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} \tag{1.4}$$



(a) (b)

Figure 1.6: Thresholding a grayscale image (a) results in two regions: Pixels in region 0 have a grayscale value smaller than the threshold θ, all other pixels belong to region 1 (b). In this example, the threshold of $\theta = 144$ was determined with a MATLAB implementation of the Otsu (1979) algorithm by Damien Garcia.

The partial derivatives I_x and I_y are not isotropic, hence the magnitude of the gradient can be applied to achieve rotation invariance:

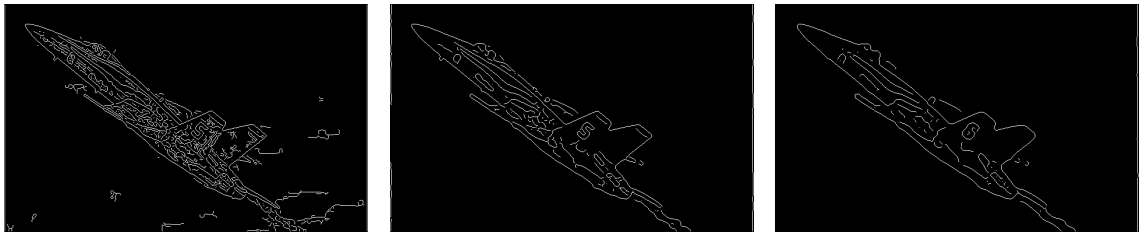
$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2} \tag{1.5}$$

An example for the gradient magnitude is shown in Figure 1.7(b).

(a) (b)

Figure 1.7: The magnitude of the image gradient shown in (b) is a powerful indicator for possible segment borders. Note that (a) was converted to grayscale before gradient computation and that the gradient was contrast enhanced for better visualization.

Note, that the magnitude of the gradient is typically noisy and does often not produce distinctive gradients (especially in highly textured regions). Therefore, additional post-processing steps are employed in order to find locally maximal gradients, closed contours or even complete shapes in the gradient image. A popular method of that kind has been presented by Canny (1986): Before calculating the gradient magnitude, the images are smoothed by convolution with a Gaussian filter with a specific standard deviation σ in order to reduce noise. Then, non-maximum suppression is performed to ensure that the width of an edge cannot exceed one pixel. Finally, pixels with a gradient magnitude between two thresholds $\theta_1 < \theta_2$ are marked as edge pixels. See Image 1.8 for an example with different values of σ.



(a) (b) (c)

Figure 1.8: The Canny edge detector employs Gaussian smoothing, non-maximum suppression as well as hysteresis thresholding to find distinctive edges in gradient maps. The Canny edge detector response of the image shown in 1.7(a) with a smoothing filter with $\sigma = 1$ is shown in (a), $\sigma = 2$ in (b) and $\sigma = 3$ in (c).

A huge class of segmentation algorithms aims at finding connected contours in the intensity gradient maps, an extensive overview of such methods is given in (Sonka et al., 2007). Another important family of segmentation methods operating on image gradients employ the Hough transform introduced by Hough and Powell (1960): While initially invented to detect straight lines in an image, there exist many extensions that allow for detection of arbitrary parameterizable shapes (cf. the survey of Illingworth and Kittler (1988)).

1.3.1.3 Region-based Segmentation

Contrary to edge-based segmentation, region-based segmentation directly aims at finding and exploiting homogeneity criteria. A basic and intuitive approach for region-based segmentation is Region Merging, where the homogeneity criterion is expressed by an

arbitrary binary evaluation function $f(E_i)$, which is true when E_i is homogeneous. Such a function could be based on comparing e.g. grayvalues or color properties of the region E_i. In Region Merging, one starts of with assigning every image pixel a separate region. Then, adjacent regions E_i and E_j are merged iteratively as long as $f(E_i \cup E_j)$ stays true. Note that the result of this algorithm depends heavily on the order of the homogeneity checks.

Another elementary region-based segmentation approach is the Watershed segmentation, where the regions are found as local valleys in the topographic surface spanned by the intensity values of the image. An efficient algorithm for Watershed segmentation has been presented by Vincent and Soille (1991): Their algorithm starts by assigning each of the lowest pixels in the topographic surface (i.e. the pixels with the smallest grayvalue) to a separate region. These regions are then grown by adding previously unassigned pixels, which are located higher in the topographic surface. This process can be interpreted as flooding the valleys of the surface with water until watersheds are found. As the water level increases, regions meet at these watersheds, which subsequently form the borders between the regions (see Figure 1.9).

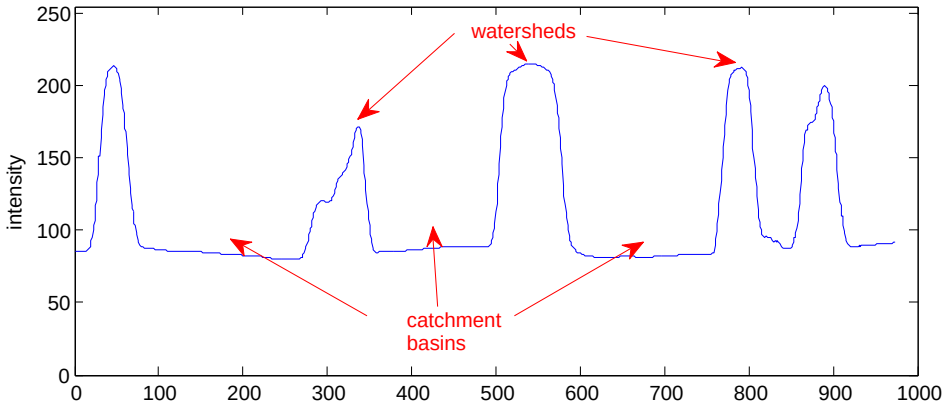
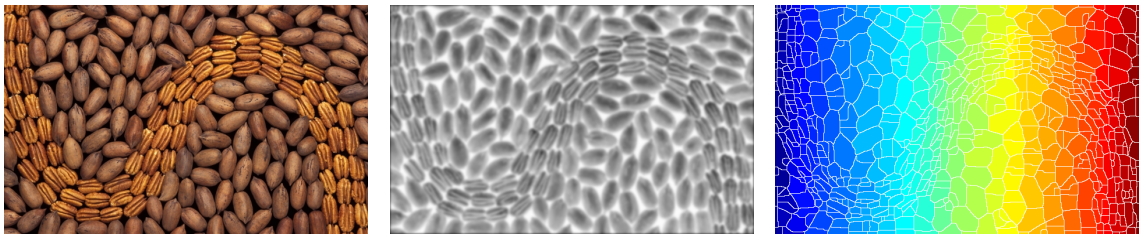


Figure 1.9: Watersheds in a one-dimensional example: The intensity profile of an image is interpreted as a topographic surface, which is immersed with water. Regions are formed by catchment basins, borders between regions are identified where these basins meet (i.e. watersheds).

For natural images, the Watershed algorithm typically yields a large number of regions i.e. it over-segments the image. In order to produce less segments, images are often smoothed with e.g. Gaussian filters before segmentation (see Figure 1.10).



(a) (b) (c)

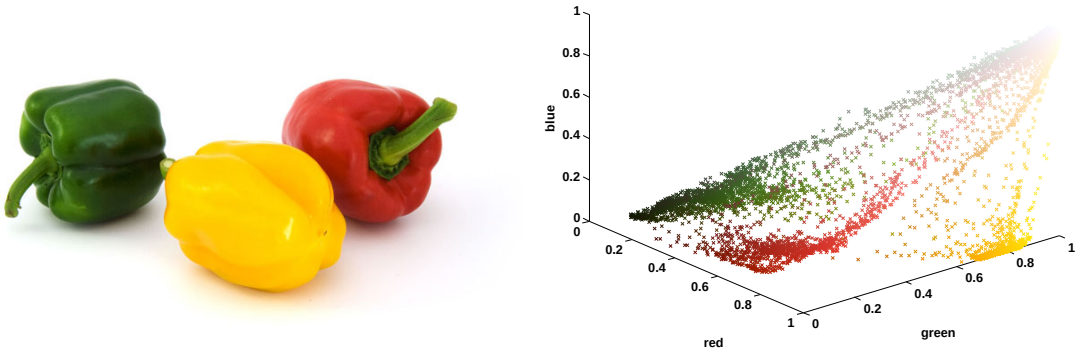
Figure 1.10: An example of a Watershed segmentation: The input image (a) is converted to grayscale, inverted and smoothed with a Gaussian filter ($\sigma = 2$) to yield (b). The Watershed segmentation is given in (c), where the white lines depict the watersheds between the colored segments. The employed Watershed algorithm (Meyer, 1994) is implemented in MATLAB.

1.3.1.4 Advanced Techniques

The unsupervised image segmentation methods described so far are early works of image segmentation research. They mostly operated on the grayscale range of images solely due to the importance of grayscale images as well as the limited computational power available at the time of their invention. Moreover, they try to solve the segmentation problem based on either edge or region characteristics independently, both of which have specific drawbacks. In contrast, state-of-the-art segmentation algorithms are often a useful combination of edge-based and region-based elements, which operate on color or even higher dimensional feature spaces. Furthermore, many algorithms employ additional constraints e.g. that resulting regions have to be spatially compact.

A very popular segmentation approach is the mean shift segmentation algorithm of Comaniciu and Meer (1997), which aims at finding modes in a feature space composed from color values and spatial pixel coordinates. See Figure 1.11 for an example: The red, green and yellow peppers in 1.11(a) form modes in the RGB-color space shown in 1.11(b), which can be detected by a mean shift procedure in order to obtain regions with coherent color signatures. The additional incorporation of the pixel coordinates yields spatially compact regions.

An important line of research is represented by energy minimization approaches, where cost functions are designed to control the behavior of an algorithm. These cost functions are then minimized e.g. by using variational approaches (Mumford and Shah, 1989) or using graph based methods (Shi and Malik, 1997; Wu and Leahy, 1993). An



(a) (b)

Figure 1.11: Cluster analysis can be used to segment images automatically: The peppers (a) form distinctive modes in the RGB-color space (b), which can be detected using basic unsupervised learning algorithms.

example for a graph-based algorithm is the method of Felzenszwalb and Huttenlocher (2004): The image is represented as a graph $G = (V, E)$, with the pixels representing vertices V with edges E between them. The edges are employed to encode a dissimilarity measure $w(p, q)$ between adjacent pixels p and q. Felzenszwalb and Huttenlocher define a segmentation to be a subset of the edges $E' \subseteq E$ forming a connected component C in the graph. The basic observation is that edges between vertices in the same component should exhibit a lower dissimilarity than edges between vertices of different components. This is modeled by an internal difference of a component C that is represented by the largest edge weight in the minimum spanning tree of the component:

$$Int(C) = \max_{\{p,q\} \in MST(C,E)} w(p, q). \tag{1.6}$$

Furthermore, the difference between two components is defined as the smallest edge weight between these components:

$$Dif(C_1, C_2) = \min_{p \in C_1, q \in C_2, \{p,q\} \in E} w(p, q). \tag{1.7}$$

Based on these models, they evaluate the evidence for a boundary between two compo-

nents as

$$D(C_1, C_2) = \begin{cases} true: & \text{if } Dif(C_1, C_2) > min(Int(C_1) + \frac{k}{|C_1|}, Int(C_2) + \frac{k}{|C_2|}) \\ false: & \text{else,} \end{cases} \tag{1.8}$$

where the terms $\frac{k}{|C_1|}$ and $\frac{k}{|C_2|}$ with a constant parameter k act as threshold functions to control the size of the resulting components. Initially, all pixels form separate components. Then, based on sorted edge weights, components are merged iteratively according to the boundary evidence predicate (1.8). This predicate allows for capturing fine details in image regions with low variability while suppressing details in regions with high variability.

For performance comparison of unsupervised image segmentation algorithms, Martin et al. (2001) presented the Berkeley Segmentation Benchmark BSDS300 featuring 300 labeled images. This benchmark dataset has become one of the most employed datasets for the evaluation of unsupervised segmentation methods, therefore Arbelaez et al. (2010) recently released the superset BSDS500 with an additional 200 annotated images. Due to the popularity of these datasets, performance results exist for the most important unsupervised segmentation algorithms. Hence, the benchmark papers of Martin et al. and Arbelaez et al. give a detailed overview of the current state-of-the-art.

1.3.2 Supervised Segmentation

Supervised (a.k.a. semantic) segmentation aims at assigning every pixel a label out of a predefined set of labels from a training database. E.g., given a training database with many labeled images depicting dogs, horses and grass, assign each pixel in an unseen image one of these three labels. To accomplish this task, state-of-the-art supervised segmentation methods typically perform four steps:

- Find a good description for the visual and/or spatial properties of all labels. These descriptions can be based on arbitrary features, starting from simple color models up to sophisticated local feature descriptions.
- Build an appropriate model for the labels in the database. This step amounts to training a supervised machine learning algorithm based on the representation computed in the previous step.
- Find for every pixel the label with the highest likelihood based on the trained models.

- Incorporate some kind of spatial regularization to produce compact regions from the noisy posterior probabilities obtained from the models.

Note that the need for a training set limits the applicability of such segmentation methods drastically, as they can only be applied when the appearance and content of images are known beforehand. Furthermore, to have a descriptive training set for the labels to be covered, the database needs to consist of many different images, which in turn requires lots of labeling effort. Also, as the number of different labels increases, the image representation needs to get more and more sophisticated.

A recent example for a semantic segmentation algorithm has been presented by Schroff et al. (2008): They employ Random Forests (Breiman, 2001) to model high-dimensional features obtained from concatenating RGB values, textons, a filterbank response as well as Histogram of oriented Gradients (HoG) features (Dalal and Triggs, 2005). With additional regularization of the posterior probabilities obtained from the Random Forests, the approach of Schroff et al. yields state-of-the-art results.

In (Shotton et al., 2008), the authors present a powerful image representation called semantic texton forests. Furthermore, they employ an additional algorithm stage to model spatial relations, e.g. that pixels labeled as 'sheep' are more likely to occur in the vicinity of 'grass' pixels than in the vicinity of 'building' pixels.

There exist several benchmark datasets for evaluation of supervised segmentation methods, two of which are the MSRC and the PASCAL VOC dataset: The MSRC object recognition database by Shotton et al. (2006) consists of 591 images in which every pixel is assigned one of 21 predefined labels, such as e.g. building, tree, sky, water, flower, bird, cat or dog. The PASCAL Visual Object Classes Challenge (VOC) (Everingham et al., 2010) is a yearly object recognition competition, which since 2007 also contains a 20-class segmentation benchmark.

1.4 Interactive Segmentation

Based on the considerations presented so far, we develop a powerful interactive segmentation framework in this thesis. Interactive (a.k.a. semi-supervised) segmentation deals with partitioning an image into regions according to user-provided input.

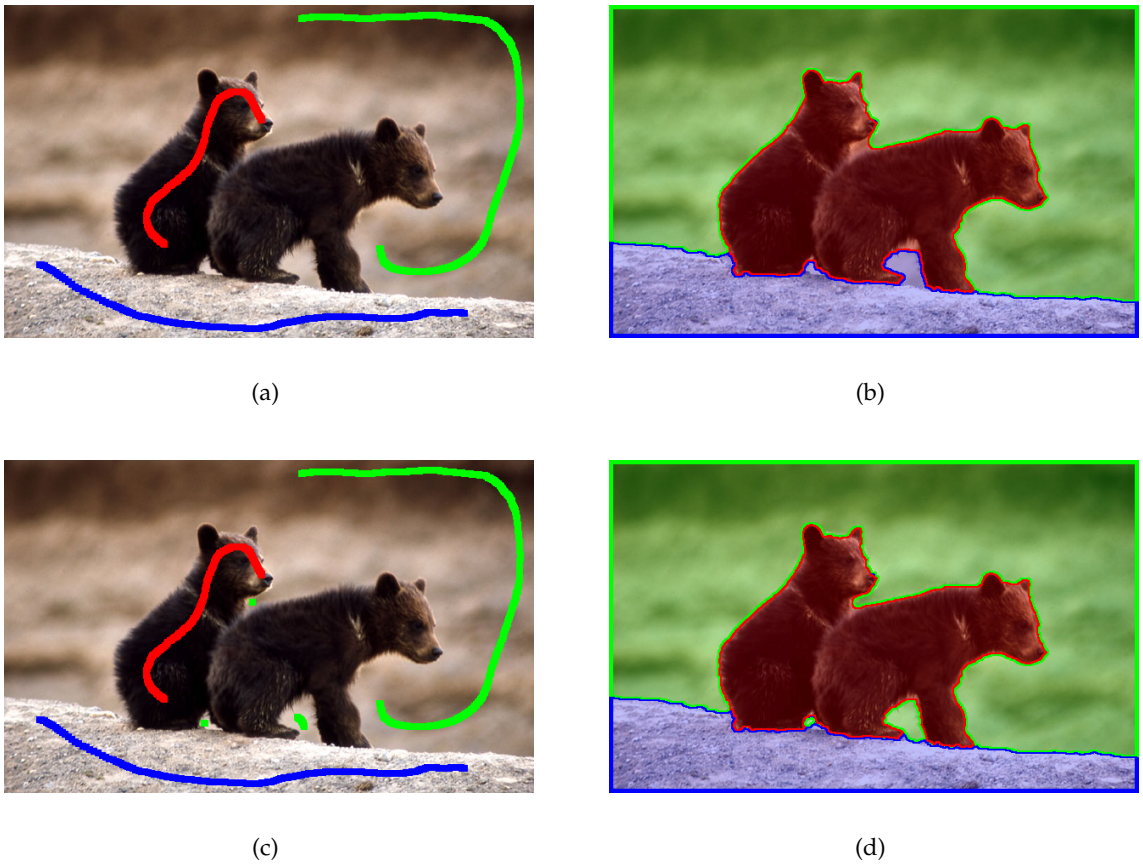
1.4.1 Conceptual Differences

The existence of a human operator makes a large difference in terms of properties and requirements of interactive methods compared to supervised and unsupervised algorithms. The main challenges of unsupervised segmentation come from the employed unsupervised machine learning algorithms: Important variables such as the number of image segments, a suitable feature space or the shape of the clusters in the feature space have to be estimated from the data. In supervised segmentation, these problems do not occur due to the usage of supervised machine learning algorithms. Therefore, such segmentation methods are restricted to the small number of segment categories trained beforehand from a database and cannot be employed to segment an arbitrary image. Interactive segmentation methods do not suffer these conceptual shortcomings: In contrast to unsupervised segmentation, the problem is well-defined by the input provided by the user. Furthermore, in contrast to supervised segmentation, any image can be processed without the need for a huge training database.

1.4.2 Interaction

The basic interactive segmentation process is depicted in Figure 1.12: Initially, the human operator marks objects he wants to segment. In Figure 1.12(a), these markings are brush strokes drawn inside the regions one wants to segment, other possibilities include marking contours between segments, drawing bounding rectangles or polygons or providing single seed pixels for segments. After the user has specified what he wants to segment, the segmentation algorithm tries to "understand" the users intention and produce a segmentation (Figure 1.12(b)). This step amounts to generalizing knowledge gained from the user input to the rest of the image. Afterwards, the user can refine segmentation results by providing additional input (Figure 1.12(c) and 1.12(d)).

These steps are repeated until the user is satisfied with the result, which directly leads to two essential properties of an interactive segmentation algorithm: First, it needs to be fast to compute, i.e. there should be only little time passing between drawing inputs and getting a segmentation. Second, the amount of user input needed to obtain the desired segmentation should be as little as possible. This implies, that the algorithm has to quickly and accurately capture what the user wants to segment. These requirements are mutually exclusive to a certain extent: Keeping the amount of user input needed small imposes the need of sophisticated feature representation and good learning algorithms, which in turn increases the computational complexity of the problem.



(a) (b)

(c) (d)

Figure 1.12: Interactive Segmentation is a process, where an operator guides an algorithm until a desired result is obtained: Image (a) shows user provided input which leads to an intermediate segmentation (b). The operator corrects this suboptimal result by providing additional input (c), which leads to the final segmentation (d).

1.4.3 Related Work

The importance of interactive image segmentation is visible not only in the vast body of literature on this topic, but also in their numerous occurrence in commercial and open source image editing software packages. In this section, we give an extensive overview of common approaches as well as recent methods in the field of interactive image segmentation.

1.4.3.1 Manual Drawing Tools

The simplest user-guided segmentation methods are tools that allow to mark regions or contours between regions manually. These approaches are completely independent

of the content of the image and therefore more a drawing tool than a segmentation method. However, due to their simplicity, such methods are implemented in all major graphics processing programs such as Adobe Photoshop, Corel Draw or GIMP.

An intuitive approach would be to assign each pixel a label manually with brush-like tools. As digital images nowadays have several millions of pixels, this method would require an unacceptable amount of user input. Therefore, the simplest common segmentation tools require the user to mark only the border pixels between segments. An example would be the Lasso tool of Adobe Photoshop illustrated in Figure 1.13(a): Starting with a mouse click, all pixels under the cursor are added to the segment border. When the mouse button is released, the contour is closed linearly between the first and the last point of the contour. The major drawback of this tool is that the marked border points cannot be changed, thus, once the user makes a labeling mistake, he has to start from scratch. Also, even little tremor in guiding the mouse leads to jittered borders. Furthermore, the amount of user input is still very high.

Another approach is to approximate the segment border with geometric primitives: Tools such as the Polygonal Lasso of Adobe Photoshop (Figure 1.13(b)) or the *roipoly()* command of MATLAB let the user specify the border between segments by drawing corner points of a polygon. While drawn points are still not changeable with the Polygonal Lasso tool, the *roipoly()* command allows to move polygon points afterwards.

1.4.3.2 Intelligent Scissors

One of the first common approaches to user-guided image segmentation making use of image information is the Intelligent Scissors (a.k.a. Livewire) approach of Mortensen and Barrett (1995). In their work, they take the common assumption that there is a change in pixel intensity between adjacent segments of an image. Therefore, Mortensen and Barrett used image gradients to define the local cost between a pixel p and its neighbor q as

$$d_{LIVEWIRE}(p,q) = \lambda_Z d_Z(q) + \lambda_G d_G(q) + \lambda_D d_D(p,q) \tag{1.9}$$

where λ_Z, λ_G and λ_D are weights to steer the relative influence of the three different cost functions $d_Z(q)$, $d_G(q)$ and $d_D(p,q)$: The first cost function $d_Z(q)$ of Equation (1.9) is based on Laplacian zero-crossings. The second partial derivative of an image can be

approximated by convolution of the image with a Laplacian kernel:

$$I_L = I * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{1.10}$$

The zero-crossings of the Laplacian represent the points of maximal gradient magnitude, which is exploited by setting the cost to zero at these image locations:

$$d_Z(q) = \begin{cases} 0: & \text{if } I_L(q) = 0 \\ 1: & \text{else} \end{cases} \tag{1.11}$$

The second term of Equation (1.9) is formed by the gradient magnitude:

$$d_G(q) = 1 - \frac{\|\nabla I(q)\|}{max(\|\nabla I\|)} \tag{1.12}$$

The last term d_D represents the direction of the gradient: With $D(p)$ representing the unit vector perpendicular to the gradient direction at the point p and $L(p,q)$ denoting the bidirectional edge vector between pixels p and q

$$L(p,q) = \begin{cases} q-p: & \text{if } D'(p) \cdot (q-p) \geq 0 \\ p-q: & \text{else,} \end{cases} \tag{1.13}$$

the cost $d_D(p,q)$ is given as

$$d_D(p,q) = \frac{1}{\pi} \left\{ cos[d_p(p,q)]^{-1} + cos[d_q(p,q)]^{-1} \right\}, \tag{1.14}$$

with

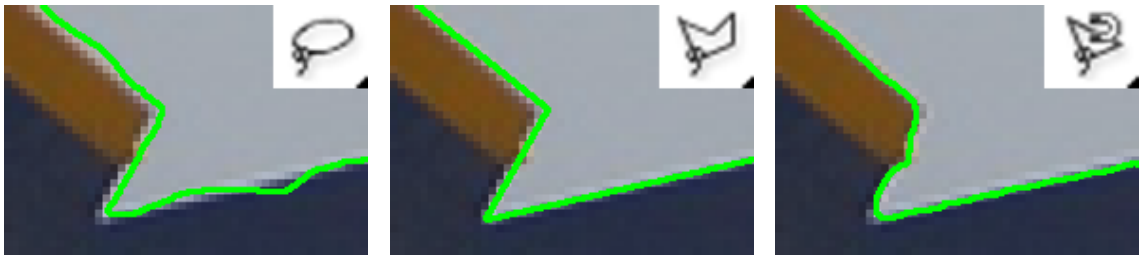
$$d_p(p,q) = D'(p) \cdot L(p,q) \tag{1.15}$$

$$d_q(p,q) = D'(q) \cdot L(p,q). \tag{1.16}$$

Hence, when the gradient directions of two adjacent pixels p and q are similar, a low cost is assigned. This leads to a penalization of sharp changes in the direction of the final segment boundary. Having calculated the local cost $d_{LIVEWIRE}(p,q)$ for every pixel and its four neighbors, a graph is constructed with a vertex for every pixel and a link between adjacent pixels representing the local cost between these pixels. Given two points in this

graph, an optimal path between the points can be computed using optimal graph search algorithms as presented by Dijkstra (1959).

Segmenting an image with the Intelligent Scissors approach then amounts to the following procedure: The user marks a starting point for a region boundary with his mouse pointer. As he moves his cursor, the optimal path between the starting point and the current cursor position is evaluated and displayed. When the user accepts the currently displayed path with a mouse click, the newly added point is taken as starting point for the next boundary segment. Finally, the optimal path between the first and the last point of the boundary is taken to close the contour. While this method performs well on images where there are distinctive intensity gradients between image segments, lots of user input is required to obtain acceptable segmentations in highly textured areas with lots of large gradients. The Intelligent Scissors method is implemented in GIMP, Adobe Photoshop features a similar approach they call Magnetic Lasso (see Figure 1.13(c)).



(a) (b) (c)

Figure 1.13: Segment boundary marking methods in Adobe Photoshop: (a) shows the Lasso tool, which allows to directly draw the segment boundary, leading to a noisy result. With the Polygonal Lasso (b) the boundary is approximated as being piecewise linear, by letting the user mark vertices of a polygon. Compared to (a), this is less noisy, therefore curvilinear boundaries and small structures are difficult to mark. (c) shows the result of the Magnetic Lasso method, which snaps the boundary to large image gradients. Note that the selection boundary has been emphasized and colored for visualization.

A related approach to the Intelligent Scissors method using gradient descent on blurred feature maps was presented by Gleicher (1995). The Intelligent Scissors algorithm itself was improved by combining it with the Watershed segmentation algorithm (Vincent and Soille, 1991). An overview over such combined algorithms, which were successfully applied e.g. in (Barrett and Cheney, 2002; Reese and Barrett, 2002), is given in (Mortensen et al., 2000).

1.4.3.3 Seeded Region Growing

In contrast to boundary-based segmentation methods which find segment boundaries by searching for gradient maxima such as the Intelligent Scissors approach, region-based segmentation methods rely on grouping pixels with similar intensity or color. A common approach to region-based segmentation are region growing (Zucker, 1976) algorithms: Region growing methods are iterative algorithms which typically start off with assigning one single pixel an initial label. Then, the intensity or color of the initial pixel and its neighboring pixels is compared. If the similarity exceeds a certain threshold, the neighboring pixels are given the same label as the initial pixel, otherwise a new region with a different label is started. This process is repeated until all pixels are labeled. While working well for images where the segments have a distinctive color signature, noise and highly textured areas affect the performance of such methods. Furthermore, the results heavily depend on the similarity threshold value chosen.

Seeded Region Growing (Adams and Bischof, 1994) is an adaption of the concepts described above which comes without the need for a similarity threshold, using a mechanism similar to the Watershed segmentation method (Vincent and Soille, 1991). The number of regions as well as an initial set of points for each region have to be specified by the user by manually placing seed pixels. The similarity between an unassigned pixel p and an adjacent region E is defined by

$$\delta(p, E) = \left| I(p) - \frac{1}{|E|} \sum_{q \in E} I(q) \right|, \tag{1.17}$$

which represents the difference of the intensity $I(p)$ to the arithmetic mean of the pixel intensities in the region E. Other similarity functions can be employed to represent color signatures or textural properties.

A related approach to Seeded Region Growing has been presented in (Tan and Ahuja, 2001). There, the authors allow to draw lines and regions in addition to points to get to an intermediate, coarse segmentation. The coarse segmentation is later refined and the border approximated to capture diffused boundaries. Therefore, the authors apply a border matting technique presented by Ruzon and Tomasi (2000), which was developed to perform soft-segmentation of objects which typically have no distinctive borders in natural images (e.g. trees, hair, water or smoke).

1.4.3.4 Magic Wand

The Magic Wand tool implemented in Adobe Photoshop is an elementary color-based algorithm for two-label segmentation. The user selects seed pixels by clicking at arbitrary pixels of the foreground object he wants to segment. All pixels having a color within a certain range to the colors of the seed pixels are added to the foreground selection. The Magic Wand tool makes no use of intensity gradients and does not impose any constraints on the shape of the foreground selection, hence the purely color driven selections tend to be very noisy. Therefore, the Magic Wand tool gives the user many ways to adjust and refine the selection:

- The user can add or remove seed pixels to the selection.
- The similarity threshold specifying the selection tolerance can be adjusted.
- The selection can be constrained to be connected.
- The contour of the selection can be modified: Operations such as anti aliasing, smoothing, expansion and contraction are supported.

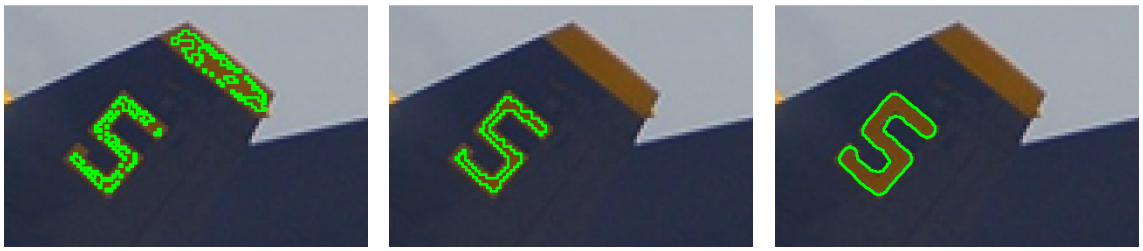
While the initial color segmentations are typically not satisfying, experienced users achieve good results using the interaction capabilities: Figure 1.14(a) shows an attempt to select a yellow digit on the blue vertical stabilizer of a jet plane. Setting one seed directly on the yellow digit produces a noisy selection where also the tip of the stabilizer is selected. In Figure 1.14(b), the similarity threshold is increased and the selection is constrained to be connected. Finally in Figure 1.14(c), the border is smoothed and expanded yielding an acceptable result.

1.4.3.5 Snakes / Active Contours

Active Contours (Kass et al., 1988) are splines that move to the border between two image segments driven by several forces in an energy minimization functional. The original energy, which the authors call snake energy due to the slithering movement during the minimization process, takes into account three force terms:

$$E_{SNAKE} = \lambda_1 \int_0^1 |C'(q)|^2 dq + \lambda_2 \int_0^1 |C''(q)|^2 dq - \lambda_3 \int_0^1 |\nabla I(C(q))| dq. \tag{1.18}$$

In this equation, $C(q) : [0,1] \rightarrow \mathcal{R}^2$ denotes a parametric planar curve describing the evolving contour and λ_1, λ_2 and λ_3 are real positive constants. The first two terms



(a) (b) (c)

Figure 1.14: Using the Magic Wand tool of Adobe Photoshop to segment the yellow digit: The initial selection (a) is purely color-driven and therefore noisy and not compact. Adjusting thresholds and adding constraints on the region (b) and contour (c) yields a good result. Note that the selection boundary has been emphasized and colored for visualization.

of (1.18) (internal energy) control the shape of the contour: The first derivative of the contour $C'(q)$ represents the tangents of the contour, hence

$$\int_0^1 |C'(q)|^2 dq \tag{1.19}$$

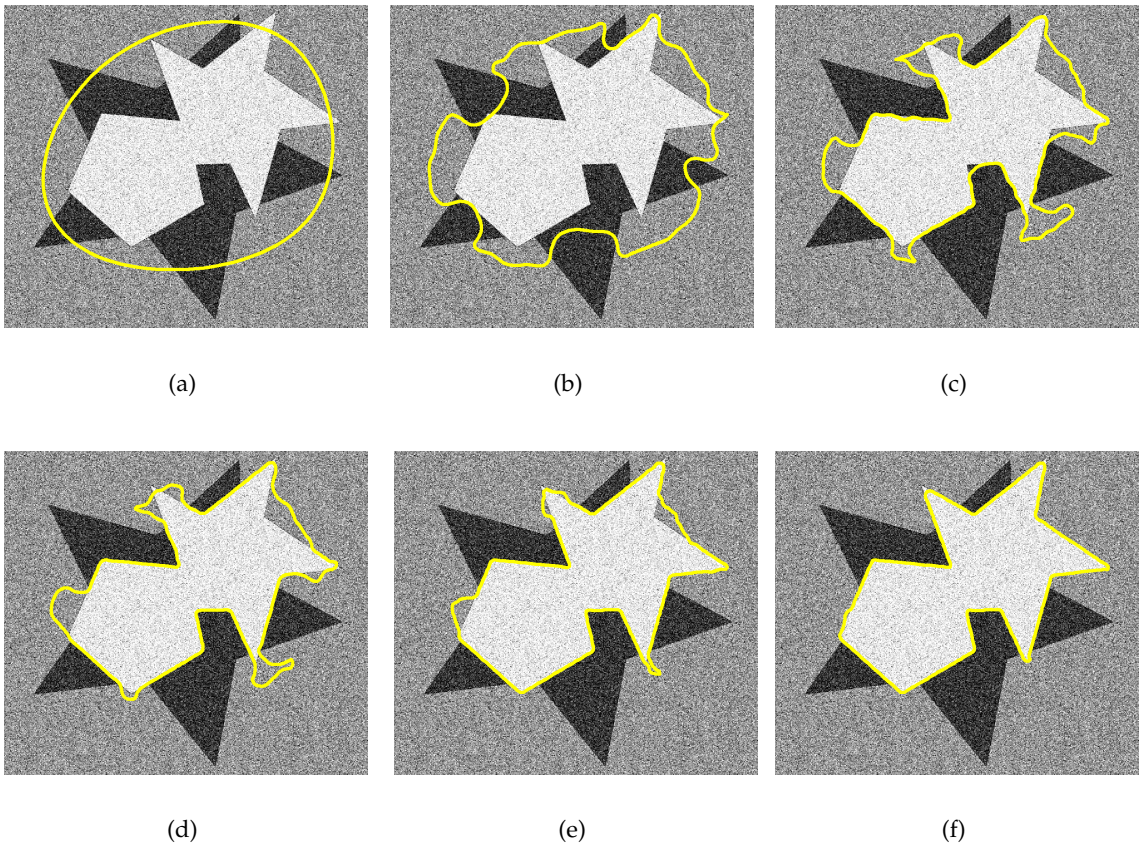
describes the integration of the norm of the tangent vectors over the whole contour, i.e. the length of the contour. Thus, when minimizing the energy E_{SNAKE}, (1.19) aims at keeping the length of the contour as small as possible. The second derivative $C''(q)$ represents the change in tangent directions of the contour, thus

$$\int_0^1 |C''(q)|^2 dq \tag{1.20}$$

yields the overall curvature of the contour. During minimization, (1.20) tries to make the contour smooth with little sharp directional changes. The third term of (1.18) (external energy) integrates the image gradient magnitude at the positions of the contour i.e. pulls the contour towards sharp gradients in the image. The constants λ_1, λ_2 and λ_3 are used to steer the influence of the three terms.

In the segmentation process, $C(q)$ is initialized by drawing a contour C_0 manually in the vicinity of the actual segment boundary. The final contour, which is obtained by performing gradient descent, is the one which minimizes E_{SNAKE} given the initial contour C_0 and the parameters λ_1, λ_2 and λ_3. Note that this is not necessarily a global mini-

mum, hence the final contour strongly depends on the initial contour C_0. An example segmentation based on Active Contours can be seen in Figure 1.15



(a) (b) (c)

(d) (e) (f)

Figure 1.15: Image segmentation using active contours: (a) shows the initial contour C_0, (b-f) show the segmentation after 1,3,5,10 and 20 iterations respectively. This segmentation has been performed using the Active Contour Toolbox by Eric Debreuve.

Two major drawbacks of the Active Contour model include that the topology is fixed with the initial contour and that the results of the energy minimization depend on the parametrization of the contour. In the Geodesic Active Contour model (Caselles et al., 1997), the contour is represented implicitly as the zero level-set of a higher order auxiliary function (Osher and Fedkiw, 2002; Sethian, 1999). In their approach, Caselles et al. try to find curves with minimum length in a space with Riemannian metric computed based on the image. In a Riemannian space, the length of a contour is given as

$$|C|_{\mathcal{R}} = \int_0^{|C|_e} \sqrt{\tau_s^T \cdot D(C(s)) \cdot \tau_s}\, ds, \tag{1.21}$$

where $|C|_\epsilon$ denotes the Euclidean length of the contour, τ_s represents the contour's unit tangent vector, $D(\cdot)$ states the local Riemannian metric at a given pixel and ds is the Euclidean element of length. If $D(\cdot)$ is given as the isotropic metric

$$D(\cdot) = diag(g(|\nabla I(\cdot))|), \tag{1.22}$$

with $g(\cdot)$ an edge indicator function vanishing at large gradient magnitudes, the Geodesic Active Contour energy is given as

$$E_{GAC} = \int_0^{L(C)} g(|\nabla I(C(q))|)\, ds, \tag{1.23}$$

with $L(C)$ denoting the length of the contour. Hence, E_{GAC} reduces to integrating the Euclidean element of length along the contour, weighted with the strength of the image gradient at the contour position. The trivial minimizer for E_{GAC} is $C = \emptyset$, thus there is a need for additional constraints to get a meaningful segmentation. When minimized with level-sets, the algorithm finds a local minimum of E_{GAC}, which depends on the initial contour. However, an algorithm based on minimal paths able to compute a global minimum of an equivalent energy functional has been presented by Cohen and Kimmel (1997). Furthermore, two popular algorithms able to compute the global minimum of E_{GAC} have been presented by Boykov and Jolly (2001) and Bresson et al. (2007): While the approach of Boykov and Jolly (cf. Section 1.4.3.6) uses a discrete graph representation, the weighted Total Variation algorithm of Bresson et al. (cf. Section 1.4.3.8) operates on a spatially continuous domain.

1.4.3.6 Graph Cut Segmentation

The high quality of segmentation results obtained with Snakes and the Geodesic Active Contour model led to the necessity of efficient ways to compute global optima of the underlying energy functionals. A seminal development is the work of Boykov and Jolly (2001), where the authors find the global minimum of the Geodesic Active Contour energy (1.23) in a discrete setting using graph-based approaches. In their work, they reformulate the problem to the cost function

$$E(u) = \lambda \cdot \sum_{p \in \mathcal{P}} R_p(u_p) + \sum_{\{p,q\} \in \mathcal{N}} [u_p \neq u_q] \cdot g(p,q), \tag{1.24}$$

where p, q are pixels of the set $\mathcal{P}$, $\{p, q\}$ is a pair of pixels from the set of all unordered pairs $\mathcal{N}$, u denotes a binary vector, which assigns pixels to either foreground / object or background, and $[u_p \neq u_q]$ is an indicator such that

$$[u_p \neq u_q] = \begin{cases} 1: & \text{if } u_p \neq u_q \\ 0: & \text{else.} \end{cases} \tag{1.25}$$

The energy functional of Boykov and Jolly consists of two terms: The first term

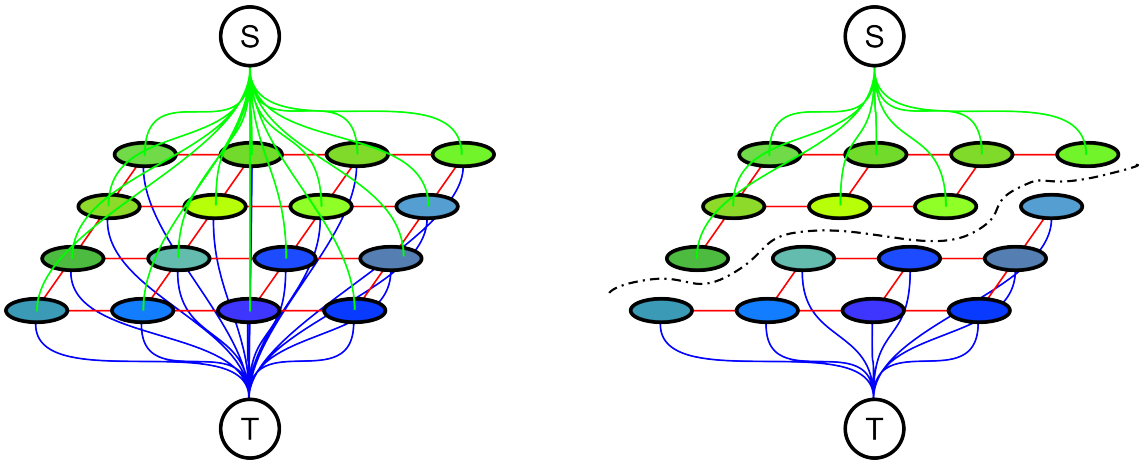
$$\lambda \cdot \sum_{p \in \mathcal{P}} R_p(u_p), \tag{1.26}$$

which is referred to as region term or unary term, employs a penalty $R_p(\cdot)$ for every pixel p, that represents the likelihood that p fits a foreground/background model. Boykov and Jolly train this model using intensity distributions of the seeded pixels with histograms. The second term

$$\sum_{\{p,q\} \in \mathcal{N}} [u_p \neq u_q] \cdot g(p, q), \tag{1.27}$$

which is referred to as boundary term or pairwise term, employs an edge indicator function $g(p, q)$, which is small when the pixels p and q exhibit different intensities.

Based on this energy, a graph $G = (V, E)$ with vertices V corresponding to the image pixels and edges E is constructed. Additionally, a source S and sink T node is added to represent the foreground and background unary potentials. Links between the nodes are established such that every pixel is connected to S and T (t-links) and pixels are interconnected according to a specific neighborhood relation such as 4-connectivity, 8-connectivity etc. (n-links) (see Figure 1.16(a)). The weights for the n-links between unseeded pixels p and q are assigned as $g(p, q)$ (pairwise term). The t-links are assigned based on the foreground/background model $\lambda R_p(\cdot)$ (unary term). Seeded pixels are treated differently: For a pixel marked as background, the weight of the t-link to the background terminal T is set to 0 and the weight of the t-link to the foreground terminal S is set to a value exceeding the sum of all n-links of the given pixel. Pixels marked as foreground are treated vice versa. In this completely defined graph, the segmentation is performed by searching for the minimum-cost cut between the two terminal nodes S and T (see Figure 1.16(b)). Boykov and Jolly employ a max-flow algorithm (Boykov and Kolmogorov, 2001, 2004), which finds the global minimum of the cost function (1.24).



(a) (b)

Figure 1.16: Graph Cut image segmentation of a 4x4 image: The image is first represented as a graph with two terminal nodes S and T (a). Pixels are linked together (red edges) based on their neighborhood relation (4-connectivity here) as well as with the terminal nodes (green and blue edges). The segmentation is then performed by finding the minimum cost cut between the terminal nodes (b).

An issue often addressed in the literature is the shrinking-bias: The pairwise term (1.27) reduces to summing up intensity gradients at the border of segments. The smaller the contour of a segment is, the less intensity gradients are accumulated which results in a smaller cost. This not only leads to spatially compact regions (which is the purpose of the pairwise term), but also introduces a bias towards smaller regions exhibiting shorter contours. Other effects of the shrinking bias are that elongated structures as well as small isolated regions are suppressed.

Another common problem are metrication errors: Boykov and Kolmogorov (2003) showed that the quality of solutions obtained with Graph Cut-based approaches suffer from low connectivity (i.e. a standard 4- or 8-neighborhood as employed in many algorithms) of the underlying graph. The length of this cut is defined as the sum of the edge weights w_e of the edges the cut C intersects:

$$|C|_G = \sum_{e \in C} w_e \tag{1.28}$$

In their paper, Boykov and Kolmogorov proved that with increasing connectivity as well as proper edge weights the length of the contour gets arbitrarily close to the Euclidean

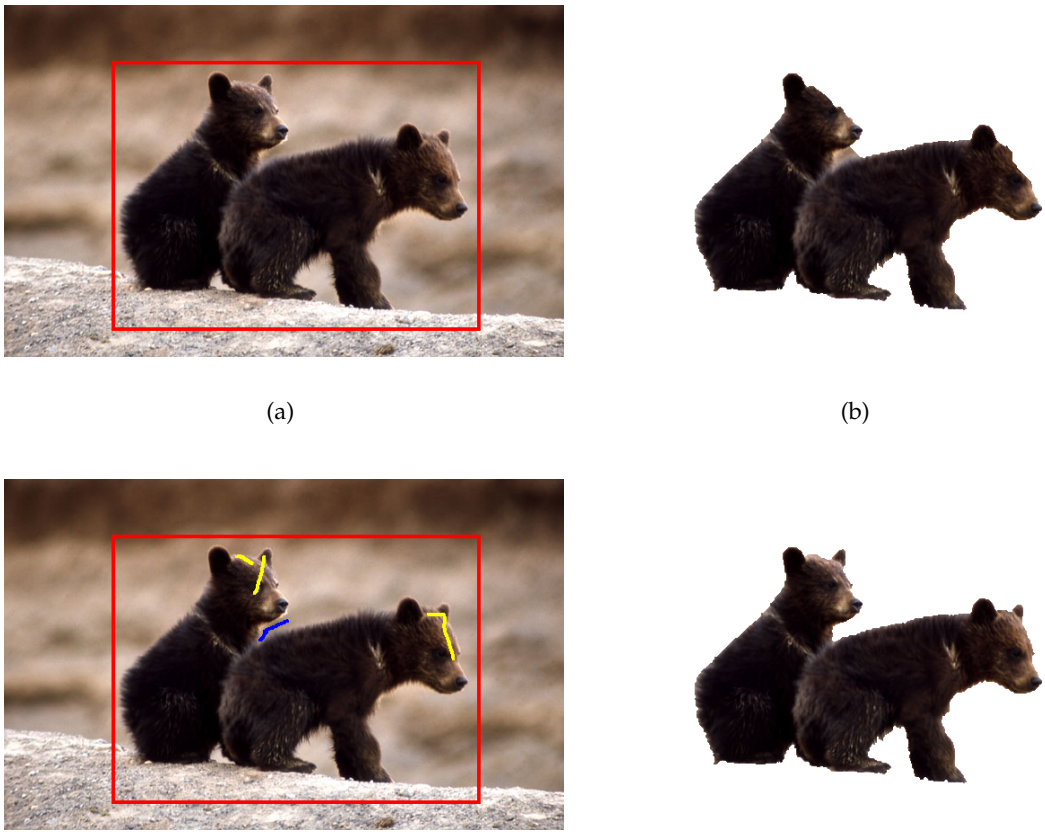
length $|C|_\epsilon$. Increasing the number of neighborhood connections however increases memory consumption and runtime of the minimization algorithm. Another drawback is, that the computation of a multi-label solution is not straightforward (Grady, 2006): Computing the globally optimal solution for more than two labels is equivalent to finding a multi way cut through a graph, which is known to be an NP-hard problem. Furthermore, the parallelization of the employed minimization algorithm is complex and does not scale well to a large number of processing cores (Goldschlager et al., 1982; Strandmark and Kahl, 2010).

In (Blake et al., 2004; Rother et al., 2004), the authors improve the performance of the initial algorithm by modeling foreground and background color signatures with Gaussian Mixture models. They reformulate the energy functional to

$$E(u) = -\sum_p log(h(z_p; u_p)) + \lambda \sum_{(p,q)\in\mathcal{N}} d(p,q)^{-1}[u_p \neq u_q]g(z_p, z_q), \tag{1.29}$$

where the image values are stated in an array $\mathbf{z} = \{z_1, \ldots, z_p, \ldots, z_N\}$, $d(p,q)$ represents the Euclidean distance between pixels p and q and $g(z_p, z_q) = e^{-\beta(z_p - z_q)^2}$ is an edge indicator function. The function $h(z_p; u_p)$ represents the consistency of the image value z_p and the current labeling u_p with the foreground/background color signatures. In their interactive tool GrabCut, Rother et al. let the user specify the object to segment by drawing a rectangle around it. The color signatures of foreground and background are deducted from pixels outside and inside the rectangle and a segmentation is performed. Based on this intermediate segmentation, the foreground color model is re-estimated and segmentation is performed iteratively until the result is stable. The converged segmentation can be altered by providing additional brush strokes (see Figure 1.17). Note that, while the intermediate segmentations are globally optimal w.r.t. the seeds and the color models, there is no guarantee for optimality of the overall iterative algorithm (Grady, 2006). Vicente et al. (2009) showed how this problem can be modeled by using graphs with higher order neighborhoods leading to algorithms with bounded optimality. In spite of the drawbacks of Graph Cut-based algorithms, the GrabCut tool is widely used due to its simplicity, speed as well as the high quality segmentation results. E.g. the software package Microsoft Office 2010 includes a foreground extraction method based on GrabCut.

A common way to increase the speed of graph-based segmentation tools is the reduction of the size of the graph: The tool Lazy Snapping by Li et al. (2004) first employs the Watershed algorithm (Vincent and Soille, 1991) to compute an oversegmentation of



(a) (b)

(c) (d)

Figure 1.17: The interactive segmentation tool GrabCut allows to specify an object of interest by marking a bounding rectangle (a). The intermediate segmentation (b) can be improved by providing additional brush-strokes (c) in order to obtain a good segmentation (d).

the image. The graph is then constructed such that the nodes represent the segments obtained with the Watershed algorithm. With this procedure, the size of the graph is reduced significantly. Recently, Liu et al. (2009) introduced their tool Paint Selection, which uses a coarse-to-fine approach to accelerate the optimization process. Instead of solving the problem for the whole image at the same time, they split up the segmentation task into consecutive local segmentations. Together with heuristic elements such as automatic deletion of misleading scribbles and viewport-based local selection, they achieve impressive results both in terms of speed and usability.

Recently, the incorporation of shape priors to Graph Cut-based image segmentation was demonstrated: Veksler (2008) showed the use of star-shape priors in order to seg-

ment star-convex objects. Star-convex objects w.r.t. a star center point c guarantee, that if a point p is part of the segmentation, any point q on a direct line between p and c must also be part of the segmentation. Gulshan et al. (2010) extended this idea such that an object can be composed out of several star-convex objects. Furthermore, they incorporated geodesic distances into this framework by allowing not only straight lines between p and c (i.e. the shortest path in Euclidean space), but the shortest path in a Riemannian space.

Other Tools based on minimizing the Geodesic Active Contour energy using graph-cuts comprise SIOX (Friedland et al., 2005), which is implemented in GIMP as Foreground Selection Tool and others (Han et al., 2009; Lempitsky et al., 2009).

1.4.3.7 Random Walker

Another interactive segmentation algorithm based on graph-representation is the Random Walker approach (Grady, 2006; Grady and Funka-Lea, 2004). Here the image is represented as a weighted graph $G = (V, E)$ with n vertices $v \in V$ and m edges $e \in E \subseteq V \times V$. Every edge e_{ij} between vertices v_i and v_j is assigned a weight w_{ij} representing the intensity gradient between the edges. Grady employ a Gaussian weighting function of the form

$$w_{ij} = e^{-\beta \cdot (I(i) - I(j))^2} \tag{1.30}$$

to represent the intensity changes between pixel values $I(i)$ and $I(j)$. Having constructed this graph, seed vertices with multiple ($k >= 2$) labels can be defined. Segmentation is performed by calculating for every vertex the probability, that a random walker starting at this location first reaches a specific seed vertex. The result of the algorithm is a probability vector with k entries for every image pixel, from which a final segmentation can be obtained by assigning the pixel the label with the highest probability value.

The computation of a biased random walk for every pixel and every label is unfeasible in practice. Instead, the fact that the Dirichlet problem

$$D[u] = \frac{1}{2} \int_{\Omega} ||\nabla u||^2 d\Omega \tag{1.31}$$

with boundary conditions at the locations of the seed vertices leads to the same solution as the random walks, is exploited. In (Grady, 2006), the solution of the combinatorial Dirichlet problem is derived as follows: First, the Laplacian matrix for the graph is

constructed, such that

$$L_{ij} = \begin{cases} d_i : & \text{if } i = j, \\ -w_{ij} : & \text{if } v_i \text{ and } v_j \text{ are adjacent vertices,} \\ 0 : & \text{else.} \end{cases} \tag{1.32}$$

Here, d_i denotes the degree of the node v_i, which sums up the weights of all adjacent edges. Then, the Dirichlet integral can be formulated as

$$D[x] = \frac{1}{2} \sum_{e_{ij} \in E} w_{ij}(x_i - x_j)^2 = \frac{1}{2} x^T L x. \tag{1.33}$$

The goal is to find the function x that minimizes the energy $D[x]$. For this purpose, the nodes are divided into two sets, containing the seed nodes V_M and the unseeded nodes V_U. Assuming that L is ordered, such that the seeded nodes come first, the energy can be decomposed to

$$D[x_U] = \frac{1}{2} [x_M^T x_U^T] \begin{bmatrix} L_M & B \\ B^T & L_U \end{bmatrix} \begin{bmatrix} x_M \\ x_U \end{bmatrix} \tag{1.34}$$

and

$$D[x_U] = \frac{1}{2} (x_M^T L_M x_M + 2 x_U^T B^T x_M + x_U^T L_U x_U). \tag{1.35}$$

After differentiation with respect to x_U, the minimization leads to a system of linear equations

$$L_u x_u + B^T x_m = 0 \tag{1.36}$$

with the number of unseeded vertices $|V_U|$ unknown variables. When solving this sparse linear system for a specific label $l \in \{1, 2, \ldots, k\}$, the probabilities for the seeded vertices with the label l are set to 1 and the probabilities for the seeded vertices with any other label are set to 0. As the probabilities for all k labels for a given vertex sum up to 1, a total of $k - 1$ sparse linear systems have to be solved. Computing the solution directly with LU-decomposition is impracticable for large image sizes due to high memory consumption. However, iterative solvers exist (Hackbusch, 1994) with a small memory footprint, that can be computed in parallel and have successfully been ported to graphics processing units (GPUs) (Bolz et al., 2005).

The algorithm presented in (Grady and Funka-Lea, 2004) leads to a unique solution with K connected regions i.e. the topology of the final segmentation is fixed by the seed points. Modifications of the algorithm including prior models (Grady, 2005) allow

for isolated segments without seed points. In (Sinop and Grady, 2007), the authors show the close relationship between the Random Walker algorithm and Graph Cut-based approaches: They reformulate the energy functional (1.33) to

$$\left[\sum_{e_{ij}\in E}(w_{ij}\cdot|I(i)-I(j)|)^q\right]^{\frac{1}{q}}, \tag{1.37}$$

such that the pairwise term of the Graph Cut energy correspond to the case where $q=1$ and the Random Walker approach corresponds to $q=2$. Recently, Zhang et al. (2010) showed the relation between the Random Walker algorithm and isotropic diffusion, and develop a novel algorithm including an anisotropic Diffusion tensor.

The Random Walker approach is popular for its simplicity, as the global optimum of the cost function can be derived solving basic sparse linear systems. However, the approach also exhibits a few drawbacks: The simplicity of the region model (i.e. only intensity differences between pixels are penalized) leads to difficulties when dealing with highly textured areas. Furthermore, the more gradients an image exhibits, the more sensitive the algorithm gets with respect to the seed positions. Another common point of criticism of the Random Walker algorithm is the lack of a geometric meaning: While the Geodesic Active Contour energy aims at minimizing the length of the segmentation border, a geometric interpretation of the Random Walker algorithm is straightforward only in the (useless) case of a homogeneous image, i.e. where the image exhibits no gradients.

1.4.3.8 Weighted Total Variation

When the Geodesic Active Contour energy is solved in a discrete setting (cf. GrabCut), one has to make assumptions on how pixels are connected. The more neighbors a pixel has (i.e. taking into account e.g. 16 neighbors instead of 8 or 4) leads to more accurate results, but also highly increases the amount of memory needed. Assuming low connectivity reduces the memory footprint and speeds up graph-based algorithms, but leads to metrication errors (Boykov and Kolmogorov, 2003).

Bresson et al. (2007) introduced weighted Total Variation as an alternative to the Geodesic Active Contour energy:

$$TV_g=\int_\Omega g(x)|\nabla u|d\Omega. \tag{1.38}$$

In this equation, $g(x)$ describes an edge indicator function and $u \in \{0,1\}$ a characteristic function indicating whether a pixel belongs to foreground or background.

When u is allowed to take an arbitrary value in the interval $[0,1]$, the problem becomes convex, allowing to compute a global minimizer. Bresson et al. perform segmentation based on the functional

$$TV_g = \int_\Omega g(x)|\nabla u| d\Omega + \lambda \int_\Omega |u - f| d\Omega, \tag{1.39}$$

which is essentially the weighted TV-norm together with a data-fidelity term. In this term, $f \in [0,1]$ is used to incorporate region information.

This framework has been used by Unger et al. (2008) in their interactive segmentation tool TVSeg. In their work, Unger et al. solve the energy functional

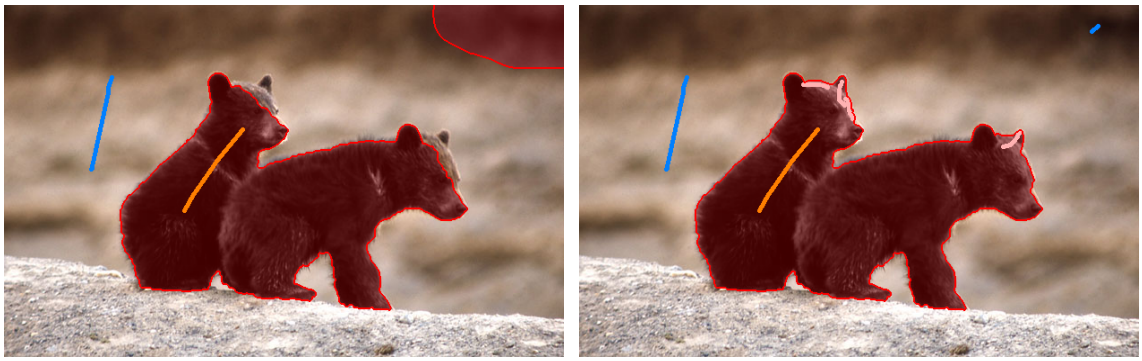
$$E = \int_\Omega g(x)|\nabla u| d\Omega + \int_\Omega \lambda(x)|u - f| d\Omega, \tag{1.40}$$

which exhibits a per-pixel weighing variable $\lambda(x)$ to incorporate user constraints: They let the user draw brush strokes in foreground and background regions and employ color histograms to model foreground and background color distributions. The values of f are then initialized by either the strokes, or the foreground/background probability based on the color histograms. The functional (1.40) also allows to set hard constraints (i.e. forcing a specific pixel to either foreground or background) via the variable $\lambda(x)$ (see also Figure 1.18).

Compared to Graph Cut-based segmentation methods, the weighted TV framework does not suffer from metrication errors. Furthermore, the employed variational energy minimization schemes perform iterative pixel updates that depend only on the pixel itself as well as its direct neighbors. These schemes can therefore be implemented in parallel in order to greatly speedup the computation time: Unger et al. implemented the core parts of their tool using NVIDIA CUDA to use the massive parallelism of graphics processors. This reduces the time needed to compute a segmentation to parts of a second for 640x480 image dimensions, allowing for convenient user interaction.

1.4.3.9 Geodesic Segmentation

The idea behind geodesic segmentation is to compute weighted distance functions between seeded and unseeded image pixels. The geodesic distance between two pixels p, q



(a) (b)

Figure 1.18: The interactive segmentation tool TVSeg models color distributions of foreground and background using color histograms based on brush strokes (a). The segmentation can be altered by providing additional brush strokes to remodel the color histograms (i.e. the blue stroke in the top right corner of (b)), or by forcing pixels to either foreground or background (the pink strokes over the heads of the bears in (b)).

over a path $C_{p,q}$ is written as

$$d(p,q) = \min_{C_{p,q}} \int_0^1 |\nabla I \cdot \dot{C}_{p,q}(s)| ds, \tag{1.41}$$

where ∇I represents the image gradient. $\dot{C}_{p,q}(s)$ is the tangent of the path, hence the integral in (1.41) reduces to accumulating the gradients over the length of the path. The solution can efficiently be computed in linear time with raster-scan (Toivanen, 1996) or fast marching methods (Yatziv et al., 2006). Without the gradient in (1.41), the geodesic distance equals the Euclidean distance and $C_{p,q}$ is a straight line between points p and q. When applied to interactive image segmentation, the seed pixel with the smallest geodesic distance is calculated for every unseeded pixel.

The image gradient in (1.41) can be replaced by more descriptive weight functions: In (Protiere and Sapiro, 2007), the authors design a weight function based on image intensity values and texture descriptions obtained with Gabor filters. This approach is extended in (Bai and Sapiro, 2007), where the weight function is calculated based on color distributions modeled with kernel density estimation. Furthermore, the authors perform border matting to obtain a soft segmentation and demonstrate the applicability of their approach to video segmentation. These approaches are similar in behavior to the Random Walker approach, and suffer from the same drawbacks, like e.g. that the

topology is fixed with positioning the seeds, that the segmentation is highly dependent on the seed positions and also the lack of a geometric meaning of the minimization process.

In (Criminisi et al., 2008, 2010), the authors create a set of candidate segmentations using a geodesic filter and find the optimal candidate within a Graph Cut framework. Their framework not only allows for topological changes between the seed positions and the final segmentation, the employed Graph Cut framework also introduces spatial coherence of the resulting segments. In (Price et al., 2010), the geodesic distance based on color models is included directly into the unary term of a Graph Cut segmentation energy.

1.4.3.10 Power Watersheds

Based on the work of Sinop and Grady (2007), where a relation between the Random Walker algorithm and Graph Cut segmentation is established, the Power Watershed algorithm of Couprie et al. (2009, 2010) incorporates Watersheds, Random Walker, Graph Cut and Geodesic Segmentation into a common framework. They state the two-label problem on a graph with edges $e_{ij} \in E$ and vertices $v_i \in V$ as

$$\min_u \sum_{e_{ij} \in E} w_{ij}^p |u_i - u_j|^q + \sum_{v_i} w_{F_i}^p u_i^q + \sum_{v_i} w_{B_i}^p |u_i - 1|^q, \tag{1.42}$$

with w_{F_i} and w_{B_i} denoting the unary potentials of the foreground and background pixels respectively. Couprie et al. show how different choices of p and q lead to different interactive segmentation algorithms (cf. Table 1.1): For zero weight power ($p = 0$), the solution of the interactive segmentation is independent of the image data and leads to Voronoi diagrams with the seed pixels as center nodes. For a finite weight power p, (1.42) corresponds to Graph Cut for $q = 1$ and the Random Walker algorithm for $q = 2$ (Sinop and Grady, 2007).

	$p = 0$	$p = finite$	$p = \infty$
$q = 1$		Graph Cut	Watershed
$q = 2$	Voronoi, L2 norm	Random Walker	Power Watersheds
$q = \infty$	Voronoi, L1 norm	Voronoi, L1 norm	Geodesic Segmentation

Table 1.1: Couprie et al. (2009, 2010) unify Watershed, Random Walker, Graph Cut and Geodesic segmentation in a common energy minimization framework based on two parameters p and q.

The work of Couprie et al. studies the case where the weight power p goes to infinity: In (Allène et al., 2010), the authors show that for $q = 1$ and $p \to \infty$, the minimizer of (1.42) can be computed by a fast maximum spanning forest algorithm instead of computationally costly max-flow methods. In this case, the Graph Cut solution corresponds to the solution of a Watershed segmentation. Couprie et al. prove that this also holds for any finite value of q, and present an minimization algorithm with a quasi-linear best-case complexity. For the case $q = 2$, their algorithm is able to find a unique global optimum of (1.42). This new family of segmentation algorithms, which Couprie et al. call Power Watersheds, allows to incorporate unary terms and spatial regularization into traditional watershed segmentation. An advantage of Power Watersheds over standard Graph Cut segmentation is, that the algorithm is able to compute multi-label solutions. A major drawback of this algorithm is, that the topology of the solution is fixed by the position of the seed pixels.

In their papers, Couprie et al. show comparable or better results than state-of-art-algorithms such as Graph Cut, Random Walker and Geodesic Segmentation on the GrabCut database at a lower runtime. They incorporate color information by employing the weight function

$$w_{ij} = \sqrt{\max\left((R_i - R_j)^2, (G_i - G_j)^2, (B_i - B_j)^2\right)}, \tag{1.43}$$

where R_i, G_i, B_i denote pixel values in the RGB color space.

1.4.3.11 Summary

Based on the vast amount of literature dealing with interactive segmentation, we now want to provide a brief overview describing the key properties and employed energy functionals of the most important methods. The following table states a set of properties of interactive segmentation algorithms:

Intensity (I) / Color (C) / Texture (T) Indicates whether an algorithm segments images based on intensity only, or employs higher dimensional features such as color or local structure descriptions. Note that these properties are stated according to the respective publication, and thus indicate the capabilities of the published version and not the theoretical capabilities of the algorithm.

Region Scribbles (RS) Indicates the type of user interaction an algorithm needs. If ticked, the algorithm is seeded by drawing scribbles directly into the regions. If not

ticked, the user has to e.g. loosely draw the contour of the object to be segmented.

Global Optimum (GO) Indicates, whether the algorithm is able to compute a globally optimal solution based on the input image and the seeds specified.

Free Topology (FT) States, whether the topology of the solution is necessarily identical with the topology specified by the seed pixels or not.

Multi-Label (ML) Indicates the capability of a segmentation method to partition the image into several (more than two) regions at the same time.

Method / Author	Description	Data			Properties			
		I	C	T	RS	GO	FT	ML
Intelligent Scissors / Livewire (Mortensen and Barrett, 1995)	Estimates the optimal path between two points p, q by minimizing the cost $$d_{LIVEWIRE}(p,q) = \lambda_Z d_Z(q) + \lambda_G d_G(q) + \lambda_D d_D(p,q)$$ using optimal graph search algorithms. The cost functions d_Z and d_G pull the path towards high gradients, d_D penalizes sharp directional changes.	√	×	×	×	√	×	×
Seeded Region Growing (Adams and Bischof, 1994)	Grows segments from seed points based on the similarity $$\delta(p,E) = \left\lvert I(p) - \frac{1}{\lvert E\rvert}\sum_{q\in E} I(q)\right\rvert$$ between a pixel p and an adjacent region A.	√	×	×	√	√	×	√
Snakes / Active Contours (Kass et al., 1988)	Snakes are splines driven by an energy of the form $$E_{SNAKE} = \lambda_1 \int_0^1 \lvert C'(q)\rvert^2 dq + \lambda_2 \int_0^1 \lvert C''(q)\rvert^2 dq - \lambda_3 \int_0^1 \lvert\nabla I(C(q))\rvert dq.$$ While the first and second derivatives of the contour C keep the contour short and smooth, the third term pulls it towards large gradients.	√	×	×	×	×	×	×
Geodesic Active Contours (Caselles et al., 1997)	Improvement of the active contours model with the energy term $$E_{GAC} = \int_0^{L(C)} g(\lvert\nabla I(C(q))\rvert) ds,$$ where $L(C)$ denotes the length of the contour, $g()$ is an edge detection function and ds is the Euclidean element of length.	√	×	×	×	×	√	×

Method / Author	Description	Data			Properties			
		I	C	T	RS	GO	FT	ML
Interactive Graph Cuts (Boykov and Jolly, 2001)	Similar to the Geodesic Active Contour model, where the energy $$E(u) = \lambda \cdot \sum_{p \in \mathcal{P}} R_p(u_p) + \sum_{\{p,q\} \in \mathcal{N}} [u_p \neq u_q] \cdot g(p,q)$$ is solved globally optimal on a graph using max-flow algorithms. The unary term $R_p(u_p)$ represents consistency with an intensity model and the pairwise term $[u_p \neq u_q] \cdot g(p,q)$ pulls the region border towards edges.	√	×	×	√	√	√	×
GrabCut (Rother et al., 2004)	Performs Graph Cut optimizations and foreground model updates iteratively until convergence. The minimized energy includes a color model as unary term: $$E(u) = -\sum_{p} log(h(z_p; u_p)) + \lambda \sum_{(p,q) \in \mathcal{N}} d(p,q)^{-1} [u_p \neq u_q] g(z_p, z_q).$$	√	√	×	√	×	√	×
Random Walks (Grady and Funka-Lea, 2004)	A graph-based algorithm minimizing the energy $$E(u) = \int_{\Omega} ||\mathbf{W}^{\frac{1}{2}} \nabla u||^2 d\Omega,$$ with the inhomogeneous metric $\mathbf{W}$ encoding intensity changes. The energy can be minimized optimally using iterative solvers for sparse linear systems.	√	×	×	√	√	×	√
Lazy Snapping (Li et al., 2004)	Employs the algorithm of Boykov and Jolly (2001) on a graph based on superpixels obtained with a Watershed segmentation.	√	√	×	√	√	√	×
Random Walks (Grady, 2005)	A modification of (Grady and Funka-Lea, 2004) able to handle topology changes.	√	×	×	√	√	√	√

Method / Author	Description	Data			Properties			
		I	C	T	RS	GO	FT	ML
Adaptive Weighted Distances (Protiere and Sapiro, 2007)	Finds the minimal geodesic distance $$d(p,q) = \min_{C_{p,q}} \int_0^1 \lvert W_i \cdot \dot{C}_{p,q}(s) \rvert ds$$ between an unseeded pixel p and a seeded pixel q over a path $C_{p,q}$, where W_i encodes color and texture information gained from Gabor filter responses.	√	√	√	√	√	×	√
Geodesic Segmentation (Bai and Sapiro, 2007)	Closely related to (Protiere and Sapiro, 2007), with color signatures modeled using kernel density estimation.	√	√	×	√	√	×	√
Weighted Total Variation (Bresson et al., 2007)	Solves the variational problem $$E = \int_\Omega g(x) \lvert \nabla u \rvert d\Omega + \lambda \int_\Omega \lvert u - f \rvert d\Omega,$$ where the first term is equivalent to the Geodesic Active Contour Energy and the second term represents data-fidelity w.r.t. to the image f.	√	×	×	×	√	√	×
GeoS (Criminisi et al., 2008)	Extension of (Bai and Sapiro, 2007), where a Graph Cut framework is employed to select the best segmentation within a set of candidates obtained with a geodesic filter.	√	√	×	√	√	√	√
TVSeg (Unger et al., 2008)	Extends the energy of (Bresson et al., 2007) to $$E = \int_\Omega g(x) \lvert \nabla u \rvert d\Omega + \int_\Omega \lambda(x) \lvert u - f \rvert d\Omega,$$ to incorporate per-pixel user constraints via $\lambda(x)$ and incorporates color information with histograms in the data fidelity term f.	√	√	×	√	√	√	×
Star Shape Prior (Veksler, 2008)	Extension of (Boykov and Jolly, 2001) including a star-shaped prior to reduce the shrinking bias and the number of necessary seed pixels.	√	√	×	√	√	×	×

Method / Author	Description	Data			Properties			
		I	C	T	RS	GO	FT	ML
Paint Selection (Liu et al., 2009)	A method based on (Boykov and Jolly, 2001), where speed and usability are highly improved using heuristic mechanisms such as viewport-based local optimization or seed competition.	√	√	×	√	×	√	×
Power Watersheds (Couprie et al., 2010)	Unifies Graph Cuts, Random Walks, Watersheds and Geodesic Segmentation with the energy $$E(u) = \sum_{e_{ij} \in E} w_{ij}^p |x_i - x_j|^q + \sum_{v_i} w_{F_i}^p u_i^q + \sum_{v_i} w_{B_i}^p |u_i - 1|^q,$$ with p and q specifying the segmentation algorithm. Power Watersheds ($q \geq 2$ and $p \to \infty$) extend traditional Watershed segmentation by unary potentials and spatial regularization.	√	√	×	√	√	×	√
Geodesic Star Convexity (Gulshan et al., 2010)	Extension of (Veksler, 2008) including multiple stars and geodesic instead of Euclidean paths.	√	√	×	√	√	×	×
Geodesic Graph Cut (Price et al., 2010)	Adds geodesic distances into the unary term of the Graph Cut segmentation energy, and adjusts the combination weights automatically according to the quality of the geodesic segmentation.	√	√	×	√	√	√	×
Diffusion (Zhang et al., 2010)	Extends (Grady and Funka-Lea, 2004) by anisotropic diffusion $$E(u) = \int_\Omega ||\mathbf{T} \nabla u||^2 d\Omega,$$ where $\mathbf{T}$ is a tensor which adds anisotropic diffusion $\mathbf{D}$ to the metric $\mathbf{W}$.	√	√	×	√	√	×	√

Table 1.2: Comparison of selected interactive segmentation algorithms based on energy functionals, methodology and several key properties.

1.5 Parallel Computing

As stated in the previous sections, a key requirement for interactive image segmentation algorithms is their computation time. For applications where computation time is an issue, the recent development of computing hardware raised the need for algorithms that can be computed in multiple threads in parallel. Hence, in our framework, we have to take care that employed algorithms are parallelizable and can be run on several central processing unit (CPU) cores or even on graphics processors. Without these multi-core implementations, the processing time would be too large to allow for convenient interaction. Throughout this thesis, we describe the parallelization strategy for every multi-core part of our framework and compare runtimes to single core implementations. For these comparisons, we use a desktop computer featuring an Intel Q9450 Core 2 Quad CPU running at 2.66 GHz with 3 GB of RAM and a NVIDIA Geforce 280 GTX graphics card. In this section, we first explain the need for parallelizable algorithms and show how multi-core CPU as well as GPU techniques can help in significantly reducing computation time.

1.5.1 CPU - Central Processing Units

From the early microprocessors such as Intel's 4004 (November 1971) or 8086 (June 1987), the increase in computational power came from increasing the number of transistors on the chips (Moore, 1998) as well as increasing the chip's clock speed. This was made possible by continuously decreasing the structure width on the chips: In 1989, Intel introduced their famous 80486 CPU (central processing unit) with 1.2 million transistors in a $0.8\mu m$ process which ran at a clock speed of 50 MHz. Ten years later, in 1999, they launched the Pentium III (codename Coppermine) featuring 28.1 million transistors manufactured at $0.18\mu m$ operated at up to 1133 MHz. The increasing clock speeds boosted the performance of every computer program with each new processor generation. However, the clock speed increase stopped in 2004 with Intel's Pentium D (codename Presler) at 3.8 GHz due to thermal limitations: Increasing the clock speed of an integrated circuit (IC) requires an increase of its operating voltage, which in turn increases the power consumption of the circuit. A high power consumption of an IC leads to more sophisticated and expensive cooling systems needed to dissipate the heat produced during operation. Therefore, the leading chip manufacturers combined several independent CPUs in one package with decreased clock rates: In 2006, Intel launched their Core 2 CPU family featuring two (codename Conroe, 65nm, up to 3 GHz) and

four (codename Kentsfield, 65nm, up to 2.67 GHz) independent processor cores. The currently most powerful consumer CPU is the Intel Core i7-980X (codename Gulftown, 32nm, 3.33 GHz), which features six independent processing cores able to process 12 independent program threads concurrently.

This development leads to the following observation: A computer program will not get faster with new processor generations unless it is able to use several independent cores simultaneously. When a program exhibits this property, it is called parallelizable. In speed-critical applications, future-proof programs nowadays need their key parts to be parallelizable.

Multi threaded programming on the CPU Compared to GPUs, the development of a program for a multi-core CPU environment is relatively straightforward: A computer program is represented as a process, working through all instructions of the program until it is finished. When time-consuming parts of the program are independent from each other, a process can calculate these parts in parallel by starting several program threads, which operate independently from each other on different CPU cores. When each of the threads is finished, the main process collects their results and continues its operation. Due to its importance, threading is a built-in feature of nearly every important programming language.

The speedup reached with threading depends on the number of cores of the CPU, which gives an upper bound for the decrease in computation time: When running a program multi-threaded on a four-core processor, the execution is typically faster than a single-threaded implementation by a factor of 3 to 3.5.

1.5.2 GPU - Graphics Processing Units

Driven by the computer games industry, graphics processing units (GPUs) can nowadays be found in desktop computers, workstations, notebooks and even mobile phones. They have emerged from simple graphics controllers needed to display geometric primitives such as lines, rectangles, circles and arcs onto a screen to highly sophisticated 3D accelerators able to render three-dimensional worlds with complex illumination at high frame rates. Early GPUs featured a fixed-function pipeline, with specific hardware for every rendering operation needed during the displaying process: E.g. for transforming geometric primitives, every GPU featured several vertex processors, for computing the final color for a pixel there were so-called pixel shaders. The theoretical computational performance of GPUs boosted, as the number of parallel rendering pipelines in-

creased: While the NVIDIA Geforce2 GTS introduced in 2000 had only 2 pipelines, the Geforce 6800 GT of 2004 featured 16 and the Geforce 7800 GTX of 2005 even 24 pipelines. As these pipelines were specifically designed for rendering three-dimensional scenery, graphics processors were able to perform an immense amount of basic calculations such as vector products or interpolations at high speed. As can be seen in Figure 1.19, the theoretical peak performance of recent GPUs is an order of magnitude greater than the theoretical peak performance of CPUs.

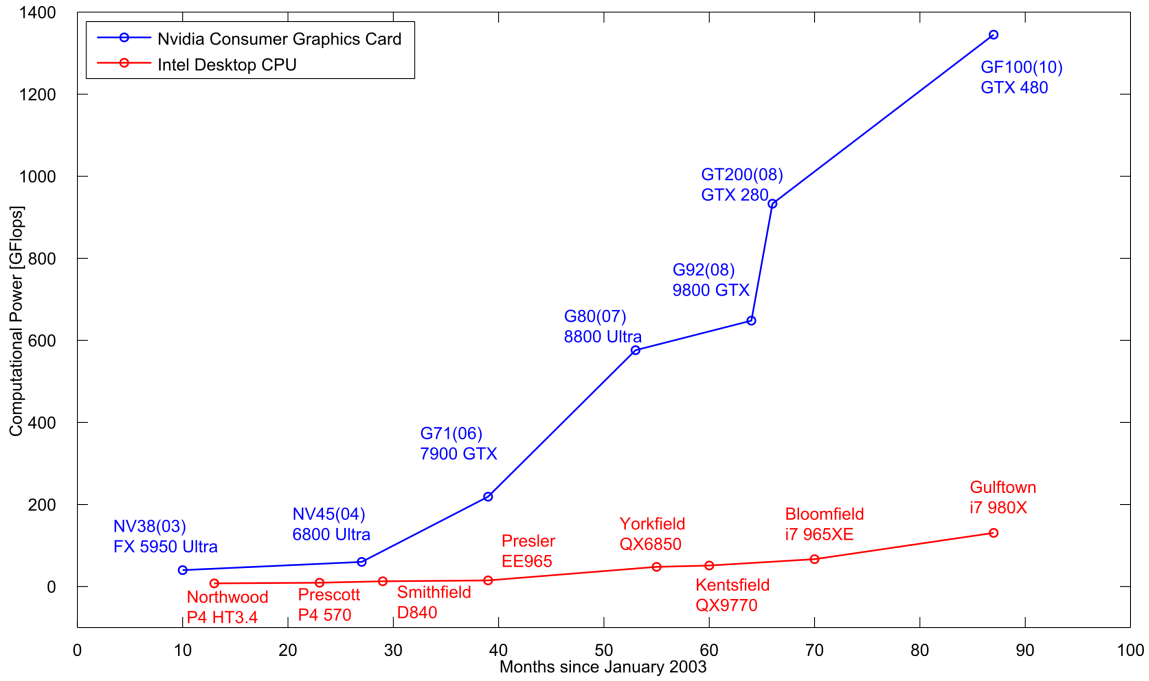


Figure 1.19: A comparison of the theoretical peak performance of recent Intel desktop processors and NVIDIA graphics processors in GFLOPS (billions of floating point operations per second). Note that most of the data is taken from NVIDIA and Intel product documentations and not from real world benchmarks.

Taking their special hardware design into account, graphics processors were interfaced with specific shader programming languages such as GLSL (OpenGL shading language), HLSL (DirectX High-Level Shader Language) or Cg (C for Graphics). As of the vast computational power, graphics processors became more and more useful to the scientific community in spite of the complex programming model induced by shader languages. Therefore, instead of rendering polygons, the GPUs were used as a high-performance mathematical co-processor. Some works in the field of computer vision using GPUs interfaced with shader languages include (Griesser et al., 2005; Labatut

et al., 2006; Sharp, 2008; Yang and Pollefeys, 2003; Zach et al., 2007).

In 2006, NVIDIA introduced the Unified Shader architecture with their new G80 chip featured in their Geforce 8800 GTX graphics card. In contrast to the Geforce 7800 GTX, which featured 8 vertex shaders and 24 pixel shaders, the Geforce 8800 GTX came with 128 unified shaders. Depending on the application, these 128 cores could act either as a vertex shader or as a pixel shader. While traditional shading languages were still supported, NVIDIA presented a new programming interface called CUDA (Compute Unified Device Architecture). CUDA allows developers to execute standard C code on the GPU by providing a set of instructions for e.g. copying data between system memory and video memory or setting up the graphics card. With the G80 architecture, GPGPU (General-Purpose Computing on Graphics Processing Units) became a widely spread technology in many scientific and commercial areas, including not only image and video processing but also machine intelligence, energy, finance as well as medical applications.

Programming on the GPU When using CUDA to speedup the computation of a specific algorithm, there are two major design decisions to be made:

First, one has to decide how to divide the problem into lots of equal, parallelizable parts: Therefore, the CUDA programming model allows the user to define so-called kernels, which are small programs executed in parallel on the GPU, operating on different data. Starting with the G80 architecture, NVIDIA GPUs consist of several so-called multiprocessors featuring eight cores each, e.g. a Geforce GTX280 consists of 30 multiprocessors which makes a total of 240 cores. The threads are grouped together into blocks, which are distributed to the multiprocessors, such that every multiprocessor is assigned several blocks. The number of threads in a block should be at least 128 (i.e. four times the number of threads which are executed simultaneously on one multiprocessor (a.k.a warp size)). For each kernel, the user has to specify the size of the block as well as the number of blocks to be computed. The number of blocks should exceed the number of multiprocessors (ideally be a multiple of the number of multiprocessors), otherwise parts of the GPU run idle. For the Geforce GTX280 and a block size of 128, the number of threads should either be a multiple of 3840 or much larger than 3840 to not waste computational performance on idle GPU cores.

Second, one has to choose the employed memory type carefully: A recent GPU features up to 4 GiB of global memory, where each thread can perform unsynchronized read/write operations, which are relatively slow compared to other memory types. The

G80 architecture also features 16 KB of so-called shared memory per multiprocessor, which allows synchronized read/write operations for all threads executed on a multiprocessor. While the global memory typically consists of discrete memory chips located on the graphics card, the shared memory resides directly on the chip and is therefore up to two orders of magnitude faster than global memory. As a third important possibility, for read-only access, CUDA offers the possibility to perform cached texture fetches from global memory, with interpolation routines implemented in hardware.

However, having chosen a suitable division into threads and the right type of memory for the problem does not necessarily guarantee a good performance on the GPU. There are also several important architectural limitations, which can lead to low performance or even impede algorithms from being ported to the GPU:

- Every multiprocessor features only one single instruction unit, thus all threads running on the same multiprocessor execute the same instruction (SIMD). In some cases (e.g. code branches or wrong shared memory access patterns), the threads need to run different instructions and are therefore serialized, which leads to significant performance drops.

- The amount of shared memory, registers, texture cache etc. is very limited, hence, as computation kernels get complex, they quickly run into resource problems. Splitting up large kernels into several smaller kernels is often hardly possible and can also lead to performance issues.

- There is typically no stack in GPU architectures, hence recursive algorithms cannot be implemented.

- The support for double precision floating point arithmetic is currently very limited.

- There is no dynamic memory allocation at kernel level, thus the amount of memory a kernel needs has to be known before its invocation.

Because of these and other limitations of GPU architectures, specific algorithms might not be suitable for GPU implementations. However, if an algorithm maps well to the GPU programming model, it is typically substantially faster than multi core CPU implementations.

1.6 Proposed Framework

In the previous sections, we gave a detailed insight to state-of-the-art interactive image segmentation algorithms and their shortcomings. Based on these considerations, we develop a powerful interactive segmentation framework with the following properties:

Multi-label Capability Many of the interactive methods described in the previous section can only tackle two-label problems, i.e. the segmentation of an image into foreground and background. When a user wants to select more than two distinctive regions in an image, he has to use a two-label tool several times. Our method is capable of partitioning the image into more than two regions simultaneously.

Region Scribbles The user interaction is performed by drawing scribbles into the region to be segmented rather than loosely drawing its boundary. Many segmentation tasks can satisfactorily be performed with only a few scribbles.

Free Topology The topology of the final segmentation is independent from the topology of the seed points.

Arbitrary Features In contrast to most segmentation algorithms which operate only in the color or intensity feature space, our framework is able to use arbitrary high-dimensional local image features, which allows e.g. to capture complex textural properties of the image regions.

High-Speed While capturing textural properties usually comes with the computationally costly analysis of high-dimensional feature spaces, our method is still fast enough for convenient user interaction by massively using GPU-implementations.

1.6.1 Contributions of this Thesis

The main contributions of this thesis are based on two papers on interactive segmentation.

In our initial work (Santner et al., 2009), we extended a weighted Total Variation framework to texture segmentation. Instead of employing intensities or modeling color with histograms, we learned higher-dimensional local structure descriptors such as the Histogram of oriented Gradients descriptor (Dalal and Triggs, 2005) with Random Forests (Breiman, 2001) to capture textural properties. Most of the parts were implemented on graphics processors for convenient interaction speed.

In our second work on segmentation (Santner et al., 2010), we extended the framework to be able to segment multiple labels ($k \geq 2$). We also extended the employed feature space with HSV and CIELAB color models as well as new features capturing local structure. We furthermore addressed the problem of assessing the quality of an interactive segmentation approach and presented a novel benchmark dataset for interactive segmentation algorithms. With this benchmark dataset, we measured the segmentation performance of the building blocks of our framework, i.e. different color and texture descriptions, multi-label learning algorithms as well as segmentation model parameterizations.

1.6.2 Organization

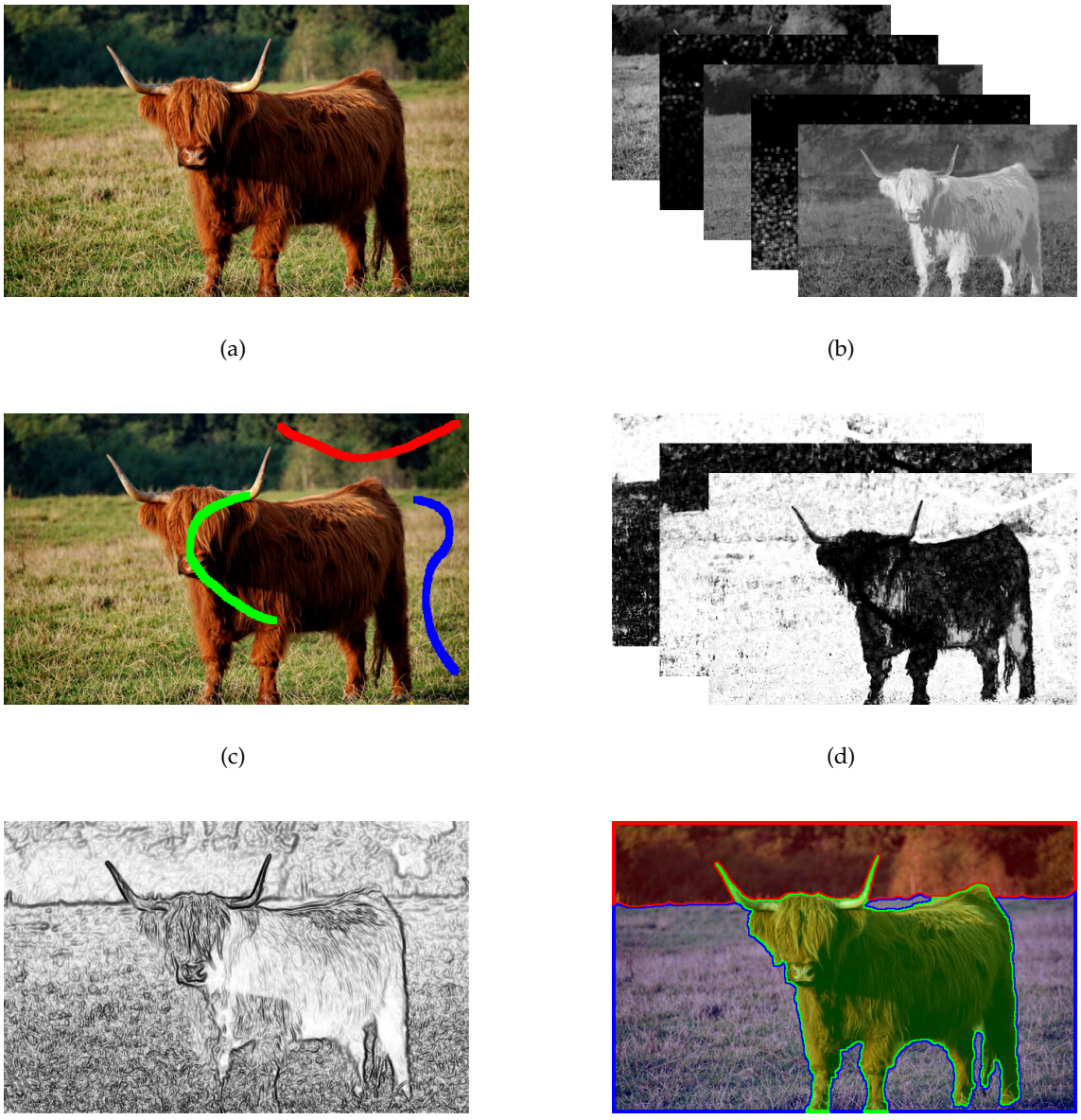
The remainder of this thesis is organized in the order of the work flow of our framework (see also Figure 1.20):

Given an image to be segmented, the first step involves the calculation of multiple feature representations. This not only includes several color models, but also features representing local structure. In Chapter 2, we describe several image features and investigate their applicability for interactive segmentation especially in terms of memory consumption and computation time.

In the second step, we generate models for every image region based on input provided by the user. These models are the product of training a supervised learning algorithm with the user-marked pixels as training set. Based on these models, the likelihood that a pixel belongs to a certain region is computed for every image pixel. In Chapter 3, we describe how such descriptive region models can be constructed efficiently based on the high-dimensional features obtained from the feature stage.

Finally, these pixel likelihoods are regularized with a continuous Total Variation energy functional for multiple labels (Pock et al., 2009). The development of this energy functional starting from the two-label weighted Total Variation formulation (Bresson et al., 2007) as well as discrete counterparts are presented in Chapter 4.

The remainder of this thesis includes the generation of a benchmark dataset for multi-label interactive image segmentation (Chapter 5) as well as extensive experiments evaluating the performance of all building blocks of our framework (Chapter 6). We finally give a conclusion and an outlook to future work in Chapter 7.



(a) (b)

(c) (d)

(e) (f)

Figure 1.20: Workflow of our interactive segmentation framework: Given an image (a), the color models as well as local features are computed (b). Then, upon user input (c), these features are employed to train a model for each region. This model is evaluated for every pixel, yielding the probability that a pixel belongs to a certain label. (d) shows the probabilities that a pixel belongs to a certain region, where dark means a high likelihood. Finally these likelihoods are regularized together with the gradient magnitude (e) in order to produce a final segmentation result (f). When seed pixels are added or removed, only the steps (d) and (f) need to be computed.

2 Image Features

In this chapter, we investigate the applicability of several image features for interactive segmentation. An image feature is the mapping of image values to a given feature space, which allows to capture certain properties of an image. By mapping e.g. a color image from the RGB feature space to the grayscale feature space, one obtains a good interpretation for the image intensity. Besides color models, where the value of one pixel in the target feature space depends only on the value of the same pixel in the original feature space, also image representations exist which allow to capture local structure. These features take into account the pixels in a given neighborhood of the original pixel, making them able to represent textural properties of the pixel neighborhood.

To be suitable for our interactive segmentation framework, an image feature needs to be very fast to compute. In this chapter, we therefore describe several color models and texture features and evaluate their runtime both on CPU and GPU.

2.1 Color Models

In this section, we will briefly describe the color models used in our segmentation framework. An extensive overview on color theory is given in (Gonzalez and Woods, 2001). The elementary feature spaces for our framework are the RGB color space and the grayscale range. In the three-dimensional RGB feature space, the color of a pixel is encoded by three scalar values representing the primary spectral components red, green

and blue. We denote the color of a pixel P in the RGB space as

$$P_{RGB} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}, \tag{2.1}$$

where the components R, G and B vary continuously between $[0, 1]$. Conversion between the RGB feature space and the grayscale range is usually performed by a linear combination of the red, green and blue value:

$$P_{grayscale} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B. \tag{2.2}$$

The weights for the spectral components are stated in the NTSC (National Television Systems Committee) norm (Smith, 1978), and are identical to the weights used to compute the luminance value in the YUV color space.

(a) (b)

(c) (d) (e)

Figure 2.1: The RGB color space: (a) shows a color image which has been converted to grayscale (b). Images (c),(d) and (e) show the red, green and blue channels of (a) separately. Note the high correlation between these color channels.

While the RGB feature space is well suited for displaying purposes, the high correlation between the three channels makes it a bad choice for color image segmentation (Cheng et al., 2001; Pietikainen et al., 1996). Color models based on linearly transforming the RGB color space such as e.g. YUV or YIQ, do not solve this problem. However, there also exist color models based on non-linear transformations such as the HSV or CIELAB model, which are a better choice for general image analysis problems.

2.1.1 HSV

The HSV (hue-saturation-value) color model separates intensity information from color information. A pixel in the HSV color space has three components (see also Figure 2.2): The H component denotes the hue of the pixel as angle on a color wheel. The primary colors are distributed over the color wheel in equal angular distance, such that the red primary is located at 0°, the green primary at 120° and the blue primary at 240°. The center of the color wheel is the white color. The S component encodes the saturation of the pixel i.e. the distance from the center of the color wheel. The V component represents the intensity of the pixel. The transformation between the RGB color model and the HSV color model is described in detail in (Smith, 1978).

(a) (b) (c)

Figure 2.2: Converting the cow from Figure 2.1 from the RGB color space to the HSV color space: (a) shows the hue, (b) the saturation and (c) the value component of the original image.

2.1.2 CIELAB

Another color space encoding intensity information independently from color information is the CIELAB (a.k.a. CIE L*a*b*) space (Robertson, 1977). An important property of this color space is the perceptual uniformity: The Euclidean distance between two

color values approximates the perceptual difference between the colors better than in other color spaces.

To calculate the CIELAB coefficients from and RGB triplet, first the tristimulus values XYZ need to be computed. This can be performed as a linear transformation with a specific transformation matrix, e.g. as stated in (Cheng et al., 2001):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.3}$$

With X_W, Y_W and Z_W describing the tristimulus values of a reference white, the CIELAB color components are derived with the following equations

$$P_{CIELAB} = \begin{pmatrix} L \\ a \\ b \end{pmatrix} = \begin{pmatrix} 116 \cdot h(\frac{Y}{Y_W}) - 16 \\ 500 \cdot \left[h(\frac{X}{X_W}) - h(\frac{Y}{Y_W}) \right] \\ 200 \cdot \left[h(\frac{Y}{Y_W}) - h(\frac{Z}{Z_W}) \right] \end{pmatrix} \tag{2.4}$$

where

$$h(x) = \begin{cases} \sqrt[3]{(x)} & \text{if } x > 0.008856 \\ 7.787 \cdot x + \frac{16}{116} & \text{else} \end{cases} \tag{2.5}$$

2.1.3 Runtime

We implemented the transformations between these color models entirely on the GPU. As the target value(s) of every pixel depends solely on the original RGB triplet, the parallelization is straightforward: Every GPU thread is responsible for the transformation of one single pixel. The measured runtimes are given in Table 2.1: As the conversions between RGB and grayscale as well as RGB and HSV are arithmetically very simple, the GPU implementation is not significantly faster than the CPU implementation. The arithmetically more complex conversion to CIELAB allows the GPU to outperform the CPU by a factor of more than 20 for images of the size 512×512.

Furthermore, the GPU implementation scales better to increasing image sizes than the CPU implementation: Between the size of the test images (256×256, 512×512 and 1024×1024) lies a four-fold increase of the number of pixels. While the runtime of the CPU implementation approximately reflects this four-fold increase, the increase of GPU computation time is less progressive. One reason for this behavior is, that the times reported for the GPU implementations include copying the images to and from the GPU

Operation	Image Size	CPU [ms]	GPU [ms]	Speedup
RGB -> grayscale	256×256	0.55	1.47	0.37
	512×512	2.32	2.70	0.86
	1024×1024	8.35	7.19	1.16
RGB -> HSV	256×256	2.68	2.30	1.17
	512×512	10.69	6.11	1.75
	1024×1024	42.37	19.49	2.17
RGB -> CIELAB	256×256	31.89	2.46	12.96
	512×512	127.4	6.36	20.03
	1024×1024	517.0	22.57	22.91

Table 2.1: Runtime comparisons between single threaded CPU implementations and GPU implementations of color space conversion functions. Note that the GPU times include copying to and from GPU device memory.

memory. Another reason is the SIMD structure of the GPU (cf. Section 1.5.2): The larger the images are, the more pixels have to be computed, hence the more blocks can be evaluated simultaneously on the GPU. This not only reduces the relative time wasted on kernel launch overhead, but also decreases the probability that multiprocessors run idle during execution of a kernel.

2.2 Texture Description

Besides color, the local structure of an image is often a very important cue for visual grouping and scene understanding. In segmentation, there exist many images where the usage of color cues alone is not sufficiently descriptive for an accurate segmentation. See the images shown in Figure 2.3 for an example: In both images, a segmentation of the foreground object is very difficult based on color cues solely. Therefore, interactive segmentation methods exist which additionally include information deducted from shape priors (Gulshan et al., 2010; Veksler, 2008; Werlberger et al., 2009) and texture descriptors (Han et al., 2009; Protiere and Sapiro, 2007; Santner et al., 2009).

In this section, we describe several ways to incorporate local structure information into our segmentation framework. For describing local structure of a pixel, the aim is to capture characteristic information from a given surrounding of the image pixel. There is a vast amount of such texture description / classification methods, which are in general computationally more intensive than the color space transformations presented earlier in this chapter. In the following, we describe several such methods and measure their runtime in order to evaluate their applicability for our framework.

(a) (b)

Figure 2.3: The segmentation of the foreground objects in (a) and (b) based on estimating color signatures is hardly possible. To accurately segment these objects, one has to include additional cues such as shape priors or local structure.

2.2.1 Image Patches

As elementary feature describing local structure, we employ square patches cropped from the original grayscale image (Varma and Zisserman, 2009). This feature exhibits no common texture descriptor properties like illumination, rotation or scale invariance, but still is the easiest way to represent local structure. The only free parameter of this feature representation is the size s of the square environment, which directly defines the dimension of the feature space (s^2). As the patches are constructed by loading and storing grayscale values only, the runtime is negligible.

2.2.2 Haralick Features

A still very popular early approach to texture classification is the method of Haralick et al. (1973). The Haralick texture features are based on computing 14 statistical measures from a graylevel co-occurrence matrix derived from a given environment around a pixel. We implemented the creation of the graylevel co-occurrence matrix as well as the computation of the statistical measures on the GPU, where we took an approach similar to (Gipp et al., 2009). Note that the last statistical measure described in the original paper is often omitted due to its high computational complexity. We follow this principle, which leads to a 13-dimensional feature

$$P_{Haralick} = (H_1, H_2, H_3, \ldots, H_{13})\,. \tag{2.6}$$

2.2.2.1 Graylevel-Cooccurrence Matrix

A graylevel co-occurrence matrix (GLCM) represents the spatial dependence of intensity levels based on a grayscale image. For every pixel in the image, one $N \times N$ GLCM is computed, where N represents the number of possible grayvalues. The typical radiometric resolution of 8 bits per pixel would lead to a 256×256 GLCM, which in turn would be populated very sparsely. To reduce the memory consumption, the graylevel values are mapped to a smaller radiometric resolution such as e.g. 5 bits, which leads to a better populated GLCM with 32×32 elements.

The GLCM accumulates co-occurring intensity levels over a sampling environment around the center pixel: Given the center pixel has intensity a and a neighboring pixel has intensity b, then the matrix values at the positions (a, b) and (b, a) are incremented. After all neighboring pixel combinations within the sampling environment have been accumulated, the symmetric GLCM is normalized.

The neighborhood relation employed for the accumulation of the GLCM can differ between applications. Typically, four neighbors are taken into account for every pixel within the sampling environment (see Figure 2.4). To achieve rotation invariance, the four neighbors representing four different directions are accumulated in the same GLCM.

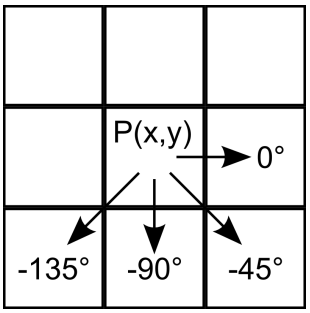


Figure 2.4: To achieve rotation invariance, four neighbors from four different directions are accumulated in the graylevel co-occurrence matrix.

An important issue for computing the GLCM is the memory consumption: To compute Haralick features densely over the image, a separate GLCM has to be computed for every pixel. After the statistics are computed, the GLCM can be discarded. This procedure maps well to the CPU, where the whole algorithm can be calculated sequentially. On the GPU, the naive approach to parallelize on pixel level (i.e. every thread computes the result for one single pixel) leads to memory issues: As all pixels are evaluated in par-

allel, one would need to store the GLCM for every pixel in the graphics card's memory. An example: For a 5-bit graylevel resolution ($N = 32$), an image with 640×480 pixels and a 4-bit floating point GLCM representation, the memory consumption for storing all GLCMs amounts to

$$2^5 \cdot 2^5 \cdot 640 \cdot 480 \cdot 4 = 1258291200 \text{ Byte} \approx 1.2\text{GB}.$$

To not run into memory problems, we apply a hybrid parallelization approach to compute the Haralick features: The image is partitioned into smaller chunks, which consist of several image lines. These chunks are computed sequentially, while all pixels within a chunk are computed in parallel. The size of the chunks is adapted, such that the amount of memory needed to store all of its GLCMs does not exceed 200 MB. After the computation of the statistical measures is finished, the GLCMs are discarded and the next chunk is processed.

2.2.2.2 Statistical Measures

From the normalized GLCM, 13 statistical measures are computed. Similar to Haralick et al. (1973), $p(i,j)$ denotes the GLCM entry at position (i,j). The marginal probabilities of the GLCM are given as

$$p_x(i) = \sum_{j=1}^{N} p(i,j) \tag{2.7}$$

and

$$p_y(j) = \sum_{i=1}^{N} p(i,j), \tag{2.8}$$

With μ_x, μ_y and σ_x, σ_y denoting their means and standard deviations. The sums of the minor diagonals are denoted as

$$p_{x+y}(k) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ i+j=k}}^{N} p(i,j), \qquad k \in \{2,3,\ldots,2\cdot N-1, 2\cdot N\} \tag{2.9}$$

and

$$p_{x-y}(k) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ |i-j|=k}}^{N} p(i,j), \qquad k \in \{0,1,\ldots,N-2,N-1\} \tag{2.10}$$

Based on these intermediate values, the Haralick textural measures are then defined as follows:

1. Angular Second Moment

$$H_1 = \sum_{i=1}^{N}\sum_{j=1}^{N} p(i,j)^2 \tag{2.11}$$

2. Contrast

$$H_2 = \sum_{n=0}^{N-1} n^2 \cdot \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{|i-j|=n} p(i,j) \tag{2.12}$$

3. Correlation

$$H_3 = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} i \cdot j \cdot p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{2.13}$$

4. Sum of Squares: Variance

$$H_4 = \sum_{i=1}^{N}\sum_{j=1}^{N} (i-\mu)^2 \cdot p(i,j) \tag{2.14}$$

5. Inverse Difference Moment

$$H_5 = \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{1}{1+(i-j)^2} \cdot p(i,j) \tag{2.15}$$

6. Sum Average

$$H_6 = \sum_{i=2}^{2N} i \cdot p_{x+y}(i) \tag{2.16}$$

7. Sum Variance

$$H_7 = \sum_{i=2}^{2N} (i - H_8)^2 \cdot p_{x+y}(i) \tag{2.17}$$

8. Sum Entropy

$$H_8 = -\sum_{i=2}^{2N} p_{x+y}(i) \cdot log(p_{x+y}(i)) \tag{2.18}$$

9. Entropy

$$H_9 = -\sum_{i=1}^{N}\sum_{j=1}^{N} p(i,j) \cdot log(p(i,j)) \tag{2.19}$$

10. Difference Variance

$$H_{10} = \sum_{i=0}^{N-1} (i - H_{11})^2 \cdot p_{x-y}(i) \tag{2.20}$$

11. Difference Entropy

$$H_{11} = -\sum_{i=0}^{N-1} p_{x-y}(i) \cdot log(p_{x-y}(i)) \tag{2.21}$$

Note that the logarithms are computed with an offset $log(p) = log(\epsilon + p)$ to avoid numerical problems. For the last two statistical measures, the entropies of the marginal probabilities p_x, p_y are denoted as HX and HY. With

$$HXY1 = -\sum_{i=1}^{N}\sum_{j=1}^{N} p(i,j) \cdot log(p_x(i) \cdot p_y(i)) \tag{2.22}$$

and

$$HXY2 = -\sum_{i=1}^{N}\sum_{j=1}^{N} p_x(i) \cdot p_y(i) \cdot log(p_x(i) \cdot p_y(i)), \tag{2.23}$$

the last two measures are defined as follows:

12. Information Measures of Correlation 1

$$H_{12} = \frac{H_9 - HXY1}{max(HX, HY)} \tag{2.24}$$

13. Information Measures of Correlation 2

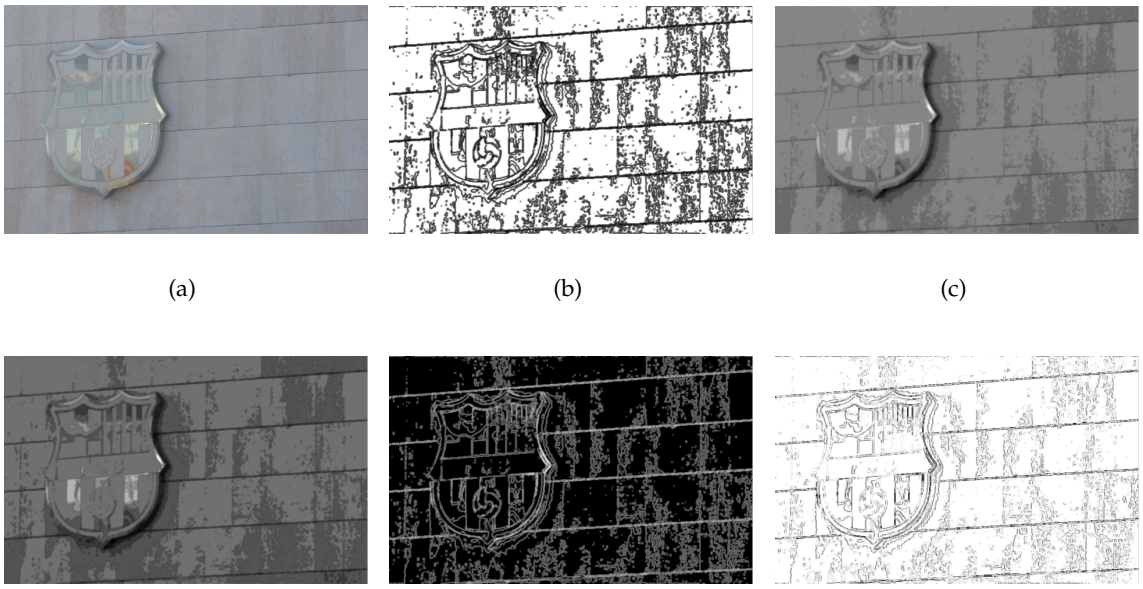
$$H_{13} = \sqrt{1 - e^{-2 \cdot (HXY2 - H_9)}} \tag{2.25}$$

Note that due to the symmetry of the GLCM, several simplifications can be performed: The equal marginal probabilities $p_x(i) = p_y(i)$ lead to $\mu_x = \mu_y$ and $\sigma_x = \sigma_y$ as well as $HX = HY$. A subset of the Haralick features is depicted in Figure 2.5.

Concerning the GPU implementation, we follow a parallelization strategy similar to the one presented in (Gipp et al., 2009). Taking into account the interdependence of the 13 features, we divide the computation into three different kernels: The first kernel computes the features which depend on the marginal probabilities, i.e. $H_1, H_2, H_3, H_4, H_5, H_9, H_{12}$ and H_{13}. The second kernel evaluates the features depending on $p_{x+y}(i)$ (H_6, H_7, H_8) and the last kernel the remaining two features H_{10} and H_{11}.

2.2.2.3 Runtime

Our implementation of the Haralick texture features offers two parameters: The number of graylevels N as well as the size of the quadratic sampling environment s employed for



(a) (b) (c)

(d) (e) (f)

Figure 2.5: Haralick texture features computed densely from image (a): H_1 is shown in (b), H_6 in (c), H_7 in (d), H_8 in (e), and H_{12} in (f).

accumulation of the GLCM. Figure 2.6 shows a runtime comparison between the CPU and GPU implementations for three different image sizes, $N \in \{8, 10, \ldots, 24\}$ and $s \in \{3, 5, \ldots, 13\}$. Using the GPU implementation, we are able to compute 13-dimensional Haralick features densely in less than two seconds for 256×256 and 512×512 images with any $N \leq 20$. For images of the size 1024×1024, computation times less than two seconds can be reached for $N < 12$ and $s < 7$. The GPU implementation yields speedup factors of 10 - 25 over the CPU version depending on the image size and parameters, however, the runtimes are still relatively high for an interactive application.

2.2.3 Filter Banks

A common approach to texture description is the analysis of image responses to a given set of linear filters. This usually consists of two steps: Given an image, first the response of the image to the filter bank is evaluated. This response then serves as starting point for further analysis, e.g. the calculation of specific energy measures or texton clustering (Leung and Malik, 2001; Shotton et al., 2006): Textons are cluster centers in the space of filter bank responses obtained from a given set of training images. These textons form a

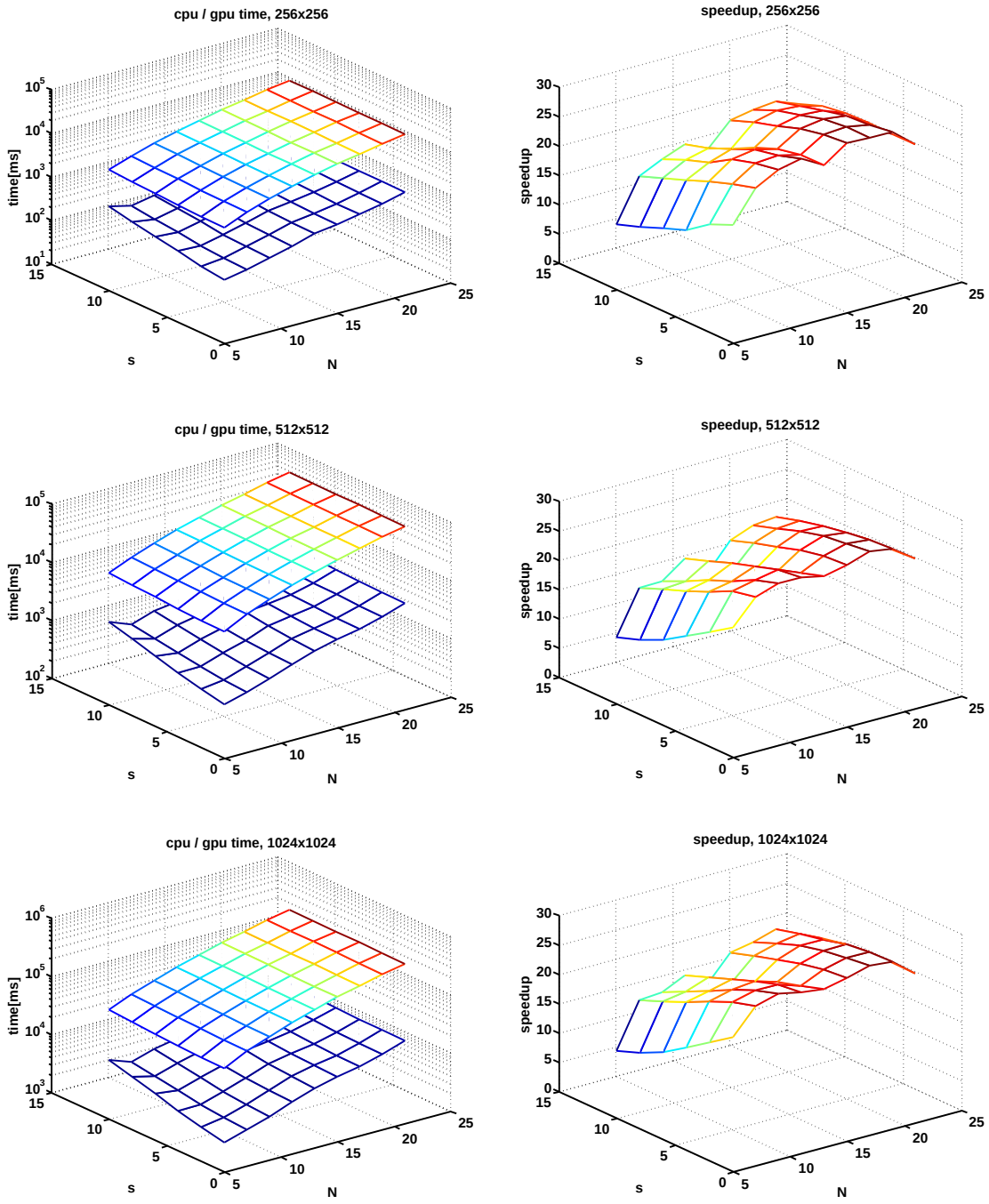


Figure 2.6: Runtime comparisons for Haralick texture features with three different image sizes (256×256, 512×512 and 1024×1024): The first column shows the runtime comparisons for different parameters N, s between the CPU (upper mesh) and GPU implementations (lower mesh), note the logarithmic scale. The second column depicts the speedup factor between CPU and GPU.

vocabulary which can be employed for classification of textures with properties similar to the textural properties of the training images.

In an early example, Laws (1980) employs five one-dimensional filters

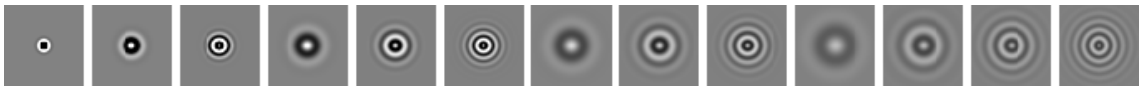
$$\begin{aligned} L5 &= \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \\ E5 &= \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix} \\ S5 &= \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix} \\ W5 &= \begin{bmatrix} -1 & 2 & 0 & -2 & 1 \end{bmatrix} \\ R5 &= \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \end{bmatrix} \end{aligned}$$

which are multiplied in order to create 25 filters of size 5×5. E.g., the filter $E5S5$ is created by the multiplication

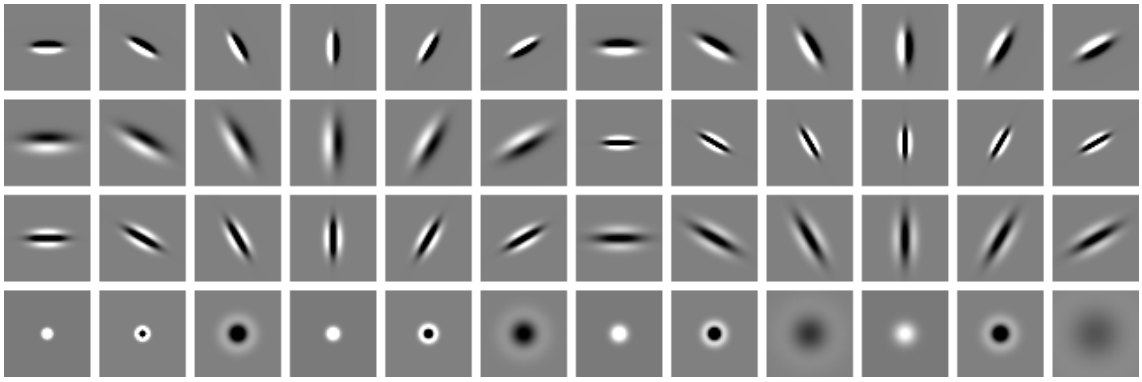
$$E5S5 = E5^T \cdot S5 = \begin{bmatrix} 1 & 0 & -2 & 0 & 1 \\ 2 & 0 & -4 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 0 & -2 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix}.$$

These filters have been designed heuristically to capture basic structural elements such as edges, spots or ripples. Based on the success of the Laws filter set, many filter banks able to model different textural phenomena have been presented, e.g. Gabor filter banks (Jain and Farrokhnia, 1990) or Quadrature Mirror Filters (Randen and John, 1994) (cf. also the the survey of Randen and Husoy (1999) on filter bank approaches).

Examples for more advanced filter banks are the Schmid13 set (Schmid, 2001) consisting of 13 rotationally invariant Gabor-like filters and the LM48 filter bank (Leung and Malik, 1999, 2001) featuring 48 filters with varying shape, orientation and scale (see Figure 2.7). Computing the image response to such a filter bank amounts to evaluating computationally expensive convolution operations. Table 2.2 gives a runtime comparison for the Schmid13 and LM48 filter banks: The GPU is able to outperform the CPU by factor of ≈ 50, however, it still needs more than two seconds for the smaller Schmid13 and eight seconds for the LM48 filter bank at an image size of 1024×1024. There exist ways to speed up convolution processes, e.g. by using separable filters such as the Laws filters or by performing convolution by multiplication in the Fourier domain, hence the



(a)



(b)

Figure 2.7: Two filter bank examples: The Schmid13 filter bank (a) consists of 13 rotationally invariant Gabor-like filters. The LM48 filter bank (b) combines 48 filters with varying shape, orientation and scale. The creation of these filter banks is based on source code of Varma and Zisserman (2002), note that the filters have been contrast enhanced for visualization.

response of certain filter banks may be computed faster than reported here.

However, there is typically additional computational effort needed for calculating energy measures or texton evaluation. Also, in the latter case, there is a training database needed for the creation of a texton vocabulary. Based on these observations, we do not consider filter bank approaches in our framework.

Operation	Image Size	Runtime CPU [ms]	Runtime GPU [ms]	Speedup
Schmid13	256×256	7945	160.1	49.64
	512×512	31516	595.9	52.89
	1024×1024	125430	2325	53.95
LM48	256×256	29288	583.4	50.2
	512×512	115794	2199	52.66
	1024×1024	461388	8645	53.37

Table 2.2: Runtime comparisons between CPU and GPU for calculating the response of an image to the Schmid13 and LM48 filter banks.

2.2.4 Local Binary Patterns

A powerful texture descriptor is the Local Binary Pattern (LBP) approach (Ojala and Pietikainen, 1999; Ojala et al., 2002). Besides being very efficient to compute, this descriptor is invariant to rotation as well as monotonic grayscale transformations.

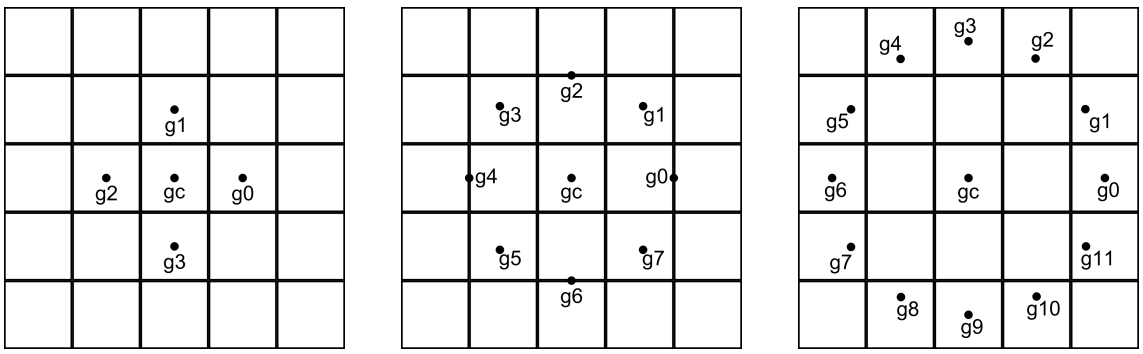
The local binary pattern algorithm works as follows: Given a center pixel g_c at location (x, y), a specific number of points P are sampled from a circle with radius R around the center pixel from the grayscale image. The coordinates for the p-th point g_p are calculated as

$$g_p = g_c + R \cdot \begin{pmatrix} -sin(\alpha(p)) \\ cos(\alpha(p)) \end{pmatrix} \tag{2.26}$$

with

$$\alpha(p) = \frac{2\pi p}{P}. \tag{2.27}$$

Figure 2.8 shows three examples for such a circular sampling. As some points drop off the regular image grid for most of the P, R pairs, the grayscale value for the sampled points is interpolated bilinearly. This maps well to the GPU, as bilinear interpolation is implemented in hardware and therefore is a free operation. The sampled intensity



(a) (b) (c)

Figure 2.8: Circular sampling for Local Binary Patterns: (a) shows the sampling pattern for $P = 4, R = 1.0$, (b) for $P = 8, R = 1.5$ and (c) for $P = 12, R = 2.0$

values are then compared with the center pixel to calculate a specific number from the pattern: A number with P bits is created, where every sampled point g_p represents one bit. If the sampled intensity $I(g_p)$ is larger than the intensity of the center pixel $I(g_c)$, the bit is set to one, otherwise to zero (see Figure 2.9). Mathematically expressed, this

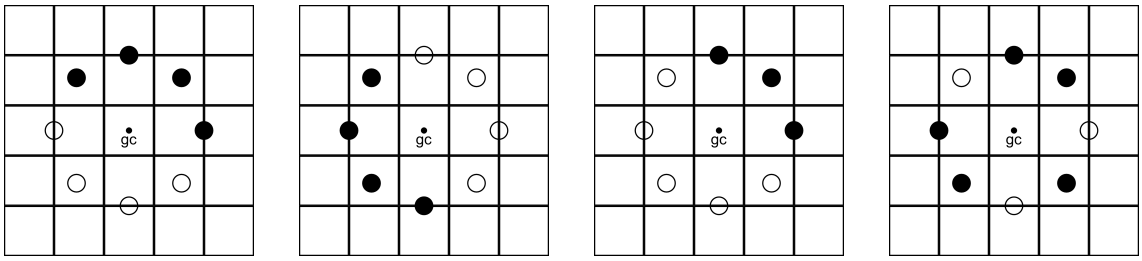
number is created as follows:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(I(g_p) - I(g_c)) \cdot 2^p, \tag{2.28}$$

with

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \tag{2.29}$$

indicating the comparison between a sampled point and the center pixel. The thresholding of the local neighborhood of the center pixel to a binary pattern leads to the name Local Binary Pattern.



(a) (b) (c) (d)

Figure 2.9: Examples for LBPs with $P = 8, R = 1.5$: A white dot represents that the interpolated intensity at that point is smaller than the intensity of the center pixel i.e. $s(I(g_p) - I(g_c)) = 0$. The patterns (a) and (b) could be interpreted as image edges, (c) could represent an image corner. The pattern in (a) amounts to 11110000_2, (b) is represented as 00011110_2. After applying the rotation invariance transform by shifting the patterns circularly until a minimum is obtained, both (a) and (b) amount to the same uniform pattern 00001111_2. The pattern in (c) is transformed from 11100000_2 to 00000111_2, which is also uniform. The pattern 01101101_2 in (d) is non-uniform.

In their work, Ojala et al. (2002) express the importance of uniform patterns. A uniform pattern exhibits at most two 0/1 transition: E.g. the patterns 00000000_2, 00000111_2 or 00110000_2 are uniform, while the patterns 01000100_2 and 00110111_2 are not. Rotation invariance is obtained by a bitwise rotational shift until the minimum value is obtained (e.g., 00011100_2 would become 00000111_2). By taking into account only rotation invariant, uniform patterns, the possible number of patterns reduces to $P + 1$. When each of these patterns is assigned the number of '1' bits as label (i.e. labels range from 0 to P), and all non-uniform patterns are assigned the label $P + 1$, there are $P + 2$ possible labels

for any pattern. To compute the final descriptor, a histogram counting the occurrence of the $P+2$ possible labels in a given square environment is created and normalized. In our implementation, we fix the size of the square environment to $s = 3 \cdot R$, e.g. for $R = 3.0$ the histogram is accumulated over a $9x9$ patch.

Implementing the LBP descriptor on the GPU can be realized with only two kernels: The first kernel computes the rotation invariant, uniform local binary pattern for every pixel in parallel. The second kernel accumulates the normalized histogram forming the final LBP descriptor. Figure 2.10 shows a runtime comparison between the CPU and GPU implementations for three different image sizes and $P \in \{2, 4, \ldots, 24\}$ and $R \in \{2, 3, \ldots, 12\}$. For the largest image size of 1024×1024, the GPU implementation is able to compute the $P+2$-dimensional LBP-descriptors densely for most parameter pairs in less than 100ms. Furthermore, it scales better with the size of the image, and is at least 20 times faster than the CPU implementation for any $R \geq 3$.

2.3 Conclusion

In this chapter, we have described and evaluated several color models and texture features with respect to their applicability for our segmentation framework. Therefore, we have conducted detailed runtime experiments for both CPU and GPU implementations.

Based on the source image being given in RGB space, we have evaluated transformations to the grayscale range as well as HSV and CIELAB feature spaces, which led to two basic observations: First, the computational complexity of these color space transformations is negligible compared to the computational effort of textural features. Second, the color spaces HSV and CIELAB are well suited for our task due to the fact that the correlation between the color channels is not as high as in the RGB color space.

Furthermore, we have described four types of texture features: Basic grayscale patches, Haralick texture features, Local Binary Patterns as well as two filter Banks (Schmid13 and LM48). There exist many more approaches to texture classification, however, we have chosen the ones which we believe are best suited for dense computation on a GPU. Experiments on 1024×1024 images showed, that Local Binary Patterns can be computed very efficiently, with runtimes lower than 100ms. Haralick texture features can be computed in the range of two seconds, depending on the parametrization. Computing the filter bank responses yields runtimes comparable to Haralick features, however, these approaches would need further steps such as clustering or energy calculations. Therefore, we do not include filter bank methods in our framework.

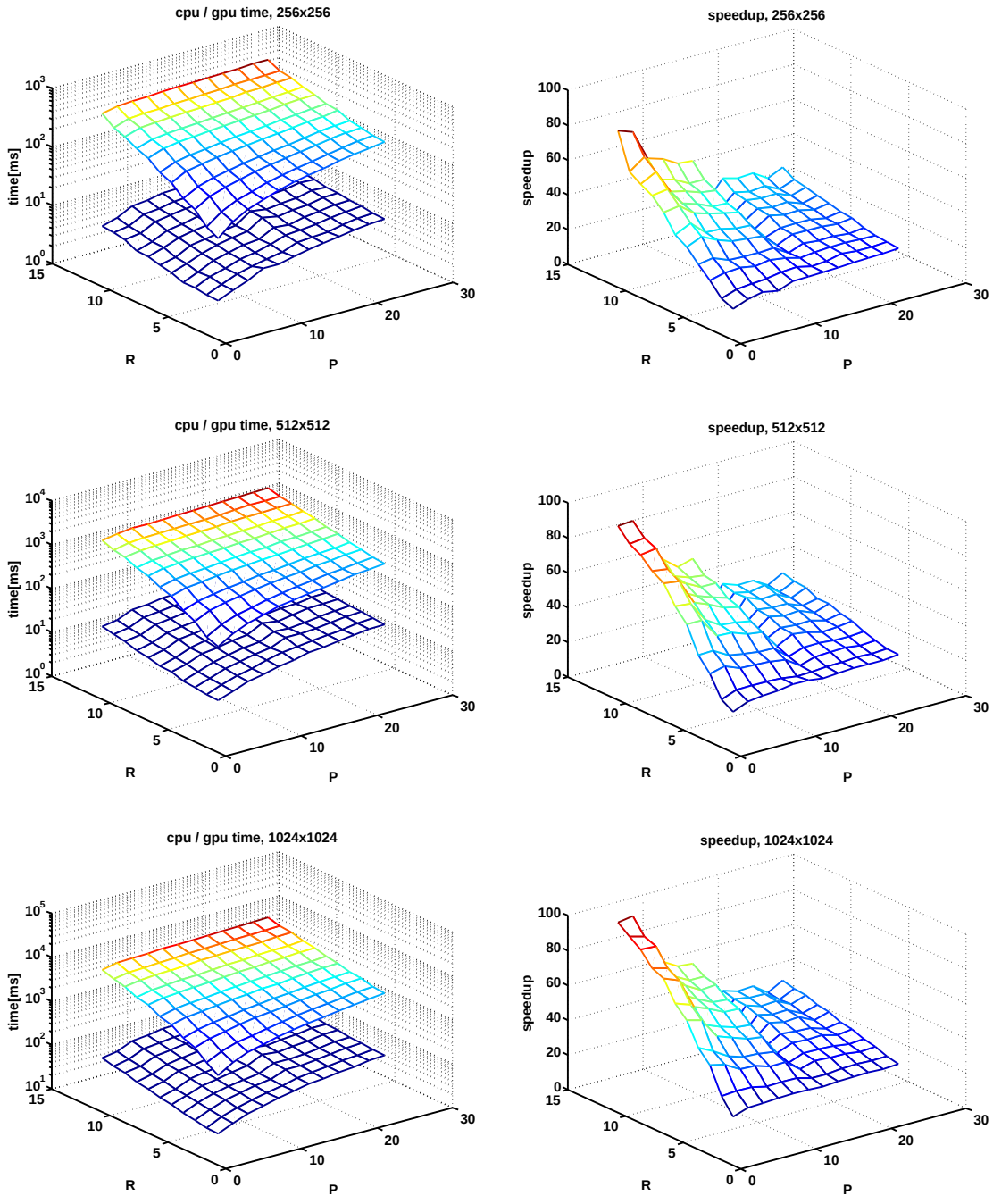


Figure 2.10: Runtime comparisons for Local Binary Patters with three different image sizes (256×256, 512×512 and 1024×1024): The first column shows the runtime comparisons for different parameters P, R between the CPU (upper mesh) and GPU implementations (lower mesh), note the logarithmic scale. The second column depicts the speedup factor between CPU and GPU.

3 Region Model

In the previous chapter, we have presented efficient ways to represent color as well as local structure in an image. In this chapter, we describe the generation of a descriptive model for our image segments. This problem can be stated as follows: The image pixels can be represented as a set $\mathcal{F}$ of $d-$dimensional feature vectors $\mathbf{f} \in \mathcal{F} \in \mathbb{R}^d$ derived in the previous chapter. Given user-defined seed pixels, we can assign them a label

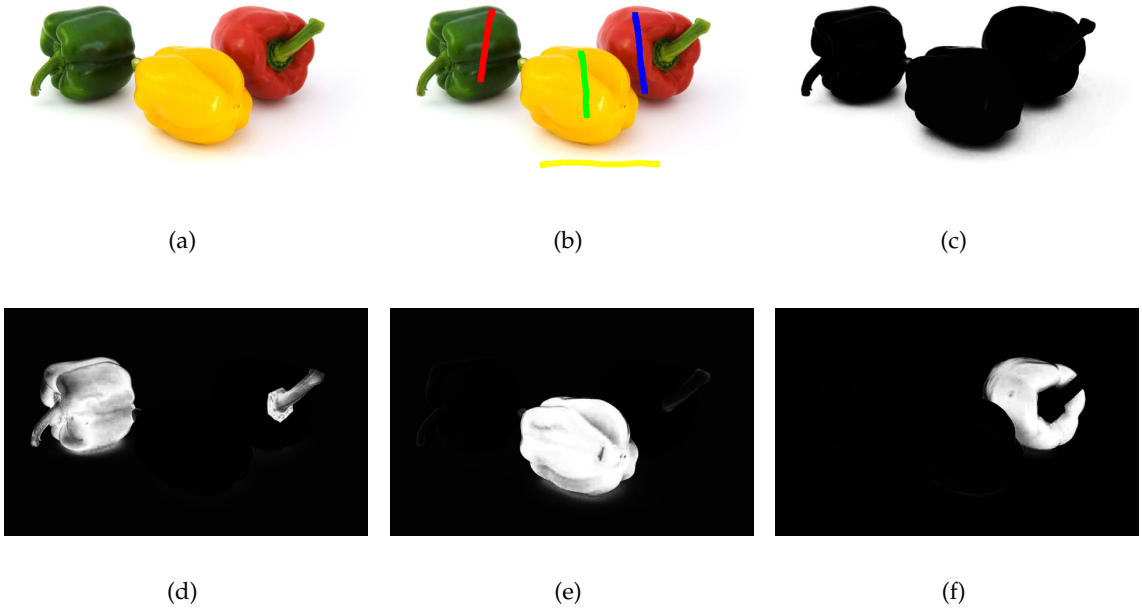
$$l \in \mathcal{K} = \{0, 1, \ldots, k-1\}, \tag{3.1}$$

where k is the number of distinctive regions. The two-label problem $k = 2$ amounts to a standard object / background segmentation. The set of pixels is divided into a subset

$$\mathcal{F}_s \subset \mathcal{F} \times \mathcal{K} = \{(\mathbf{f_0}, l_0), (\mathbf{f_1}, l_1), \ldots, (\mathbf{f}_{|\mathcal{F}_s|}, l_{|\mathcal{F}_s|})\}, \tag{3.2}$$

where the labels are known from seeds, and a subset $\mathcal{F}_u$ containing all unlabeled pixels. The goal is to derive a model from the seed pixels $\mathcal{F}_s$ in order to find the most probable label for every unseeded pixel from $\mathcal{F}_u$. See Figure 3.1 for a simple example: There, a model is created by taking the arithmetic mean in the CIELAB color space over all pixels in $\mathcal{F}_s$ belonging to the same segment. The probability for the unlabeled pixels in $\mathcal{F}_u$ is then derived over the distance to the center points in feature space.

This problem is a basic supervised learning problem, where a model is generated from the seed pixels in the training phase, and probabilities are assigned to every unlabeled pixel in the evaluation phase. Hence this problem can be tackled using supervised machine learning algorithms such as e.g. k-Nearest Neighbors, Gaussian Mixture Models, Histograms, Support Vector Machines or Decision Trees. In order to find a suitable



(a) (b) (c)

(d) (e) (f)

Figure 3.1: A simple region model is the arithmetic mean: Given the image (a) and four user-provided brush strokes (b), the arithmetic mean over all labeled pixels belonging to the same class is calculated in the CIELAB space. Then, the probability for every segment is derived using the distance to the center pixels in the feature space: (c) shows the likelihood that a pixel belongs to the background label with the yellow seeds, lighter means higher probability. (d),(e) and (f) show the probability for the green, yellow and blue pepper respectively. Note that the green stems of the red and yellow pepper have a significantly higher probability to belong the green pepper.

algorithm for our framework, we take into account the special requirements of our application:

Multi-Class We want to partition our image into multiple regions at the same time. Hence, the applied algorithm has to be capable of solving multi-class problems.

Robustness Users might make mistakes while marking seed pixels, thus the algorithm should not be prone to label noise.

High-Dimensionality The feature space employed in our framework has many dimensions and might incorporate non-informative or misleading features. The algorithm has to be capable to selecting suitable features itself in high-dimensional feature spaces.

Computational Complexity As for each part of our framework, computational performance is a crucial requirement, therefore the model must be fast to train and evaluate and ideally be portable to the GPU.

These requirements allow to exclude commonly used learning algorithms from the selection: Histograms and Gaussian Mixture Models are impractical for high-dimensional feature spaces, Boosting and k-Nearest Neighbors are prone to label noise and Support Vector Machines are not inherently multi-class. Therefore, as in previous work (Santner et al., 2009), we employ Random Forests introduced by Breiman (2001). Random Forests are inherently multi-class and feature selective, robust to label noise, and their parallel nature allows for implementing them on the GPU (Sharp, 2008). They have been used in various computer vision tasks, showing classification performance rates competitive to Boosting and Support Vector Machines (e.g. (Bosch et al., 2007; Breiman, 2001)).

In this chapter, we first describe Decision Trees and Random Forests as well as the process of implementing them on the GPU. In the experiments in Section 3.3.3 and Chapter 6, we compare Random Forests with a k-Nearest Neighbors implementation as well as linear Support Vector Machines (Fan et al., 2008).

3.1 Decision Trees

A decision tree is an elementary supervised machine learning algorithm, which can be employed for classification and regression. Due to the underlying tree structure (see Figure 3.2 for an example), its evaluation is very fast. The tree consists of two types of nodes: Split nodes (i.e. the elliptic nodes in Figure 3.2) incorporate some kind of decision function to find out to which child node a sample is passed. If all decision functions are binary (i.e. every node in the tree has at most two child nodes), the tree is called Binary Decision Tree. Leaf nodes (i.e. the rectangular nodes in Figure 3.2) assign a sample a probability for each class.

3.1.1 Evaluation

Given a previously trained binary decision tree and the evaluation set $\mathcal{F}_u$, the evaluation works as follows: A sample from $\mathcal{F}_u$ is dropped onto the root note of the tree. Depending on the outcome of the decision function, it is propagated to either the left or right child node, until it reaches a leaf node. It is then assigned the probabilities stored in the leaf node reached. This procedure is performed for every sample from the

evaluation set $\mathcal{F}_u$. See Figure 3.2 for an example: There, a Binary Decision Tree modeling a two-class problem in a four-dimensional feature space is depicted. The sample $\mathbf{f} = [0.2, 0.7, 0.1, 0.9]$ is evaluated, starting in the root node with the decision function $\mathbf{f}(2) > 0.7$. Our sample has $\mathbf{f}(2) = 0.1$, hence the statement is false and the sample is propagated to the right child node. In this child node, the decision function $\mathbf{f}(1) > 0.4$ is true and the sample is propagated to the left child node. The sample follows the path marked in red all the way to a leaf node, which assigns the probabilities $p_l = [0.7, 0.3]$. This means, that the sample has a probability of 0.7 for label 0 and 0.3 for label 1.

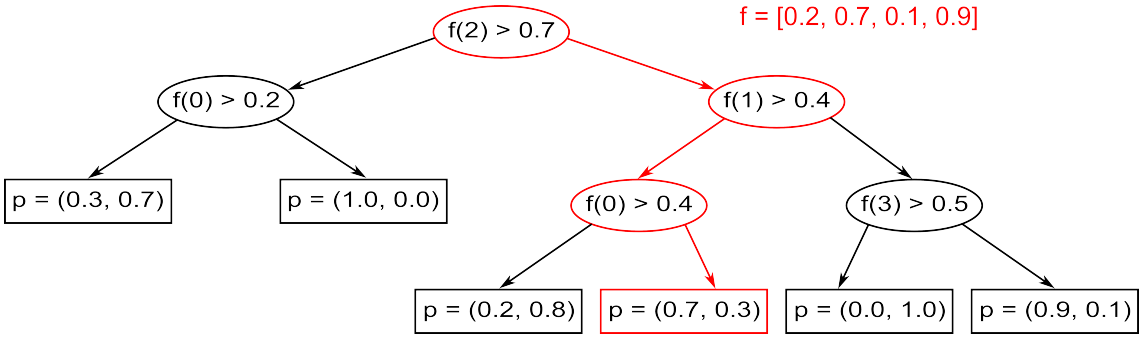


Figure 3.2: A binary decision tree consists of split nodes (elliptic) with decision functions and leaf nodes (rectangular) with assigned probabilities. The tree depicted in the figure above represents a two-label problem in a four-dimensional feature space. When the sample $\mathbf{f} = [0.2, 0.7, 0.1, 0.9]$ is evaluated, it takes the path marked in red color through the tree and is assigned the probabilities stored in the leaf node.

3.1.2 Training

The training procedure of a decision tree is more complex than the evaluation: Given the training set $\mathcal{F}_s = \{(\mathbf{f_0}, l_0), (\mathbf{f_1}, l_1), \ldots, (\mathbf{f}_{|\mathcal{F}_s|}, l_{|\mathcal{F}_s|})\}$, one tries at every node to find a suitable binary decision function. With k the number of classes in our problem and p_l the fraction of samples with label l in the set, there are two commonly used mechanisms for tree construction: While the CART algorithm (Breiman et al., 1984) employs the Gini impurity

$$I_G = 1 - \sum_{l=0}^{k-1} p_l^2, \tag{3.3}$$

the ID3 (Quinlan, 1986), C4.5 and C5.0 (Quinlan, 1993) algorithms employ the Information Gain

$$I_E = -\sum_{l=0}^{k-1} p_l log(p_l), \tag{3.4}$$

to measure how good a decision function separates the set. When the best decision function is found, the training set is distributed to the new child nodes according to the decision function and the process starts from scratch. Decision functions can e.g. be thresholds on a single feature (as depicted in Figure 3.2) or hyperplanes constructed from several features. Selecting the best decision function from a large number of candidate functions is a computationally expensive process, but in turn automatically selects features which are discriminative for the learning problem. In the Randomized Decision Trees approach proposed by Amit et al. (1996), the best decision function is not searched in the pool of all candidate functions, but e.g. in a randomly sampled subset. This typically leads to larger decision trees, but in turn greatly reduces the computational workload during training. A leaf node is inserted when the node is pure (i.e. holds only samples with the same label), a maximum depth is reached or no suitable decision function can be found. The leaf node probabilities are then assigned according to the class distribution p_l of the remaining samples in the leaf node.

3.2 Random Forests

Random Forests combine the bagging approach of Breiman (1996) with Randomized Decision Trees. Bagging (short for Bootstrap Aggregating) is a technique to improve the stability and generalization of a machine learning algorithm: Instead of training a model on the whole training set $\mathcal{F}_s$, several models are trained on subsets $\mathcal{F}_{s,i} \subseteq \mathcal{F}_s$ of the training set. The subsets of the training set are randomly sampled from $\mathcal{F}_s$ with replacement. Thus, a Random Forest is a set of N binary decision trees, each of which is trained with the random subset $\mathcal{F}_{s,i}$.

In contrast to standard decision trees, where one aims at finding the optimal decision function in every split node, the decision trees applied in Random Forests exhibit more randomness: Breiman (2001) and Bosch et al. (2007) employ decision functions of the form

$$\mathbf{f}^T \mathbf{w} \leq \theta, \tag{3.5}$$

where $\mathbf{f}$ is the feature vector, $\mathbf{w}$ is an arbitrary weight vector and θ is a scalar threshold.

The weight vector $\mathbf{w}$ exhibits only few non-zero elements, which amounts to a separating hyperplane taking into account only a subset of the dimensions of the feature space. Even thresholding only one single feature is possible by allowing only one non-zero element in $\mathbf{w}$. In every split node, several weight vectors are generated randomly. The splitting function is then derived using only the randomly chosen weight vectors, i.e. only in some dimensions of the input feature space. This reduces the computational complexity of finding suitable decision functions largely but still preserves the feature selective capabilities of decision trees.

In our framework, we only use binary thresholding on one single feature i.e. $\mathbf{w}$ has only one non-zero element e.g. $\mathbf{w} = (0,0,0,0,1,0,0,0)$. The number of randomly chosen weight vectors per split node is set to one fifth of the feature space dimensions d, e.g. in a 30-dimensional feature space, $d/5 = 6$ random features are selected and tested in every split node. For finding the threshold θ, first the minimum and maximum response of $\mathbf{f}^T\mathbf{w}$ over all samples is computed. Then the best threshold is selected out of ten candidates equally distributed between the minimum and maximum response. An example: Given a $d = 25$-dimensional feature space, five features are randomly selected. For each of these features, ten threshold value candidates are computed. Finding the best decision function then amounts to selecting the feature/threshold combination out of $5x10 = 50$ possibilities that maximizes the Gini impurity (3.3).

For the evaluation of Random Forests, each sample of the test set $\mathcal{F}_u$ is propagated through each tree resulting in a probability $p_n(l|\mathbf{f})$, for the n^{th} tree. These probabilities can be combined to a forest's joint probability

$$p(l|\mathbf{f}) = \frac{1}{N}\sum_{n=1}^{N} p_n(l|\mathbf{f}). \tag{3.6}$$

3.2.1 Multi-Core Implementation

Random Forests consist of a considerable amount of independent binary decision trees. This inherent parallelism makes them an ideal choice for multi-core implementations. The first reported GPU implementation of Random Forests was presented by Sharp (2008), where both training and evaluation were ported using the HLSL shader language. Decision trees are typically trained in a recursive manner by growing a tree data structure with nodes and pointers between nodes. As stacks do not exist in current GPU architectures, recursion and dynamic tree data structures can not be used to grow a tree. Therefore, similar as Sharp (2008), we employ a linear data structure to represent the

Random Forest in GPU memory. See Table 3.1 for an example: For every split node, we store its own index, the index of its left child node and the decision function represented by the feature indices i with non-zero weights $\mathbf{w}_i$ and the threshold θ. The index of the right child node is per definition the index of the left child node incremented by one. For leaf nodes, we store its own index, as well as the probabilities for each class $p(l|\mathbf{f})$. The index of the left son is set to -1 to indicate that the node is a leaf node. Hence we are able to represent a whole unbalanced binary decision tree as a matrix. Storing a Random Forest then amounts to concatenating several decision tree matrices horizontally to form a larger matrix.

Node Index	Child Node Index	Feature Index i or $p(0\|\mathbf{f})$	Weight $\mathbf{w}_i$ or $p(1\|\mathbf{f})$	Threshold θ
0	1	2	1.0	0.7
1	3	0	1.0	0.2
2	5	1	1.0	0.4
3	-1	0.3	0.7	
4	-1	1.0	0.0	
5	7	0	1.0	0.4
6	9	3	1.0	0.5
7	-1	0.2	0.8	
8	-1	0.7	0.3	
9	-1	0.0	1.0	
10	-1	0.9	0.1	

Table 3.1: A linear data structure for representing binary decision trees. Here, the exemplary tree depicted in Figure 3.2 is shown. The node indices start at 0 (root node) and then grow first from left to right and then with the depth of the tree. The field 'Child Node Index' stores the index of the left child node or -1 for leaf nodes. In the remaining fields, either the decision function is stored (with feature indices i and corresponding weights $\mathbf{w}_i$ as well as the threshold θ, or in case of a leaf node, the probabilities $p(k|\mathbf{f})$.

3.2.1.1 Training

The training process of Random Forests does not map well to the GPU architecture mainly because of two reasons:

- The randomness introduced by the bagging process as well as the random feature selection leads to non-uniform memory access patterns between different trees, which impede a high memory bandwidth on the GPU.

- The parallelization strategy is difficult to chose: If one assigns each thread a separate tree to train, most of the GPU would run idle. The number of trees employed in a segmentation problem ranges between 50 and 500, which leads to lots of idle GPU cores (cf. Section 1.5.2). Other parallelization strategies quickly lead to very complex implementations.

In our framework, we therefore perform training of the Random Forests using a multi-core CPU implementation. We parallelize by separating the decision trees, such that the total number of trees is distributed among the available CPU cores. This leads to a speedup factor of $3 - 3.5$ on a current quad-core processor. As the CPU supports recursion, the trees are built using a recursive data structure. After the training has finished, this data structure is translated to the linear data structure for GPU evaluation.

3.2.1.2 Evaluation

The evaluation of a Random Forest can be performed in two separate steps: The first kernel evaluates every sample of the evaluation set $\mathcal{F}_u$ on every tree independently, yielding for every sample the leaf-node probabilities of every tree. The second kernel then performs the combination of the leaf-node probabilities for every sample.

An example: Given a $k = 3$-class problem for an image of the size $640x480$ and a Random Forest with $N = 100$ trees. The first kernel then computes $640x480x100 \approx 30M$ threads and produces $640x480x100x3$ probability values. In the second kernel, the sum

$$p(l|\mathbf{f}) = \frac{1}{N} \sum_{n=1}^{N} p_n(l|\mathbf{f}). \tag{3.7}$$

is calculated over all N trees yielding $k = 3$ probability values for every pixel.

Note that the intermediate probabilities consume lots of memory: If the probability values are stored as 4-byte floating point variables, the intermediate output consumes $640x480x100x3x4 \approx 352MB$ of graphics card memory. To not run into memory issues, the whole problem is divided into several smaller chunks similar as during the computation of the GLCM described in Section 2.2.2.1.

3.3 Performance Evaluation

In this section, we evaluate the performance of the region models trained with Random Forests. We therefore compare to two other learning algorithms: An implementation of

the k-Nearest Neighbors (kNN) algorithm and a linear Support Vector Machine (SVM).

3.3.1 K-Nearest Neighbors

The k-Nearest Neighbors algorithm finds for every sample of the test set the k samples of the training set with the smallest Euclidean distance in the feature space. This leads to virtually no training time but lots of memory consumption, as in the training phase only the entire training set is stored. In the test phase, the distance between all samples of the training set and all samples of the test set needs to be computed and compared. This can be easily parallelized: We employ a GPU-based approach similar to the one presented by Garcia et al. (2008), with iterative minimum search instead of sorting.

3.3.2 Support Vector Machines

Support Vector Machines are learning algorithms which try to find the optimal decision boundary between samples of the training set with different labels by maximizing the margin between the decision boundary and the training samples. We employ an approach with linear decision boundaries (i.e. hyperplanes in the feature space) implemented in the LIBLINEAR package (Fan et al., 2008), using an L2-regularized logistic regression solver.

3.3.3 Runtime

For performance evaluation, we employ two common machine learning datasets: The Statlog (Landsat Satellite) and the Letter Recognition data set. The Statlog (Landsat Satellite) data set represents a 6-class problem in a 36-dimensional feature space, with 4435 training samples and 2000 test samples. The Letter Recognition data set is a 26-class problem in a 16-dimensional feature space, with 16000 training samples and 4000 test samples. In our segmentation framework, we typically deal with differently sized problems: As our training sets consist only of the seeded pixels and our test sets are built from every single image pixel, we have the situation that $|\mathcal{F}_s| \ll |\mathcal{F}_u|$. For that reason, we compare on a third artificial dataset (which we call Segmentation data set) featuring a 5-class problem in a 3-dimensional feature space with 12500 training samples and 300000 test samples. The artificial segmentation dataset, which consists of five Gaussian modes with different symmetric covariance matrices, is depicted in Figure 3.3.

We compare the Random Forest algorithm described in this chapter as well as an implementation of the k-Nearest Neighbors (kNN). Results are given in Table 3.2: For

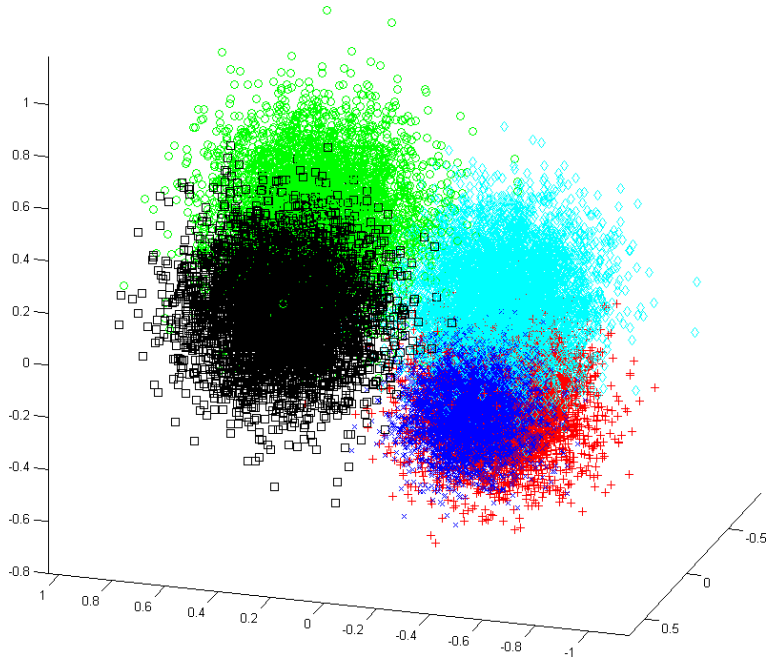


Figure 3.3: To benchmark our learning algorithm, we have created an artificial dataset resembling conditions occurring in our segmentation framework. This dataset consists of five classes in a 3-dimensional feature space, with 12500 training samples and 300000 test samples.

the relatively small machine learning data sets Letter and Landsat, the kNN algorithm outperforms the Random Forest in terms of speed while reaching comparable accuracy. Here, due to $|\mathcal{F}_s| > |\mathcal{F}_u|$, the training phase of the Random Forests strongly influence the overall runtime. An important issue is that the accuracy of the kNN algorithm strongly depends on the right choice of the parameter k: When k is too large, the algorithm tends to overfit the training data. Note that the Random Forests do not exhibit this behavior. On the larger artificial dataset, where $|\mathcal{F}_s| \ll |\mathcal{F}_u|$, the Random Forests yield more accurate results in less time than the kNN algorithm.

3.3.4 Model Quality

In this section, we give a qualitative comparison of the learning algorithm results on interactive segmentation problems. We employ a $d = 17$-dimensional feature space consisting of CIELAB color features and Local Binary Patterns with $P = 12, R = 2$ as texture descriptor. We compare Random Forests with 100 trees, the kNN algorithm with $k = 10$ and a linear SVM with $C = 0.045$. In the following figures, the resulting

Data	Algorithm	Parameters	T_{train}[ms]	T_{test}[ms]	Accuracy
Landsat	Random Forest	$N = 10$	433.55	0.65	86.95
		$N = 20$	845.57	0.79	88.05
		$N = 50$	2092.77	1.76	89.90
		$N = 100$	4093.82	3.64	90.40
		$N = 200$	8266.54	9.63	90.45
		$N = 500$	20502.85	25.27	90.60
	kNN	$k = 1$	xxx	54.92	89.35
		$k = 2$	xxx	57.70	89.35
		$k = 5$	xxx	62.90	90.65
		$k = 10$	xxx	73.65	90.05
		$k = 20$	xxx	96.58	89.10
Letter	Random Forests	$N = 10$	1492.18	3.32	85.35
		$N = 20$	2762.56	7.31	90.05
		$N = 50$	6869.20	24.06	94.00
		$N = 100$	13613.92	53.90	95.32
		$N = 200$	27691.55	126.86	95.70
		$N = 500$	73676.96	286.12	96.10
	kNN	$k = 1$	xxx	438.72	95.20
		$k = 2$	xxx	469.05	95.20
		$k = 5$	xxx	583.71	95.40
		$k = 10$	xxx	718.55	94.57
		$k = 20$	xxx	1005.55	93.42
Segmentation	Random Forests	$N = 10$	381.30	58.23	86.42
		$N = 20$	773.36	112.11	87.09
		$N = 50$	1872.71	387.70	87.96
		$N = 100$	3809.13	971.98	87.95
		$N = 200$	7843.54	1980.04	88.09
	kNN	$k = 1$	xxx	10924.49	83.96
		$k = 2$	xxx	12388.69	83.96
		$k = 5$	xxx	16560.48	87.30
		$k = 10$	xxx	23283.90	88.08
		$k = 20$	xxx	37009.84	88.71

Table 3.2: Performance comparison on standard machine learning datasets (Landsat, Letter) and an artificial dataset (Segmentation) resembling conditions occurring in our segmentation framework. While Random Forests are slower than the k-Nearest Neighbors algorithm on Landsat and Letter due to the training phase, they outperform kNN on the Segmentation data set, where the amount of test samples is much higher than the amount of training samples. Note also, that Random Forests do not tend to overfit the training set.

probabilities are encoded in the grayscale range for every label k separately. For the Random Forests and the kNN algorithm, where $p(l|\mathbf{f}) \in [0, 1]$, the intensity is set to $-p(l|\mathbf{f})$. For the SVM, the intensity is set to $-log(p(l|\mathbf{f}))$. This leads to high probability regions appearing dark and low probability regions light. For a better visibility, the logarithmic SVM results are contrast enhanced.

Figure 3.4 shows the inherent feature selective capabilities of Random Forests: The 14-dimensional Local Binary Pattern descriptor exhibits dimensions which are mislead-

ing. While the kNN algorithm as well as the linear Support Vector Machine results are influenced by these misleading dimensions (cf. the 'noisy' patterns in Figure 3.4), the Random Forest algorithm nearly completely neglects them.

The same behavior can be observed in Figure 3.5: There, the problem is a three-label problem in a $d = 17$-dimensional feature space, with $|\mathcal{F}_s| = 33360$ training samples and $|\mathcal{F}_u| = 154401$ evaluation samples. The task is to find a good description for a bridge, surrounding grassland and the background. A difficulty is given by the similarity of the color of the bridge to the color of the grassland it is standing on. The kNN algorithm, which is per definition a bad generalizer, leads to a very noisy result. Also the time for evaluation (103810 ms) is unacceptable for an interactive framework. The linear SVM also has difficulties separating the bridge from the grassland, especially at the upper border of the bridge, however, the training time (467.82 ms) and the evaluation time (518.54 ms) are very low. The Random Forest is able to select discriminative features and yields a well generalized result. While the training takes longer than the SVM training (5444.8 ms), the evaluation is completed in 181.29 ms.

The time spent during training can be reduced significantly by replacing the solid brush with a spraygun-like brush: As neighboring points typically have very similar properties, lots of redundancy is given in a training set deducted from seeds drawn with a solid brush. By randomly sampling a given percentage from these seed pixels, the training set is significantly reduced while keeping most of the important information. Figure 3.6 shows a segmentation problem where a spraygun-like brush is applied instead of a solid brush. There, a five-label problem is solved with $|\mathcal{F}_s| = 1818$ training samples and $|\mathcal{F}_u| = 154401$ evaluation samples. The regions to segment are the sky (red seed pixels), the roof (green seed pixels), the church towers (blue), the mountain (yellow) as well as the leaves in the top right of the image (magenta). In Figure 3.6, only the probabilities for the labels sky (red), roof (green) and mountain are shown (yellow). Here again, the Random Forests lead to cleaner, more noise-free results than the other two algorithms. With the spraygun brush, also the runtime characteristics change massively: While the kNN algorithm needs 3243.04 ms for the evaluation, the Random Forest takes 824.76 ms to train and 319.92 ms to evaluate. Thus with Random Forests, we obtain a high-quality probability estimate for this difficult five-label problem in little more than one second.

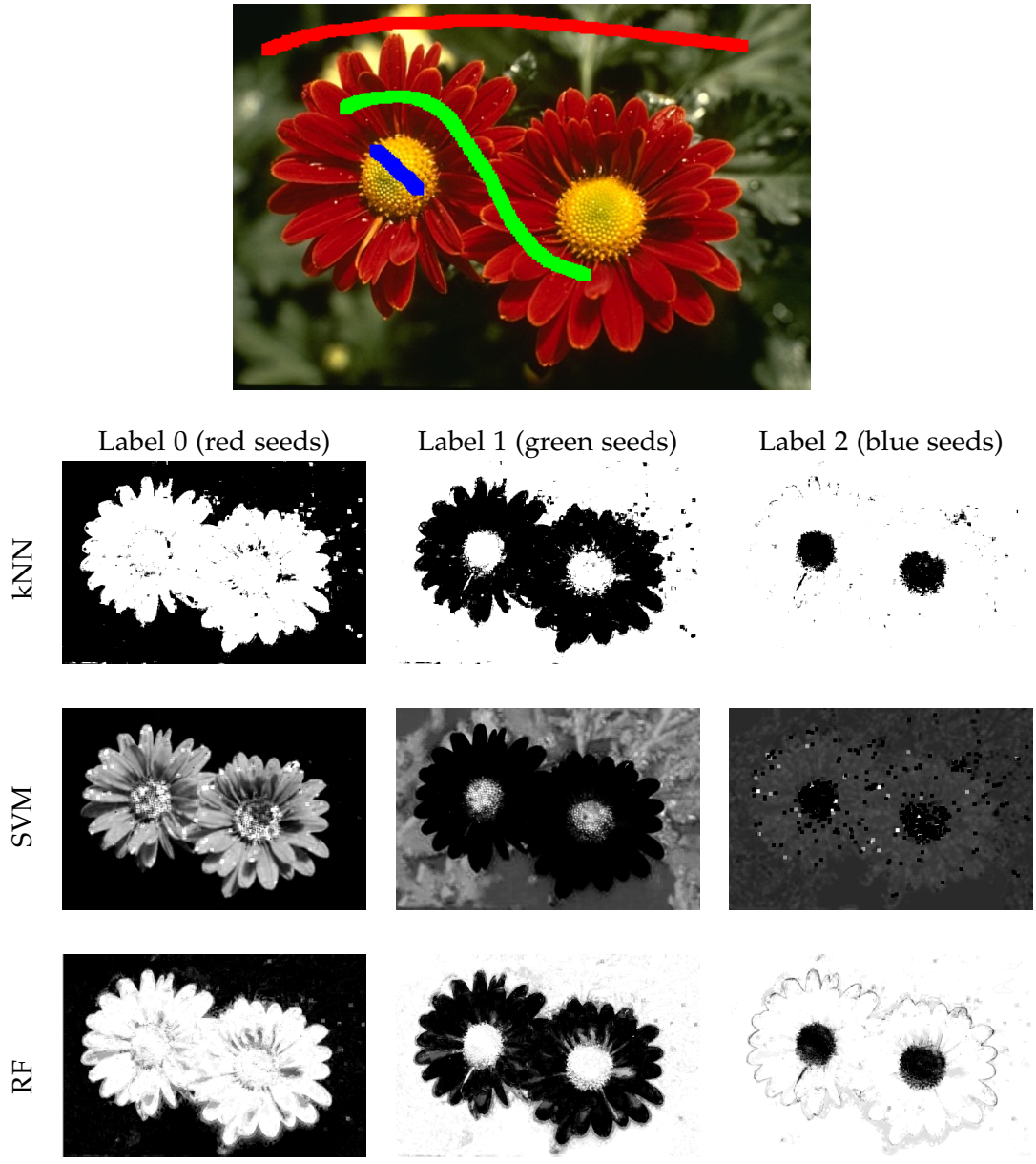


Figure 3.4: Region models trained with three labels: Label 0 (red seeds) is depicted in the left column, label 1 (green seeds) in the middle column, label 2 (blue seeds) in the right column. The first row shows the probabilities for the separate labels obtained with a kNN algorithm with $k = 10$, the second row the result of a linear SVM with $C = 0.045$. The third row depicts results obtained with a Random Forest consisting of 100 trees.

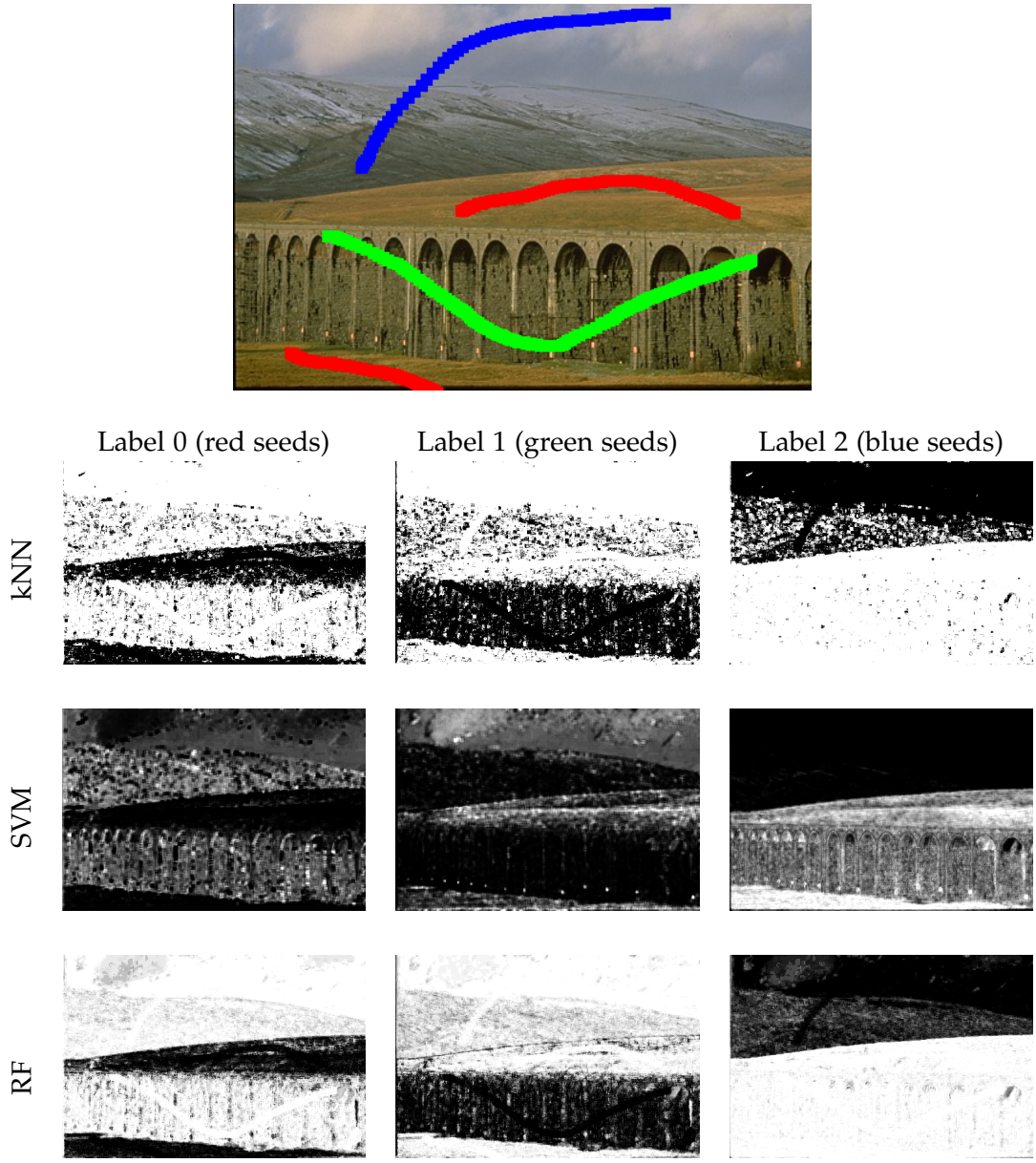


Figure 3.5: Region models trained with three labels: Label 0 (red seeds) is depicted in the left column, label 1 (green seeds) in the middle column, label 2 (blue seeds) in the right column. The first row shows the probabilities for the separate labels obtained with a kNN algorithm with $k = 10$, the second row the result of a linear SVM with $C = 0.045$. The third row depicts results obtained with a Random Forest consisting of 100 trees.

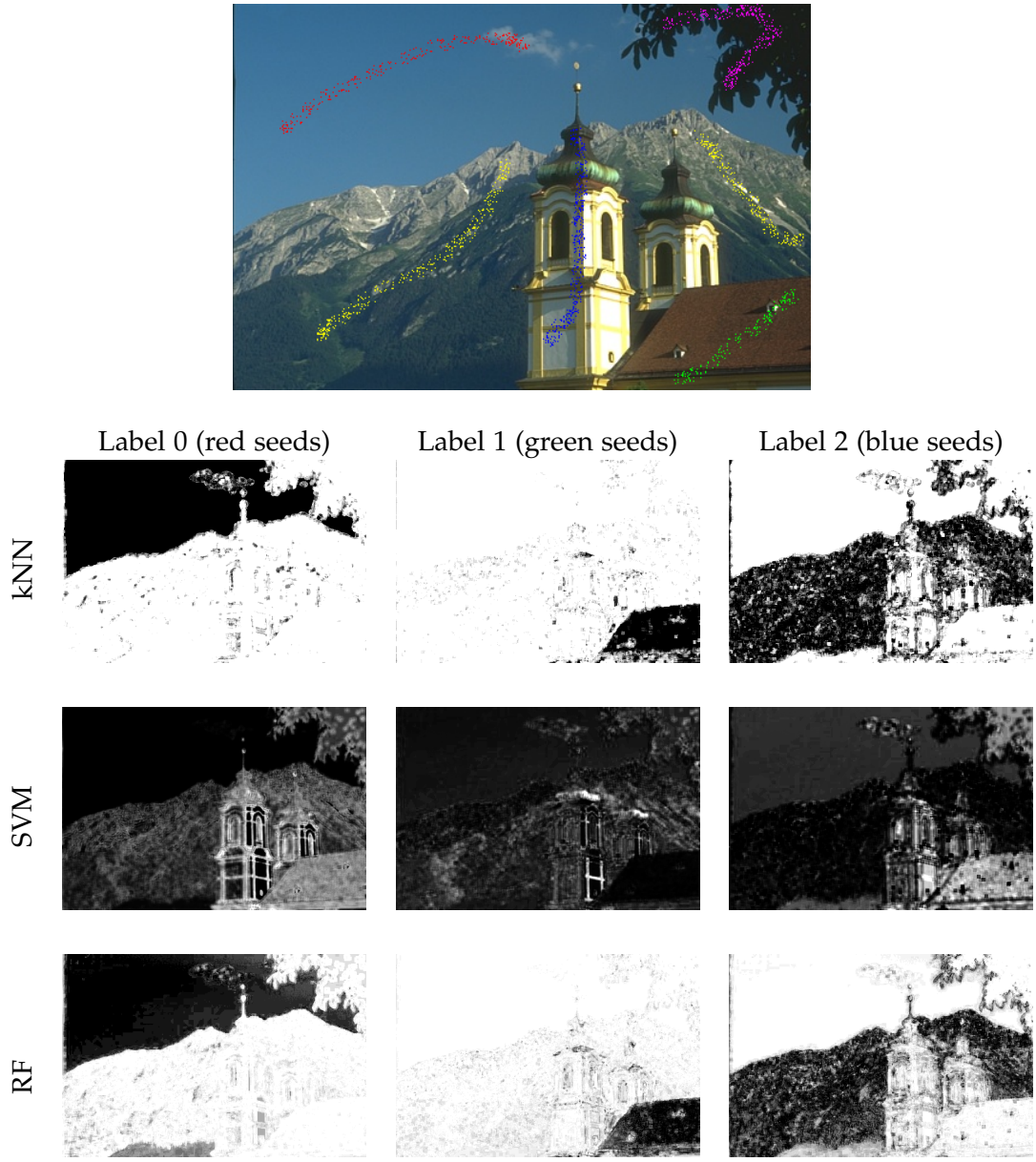
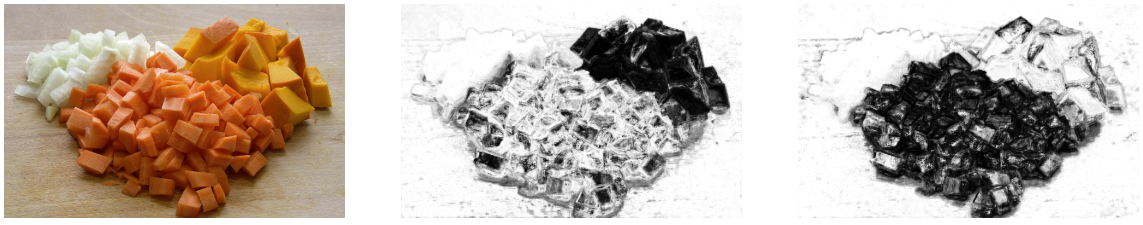


Figure 3.6: Using a spraygun-like brush for marking seed pixels leads to a significant reduction of the model training time: The regions to segment are the sky (red seed pixels), the roof (green seed pixels), the church towers (blue), the mountain (yellow) as well as the leaves in the top right of the image (magenta). The learning algorithm outputs are arranged similarly to Figure 3.5. Due to spatial reasons, only three labels are shown: The sky (left), the roof (middle) as well as the mountain (right).

4 Segmentation

The previous chapters described how we can efficiently compute powerful pixel-wise posterior probabilities $p(l|\mathbf{f})$, encoding the probability that a pixel represented by a feature vector $\mathbf{f}$ belongs to a given label $l \in \{0, 1, \ldots, k-1\}$. The quality of these probabilities depends on many factors, e.g. the employed feature space, the type and parametrization of the learning algorithm or the segmentation problem itself. Hence, if one of these subparts is badly conditioned or not parametrized reasonably, the probabilities might get very noisy. Figure 4.1 shows a 4-class segmentation problem together with the posterior probabilities for two exemplary segments. These exemplary segments have a very similar color distribution, i.e. some of the pixels of both segments lie close together in the color feature space. While most of the pixels of the two segments can be classified correctly, some pixels have high probabilities for the wrong segment.



(a) (b) (c)

Figure 4.1: Given the 4-class segmentation problem in (a), (b) and (c) depict pixel-wise probabilities for two segments with similar appearance (using the same encoding as in the previous chapters, where a high probability corresponds to dark regions and vice versa). This leads to some pixels exhibiting a high probability for the wrong label.

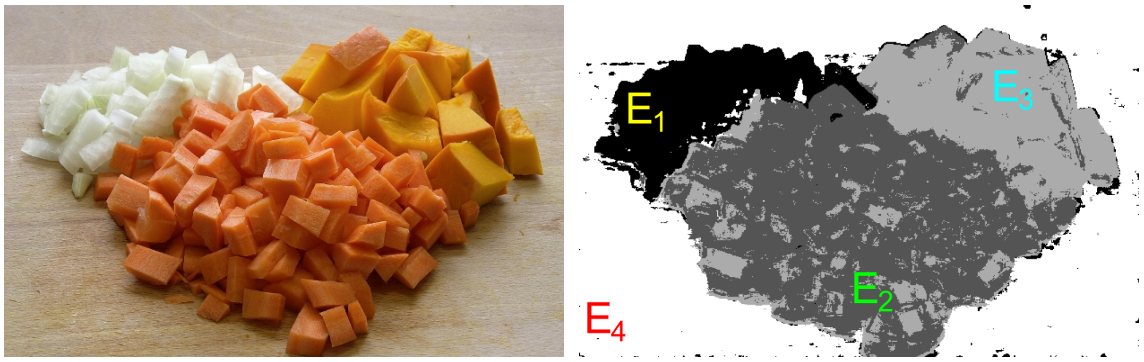
The overall problem is to find a suitable split of the image domain Ω into k sets E_l with $l \in \{0, 1, \ldots, k-1\}$

$$\bigcup_{l=0}^{k-1} E_l = \Omega, \tag{4.1}$$

such that every pixel is part of exactly one set

$$E_i \cap E_j = \emptyset \; \forall \; i \neq j. \tag{4.2}$$

To find such a labeling based on noisy data is a typical and common problem which not only occurs in segmentation, but in many other computer vision problems such as stereo, motion estimation or inpainting. In a naive approach, one could find such a labeling by simply assigning each pixel the label with the highest probability. Based on noisy posterior probabilities, this would lead to noisy labelings (cf. Figure 4.2).



(a) (b)

Figure 4.2: (b) shows labels obtained from taking the maximum posterior probability for the 4-class problem in (a): The pixels encoded in black belong to region E_1, dark gray to E_2, light gray to E_3 and white to region E_4.

A suitable labeling should exhibit spatially compact regions, which have borders aligned with large image intensity changes. These requirements match the properties of many binary segmentation methods described in the introduction of this thesis, e.g. the Geodesic Active Contours model (Caselles et al., 1997), Graph Cut segmentation (Boykov and Jolly, 2001), weighted Total Variation (Bresson et al., 2007), the Random Walker algorithm (Grady and Funka-Lea, 2004) or Geodesic Segmentation (Criminisi et al., 2008, 2010). While some methods such as the Random Walker algorithm and Geodesic Segmentation are inherently able to tackle multi-label segmentation problems,

the extension of Graph Cut segmentation or weighted Total Variation to $k > 2$ is not straightforward.

In this chapter, we first explain the role of the regularization terms in segmentation energy functionals. Then, we describe how current multi-label segmentation methods have evolved from their binary counterparts in both the discrete as well as continuous domain. Note that we focus on the employed models and their properties only. For details on the energy minimization procedures, refer to the respective papers. Finally, we show the incorporation of a multi-label segmentation method into our framework.

4.1 Regularization Terms

When image segmentation is cast as an energy minimization problem, the energy typically takes the form

$$E(u) = E_R(u) + \lambda \cdot E_D(u), \tag{4.3}$$

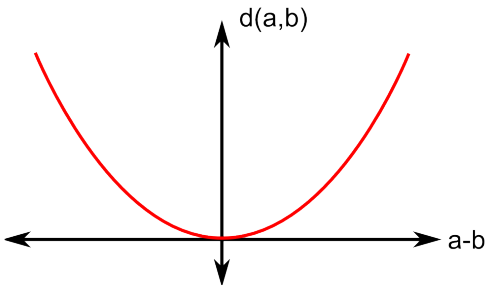
with u denoting the labeling. The term $E_D(u)$ is referred to as unary term or as data term, which forces the segmentation to match the probabilities obtained from the region models. The regularization term or pairwise term E_R typically takes care of pulling the contours towards image gradients and keeping the overall length of the contour small. In order to yield spatially compact regions, different labels between neighboring pixels are penalized: In Graph Cut segmentation, regularization terms take the form

$$E_R(u) = \sum_{\{p,q\} \in \mathcal{N}} d(u_p, u_q), \tag{4.4}$$

where $\mathcal{N}$ denotes the set of neighboring pixels p, q, and $d(u_p, u_q)$ denotes a function computing a penalty depending on the label similarity of pixels p and q. In the continuous setting, a common regularization term is the weighted Total Variation

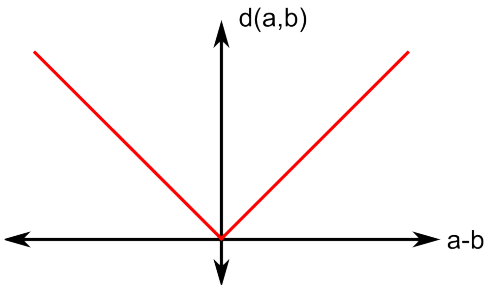
$$E_R(u) = \int_\Omega g(x) |\nabla u| dx, \tag{4.5}$$

where the penalty function is a L1-norm and $g(x)$ is an edge indicator function. Table 4.1 shows different penalty functions employed in regularization terms of computer vision energy functionals.



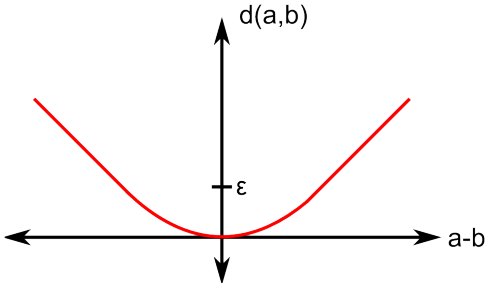
Quadratic (L2) Norm:

$$d(a,b) = (a-b)^2$$



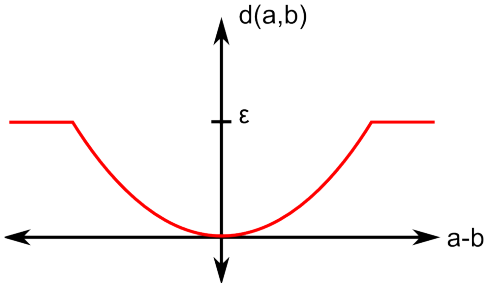
L1 Norm:

$$d(a,b) = |a-b|$$



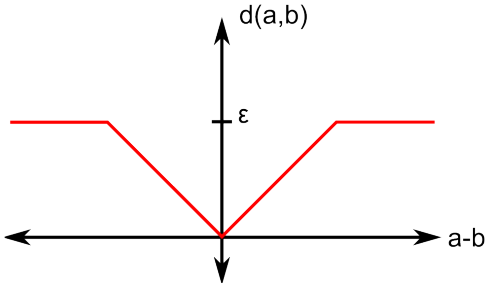
Huber Norm:

$$d(a,b) = \begin{cases} \frac{(a-b)^2}{2\epsilon} : & \text{if } |a-b| \leq \epsilon \\ |a-b| - \frac{\epsilon}{2} : & \text{else} \end{cases}$$



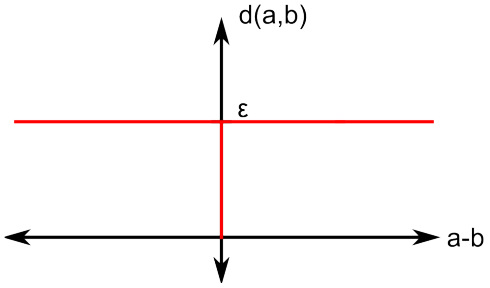
Truncated Quadratic:

$$d(a,b) = min((a-b)^2, \epsilon)$$



Truncated L1:

$$d(a,b) = min(|a-b|, \epsilon)$$



Potts:

$$d(a,b) = \epsilon \cdot [a \neq b]$$

Table 4.1: Different regularization terms employed in energy minimization functionals.

4.1.1 Properties

In order to investigate the properties of different penalty functions $d(a, b)$, we assume that the labels vary continuously in the range $[0, 1]$.

4.1.1.1 Convexity

A very important property of a regularizer is convexity: Formally, a function $d(x)$ is called convex, if

$$d(\lambda \cdot p + (1 - \lambda) \cdot q) \leq \lambda \cdot d(p) + (1 - \lambda) \cdot d(q) \tag{4.6}$$

for any two points p, q and any continuous variable $\lambda \in [0, 1]$. This means, that any straight line connecting two points of the function must not intersect the function. If

$$d(\lambda \cdot p + (1 - \lambda) \cdot q) < \lambda \cdot d(p) + (1 - \lambda) \cdot d(q) \tag{4.7}$$

holds, $d(x)$ is called strictly convex. Examples for convex functions are the L2-norm $d(x) = x^2$ or the L1-norm $d(x) = |x|$. The importance of convexity lies in its benefit for minimization: If a function is convex and the minimization is performed over a convex set, the global minimizer of the function can be found.

4.1.1.2 Discontinuity Handling

Another important property of penalty functions is the treatment of discontinuities: Consider the three signals depicted in Figure 4.3, which perform a transition from 0 to 1 via five (s_5), three (s_3) and one (s_1) separate steps respectively. Note that the overall step size is identical for all three signals, only the number of discontinuities and their strength changes.

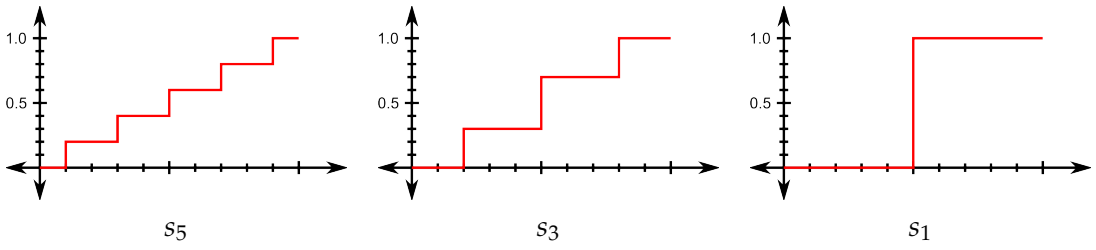


Figure 4.3: Signals for evaluating the handling of discontinuities of penalty functions.

In order to demonstrate the differences between the penalty functions, we calculate the energy of every signal with different penalty functions (Table 4.2). Based on how the

Penalty	s_5	s_3	s_1
L2-norm	0.2	0.34	1.0
L1-norm	1.0	1.0	1.0
Potts ($\epsilon = 0.1$)	0.5	0.3	0.1
Truncated L2 ($\epsilon = 0.1$)	0.2	0.28	0.1
Truncated L1 ($\epsilon = 0.1$)	0.5	0.3	0.1

Table 4.2: Different regularization terms lead to different energies for the signals depicted in Figure 4.3: While the L2-norm penalizes discontinuities, the L1-norm preserves them. Non-convex functions like the truncated L1 / L2 and the Potts function enhance discontinuities.

energies change with the number and strength of the discontinuities, one can divide the penalty functions into three types:

Discontinuity penalizing regularizations such as the L2-norm clearly favor small discontinuities over large steps. This leads to smooth results without sharp edges.

Discontinuity preserving regularizations treat all discontinuities equally independent of their step size. An example for a discontinuity preserving function is the L1-norm.

Discontinuity enhancing penalty functions like the truncated L1 and L2 norms as well as the Potts function favor large discontinuities over small ones. While this leads to sharp boundaries in the solution, it comes at the cost of being non-convex.

4.1.2 Applicability

Finally, we want to investigate which regularization terms can be applied for image segmentation. Generally, we want the energy to be convex in order to be able to compute a global minimizer. For image segmentation, we want discontinuities to be preserved or enhanced. Based on these observations, from the penalizers stated in Table 4.1 only the L1-norm seems to be suitable for image segmentation. Note that this is only applicable for the binary case: For multiple labels ($k > 2$), convex regularization terms are only applicable, when the order of the label space has a meaning for the problem it is representing. Consider the segments E_0, E_1 and E_2 represented by the labeling $u : \Omega \to \{0, 1, 2\}$. When applying any convex penalization (except a straight line), a jump between labels E_0 and E_2 would be more expensive than a jump between E_0 and E_1. While these convex regularizers are well suited for multi-label computer vision tasks

such as e.g. image restoration or stereo, they are not applicable to segmentation where the labels have only a symbolic character.

4.2 Binary Segmentation

Many approaches have been presented to obtain binary labelings ($k = 2$) using energy minimization. Refer to the introduction of this thesis for a comprehensive overview. Generally, the global minimum of binary segmentation energies can be computed in the discrete setting as well as in the continuous setting.

4.2.1 Discrete

A binary labeling u can be obtained with graph-based approaches (Boykov and Jolly, 2001) by minimizing the energy

$$E(u) = \lambda \cdot \sum_{p \in \mathcal{P}} R_p(u_p) + \sum_{\{p,q\} \in \mathcal{N}} [u_p \neq u_q] \cdot g(p,q), \tag{4.8}$$

with $R_p(u_p)$ (unary term) representing the pixel-wise probabilities obtained from the region models, and $g(p,q)$ (pairwise term) representing the image gradient. The global minimizer of (4.8) can be found using max-flow algorithms. Refer to Section 1.4.3.6 for a detailed description of Graph Cut-based binary segmentation.

4.2.2 Continuous

In the continuous domain, Bresson et al. (2007) employ the energy functional

$$E_{TV}(u) = \int_\Omega g(x)|\nabla u| d\Omega + \lambda \int_\Omega |u - f| d\Omega, \tag{4.9}$$

with u denoting the labeling and f describing the posterior probabilities from the region models. The first term, which corresponds to the pairwise term in Graph Cut segmentation is the weighted Total Variation, where the weight is represented by the edge detection function $g(x)$. The energy (4.9) is convex, but the set over which the energy is minimized is not: As we want to obtain a binary solution, u is only allowed to take a value in $\{0,1\}$ which is a non-convex set. Bresson et al. overcome this problem by relaxation: For the minimization, they allow u to vary continuously in $[0,1]$ which is a convex set. After computing the globally optimal continuous solution, the binary

segmentation can be obtained by thresholding. Bresson et al. show that the thresholded solution equals the global optimum of the binary problem for any threshold $\in [0, 1]$.

4.3 Convex Multi-label Functionals

In order to solve for multiple labels, one could easily employ binary energy minimization functionals and allow the field u to take more than two labels i.e. $u : \Omega \rightarrow \{0, 1, \ldots, k-1\}$. Ishikawa (2003) showed how a global minimizer for such functionals can be found in the discrete domain, Pock et al. (2008) demonstrated a globally optimal solution of this labeling task in the continuous domain. Both approaches require a convex regularization term as well as a linearly ordered label space. As stated in Section 4.1.2, the convex regularization terms would penalize label jumps differently depending on the labels involved: E.g. the transition of the set E_1 to the set E_3 would be more expensive than the transition between the sets E_1 and E_2. Such functionals hence can only be applied when the ordering of the label set is characteristic for the problem, which is the case for stereo or image restoration. In segmentation, where the label set is not ordered linearly but has only symbolic character, these functionals can not be applied.

4.4 Sequential Methods

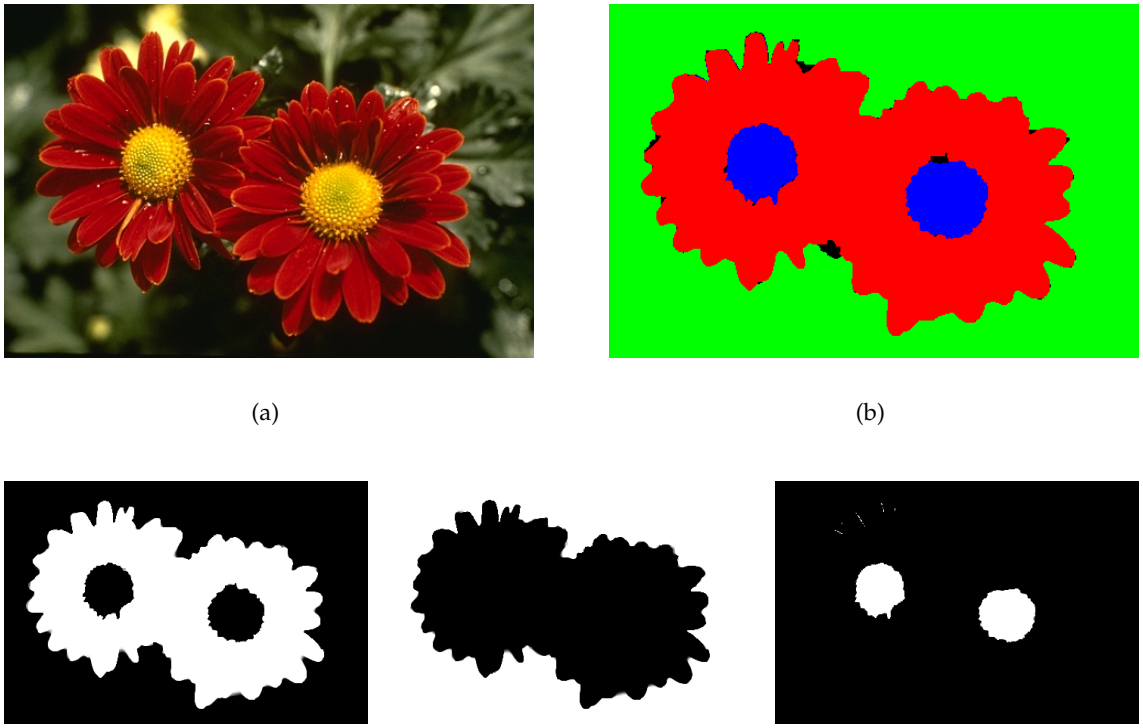
Based on the considerations of the previous Section, we need a regularization term that is able to treat the unordered labels occurring in segmentation equally. Employing the Potts regularizer

$$d(a, b) = \epsilon \cdot [a \neq b] \tag{4.10}$$

meets this requirement by penalizing neighboring pixels with different labels by a fixed scalar value ϵ. However, it has been shown that minimizing the Potts model for multiple labels can be reduced to finding the minimum cost multi-way cut, which is an NP-hard problem (Boykov et al., 2001; Dahlhaus et al., 1992). Therefore, a popular class of segmentation algorithms exist, that solve multi-label problems by combining several binary problems that are solved sequentially. Basically, none of them is able to find the global minimizer of the multi-label problem, however, they provide useful approximations.

4.4.1 1-vs-All

An approach, which is common for solving multi-label machine learning problems with binary classifiers, is to solve the multi-label problem in a 1-versus-All manner. Instead of solving one single segmentation with k labels, one splits up the problem into k binary problems. See Figure 4.4 for an example: Here, three binary segmentations are computed and fused to obtain the multi-label result. Note the small black regions in the final segmentation: When solving k binary problems, a pixel might not be part of any of the k segmented objects which leaves them unassigned in the result.



(a) (b)

(c) (d) (e)

Figure 4.4: A multi-label segmentation problem (a) can be solved as a combination of binary segmentations. The final segmentation (b) is obtained by combining the results of the binary problems (c-e). Note that a pixel might not be part of any of the binary segmentations and therefore remain unassigned in the result (cf. the black regions in (b)).

Figure 4.5 illustrates this phenomenon with a 3-label toy example: The gray center dot in Figure 4.5(a) has exactly the same distance in the RGB-space to all three border segments, and therefore the identical posterior probability for all three segments. The

corresponding binary segmentations for the three labels (Figures 4.5(b) - 4.5(d)) therefore yield the respectively shortest boundary through the center dot. The combined binary segmentations hence leave a triangle which remains unassigned. Donoser et al. (2009) approached this problem of unassigned pixels with a heuristic post-processing step in order to allocate unassigned regions to a specific label.

(a) (b) (c) (d) (e)

Figure 4.5: The gray center dot in (a) has the identical posterior probability for all three border segments, hence the optimal result is a 120° triple junction. Binary segmentations (b-d) yield the shortest boundary through the center dot, hence any combination of the binary results leaves an unassigned center triangle (e).

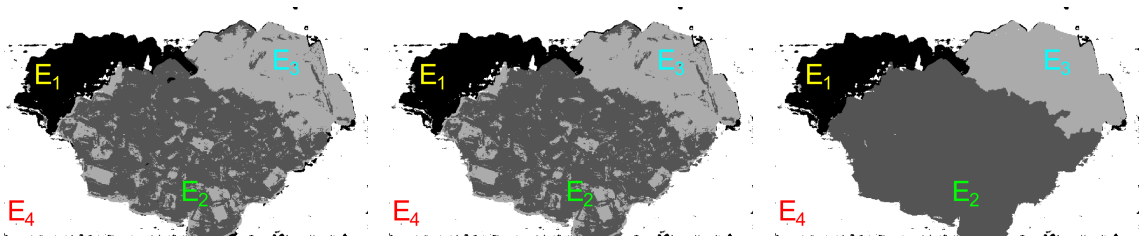
4.4.2 Expansion / Swap Moves

A popular approach for solving labeling problems for $k > 2$ using graph cuts has been presented by Boykov et al. (2001). Starting with an initial labeling, they iteratively minimize the energy by moving pixels between labels. They allow two types of moves:

$\alpha - \beta$ **Swap moves** (cf. Figure 4.6) between two sets E_α and E_β allow any pixel in the sets E_α and E_β to change its label. Minimization is performed until no swap move for any pair of labels yields a smaller energy.

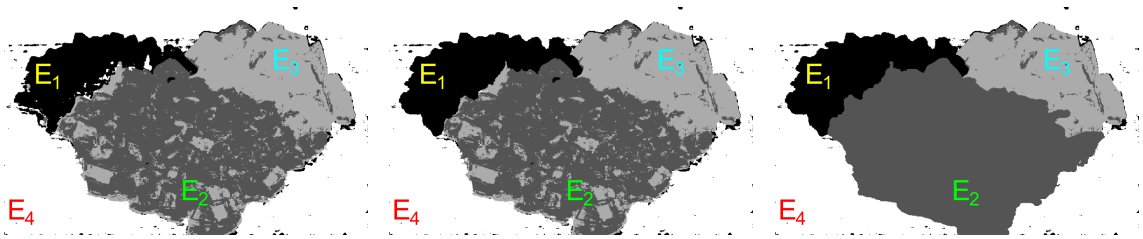
$\alpha-$**Expansion moves** (cf. Figure 4.7) with respect to a label α are moves which allow any pixel to change its label to α. Minimization is performed until no expansion move for any label yields a smaller energy.

Finding the optimal expansion/swap move is performed in a discrete setting, hence the method suffers from metrication errors and bad parallelizability (cf. Section 1.4.3.6). Furthermore, the multi-label problem is again approximated by a sequence of binary problems, and hence suffers similar drawbacks as 1-vs-All segmentation (cf. Figure 4.8).



(a) (b) (c)

Figure 4.6: Swap moves: Given the initial labeling in (a), in (b) pixels in E_1 and E_2 are allowed to swap labels. In (c) pixels of E_2 and E_3 are allowed to swap. This process is continued until no swap decreases the segmentation energy.



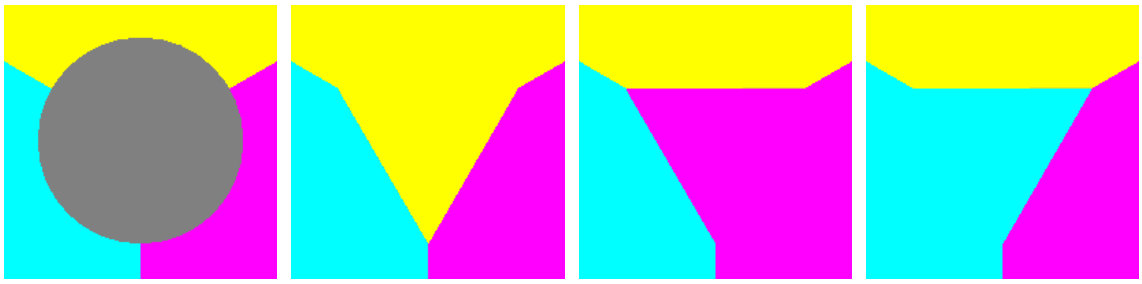
(a) (b) (c)

Figure 4.7: Expansion moves: Given the initial labeling in (a), in (b) all pixels are allowed to change to E_1 (i.e. E_1 is expanded). In (c) all pixels are allowed to change to E_2. This process is continued until no expansion decreases the segmentation energy.

In their paper, Boykov et al. prove bounded optimality within a factor of 2 for the α-expansion algorithm, i.e. the energy of the minimum obtained by their method is at most twice the energy of the global minimum. A continuous version of the $\alpha-$expansion algorithm with the same bounded optimality has been presented by Olsson et al. (2009).

4.5 Convex Approximations

As stated in the previous section, the computation of the global minimizer of the multi-label Potts model is NP-hard. The sequential $\alpha-$expansion algorithms have a bounded optimality with a factor of 2. However, there exist methods with better optimality bounds: Dahlhaus et al. (1992) present an approximation to the multiway-cut problem



(a) (b) (c) (d)

Figure 4.8: Approximating a multi-label segmentation problem by a series of binary segmentations leads to problems with non-descriptive posteriors similar to those observed in 1-vs-All segmentation: In the toy example from Figure 4.5 (a), the result of a segmentation by expansion or swap moves depends solely on the initial labeling of the gray region, hence any result of (b-d) would be possible.

with an optimality bound of $2 - \frac{2}{k}$. This means, that for the binary case $k = 2$, the global optimum can be found. The more labels are involved in the segmentation process, the worse the optimality bound gets.

An approximation in a spatially continuous setting has been presented by Pock et al. (2009). In their paper, they state the Potts energy as

$$E_{POTTS} = \frac{1}{2} \sum_{l=0}^{k-1} Per(E_l; \Omega) + \sum_{l=0}^{k-1} \int_{E_l} f_l(x) dx. \tag{4.11}$$

In this energy, the regularization term includes the function $Per(E_l; \Omega)$, which measures the perimeter of the set E_l. The binary Potts penalization function $d(a, b) = k \cdot [a \neq b]$ counts the number of neighboring pixels exhibiting different labels. For a sufficiently large number of neighborhood relations, this approximates the perimeter of the set (or the area of the interface in three dimensions).

In the binary case, Chan et al. (2006) minimize the Potts energy via the functional

$$\int_\Omega |D\theta| + \int_\Omega (1 - \theta(x)) \cdot f_0(x) + \theta(x) \cdot f_1(x) dx. \tag{4.12}$$

In this functional, θ is a binary labeling and f_0 and f_1 represent the data term. With $D\theta$ denoting the distributional derivative of the labeling, $\int_\Omega |D\theta|$ is the Total Variation of θ. The distributional derivative allows to express the Total Variation for non-smooth

functions, reversely if θ was a smooth function, $\int_\Omega |D\theta|$ would equal $\int_\Omega |\nabla\theta|$. Note that this model employs the L1-norm as penalizer in the regularization term, which is legitimate, as the Potts penalizer is identical to the L1-norm for binary functions. In other words, the perimeter of the set $Per(E_l;\Omega)$ is approximated equally by the Potts penalizer and the L1-norm for binary functions. The benefit of this substitution is, that it makes the energy functional convex, which allows for computing a global minimizer by relaxation.

In their paper, Pock et al. first define the labeling function $u : \Omega \to \{0, 1, \ldots, k-1\}$, which is represented by k binary functions $(\theta_0(x), \theta_1(x), \ldots, \theta_{k-1}(x))$. The relation between the binary functions and the labeling function is given by

$$\theta_l(x) = \begin{cases} 1: & \text{if } u(x) \geq l \\ 0: & \text{else} \end{cases} \tag{4.13}$$

and

$$u(x) = \sum_{l=0}^{k-1} \theta_l(x). \tag{4.14}$$

In order to find out whether a pixel is part of the set E_l, the following relation can be employed:

$$\theta_l(x) - \theta_{l+1}(x) = \begin{cases} 1: & \text{if } u(x) = l \\ 0: & \text{else,} \end{cases} \tag{4.15}$$

with $\theta_0 = 1$ and $\theta_k = 0$. With these relations, the Potts energy can be reformulated as

$$E_{POTTS} = \frac{1}{2} \sum_{l=0}^{k-1} \int_\Omega |D\theta_l| + \sum_{l=0}^{k-1} \int_\Omega (\theta_{l+1}(x) - \theta_l x) \cdot f_l(x) dx. \tag{4.16}$$

However, the regularization term in this equation is not correct, as the sum over the labels leads to interfaces between different labels being counted differently, e.g. the boundary between sets E_2 and E_5 would be counted three times. Pock et al. overcome this problem by employing the dual formulation of the Total Variation as regularization term, and suppress multiple boundary counts by adding constraints on the dual variables: They approximate the perimeter of the set E_l by the dual of the Total Variation

$$\sum_{l=0}^{k-1} Per(E_l;\Omega) = \sup_{\xi \in \mathcal{K}} \left\{ \sum_{l=0}^{k-1} - \int_\Omega \theta_l div \xi_l \right\}, \tag{4.17}$$

and constrain the dual variables to the set

$$\mathcal{K} = \left\{ (\xi_0, \xi_1, \ldots, \xi_{k-1}), \left| \sum_{l_1 \leq l \leq l_2} \xi_l(x) \right| \leq 1, \forall x \in \Omega, 1 \leq l_1 \leq l_2 \leq k \right\}. \tag{4.18}$$

Pock et al. show that the set $\mathcal{K}$ is convex, and compute a global optimum by continuous relaxation. As the thresholding technique employed in the binary case $k = 2$ is not applicable for multiple labels, the global optimum of the relaxed problem is not the global optimum of the binary problem. However, Pock et al. give a tight bound on the optimality: The energy of the thresholded solution $E_{POTTS}(\mathbf{1}_{u^* \geq s})$ cannot be lower than the energy of the global minimum of the relaxed problem $E_{POTTS}(u^*)$. The true binary minimizer of the problem $E_{POTTS}(u)$ cannot produce a lower energy than the relaxed problem, but in turn is at least as good as the thresholded solution, i.e. $E_{POTTS}(u^*) \leq E_{POTTS}(u) \leq E_{POTTS}(\mathbf{1}_{u^* \geq s})$. Thus the global optimum lies somewhere in between the optimum of the relaxed problem and the thresholded solution.

Other approximations of the Potts model in the continuous domain have recently been presented by Zach et al. (2008) and Lellmann et al. (2009). Note that there is no proven optimality bound for any of these three methods yet. However, in their paper, Pock et al. show qualitative dominance over the other two approaches.

4.6 Incorporation into the Framework

Based on the considerations presented so far, we employ the labeling algorithm of Pock et al. (2009) in our interactive segmentation framework. In contrast to the version published in the original paper, we employ a GPU-based implementation which allows for the additional incorporation of an edge weight in the regularization term. This modification leads to the segment borders being attracted by image gradients (cf. weighted Total Variation (Bresson et al., 2007)). Instead of regularizing with the energy

$$\int_\Omega |D\theta|, \tag{4.19}$$

this extension uses

$$\int_\Omega g(x) \cdot |D\theta| dx, \tag{4.20}$$

with g denoting the edge indicator

$$g(x) = e^{-\alpha \nabla I}, \tag{4.21}$$

with α a scalar weighing factor. In the dual formulation, this leads to additional constraints on the dual variables. With λ denoting a scalar weight between regularization and data term, the minimized energy amounts to

$$\frac{1}{2} \sum_{l=0}^{k-1} \int_{\Omega} e^{-\alpha \nabla I} \cdot |D\theta_l| + \lambda \sum_{l=0}^{k-1} \int_{\Omega} (\theta_{l+1}(x) - \theta_l x) \cdot f_l(x) dx. \tag{4.22}$$

In terms of parameters, three parameters are needed to obtain a segmentation result

$$u(x) = \sum_{l=0}^{k-1} \theta_l(x) \tag{4.23}$$

given the image I and the posterior probabilities $\mathbf{f} = \{f_0, f_1, \ldots, f_{k-1}\}$: The edge weight α, the weighing between the regularization term and the data term λ as well as the number of iterations for the minimization algorithm.

4.7 Conclusion

In this chapter, we have reviewed binary and multi-class labeling algorithms in the discrete as well as continuous setting (cf. the overview in Table 4.3).

Scenario	**Discrete Method**	**Continuous Method**
Two-label ($k \in \{0,1\}$) (any binary labeling task)	Boykov and Jolly (2001) global optimum	Bresson et al. (2007) global optimum
Multi-label ($k \geq 2$) linearly ordered label sets (stereo, denoising)	Ishikawa (2003) global optimum	Pock et al. (2008) global optimum
Multi-label ($k \geq 2$) expansion/swap moves (any labeling task)	Boykov et al. (2001) bounded optimality with factor 2	Olsson et al. (2009) bounded optimality with factor 2
Multi-label ($k \geq 2$) convex approximations (any labeling task)	Dahlhaus et al. (1992) bounded optimality with factor $2 - \frac{2}{k}$	Pock et al. (2009) bounded optimality with no proved factor

Table 4.3: Binary and multi-class labeling algorithms in the discrete and continuous setting as well as their optimality.

As there is no globally optimal way to solve the labeling problem for the unordered label space occurring in segmentation (i.e. the labels have only symbolic character), we have to employ a method calculating an approximation of the global minimizer. We have shown, that the popular family of expansion/swap move algorithms have a systematic deficiency for weak unary potentials and also exhibit a large bound on the optimality. Better results can be obtained with convex approximations, e.g. the discrete approximation of the multiway-cut problem by Dahlhaus et al. (1992) reaches bounded optimality with a factor of $2 - \frac{2}{k}$. In the continuous domain, Pock et al. (2009) show excellent results for segmentation and assume a bounded optimality similar or better to that of Dahlhaus et al. Therefore, in our segmentation framework we employ a fast GPU-implementation of the convex approximation algorithm of Pock et al.

5 Benchmark

The evaluation of the quality of interactive segmentation algorithms is not straightforward. As the interaction in later steps typically depends of the segmentation result of earlier steps, there is no segmentation benchmark with identical behavior for every segmentation algorithm evaluated. Furthermore, there are many different ways of interacting, starting from different scribble types over rectangles or ellipses up to boundary marking methods. Based on these considerations, many interactive segmentation methods have been presented without any comparable quantitative scores (e.g. Bai and Sapiro (2007); Barrett and Cheney (2002); Grady (2006); Santner et al. (2009); Sinop and Grady (2007); Unger et al. (2008); Veksler (2008)). While some methods were evaluated using extensive user studies (e.g. Li et al. (2004); McGuinness and O'Connor (2010); Reese and Barrett (2002)), a recent work of Gulshan et al. (2010) introduces a robot user, which performs automatic interaction based on intermediate results. This method allows for a consistent comparison of interactive segmentation algorithms, however, the compliance of the automated interaction with that of human users is not clear.

In order to evaluate our framework thoroughly, we have created a new benchmark dataset for interactive image segmentation which is described in this chapter. For this dataset, we gathered many images of varying objects, people, animals and scenes. We then let different people provide seed pixels as well as ground truth segmentations corresponding to their interpretation of the images.

5.1 Related Image Segmentation Benchmarks

The most important segmentation benchmarks that have been applied to interactive segmentation are the GrabCut database (Rother et al., 2004) as well as the Berkeley Segmentation Dataset and Benchmark (Arbelaez et al., 2010; Martin et al., 2001). Other popular benchmarks used in image segmentation are the PASCAL VOC dataset (Everingham et al., 2010) and the LabelMe dataset (Russell et al., 2008). In this section, we describe these benchmark datasets and discuss why they are not suited for assessing interactive multi-label segmentation algorithms.

5.1.1 LabelMe Database

The LabelMe database by Russell et al. (2008) consists of images labeled by arbitrary people via a web annotation tool (see Figure 5.1 for an example). During annotation, the users have to identify objects within images, loosely draw their outline with polygons and freely choose a descriptive name for the object (e.g. car, building, tree, person, window, road, sidewalk, sign, sky etc.). As the LabelMe benchmark was mainly de-

Figure 5.1: An exemplary image of the LabelMe database with object annotations.

signed for object recognition, the annotations are often very imprecise and overlap each other: E.g.'wheels' and 'cars' form different object classes, however a pixel on a wheel is often annotated as being within a 'wheel' object and a car 'object' at the same time. Furthermore, the benchmark dataset does not contain seed pixels for interactive segmentation.

5.1.2 PASCAL VOC Database

The PASCAL Visual Object Classes (VOC) Challenge is a popular object recognition competition on a large database, which is extended every year. Since 2007, the challenge includes a segmentation benchmark, where the task is to assign every image pixel one out of 20 classes: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, and TV/monitor. The ground truth segmentations provide a pixel-wise labeling into either one of these 20 classes, a background class and an unlabeled class (see Figure 5.2). Compared to the groundtruth segmentations of the LabelMe dataset, the segmentations of the PASCAL VOC dataset are very precise and do not contain overlapping segments. However, the limitation to 20 specific object classes as well as the lack of seed pixels are drawbacks w.r.t. evaluating the performance of interactive segmentation algorithms.

5.1.3 GrabCut Database

The GrabCut database (Rother et al., 2004) is also known as MSRC database (not to be confused with the MSRC 21-Class Database (Shotton et al., 2006, 2009) for supervised image segmentation). This database has been widely used for evaluating interactive segmentation algorithms, recently e.g. in (Ding and Yilmaz, 2010; Duchenne et al., 2008; Friedland et al., 2005; Gulshan et al., 2010; Lempitsky et al., 2009; Price et al., 2010; Zhang et al., 2010). The database consists of 50 images, where 20 images are taken from the BSDS300 dataset. For every image, a ground truth segmentation is given as well as a rough labeling based on the object boundary, from which seed pixels are deducted. See Figure 5.3 as an example: The rough labeling divides the pixels into a set which is used for foreground model training and a set for background model training. A large amount of pixels (typically at the border) are not considered. Finally, the remaining pixels are marked as unknown, and need to be assigned by the segmentation algorithm based on the generated foreground and background model.

The GrabCut database has two drawbacks: First, as also mentioned by Price et al.

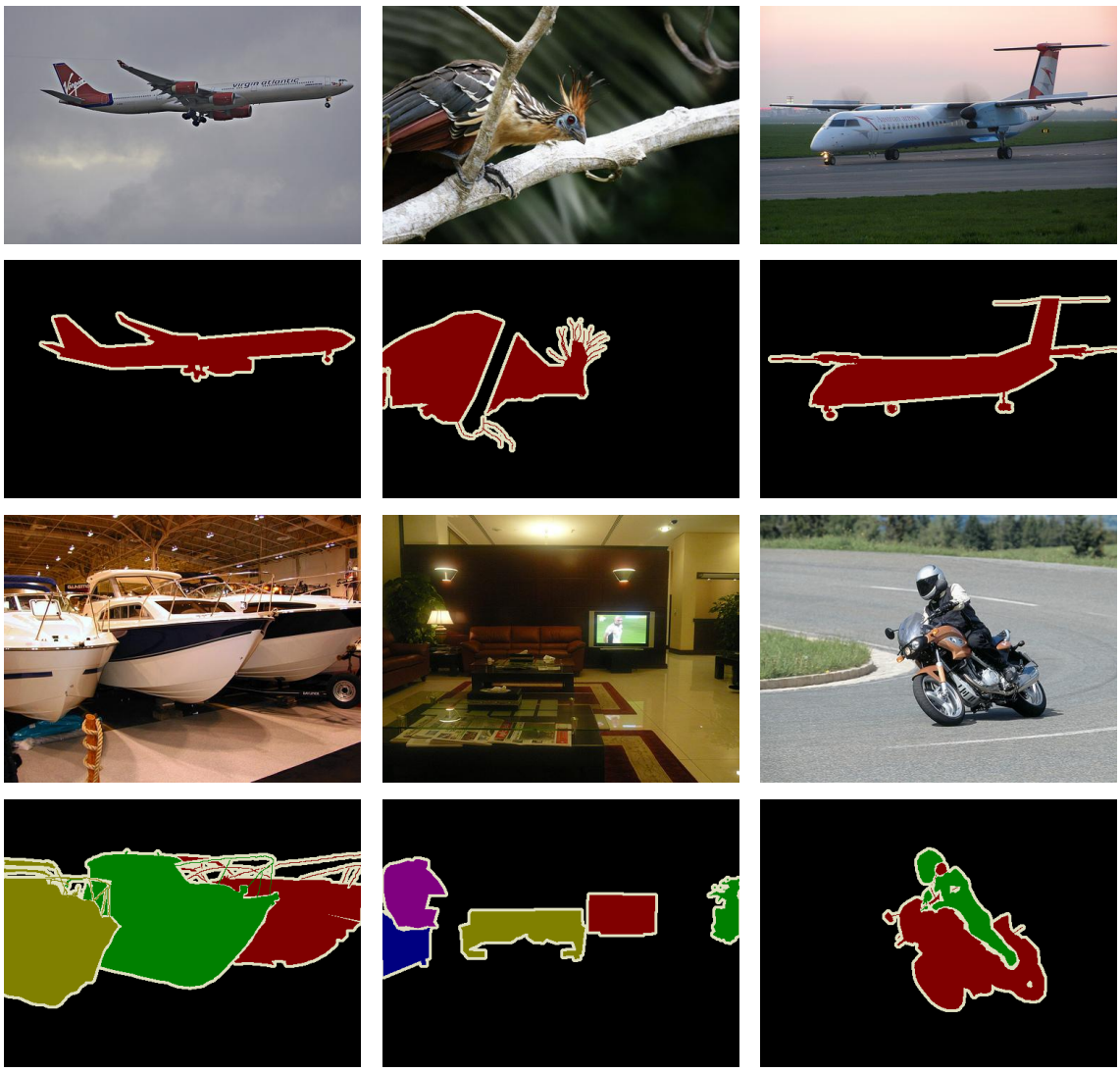


Figure 5.2: Exemplary images of the PASCAL VOC segmentation dataset together with groundtruth segmentations.

(2010) and Duchenne et al. (2008), it assumes to have a rough estimate of the object boundary. For methods working with scribble based interaction such as our framework, this estimate does not reflect the type of user interaction. Furthermore, typically much more interaction is needed to generate an approximate object boundary. Second, the framework is only handling the binary segmentation case i.e. foreground-background segmentation.

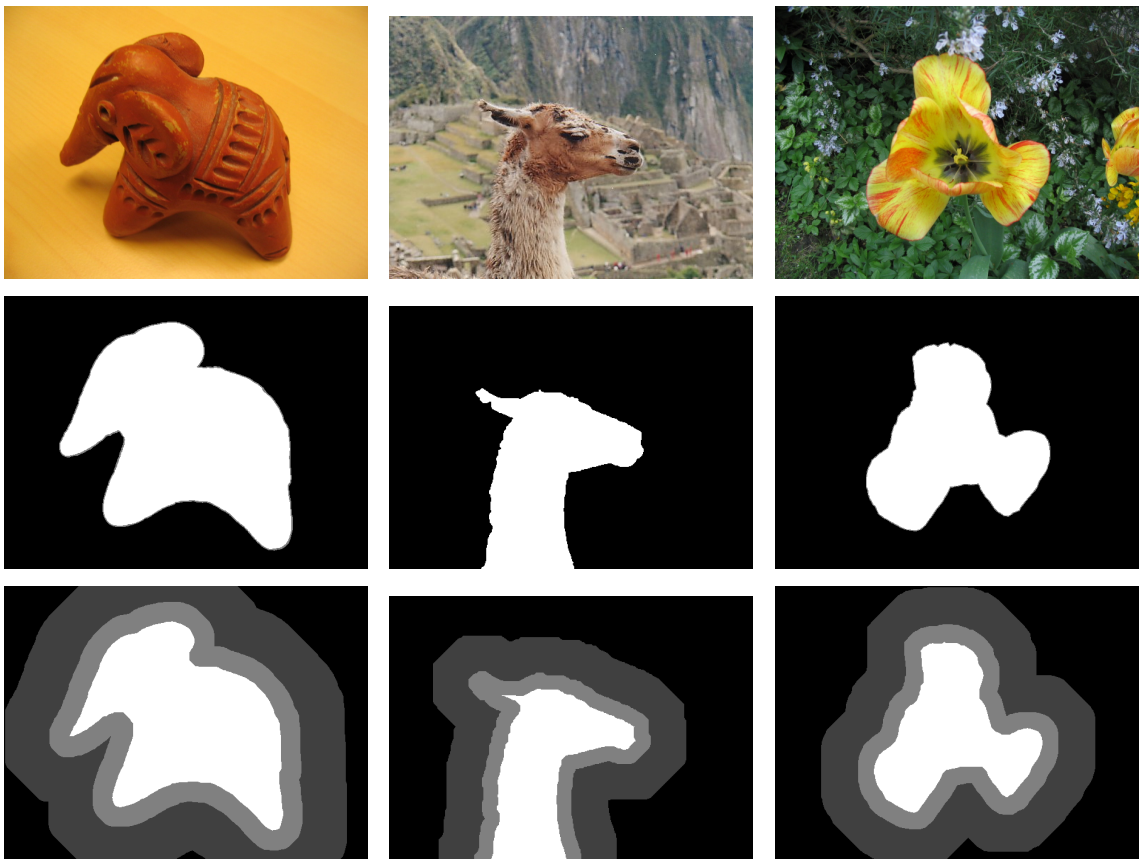


Figure 5.3: Three example images from the GrabCut database (first row) together with exemplary ground truth segmentations (second row). The third row shows the seeds for the segmentation algorithms: White pixels (intensity = 255) denote pixels for foreground model generation, dark gray pixels (64) are used for background model generation. While black pixels (0) are also background, they are not used for background model training. Light gray pixels (128) are considered unknown and have to be assigned correctly by the segmentation algorithm.

5.1.4 Berkeley Segmentation Dataset and Benchmark

The Berkeley Segmentation Dataset and Benchmark was initially presented by Martin et al. (2001) as BSDS300, consisting of 300 color images. Recently, the dataset was extended by 200 images to form the BSDS500 database (Arbelaez et al., 2010). For every image in the dataset, there exist multiple ground truth segmentations manually drawn by different users (cf. Figure 5.4). These segmentations (in average, there are five segmentations per image) differ in number of segments as well as position of the segment boundaries. Evaluation scores such as Variation of Information, Probabilistic Rand Index

or Segmentation Covering are employed to compare results of different segmentation algorithms, however, there is in general no agreement on how to evaluate results of this benchmark.

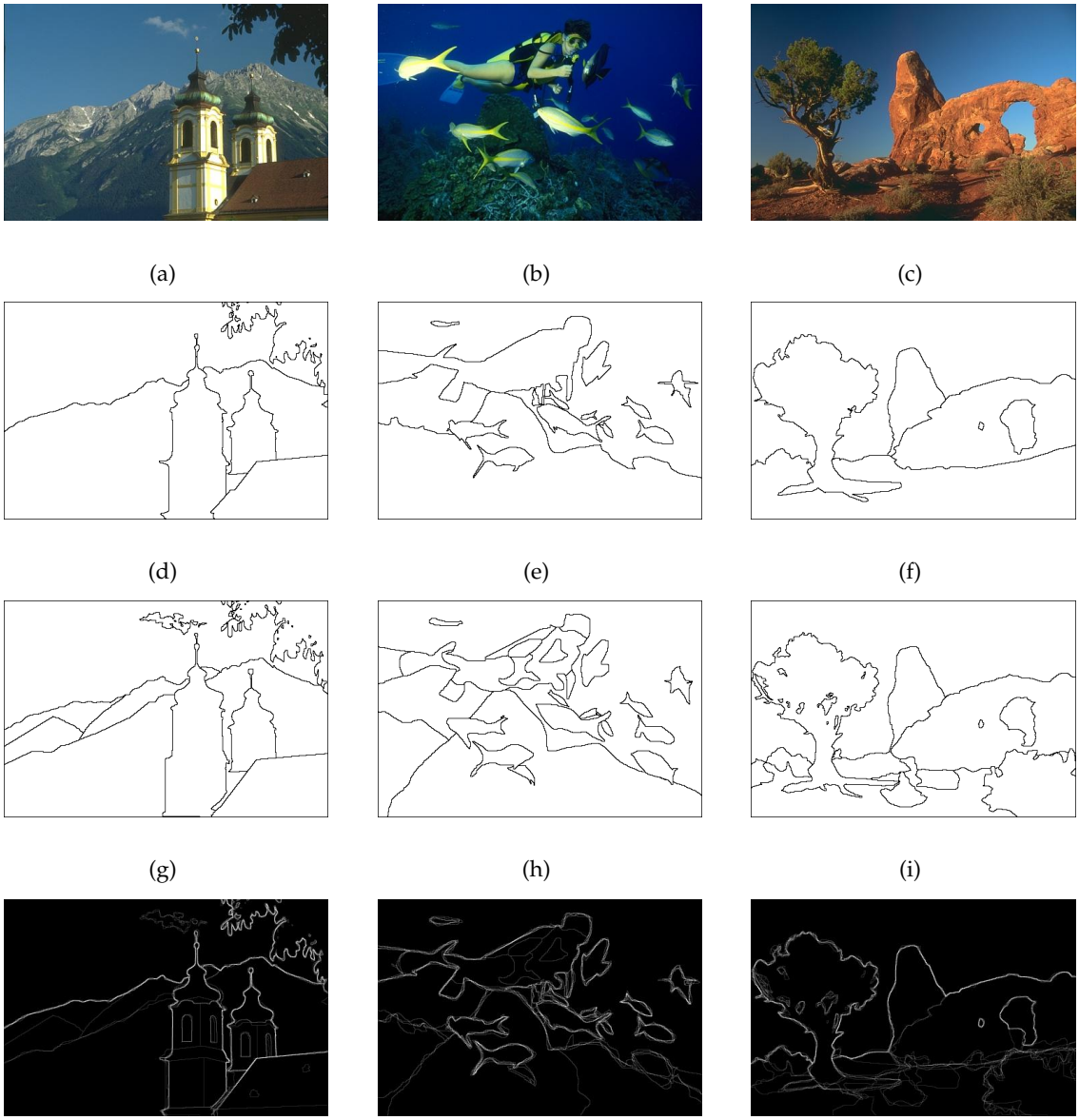


Figure 5.4: Three example images from the BSDS300 dataset (a-c) together with exemplary ground truth segmentations (d-f and g-i). Note that the ground truth segmentations are labeled by different users and therefore represent different interpretations of the image: (j-l) show the boundary probability accumulated from all segmentations.

While this benchmark dataset has been widely applied for unsupervised image segmentation algorithms, it is only partially suited for interactive segmentation methods: The benchmark consists only of ground truth segmentations but no seed pixels. McGuinness and O'Connor (2010) bypass this problem by describing the quality of an interactive method by the effort needed to segment a given object according to the ground truth. In their work, they conduct a series of user experiments where they measure the time a user needs to perform a specific segmentation as well as the accordance with the ground truth. Arbelaez and Cohen (2008) generate a single seed pixel per segment by finding the point with the largest Euclidean distance from the ground truth segment boundary. While this method would be reproducible and thus comparable, it does not reflect the way seed pixels are generated by human users.

5.2 IcgBench

As explained in the previous section, there is no benchmark dataset for proper quantitative evaluation of the performance of our framework. In this section, we therefore describe the design and generation of a new publicly available dataset. To alleviate the shortcomings of the BSDS300 and the GrabCut datasets, our new benchmark exhibits the following properties:

Multi-label The benchmark exhibits segmentations with several regions in a single image. Furthermore, it can be reduced to evaluating foreground/background segmentations by translating multiple labels to sequential binary problems.

Scribbles The seed pixels are given by scribbles drawn into the region to segment rather than by geometric primitives such as ellipses or rectangles approximating the object boundary. Drawing scribbles to generate seeds has been chosen because this type of interaction seems to be more common in recent literature as well as in commercial products.

Interactivity Interactivity is a property that can hardly be covered by automatic benchmarks. The closest approach is the robot user by Gulshan et al. (2010), that places correcting seeds automatically in the largest mislabeled region of the image until an acceptable result is obtained. While being an intuitive approach, this behavior does still not fully reflect the circumstances of a human correcting intermediate segmentation results: The position, number and shape of the correcting seeds drawn by a human operator are typically not defined. Moreover, the definition

of an acceptable result might vary greatly. In general, there is no simple way to approximate the 'average' human user of an interactive segmentation tool automatically. Therefore, we restrict our new benchmark to evaluating the quality of the initial segmentation only. The question how close an algorithm can get to a desired segmentation by adding more and more correcting scribbles (cf. Gulshan et al.) is not covered.

5.2.1 Structure

We wrote a simple annotation tool and let several different users annotate images according to their own interpretation. See Figure 5.5 for an example: Here, the image is interpreted as a three-label segmentation problem, where the surfer and the surfboard are separate objects, and the water is background. With the tool, the user specified not only the ground truth segmentation of the image, but also the seeds he would give to an interactive tool in order to obtain the desired segmentation. If one region is given no seeds, it is interpreted as background. In that case, background seeds are randomly sampled from the background region such that the number of background seed pixels equals the average number of foreground object seed pixels.

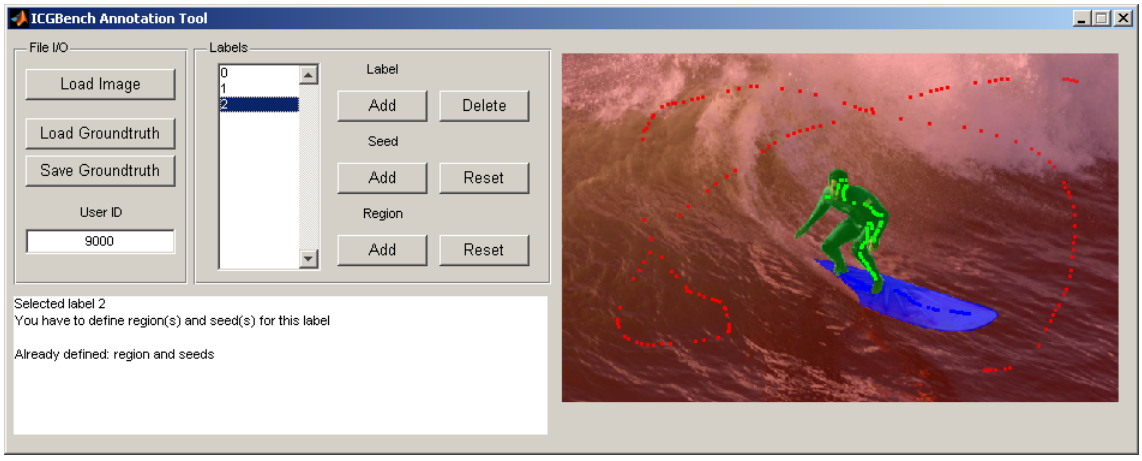


Figure 5.5: The annotation tool for creating our interactive segmentation benchmark: Given an image, the user should specify not only the ground truth segmentation according to his own interpretation of the image, but also seeds for every label in the image. Note that the seed pixels are dilated here for better visibility.

Instead of letting users provide both seeds and segmentation groundtruth, we could have reused the groundtruth segmentations of the GrabCut or BSDS300 / BSDS500

datasets. In that case, the users would only have needed to place seed pixels for existing image segments. However, this would have led to a bias towards the segment interpretations of the respective benchmarks. In the natural interactive segmentation process, the user selects the image himself, makes up his mind on what the segments in the specific image look like and places seeds accordingly. In the design of the IcgBench dataset, we wanted to get as close as possible to this process by imposing only a few restrictions: The users were allowed to freely select the images they wanted to annotate, therefore several images are annotated by several people, while some of the images are not annotated at all (see Table 5.1 for the full statistics). Furthermore, the users were allowed to collect and add images themselves freely, only the resolution of the images was defined for easier display and print.

Annotations per Image	0	1	2	3	4	5
Occurence	85	79	59	16	3	1
Percentage	35.0	32.5	24.3	6.6	1.2	0.4

Table 5.1: As users were allowed to choose images freely, the number of annotations per image varies: While 85 images were never annotated, 79 where annotated once and 79 annotated more than once.

At the time of writing, we gathered 262 seed-groundtruth pairs labeled by eight different users. Table 5.2 shows the number of annotations per user: More than half of the seed-groundtruth pairs were annotated by only two users. The images annotated by these two users have a large influence on the whole benchmark, however, note that both users had little to no knowledge on interactive segmentation before the annotations.

User ID	9000	9010	9020	9025	9035	9040	9050	9060
Number of Annotations	30	16	27	17	5	3	97	67
Percentage	11.4	6.1	10.3	6.5	1.9	1.1	37.0	25.6

Table 5.2: Number of annotated images per user. Note that the users 9050 and 9060 annotated more than half of the dataset, however, both users had little to no knowledge on interactive segmentation before the annotations.

Example images of our dataset together with seed - ground truth pairs are shown in Figures 5.6 and 5.7.

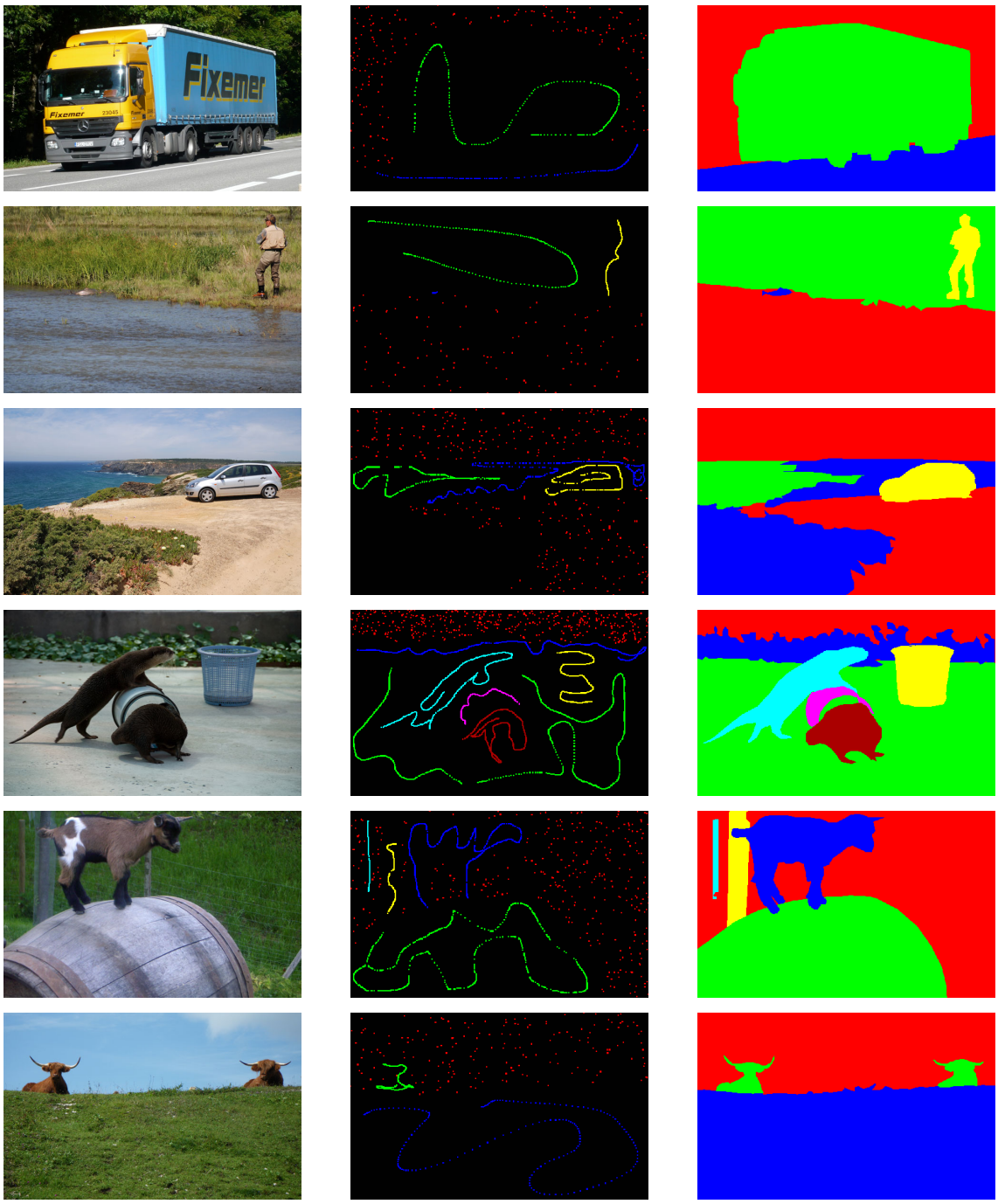


Figure 5.6: Exemplary seed - groundtruth pairs of our dataset IcgBench. Note that the images showing seed pixels (center column) have been dilated for better visibility.

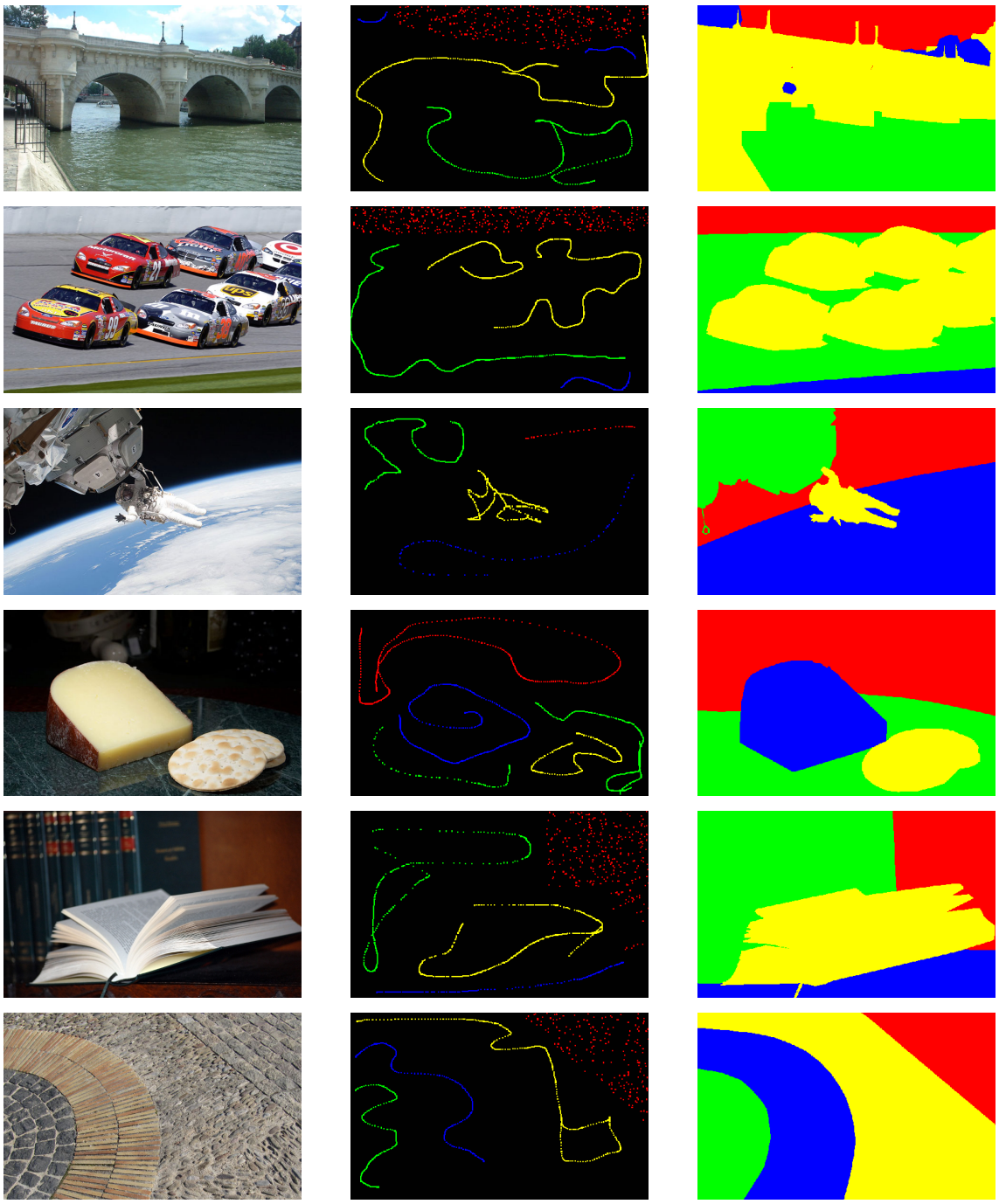
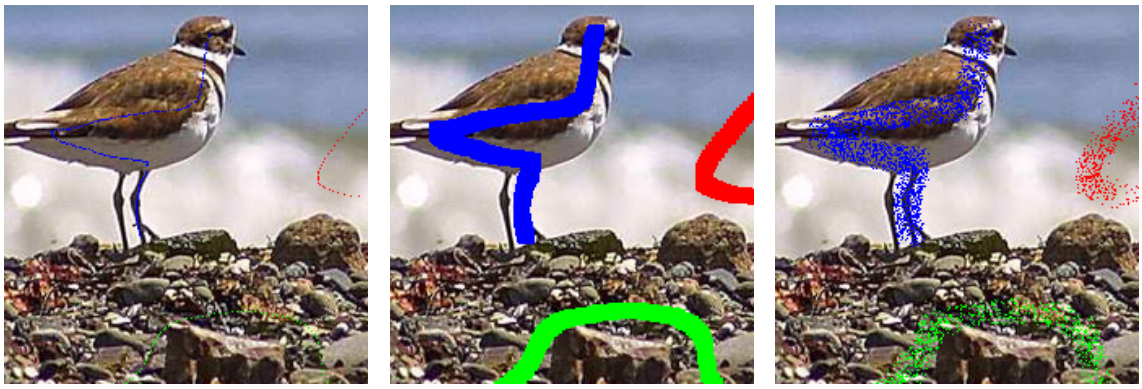


Figure 5.7: Exemplary seed - groundtruth pairs of our dataset IcgBench. Note that the images showing seed pixels (center column) have been dilated for better visibility.

5.2.2 Tooltip

As stated in the beginning of this section, our benchmark is based on seed points specified by scribbles. In our annotation tool, we therefore recorded the mouse path the user took when drawing seed pixels (See Figure 5.8(a)). As different algorithms might require larger training sets than those provided by the mouse paths themselves, anyone using our benchmark may use a different tooltip based on the recorded mouse path: To increase spatial support as well as the number of seed pixels, the user can overlay the mouse path with a solid brush with a certain radius (Figure 5.8(b)). Using a solid brush increases the number of seed pixels dramatically, but adds a lot of redundant information to the training set. Another possibility is the use of a spraygun-like brush (5.8(c)), where only a random subset of the pixels within a given radius from the mouse path are taken into account.



(a) (b) (c)

Figure 5.8: Our benchmark dataset allows for different ways of generating seed pixels: The seeds are specified only by the mouse path the user took while marking them (a). Algorithms, that need more seed pixels, are free to employ solid (b) or spraygun-like (c) brush tooltips with varying radius upon this mouse path.

5.2.3 Evaluation Multi-label Case

While many of the quality measures of established segmentation benchmarks describe the accuracy of the segment boundaries only, we want that the resulting segmentations are close to the ground-truth labeling of the user, such that the amount of further interaction to yield the desired segmentation is as small as possible. For the evaluation

of the GrabCut dataset (Rother et al., 2004), the error is computed as the average of the relation

$$\epsilon = \frac{\text{no. misclassified pixels}}{\text{no. pixels in unclassified region}} \tag{5.1}$$

over all images. A similar measure for multiple labels can be expressed by the accuracy score

$$accuracy = \frac{\text{no. correctly classified pixels}}{\text{total no. pixels}}. \tag{5.2}$$

However, this quality measure does not take into account the size of a region: Figure 5.9 shows two examples of our benchmark together with groundtruth labeling and a fictive segmentation result: In the first example (plane), the fictive result differs from the ground truth segmentation only by small inaccuracies at the region borders. This result yields a good accuracy score of 0.963 (i.e. more than 96 % of the pixels are correctly classified). The second example shows two cows on a meadow. In the fictive segmentation result, the cows are simply removed, which we think is a much worse result compared to the border inaccuracies of the previous example. However, as the cows occupy only very small fraction of the image, the accuracy of this fictional result (0.968) is even higher than the accuracy of the previous example. Hence, this notion of accuracy is not well suited to assess the quality of multi-label segmentation results.

For our dataset, we therefore chose the arithmetic mean of the Dice evaluation score (Dice, 1945) over all segments. This score relates the area of two segments $|E_1|$ and $|E_2|$ with the area of their mutual overlap $|E_1 \cap E_2|$ such that

$$dice(E_1, E_2) = \frac{2|E_1 \cap E_2|}{|E_1| + |E_2|}, \tag{5.3}$$

where $|\cdot|$ denotes the area of a segment. Given GT_i the ground-truth labeling for the i-th of N segments, the evaluation score for one image amounts to

$$score = \frac{1}{N}\sum_{i=1}^{N} dice(E_i, GT_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{2|E_i \cap GT_i|}{|E_i| + |GT_i|} \tag{5.4}$$

The overall benchmark score is computed as the average of the results of all images.

Figure 5.9 also states the average Dice score for the exemplary fictional segmentations: While the plane example with the border inaccuracies yields a good score of 0.925, the example with the cows gets a significantly worse result (0.654).

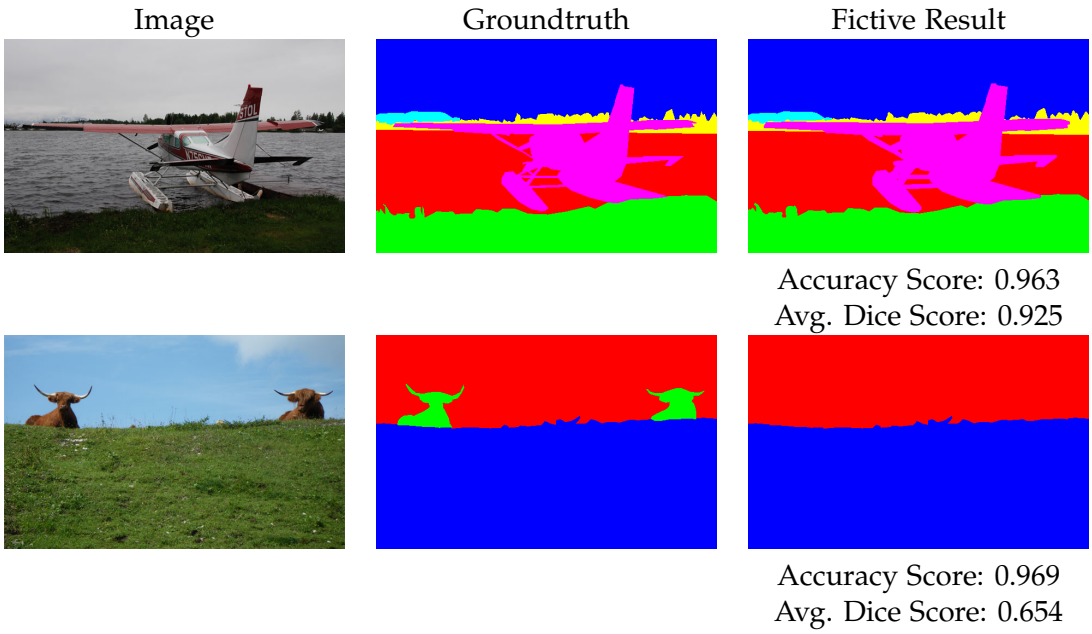


Figure 5.9: Relating the number of correctly classified pixels to the total number pixels (denoted as accuracy score here) does not take into account the relative size of the regions: The example with the plane yields about the same accuracy scores as the (perceptually much worse) example with the cows. Therefore, we employ the average Dice score over all segments in the labeling to assess the quality of a segmentation result.

5.2.4 Evaluation Binary Case

As stated before, we want our benchmark to be applicable to binary segmentation algorithms as well. Therefore, we simply let the user solve for every single N-label problem N binary segmentation problems in a one-versus-all manner. The score is then computed similarly to the multi-label case with

$$score = \frac{1}{2}(dice(E_1, GT_1) + dice(E_2, GT_2)) = \frac{|E_1 \cap GT_1|}{|E_1| + |GT_1|} + \frac{|E_2 \cap GT_2|}{|E_2| + |GT_2|}, \tag{5.5}$$

which is averaged over all binary segmentation problems for all images.

6 Experiments

In the previous chapters, we have described the essential parts of our interactive segmentation framework, as well as a dataset for its evaluation. In this chapter we first present the framework itself, and later employ our benchmark to assess its behavior with respect to different parameter settings, feature combinations as well as learning algorithms.

In these experiments, we try to adjust our segmentation framework such that it yields high scores on our benchmark at a low overall runtime. Optimizing for high scores on the benchmark does not necessarily imply that a parameter is optimal w.r.t. a certain image. Contrarily, there will always be a segmentation problem where a specific parameter setting produces bad results and the framework needs to be readjusted to a certain extent. However, based on the assumption that the benchmark covers a certain variability of segmentation problems and different user interaction patterns, the framework adjustments performed in this chapter should provide a good starting point for many interactive image segmentation problems.

6.1 Graphical User Interface

Our interactive segmentation framework is implemented as a full-featured graphical user interface, that allows for convenient evaluation of all components (see Figure 6.1 for a screenshot). The user interface allows for the following actions:

File I/O The user can load and save images and segmentation results, as well as region models. Storing region models allows for training a region model on one image, and evaluating it on any other image (e.g. in a batch processing mode).

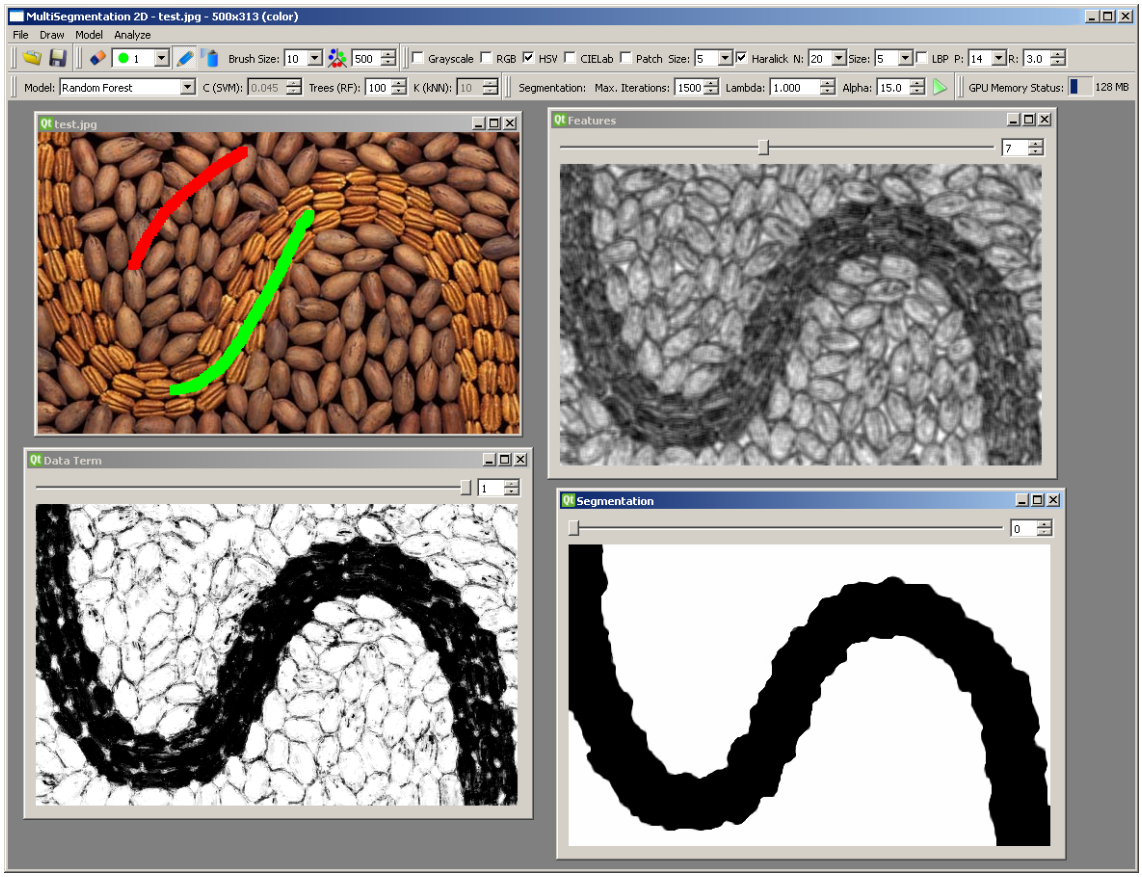


Figure 6.1: The graphical user interface of our framework. The sub-windows (starting from the top left) show the input image together with the user scribbles, the extracted image features (here the fifth Haralick texture feature is depicted), the final region model as well as the segmentation.

Scribbles The user can draw scribbles with two different brushes: A solid brush and an airbrush, both with adjustable radius. Furthermore, random scribbles can be assigned based on intermediate segmentations in order to refine the segmentation result.

Feature Extraction The user can select and parametrize several image representations and local features: Grayscale, RGB, HSV, CIELAB, Patches, Haralick texture features as well as Local Binary Patterns. (cf. Chapter 2).

Model Generation The user can select and parametrize different learning algorithms for the region model generation. This includes Random Forests, Linear Support Vector Machines and the k-Nearest Neighbors algorithm (cf. Chapter 3).

Segmentation Finally, the user can adjust the segmentation algorithm as explained in Section 4.6.

Display The user interface displays not only the input image, but optionally also every intermediate step of the segmentation algorithm (cf. Figure 6.1).

The user interface is written in C++ using the open source framework Qt. Image I/O is performed with the OpenCV libraries.

6.2 Segmentation Parameters

We start with evaluating the influence of the parameters of the segmentation algorithm. As stated in Section 4.6, the algorithm has three parameters: The edge weight α, the regularization weight λ as well as the number of iterations.

6.2.1 Number of Iterations

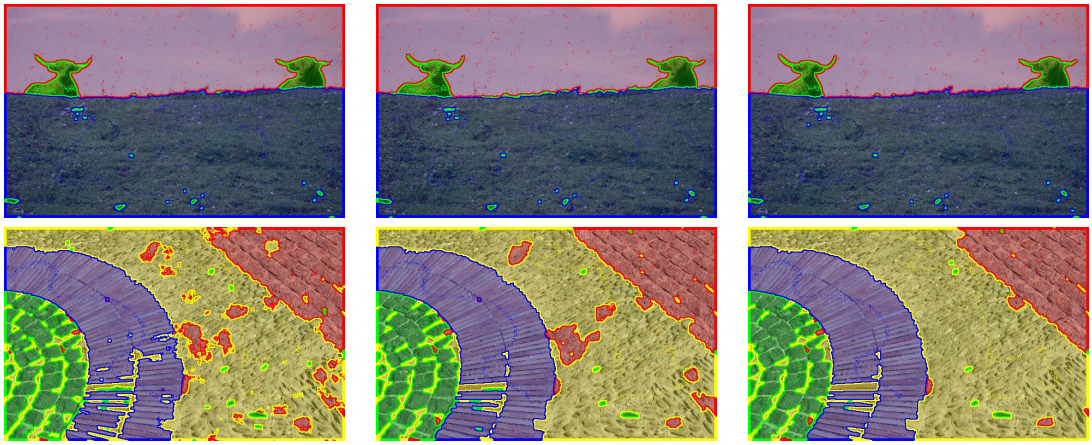
Pock et al. (2009) showed how the convergence of their minimization algorithm can be estimated by observing the gap between the primal and dual energy, which is however costly to compute. Therefore, the algorithm is simply run for a given number of iterations. In order to find out how many iterations need to be performed we conduct the following experiment: We keep the other segmentation parameters fixed ($\lambda = 1.0$ and $\alpha = 15$), perform a sweep over the number of minimization iterations (n_{iter}) and compute the score on our benchmark. As image representation, we employ RGB and 5×5 patches, and as region model we use Random Forests with 200 trees. The seed scribbles provided by the benchmark are used as a mouse path for an airbrush with radius 7. Table 6.1 shows the results as well as the overall runtime for the whole benchmark: According to these results, the benchmark score is stable for $n_{iter} \geq 100$.

n_{iter}	10	20	50	100	200	500	1000	2000	5000
Performance	0.829	0.835	0.837	0.839	0.838	0.838	0.837	0.839	0.839
Runtime [s]	1169	1164	1211	1190	1227	1321	1497	1851	2975

Table 6.1: Benchmark results for varying number of segmentation algorithm iterations.

This can also be observed with a qualitative evaluation of the resulting segmentations: For most of the images, the segmentation results change only minimally after 100 minimization iterations. However, few images need longer to converge to a stable

result (see Figure 6.2). In order to minimize the influence of this parameter, we employ $n_{iter} = 750$ for the remaining experiments in this chapter.



$n_{iter} = 20$ $n_{iter} = 200$ $n_{iter} = 2000$

Figure 6.2: The convergence of the segmentation algorithm is data-dependent: The columns show results after 20, 200 and 2000 iterations respectively. While the segmentation results are visually stable for the first image, they vary largely for the second image.

6.2.2 Regularization and Edge Weight

A stated in Section 4.6, the segmentation energy employed in our framework

$$\frac{1}{2}\sum_{l=0}^{k-1}\int_{\Omega} e^{-\alpha\nabla I}\cdot|D\theta_l| + \lambda\sum_{l=0}^{k-1}\int_{\Omega}(\theta_{l+1}(x)-\theta_l x)\cdot f_l(x)dx. \tag{6.1}$$

has two weighing parameters: α, the edge weight controls how strong segmentation boundaries are attracted by image gradients during the minimization. The regularization weight λ performs a trade-off between the regularization term and the data-fidelity term (see Figure 6.3).

Both of these parameters should be adjusted according to a specific segmentation problem, i.e. values that are well-suited for one image might be a bad choice for another image. However, in this experiment, we want to find a setting that works well for most of the images in our benchmark. Therefore, we employ the identical settings as in the previous experiment and perform two parameter sweeps, once over λ and once over α.

Figure 6.4 shows the results for these sweeps. Based on these results, we choose the regularization weight $\lambda = 0.2$ and the edge weight $\alpha = 15.0$.

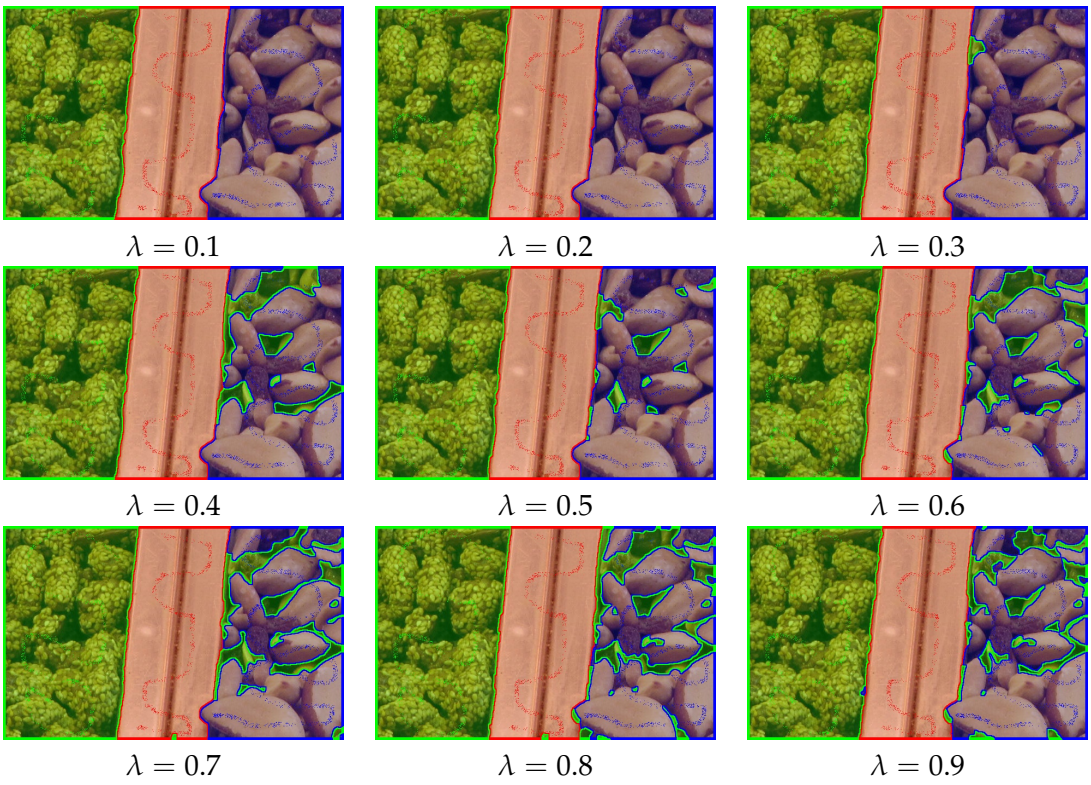
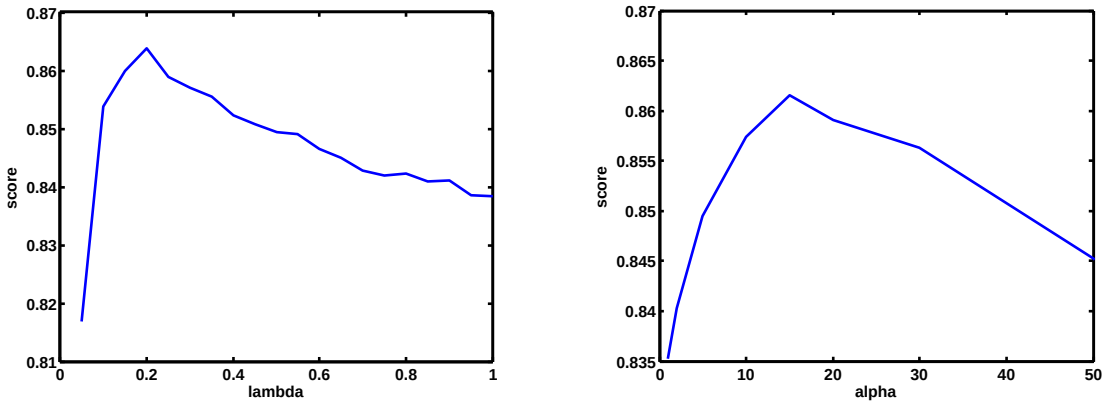


Figure 6.3: Increasing the parameter λ leads to a higher influence for the data fidelity term and a lower influence of the regularization term.



(a) (b)

Figure 6.4: Benchmark results for varying values of the regularization weight λ (a) and the edge weight α (b).

6.3 Features

We now want to evaluate the influence of the employed image features (cf. Chapter 2) to the benchmark score. The segmentation algorithm is parametrized according to the results of the previous experiments (i.e. $n_{iter} = 750$, $\lambda = 0.2$ and $\alpha = 15$).

6.3.1 Color Model

We start with comparing the performance of grayscale images to the performance of images encoded with the different color models stated in Section 2.1. The results are stated in Table 6.2:

Color Model	Gray	RGB	HSV	CIELAB
Performance	0.728	0.877	0.897	0.898
Runtime [s]	1556	967	890	898

Table 6.2: Benchmark results for grayscale values and color representations.

As can be expected, there is a significant performance gap between grayscale images and color images. Note also, that the performance of the HSV and CIELAB models are nearly the same, both slightly better than RGB. This is caused by the high correlation between the three channels in the RGB space. The non-linear transformations from the RGB space to the HSV and CIELAB spaces reduce this correlation significantly, making them better suited for the employed learning algorithm.

This can also be observed when comparing the different runtimes: The computation of the color models takes only a few ten milliseconds per image, hence the differences in runtime origin mostly from different training and evaluation times of the Random Forest. When a learning problem is very ambiguous (which is e.g. the case for segmenting the benchmark images only based on grayscale values), the Randomized Trees have difficulties in finding suitable decision functions in their split nodes. In such a case, the trees grow very large, which causes increased training and evaluation time. The lowest training and evaluation times are reached for the CIELAB and HSV color spaces in this experiment, with RGB as close runner up. Evaluating the whole benchmark based on grayscale images takes nearly twice as long.

6.3.2 Textural Features

In Chapter 2, we have described grayscale patches, Haralick texture features as well as Local Binary Patterns as suitable features for our segmentation framework. These features depend on parameters which influence the computation time as well as the descriptive capabilities of the corresponding features. In the following, we want to find good parameterizations for these features based on their benchmark performance.

6.3.2.1 Grayscale Patches

The benchmark score and runtime for grayscale image patches are given in Table 6.3:

Patch Size	3^2	5^2	7^2	9^2	11^2	13^2	15^2	17^2	19^2
Score	0.777	0.792	0.800	0.805	0.810	0.811	0.813	0.814	0.813
Runtime [s]	1266	1733	2252	2913	3607	4370	5292	6177	7329

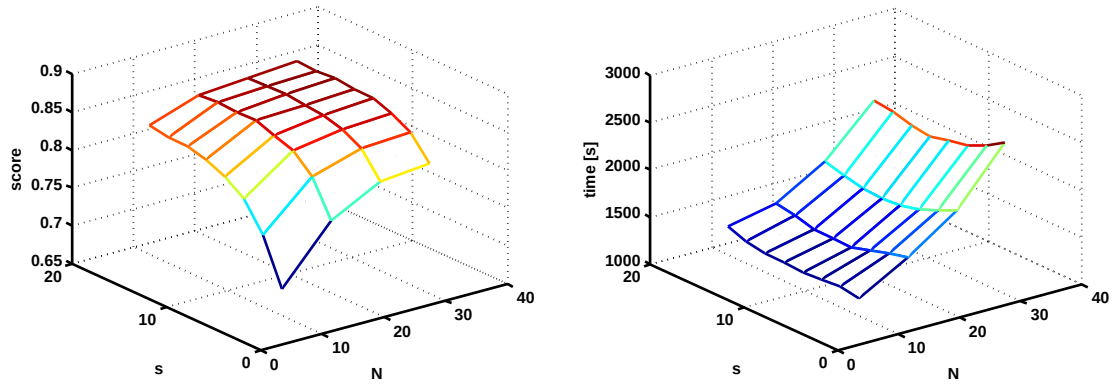
Table 6.3: Benchmark results for varying grayscale patch sizes.

The benchmark score steadily increases with the patch size starting from 0.777 for 3×3 patches until it gets stable at ≈ 0.81 for patch sizes larger than 9×9. This is due to the feature selective property of the Random Forests: At each node, the decision function is based on a single dimension of the feature vector, which is selected from a randomly sampled subset of all feature vector dimensions. As the patches increase, the dimension of the feature space increases, which leads to a larger number of possibly useful splitting functions. In this experiment, the Random Forest is able to find useful splitting functions and hence steadily improve the benchmark performance up to a patch size of 17×17. The performance drops for the first time at a patch size of 19×19 (which is a 361-dimensional feature space). However, larger feature spaces from grayscale patches also contain lots of noise, which leads to an increased runtime during the training phase of the Random Forests: While the overall runtime of the benchmark is only 1266 seconds for 3×3 patches (i.e. ≈ 4.5 seconds per image), it increases twofold for 7×7 and even sixfold for 19×19 patches. Also the memory consumption increases, not only for storing the features (≈ 337 MB for a single benchmark image with 19×19 patches), but also for the larger Random Forests.

6.3.2.2 Haralick Textural Features

The Haralick features described in Section 2.2.2 depend on two parameters: N represents the number of discrete grayvalues for the creation of the graylevel co-occurrence matrix and s defines the size of the quadratic sampling environment employed for its accumulation. E.g. the Haralick feature $H_{(16,5)}$ employs 16 discrete grayvalues sampled from a square 5×5 environment.

The results for different combinations of $N \in \{8, 16, 24, 32\}$ and $s \in \{3, 5, \ldots, 17\}$ are given in Figure 6.5: Starting with a score for 0.697 for $H_{(8,3)}$, the performance increases for more discrete grayvalues as well as larger sampling environments, and gets stable around 0.855 for $N \geq 16$ and $s \geq 11$. Similar to what has been observed for image patches, the computation time changes as the feature parameters change: While the runtime grows steadily for an increasing N, there seems to be a minimum for sampling environments around $s = 11$. The best benchmark score (0.8646) can be obtained for $N = 32$ and $s = 13$ in 2226 seconds.



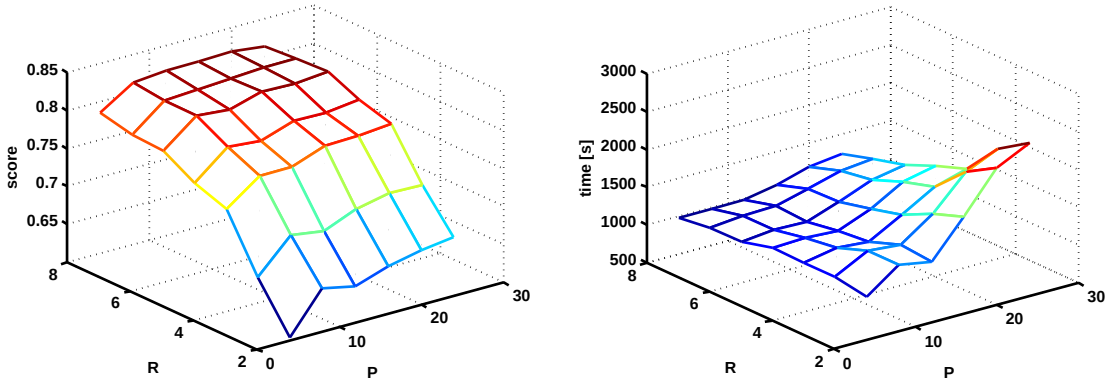
(a) (b)

Figure 6.5: Benchmark score (a) and overall runtime (b) for different parameterizations of Haralick texture features.

6.3.2.3 Local Binary Patterns

The Local Binary Patterns described in Section 2.2.4 are rotationally invariant and uniform, thus they have two free parameters: The number of points P sampled equally spaced on a circle of radius R. In this experiment, we evaluate the performance of Local Binary Patterns $LBP_{P,R}$ for $P \in \{4, 8, \ldots, 24\}$ and $R \in \{2, 3, \ldots, 8\}$. The results in Figure

6.6 show that the performance of the LBPs gets stable for $P \geq 16$ and $R \geq 6$ at a score of ≈ 0.815. The best result (0.8192) is achieved for $P = 20$ and $R = 8$ in 1186 seconds.



(a) (b)

Figure 6.6: Benchmark score (a) and overall runtime (b) for different parameterizations of Local Binary Patterns.

6.3.3 Feature Combinations

The best results from the single feature evaluations are summarized in Table 6.4.

Color Model	Gray	RGB	HSV	CIELAB	Patch 17×17	$H_{(32,13)}$	$LBP_{20,8}$
Performance	0.728	0.877	0.897	0.898	0.814	0.865	0.819
Runtime [s]	1556	967	890	898	6177	2226	1186

Table 6.4: Benchmark results for grayscale values and color representations.

Based on these results, the following observations can be made:

- The CIELAB color space reaches the best score with 0.898.
- All color models yield better scores than the local structure descriptions, which suggests that the most important information with respect to this benchmark is encoded in color.
- The textural features can give important cues for the segmentation process: As the three texture features operate on the grayscale values only, their result can directly be compared with the benchmark results obtained for grayscale images. All three

texture features yield a significantly better score (≥ 0.814) than the grayscale image alone (0.728).

- The textural features need large sampling neighborhoods for good performance: The best benchmark performance for patches has been obtained for the size 17×17, the Haralick features yield good results for $s \geq 11$. The Local Binary Patterns work well for $R \geq 6$, which corresponds to an 18×18 sampling environment (cf. Section 2.2.4).

A basic requirement for the design of our segmentation framework was, that it has to be able to handle arbitrary high-dimensional features. This allows for segmenting in even larger feature spaces by combining different feature representations, which can be accomplished by simply concatenating several feature vectors to one single larger feature vector. In the following, we want to evaluate whether such feature combinations can improve the benchmark performance.

6.3.3.1 Color Features

The best result for a single feature (0.898) has been obtained for the CIELAB color space. Table 6.5 shows results for combining several color features into a single feature vector (e.g. Gray+RGB means the combination of grayvalues and RGB color vectors to four-dimensional feature vectors). These results show that the combination of color features yields worse benchmark scores than the single CIELAB features at a higher runtime. Hence, adding grayvalues, RGB or HSV features to the CIELAB features does not add enough information to improve the benchmark score.

Color Model	CIELAB	Gray+RGB	Gray+CIELAB	Gray+RGB+HSV+CIELAB
Performance	0.898	0.876	0.894	0.896
Runtime [s]	898	953	919	973

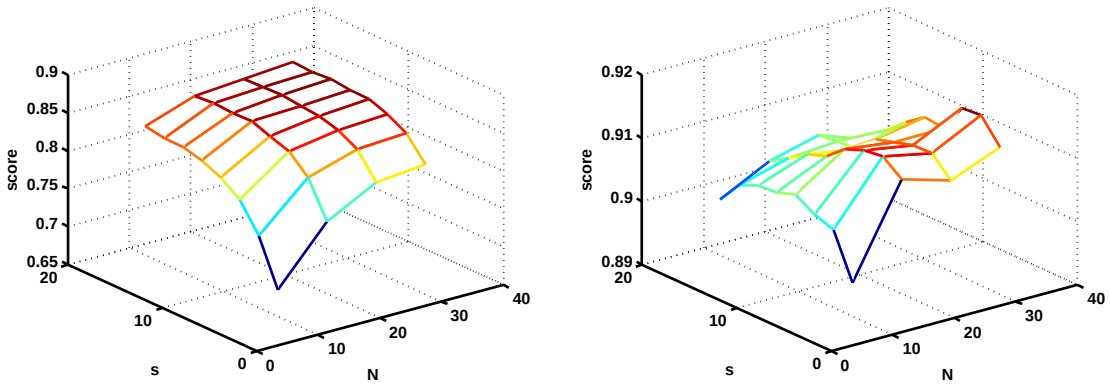
Table 6.5: Benchmark results for combined color features.

6.3.3.2 Color and Texture Features

In order to improve the results achieved with CIELAB features, we try to combine them with texture features. We employ the Haralick features as well as the Local Binary Patterns, which have achieved higher individual benchmark scores than the grayscale patches (cf. Table 6.4). In order to find out whether the best parameterizations obtained

from the individual benchmark runs are still good when the features are combined with CIELAB features, we again conduct parameter sweeps for the textural features.

The results for combined CIELAB and Haralick features for $N \in \{8, 16, 24, 32\}$ and $s \in \{3, 5, \ldots, 17\}$ compared to the results of Haralick features alone are depicted in Figure 6.7. Note that the addition of CIELAB color information not only significantly



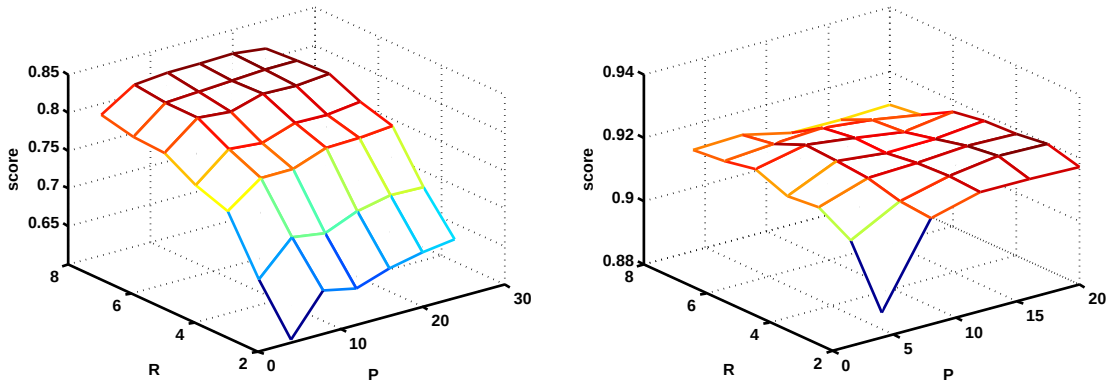
(a) (b)

Figure 6.7: Benchmark score for Haralick features alone (a) compared to the benchmark score achieved with Haralick features combined with CIELAB color vectors (b). Note that the Haralick parameters yielding the best results have changed when combined with CIELAB features.

increases the performance, but also changes the response to different parameter sweeps: While the performance of Haralick features alone steadily increased with an increasing sampling environment s, the combined features work best with $s \approx 7$. The highest score (0.916), which is achieved with $N = 32, s = 5$, is better than the score of the CIELAB color space alone (0.898).

We now perform a similar experiment for the Local Binary Patterns, with $P \in \{4, 8, \ldots, 20\}$ and $R \in \{2, 3, \ldots, 8\}$ (cf. Figure 6.8). Similar to what has been observed for the Haralick-CIELAB combinations, the addition of color information to Local Binary Patterns significantly improves the benchmark results. Also in this experiment, the color information renders large sampling environments obsolete: While the best scores for Local Binary Patterns were reached for $R \geq 6$, the performance of the combined features peaks at $R = 3$. The highest score (0.920), which is obtained with $P = 16, R = 3$, is better than what can be achieved with Haralick features.

Table 6.6 states the scores and runtime achieved with CIELAB features as well as



(a) (b)

Figure 6.8: Benchmark score for Local Binary Patterns alone (a) compared to the benchmark score achieved with Local Binary Patterns combined with CIELAB color vectors (b). Note that the Local Binary Pattern parameters yielding the best results have changed when combined with CIELAB features.

the best Haralick and Local Binary Pattern parameterizations alone and in combination. A basic observation is, that combining color and texture features yields a better

Features	CIELAB	$H_{(32,13)}$	CIELAB + $H_{(32,5)}$	$LBP_{20,8}$	CIELAB + $LBP_{16,3}$
Performance	0.898	0.865	0.916	0.819	0.920
Runtime [s]	898	2226	2183	1186	1286

Table 6.6: Scores and Runtime for CIELAB color and texture features alone and in combination.

score than what can be achieved with the respective color and texture features alone. Also the optimal parameters for the texture features change when combined with the color information: Especially the size of the sampling environment (s in the Haralick parametrization, R for the Local Binary Patterns) can be greatly reduced with the addition of the CIELAB features. Finally, the combination with Local Binary Patterns yields better results than the combined Haralick features in less time. Therefore, we employ the combination CIELAB + $LBP_{16,3}$ as combined feature for the remaining experiments.

6.4 Learning Algorithm

In every experiment in this section, we have employed Random Forests with 200 trees for the generation of our region models. In this experiment, we want to evaluate how far we can lower the number of trees in order to speedup the overall segmentation process. We also want to test the performance of the other learning algorithms described in Section 3.

Table 6.7 states benchmark scores and runtimes for Random Forests with different numbers of trees, k-Nearest Neighbours with varying k and a linear Support Vector Machine with varying regularization parameter C. When employing Random Forests

Algorithm	Parameters	Benchmark Score	Runtime [s]
Random Forest	$N = 10$	0.9084	626
	$N = 20$	0.9161	655
	$N = 30$	0.9193	686
	$N = 40$	0.9166	746
	$N = 50$	0.9172	793
k-Nearest Neighbors	$k = 1$	0.9161	2663
	$k = 2$	0.9161	2707
	$k = 5$	0.9140	2732
	$k = 10$	0.9069	2876
	$k = 15$	0.9038	2910
	$k = 20$	0.8997	3010
	$k = 25$	0.8954	3140
	$k = 30$	0.8944	3207
linear SVM	$C = 0.03$	0.8111	706
	$C = 0.035$	0.8114	702
	$C = 0.04$	0.8123	713
	$C = 0.045$	0.8128	696
	$C = 0.05$	0.8130	727
	$C = 0.055$	0.8130	736
	$C = 0.06$	0.8131	701

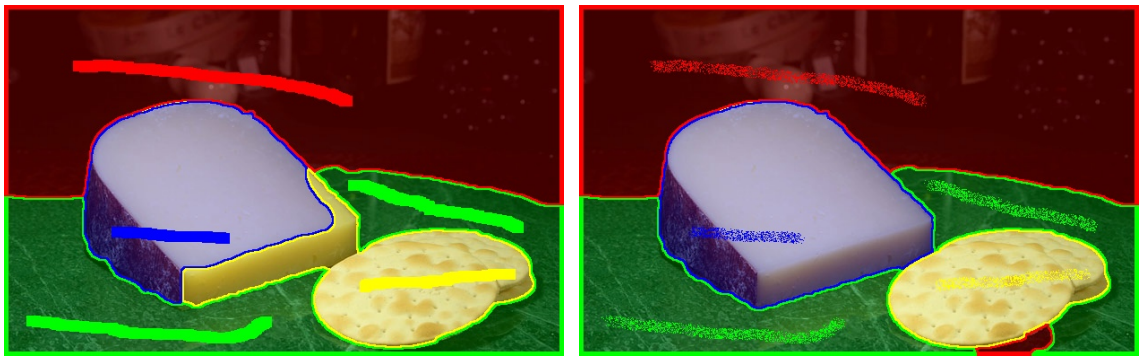
Table 6.7: Benchmark performance for different learning algorithms and parameters.

with $N = 200$, a score of 0.920 was achieved in 1286 seconds. A close performance (0.919) can be reached with 30 trees only, which boosts the overall runtime to 686 seconds (≈ 2.6 seconds per benchmark image). The k-Nearest Neighbors algorithm works surprisingly well in this scenario, reaching a score of 0.916 with $k = 1$ and $k = 2$. However, the distance computation is very expensive, which leads to a runtime of > 2650 seconds (> 10 seconds per image). The linear Support Vector Machine is comparably fast to

Random Forest with 30 trees. However, the problem is obviously not suited for linear SVMs, as the benchmark score does not exceed 0.82.

6.5 Tooltip

As described in Chapter 5, the seed pixels in the benchmark are represented by the path the user took with his mouse during annotation. The benchmark does not define how this mouse paths are interpreted, anybody using the benchmark may employ arbitrary tooltips upon the mouse paths. In this experiment, we evaluate the performance of a square solid brush and a spraygun brush with different radii. The spraygun brush is like a solid brush, where only a random five percent of the pixels are used (cf. Figure 6.9).



(a) (b)

Figure 6.9: Applying different tooltips leads to different results: (a) shows results obtained with a square brush with radius $r = 11$, (b) shows the same problem with an airbrush tooltip with $r = 11$.

The benchmark scores and runtimes are given in Table 6.8: Both the airbrush as well as the solid brush are able to yield higher benchmark scores with larger radii. As the radius of tooltips increases, the number of training samples for the Random Forest increases, which allows to build better models for the regions around the seed pixels. However, the incorporation of many spatially close pixels includes lots of redundant information in the training set. The random sampling strategy of the airbrush tooltip together with the feature selective property of the Random Forest allows to reduce this redundancy. Hence, the airbrush tooltip yields comparable results on the benchmark in

Brush	Radius	Benchmark Score	Runtime [s]
Square Solid	$r = 3$	0.9144	804
	$r = 5$	0.9167	986
	$r = 7$	0.9221	1161
	$r = 9$	0.9256	1376
	$r = 11$	0.9253	1488
	$r = 13$	0.9250	1675
	$r = 15$	0.9257	1842
Spraygun	$r = 3$	0.8945	661
	$r = 5$	0.9078	695
	$r = 7$	0.9140	731
	$r = 9$	0.9188	731
	$r = 11$	0.9220	770
	$r = 13$	0.9267	834
	$r = 15$	0.9249	876

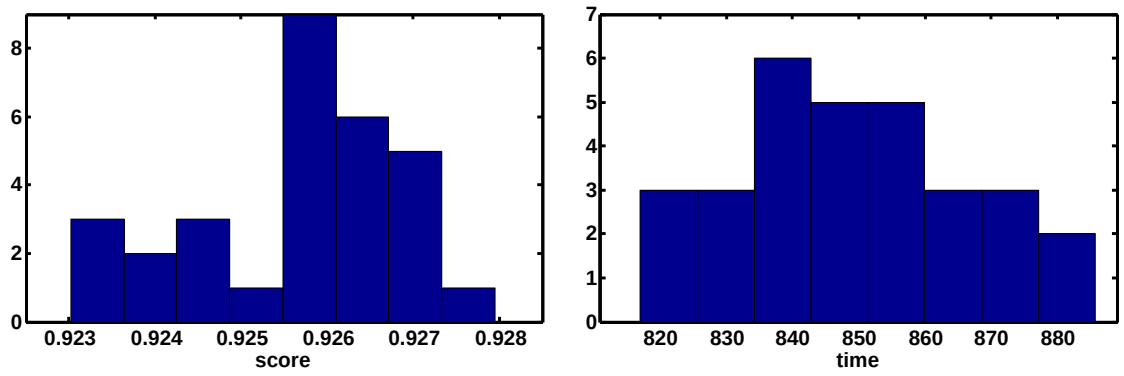
Table 6.8: Benchmark scores and runtimes for seeds generated with different tooltips: A square solid brush and an airbrush with varying radii.

significantly less time: The highest score (0.9267) can be obtained with an airbrush with $r = 13$, where the whole benchmark run takes 834 seconds to complete.

6.6 Repeatability

The Random Forests as well as the randomly sampled seed points obtained with the airbrush tooltip impose a random behavior of our framework, i.e. two identically parameterized segmentation problems might yield different results at different runtimes. In this experiment, we evaluate the influence of this randomness on the benchmark performance and runtime in order to find out how reliable the previously obtained results are. Figure 6.10 shows histograms over the performance of 50 identically parameterized benchmark runs. The average score over these 30 runs is 0.9257 with a standard deviation of 0.0013, thus at least two thirds of the benchmark runs (in our case exactly 20 of 30) achieved scores between 0.9244 and 0.9269.

The average runtime is 848.9 seconds with a standard deviation of 18.1 seconds. These observations show, that the inherent randomness in our framework influences the benchmark performance, however, this influence is not large enough to compromise the reasoning of the previous experiments. The averaged results obtained in this experiment (score: 0.9257, runtime: 848.9 seconds), form the basis for the considerations in the upcoming experiments.



(a) (b)

Figure 6.10: Evaluating the influence of the randomness induced by the Random Forests and the random seeds of the airbrush tooltip: (a) shows an histogram over the performance of 30 identically parameterized benchmark runs, (b) shows the corresponding runtime in seconds.

6.7 Runtime

In this experiment, we want to examine the runtime of our framework and its building blocks. Therefore, we analyze a single benchmark run from the previous experiment by dividing the overall runtime into several parts:

Edge and Feature represent the runtimes for the computation of the gradient image as well as the image features (CIELAB color vectors and Local Binary Patterns).

Model describes the time spent on training the Random Forests with the seed pixels forming the training set.

Data Term is the part of the algorithm, where the Random Forests are evaluated for every pixel in the image in order to obtain the likelihoods for each label.

Segmentation indicates the runtime of the regularization with the Potts model.

Overhead contains everything else, e.g. loading images and groundtruth data, the computation of the benchmark score or displaying, compressing and storing segmentation results.

The runtime for every stage except the overhead is directly measured for every of the 262 segmentation problems in the benchmark. These runtimes include not only the raw

computation times on the CPU or GPU, but also memory operations such as copying to and from GPU memory or memory reshape operations. The overhead runtime is the difference between the total benchmark runtime (812.67 for the given benchmark run) and all other measured runtimes. The average runtime and the standard deviation over all 262 segmentation problems are summarized in Table 6.9: The computation of the

	Edge	Feature	Model	Data Term	Segment.	Overhead
Avg. Runtime [ms]	62.30	338.93	768.61	103.51	1008.72	819.73
Std. Dev. [ms]	3.21	6.10	817.26	42.12	454.61	0.0

Table 6.9: Runtime of the different stages of our framework: This table summarizes the average runtime and the standard deviation over all segmentation problems within one single benchmark run. The Overhead stage has no standard deviation, as it is not measured but computed as the difference between all measured runtimes and the overall benchmark runtime.

gradient image as well as the computation of the image features has very low variation between the different segmentation problems in the benchmark. This is obvious, as these operations depend only on the size of the image and the feature parameterizations, which are the same for every problem in the benchmark. Contrarily, the runtimes of the training and evaluation of the Random Forest change largely. This can be explained by the fact that the difficulty of the learning problems varies between different segmentation problems. While simple problems results in small trees and forests with a small training and evaluation time, difficult problems lead to larger grown trees and hence longer training and evaluation times. The time spent for the segmentation algorithm also differs between the segmentation problems, although the number of iterations of the algorithm is fixed. The reason for this is, that the number of labels of a segmentation problem influences the runtime of the algorithm. In Table 6.10, the runtimes of the segmentation algorithm are grouped according to the number of labels of the different segmentation problems. This table clearly shows, that the runtimes of the segmentation

# Labels	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
# Problems in IcgBench	64	104	58	18	8	3
Avg. Runtime [ms]	586.65	876.23	1179.44	1520.28	1861.03	2205.57
Std. Dev. [ms]	2.74	1.23	0.90	2.11	3.09	2.73

Table 6.10: The runtime of the segmentation algorithm only depends on the number of labels involved in a specific problem.

algorithm for problems with the same number of labels is nearly constant. More than 85

percent of the segmentation problems in IcgBench (226 problems) have at most 4 labels, for which the segmentation algorithm finishes in less than 1.2 seconds.

Figure 6.11 shows the fraction of the runtimes of the different stages. Concerning the interactivity, the most important stages are Model, Data Term and Segmentation: The Edge and Feature stages as well as most of the Overhead can be computed before or while the user draws his initial scribbles, therefore their runtime is not very important. Every time a user adds or removes scribbles, the stages Model, Data Term and Segmentation need to be computed in order to obtain a displayable result, which makes their runtime the most important factor for convenient interaction. Based on the measurements in this experiment, these three stages make up $\approx$ 60 percent of the overall benchmark runtime, which amounts to an average of $\approx$ 1.9 seconds per benchmark problem.

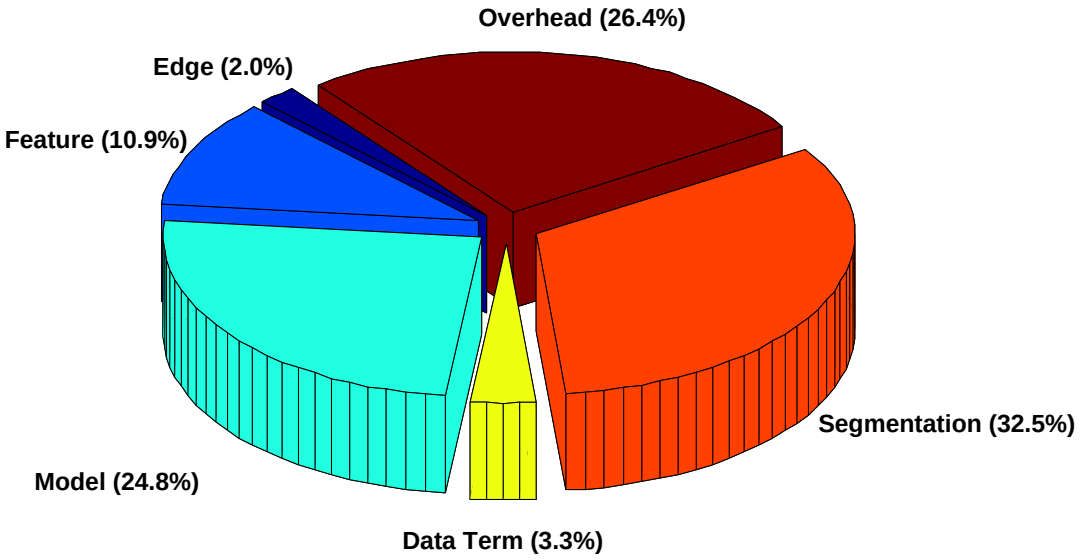


Figure 6.11: Runtime fraction of the different framework stages evaluated over the whole benchmark. The important stages for interactivity (Model, Data Term and Segmentation) make up $\approx$ 60 percent of the overall runtime.

6.8 Comparison to Power Watersheds

In this experiment, we compare our framework to the Power Watersheds algorithm by Couprie et al. (2010), which is one of few interactive multi-label segmentation methods with an publicly available implementation. The Power Watersheds method, which is

a graph-based algorithm operating in the RGB color space (cf. Section 1.4.3.10 for a detailed description), yields comparable or better results than e.g. Graph Cut or Random Walker segmentation on the two-label GrabCut benchmark dataset. In Table 6.11, we compare the benchmark scores and runtimes of Power Watersheds to our framework. For our framework, we compare to the optimized version with CIELAB color vectors and Local Binary Patterns, 30 Random Forest Trees and an airbrush tooltip with radius 13. Furthermore, we also compare to an identically parameterized benchmark run of our framework, with only RGB color vectors as features. The Power Watersheds method

	Power Watersheds	RGB	CIELAB + $LBP_{16,3}$
Runtime [s]	347	648	849
Score	0.864	0.887	0.926

Table 6.11: Benchmark score and runtime of Power Watersheds compared to results obtained with our framework.

employs a highly efficient energy minimization algorithm based on the computation of maximum spanning forests, which allows it to finish the benchmark about two times faster than our framework. In order to run this benchmark we employed a MATLAB wrapper, which prepares the benchmark problems for the C implementation of Power Watersheds by Couprie et al. (2010). Of the overall 347 seconds, only 224 seconds are spent with the Power Watershed segmentation itself, the rest is overhead produced by the wrapper. Hence, the algorithm is able to segment one single benchmark problem in $\approx$ 855 milliseconds.

Concerning the benchmark score, the Power Watersheds method yields slightly worse results than our framework with RGB values. The best results are obtained with the optimized version of our framework with CIELAB + $LBP_{16,3}$ features. Figure 6.12 shows the three segmentation problems where the Power Watersheds algorithm yields the lowest score: All of them exhibit regions with high variability and lots of strong image gradients, which suggests that Power Watersheds has difficulties with such highly textured objects and backgrounds. Another drawback can be clearly observed in the third problem (Image 008, User 9060): The topology of the ground truth solution differs from the topology of the seed pixels, which cannot be modeled with the Power Watersheds algorithm.

Figure 6.13 shows the three segmentation problems where the Power Watersheds algorithm yields the highest score: In all of them, the number of seeded pixels is very high compared to the number of seeded pixels in the problems where the algorithm

scored worst. Furthermore, the seed pixels are widely spread over the regions. This suggests, that the spatial extents of the objects need to be loosely defined by the seed pixels for a good performance of the Power Watershed algorithm.

In the second best problem (Image 216, User 9050), the Power Watershed algorithm benefits of its topological limitations coincidentally: From a feature point of view, the hole in the paper roll clearly should have the same label as the background according to the specified seed pixels. However, in the groundtruth labeling, the user assigned it the same label as the paper roll itself. While our framework assigns the hole in the paper roll as background, the Power Watersheds stick to the topology of the seed pixels and achieve a high score on this problem.

6.9 Conclusion

In this chapter, we performed extensive experiments on our benchmark dataset IcgBench in order to find suitable adjustments and parameterizations for our framework. The following adjustments were found useful in order to achieve a high benchmark score at a reasonable runtime:

Tooltip In order to generate seed points for the mouse path, we employ an airbrush tooltip with radius $r = 13$. The experiments showed, that this sampling tooltip yields comparable results as a solid brush tooltip at significantly lower runtimes.

Features We evaluated the performance of color and texture features. When evaluated alone, CIELAB was found to be the best-performing color space. The best scoring textural representation was a Haralick feature, which however requires a long time to compute. The best overall performance was reached by concatenating CIELAB color vectors and Local Binary Patterns with $P = 16$ and $R = 3$ to a 21-dimensional feature representation.

Learning Algorithm We assessed three learning algorithms for the generation of models from the seed pixels, namely Random Forests, k-Nearest Neighbors and a linear Support Vector Machine. The Random Forests and the SVM were significantly faster than k-Nearest Neighbors, however, the SVM achieved only poor benchmark scores. Based on these results, we employ Random Forests with $N = 30$ trees.

Segmentation Algorithm The experiments showed, that for most of the images, the results do not change after $n_{iter} = 750$ iterations. We obtained the best results with a regularization weight $\lambda = 0.2$ and an edge weight $\alpha = 15$.

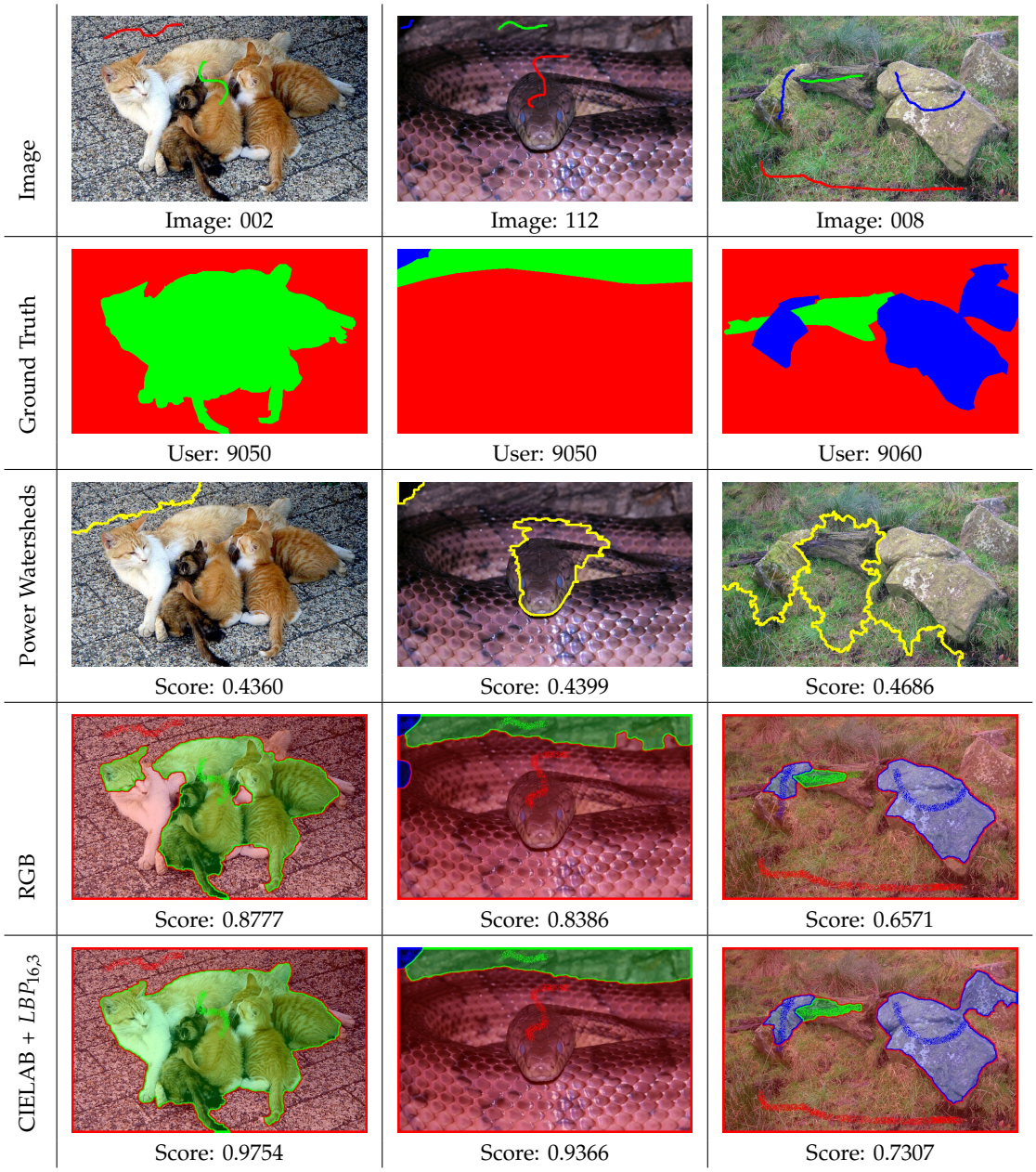


Figure 6.12: When benchmarking the Power Watersheds algorithm, the three depicted problems yield the lowest score within the entire benchmark dataset. The first and second row show the input image with seed pixels (dilated for visualization) and the ground truth segmentation. The other rows depict the results of the Power Watershed algorithm as well as the results of our algorithm with RGB values and CIELAB + $LBP_{16,3}$ features.

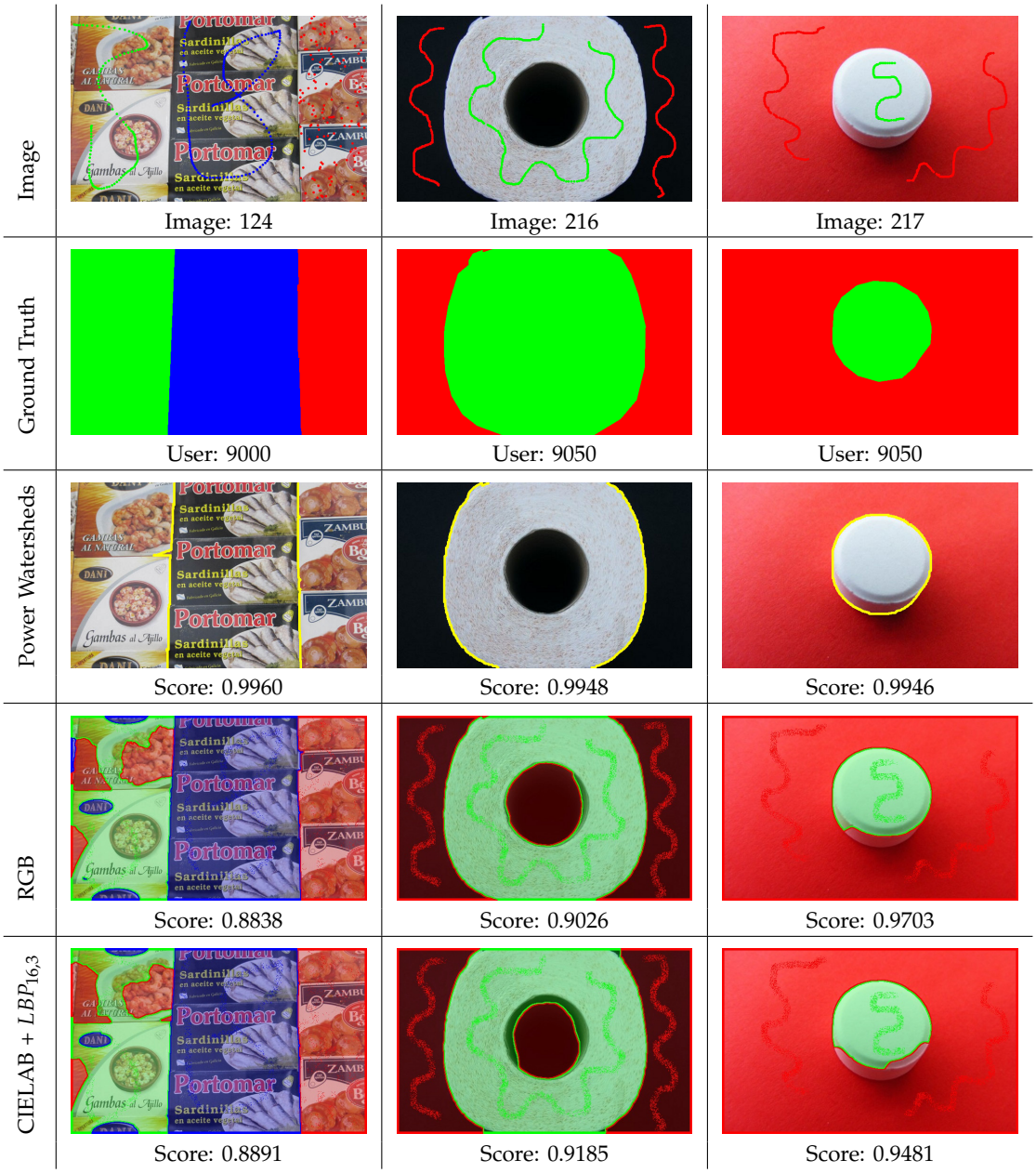


Figure 6.13: When benchmarking the Power Watersheds algorithm, the three depicted problems yield the best score within the entire benchmark dataset. The first and second row show the input image with seed pixels (dilated for visualization) and the ground truth segmentation. The other rows depict the results of the Power Watershed algorithm as well as the results of our algorithm with RGB values and CIELAB + $LBP_{16,3}$ features.

With these adjustments, a benchmark score of ≈ 0.926 can be achieved. The overall runtime with this parameterization is ≈ 830 seconds, which is little more than three seconds per benchmark image. In the interactive setting, mainly the time between drawing seeds and getting the final segmentation is of interest (i.e. the training and evaluation of the Random Forest as well as the segmentation algorithm). For most of the benchmark images, this is performed in about 1.9 seconds.

We finally compared the benchmark result of our framework to the benchmark result of the Power Watersheds algorithm by Couprie et al. (2010). While this algorithm is faster than our framework by a factor of two, the benchmark score (0.864) is even lower than what our framework can achieve with only RGB color vectors. We showed, that this is mostly because the Power Watersheds algorithm suffers from a fixed topology and has problems with highly textured areas. Furthermore, the experiments suggest that the performance of the algorithm depends strongly on the number and position of the seed pixels.

The fiveteen segmentation problems yielding the highest score within the benchmark are illustrated in Figure 6.15, the fiveteen worst results are given in Figure 6.14.

Image: 223, User: 9050, Score: 0.5719

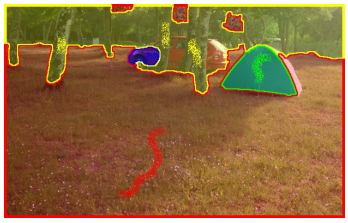


Image: 029, User: 9050, Score: 0.6064

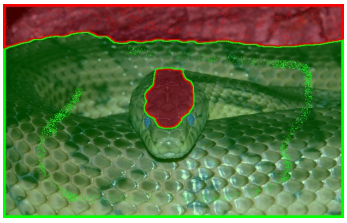


Image: 112, User: 9020, Score: 0.6167

Image: 124, User: 9050, Score: 0.6708

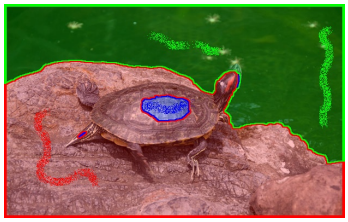


Image: 048, User: 9050, Score: 0.6741

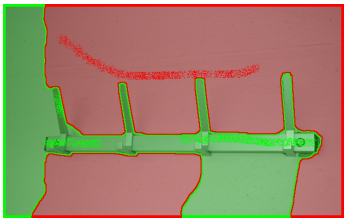


Image: 180, User: 9060, Score: 0.6751

Image: 111, User: 9050, Score: 0.6997

Image: 106, User: 9060, Score: 0.7024

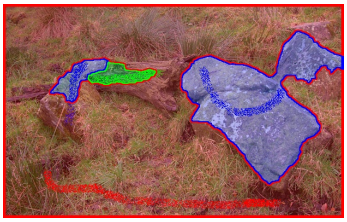


Image: 008, User: 9060, Score: 0.7073

Image: 005, User: 9020, Score: 0.7232

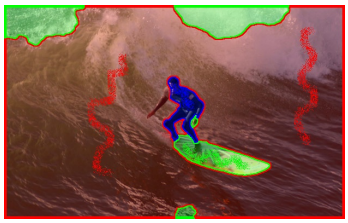


Image: 149, User: 9050, Score: 0.7328

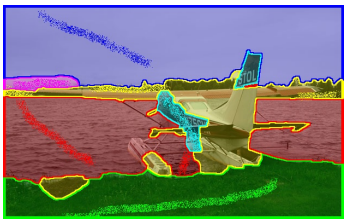


Image: 012, User: 9060, Score: 0.7330

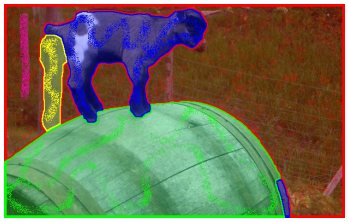


Image: 024, User: 9025, Score: 0.7413

Image: 110, User: 9050, Score: 0.7614

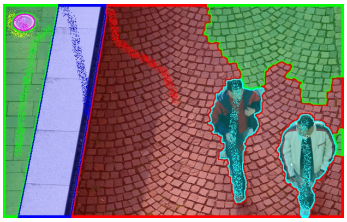


Image: 018, User: 9060, Score: 0.7640

Figure 6.14: The 15 worst-scoring segmentation problems of the benchmark

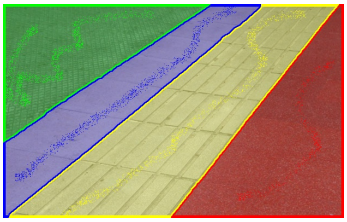


Image: 132, User: 9000, Score: 0.9959


Image: 202, User: 9050, Score: 0.9952

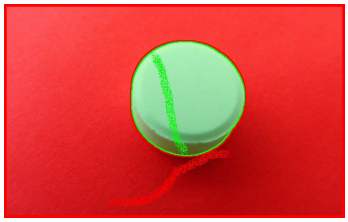


Image: 217, User: 9060, Score: 0.9948

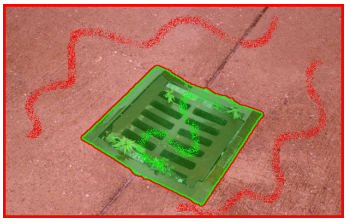


Image: 182, User: 9050, Score: 0.9946

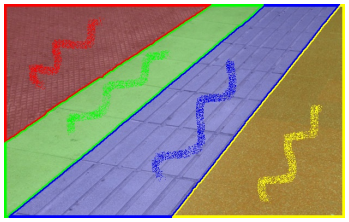


Image: 132, User: 9050, Score: 0.9941


Image: 218, User: 9050, Score: 0.9935

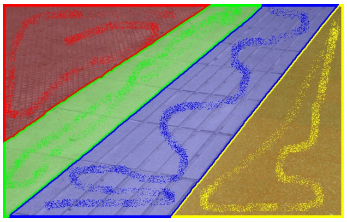


Image: 132, User: 9025, Score: 0.9916

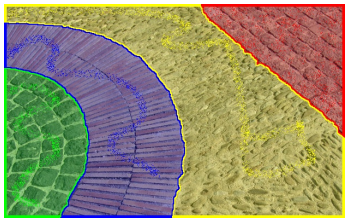


Image: 140, User: 9025, Score: 0.9914

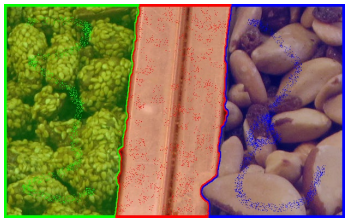


Image: 121, User: 9000, Score: 0.9911

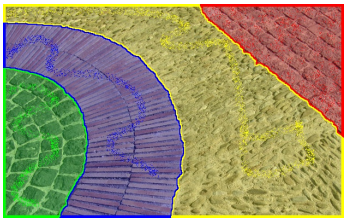


Image: 140, User: 9000, Score: 0.9909

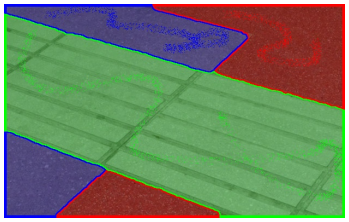


Image: 131, User: 9000, Score: 0.9908

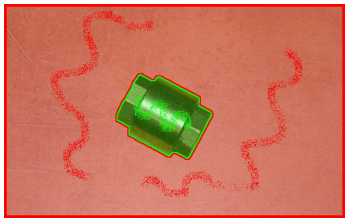


Image: 212, User: 9050, Score: 0.9907

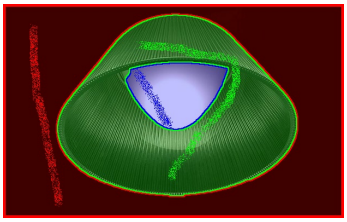


Image: 232, User: 9060, Score: 0.9902


Image: 077, User: 9050, Score: 0.9900

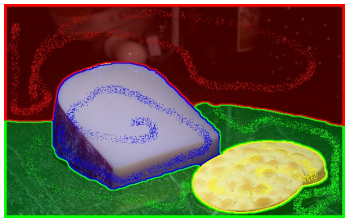


Image: 096, User: 9000, Score: 0.9899

Figure 6.15: The 15 best-scoring segmentation problems of the benchmark

7 Conclusion

In this thesis, we studied the process of interactive image segmentation. Popular interactive image segmentation methods typically only divide an image into foreground and background based on intensity or color information, as the computational effort needed for an incorporation of textural features is usually prohibitive for reaching interactive runtimes. In this work, we show that interactively segmenting an image into multiple regions based on high-dimensional feature representations is possible at interactive speeds by carefully selecting the building parts w.r.t. to parallelizability and implementing them on multi-core CPUs and GPUs.

7.1 Summary

In this work, we first described several color models and texture features and analyzed their applicability for interactive image segmentation. We demonstrated how the GPU can be employed to speedup the computation of CIELAB color features as well as Haralick texture features (Haralick et al., 1973) and Local Binary Patterns (Ojala et al., 2002) by factors of up to 90 depending on the image size: E.g., while a single threaded CPU implementation of dense Local Binary Patterns takes about 6 seconds on a 1024×1024 image depending on the parameterization, a GPU counterpart performs the same computation in ≈ 150 milliseconds.

In interactive segmentation, one typically solves a supervised machine learning problem, where the user provided seed pixels form the training set and all other pixels form the evaluation set. In this work, we compared the performance and runtime of Random Forests (Breiman, 2001) with that of a linear Support Vector Machine and a k-Nearest

Neighbors implementation on standard machine learning datasets as well as segmentation problems. Especially for the latter scenario, where the size of the test set (i.e. all image pixels) is much larger than the size of the training set (i.e. the number of seeded pixels), we showed that Random Forests are able to yield excellent results at high speed.

Furthermore, the pixel likelihoods obtained from the learning algorithm need to be regularized in order to yield spatially compact regions. We thoroughly described such segmentation algorithms in the discrete domain (cf. Graph Cut segmentation (Boykov and Jolly, 2001)) as well as in the spatially continuous setting (cf. weighted Total Variation (Bresson et al., 2007)). While these energy minimization approaches can be solved in a globally optimal way for two labels, the globally optimal solution for multiple labels is NP-hard. We evaluated several approximation techniques and selected a spatially continuous approximation of a multi-label Potts energy minimization problem, which can also be implemented on a GPU (Pock et al., 2009).

Finally, we addressed the lack of a comparative benchmark dataset for the evaluation of interactive multi-label segmentation techniques: We presented a novel benchmark dataset consisting of 262 seed-groundtruth pairs annotated by eight different people. We employed this benchmark dataset to evaluate the performance and runtime of the different building blocks of our framework: We showed that the combination of texture features with color features leads to better results than what can be achieved with color features alone. Moreover, we found out, that the best benchmark results can be obtained when CIELAB color vectors are combined with Local Binary Patterns. Finally, we showed that our framework outperforms the state-of-the-art Power Watersheds segmentation method on this benchmark dataset.

7.2 Future Work

In this thesis, we presented a powerful interactive multi-label segmentation framework and evaluated it with a novel benchmark dataset. However, there are still open questions and drawbacks, which may be addressed in future work:

7.2.1 Semi-Supervised Learning

Currently, the setting of the learning stage of our algorithm is fully supervised, i.e. there is a training set (the seeded pixels) for learning a model and a test set (the unseeded pixels) on which the model is evaluated. However, the problem of interactive segmentation can also be tackled with semi-supervised machine learning algorithms: In

the semi-supervised setting, the model is trained not only according to the seeded image pixels, but also by incorporating information of all unseeded pixels. There exists a semi-supervised version of Random Forests (Leistner et al., 2009), which could be easily employed to evaluate the benefit of learning the model in a semi-supervised setting. It is not clear whether employing such a semi-supervised learning algorithm in our interactive segmentation framework leads to better results or not. Furthermore, the effect on the runtime is an open question.

7.2.2 Computational Effort

In this work, we achieved interactive performance with our framework on images with resolutions below one million pixels (MP). In professional digital image editing, typically images with a much higher resolution ($\geq 10MP$) are of interest, which leads to the question, whether our framework can still be optimized w.r.t. runtime. We want to note that the framework was set up to be modular in order to evaluate different image features and learning algorithms. This naturally comes at the cost of runtime, e.g. many unnecessary copy and store operations. Moreover, we perform every step of our algorithm always on the entire image, which is not the case for e.g. the recently published very fast Paint Selection method (Liu et al., 2009). The speedup potential of non-global operations such as employing sequential segmentations on parts of the image needs to be evaluated.

7.2.3 Evaluation

In this thesis, we mainly employed our benchmark dataset IcgBench to perform detailed experimental evaluations of our own segmentation framework. We only compared our framework to the Power Watersheds method, as there is hardly any code available online for other interactive multi-label segmentation tools. In the original Power Watersheds paper, the authors demonstrate comparable or better results than e.g. Geodesic Segmentation or Random Walker Segmentation on the GrabCut database. However, the GrabCut database consists only of two-label problems and has objects that are spatially very well defined by the seed pixels, thus one cannot draw a conclusion on the performance of Geodesic Segmentation or Random Walker Segmentation based on the Power Watersheds result. Hence, in future work, a detailed comparison of interactive multi-label segmentation methods on IcgBench needs to be done.

7.2.4 3D / Video

The framework described in this thesis is designed to operate on 2D images only. An interesting field of future work would be the extension of our framework to 3D-segmentation for e.g. medical data or videos in spatial-temporal representation.

A List of Publications

A.1 2010

Interactive Multi-Label Segmentation

Jakob Santner, Thomas Pock and Horst Bischof
In: *Proceedings of the 10th Asian Conference on Computer Vision* (ACCV),
November 2010 (to appear), Queenstown, New Zealand
(Accepted for oral presentation, 4.7 % acceptance rate)

Abstract: This paper addresses the problem of interactive multi-label segmentation. We propose a powerful new framework using several color models and texture descriptors, Random Forest likelihood estimation as well as a multi-label Potts-model segmentation. We perform most of the calculations on the GPU and reach runtimes of less than two seconds, allowing for convenient user interaction. Due to the lack of an interactive multi-label segmentation benchmark, we also introduce a large publicly available dataset. We demonstrate the quality of our framework with many examples and experiments using this benchmark dataset.

PROST: Parallel Robust Online Simple Tracking

Jakob Santner, Christian Leistner, Amir Saffari, Thomas Pock and Horst Bischof
In: *Proc. of the 23rd IEEE Conference on Computer Vision and Pattern Recognition* (CVPR),
June 2010, San Francisco, USA
(Accepted for poster presentation, 26.7 % acceptance rate)

Abstract: Tracking-by-Detection is an increasingly popular approach in order to tackle the visual tracking task. Existing adaptive methods suffer from the drifting problem, since they rely on self-updates of an on-line learning method. In contrast to previous work that tackled this problem via reformulating the self-learning to either semi-supervised or multiple-instance learning, we show that augmenting an on-line learning method with complementary tracking approaches can lead to better results. In particular, we use a simple template model as a non-adaptive element, a novel optical-flow-based meanshift tracker as highly adaptive element and an on-line random forest as adaptive appearance-based learner. We combine these three trackers in a simple cascade. All of our system parts are chosen in order to run on GPUs or similar multi-core systems, which allows for near real-time performance. We show the superiority of our system over current state-of-the-art online tracking methods in several experiments.

FlowGames

Jakob Santner, Manuel Werlberger, Thomas Mauthner, Wolfgang Paier and Horst Bischof
In: *1st IEEE International Workshop on Computer Vision for Computer Games* (CVCG),
June 2010, San Francisco, USA
(Accepted for oral presentation, 36.4 % acceptance rate)

Abstract: Computer vision-based interfaces to games hold the promise of rich natural interaction and thus a more realistic gaming experience. Therefore, the video games industry started to develop and market computer vision-based games recently with great success. Due to limited computational resources, they employ mostly simple algorithms such as background subtraction, instead of sophisticated motion estimation or gesture recognition methods. This not only results in a lack of robustness, but also in very limited interaction possibilities and thus reduced gaming experience. In this paper, we show a couple of concepts to control video games based on optical flow. We use a state-of-the-art optical flow algorithm able to be computed densely in real-time on GPUs, which are in fact built-in in nearly every gaming hardware available. Based on the estimated motion, we develop several computer games with increasing complexity: Starting with using the flow field as force acting on moveable objects, we span the spectrum to more sophisticated concepts such as controlling widgets and action recognition.

A.2 2009

Semi-Supervised Random Forests

Christian Leistner, Amir Saffari, Jakob Santner, Horst Bischof
In: *Proceedings of the 12th IEEE International Conference on Computer Vision* (ICCV),
October 2010, Kyoto, Japan
(Accepted for poster presentation, 19.6 % acceptance rate)

Abstract: Random Forests (RFs) have become commonplace in many computer vision applications. Their popularity is mainly driven by their high computational efficiency during both training and evaluation while still being able to achieve state-of-the-art accuracy. This work extends the usage of Random Forests to Semi-Supervised Learning (SSL) problems. We show that traditional decision trees are optimizing multi-class margin maximizing loss functions. From this intuition, we develop a novel multi-class margin definition for the unlabeled data, and an iterative deterministic annealing-style training algorithm maximizing both the multi-class margin of labeled and unlabeled samples. In particular, this allows us to use the predicted labels of the unlabeled data as additional optimization variables. Furthermore, we propose a control mechanism based on the out-of-bag error, which prevents the algorithm from degradation if the unlabeled data is not useful for the task. Our experiments demonstrate state-of-the-art semi-supervised learning performance in typical machine learning problems and constant improvement using unlabeled data for the Caltech-101 object categorization task.

On-line Random Forests

Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, Horst Bischof
In: *3rd IEEE Online Learning for Computer Vision Workshop* (OLCV),
October 2010, Kyoto, Japan
(Accepted for oral presentation, 45.5 % acceptance rate)

Abstract: Random Forests (RFs) are frequently used in many computer vision and machine learning applications. Their popularity is mainly driven by their high computational efficiency during both training and evaluation while still achieving state-of-the-art results. However, in most applications RFs are used off-line. This limits their usability for many practical problems, for instance, when training data arrives sequentially or the underlying distribution is continuously changing. In this paper, we propose a novel

on-line random forest algorithm. We combine ideas from on-line bagging, extremely randomized forests and propose an on-line decision tree growing procedure. Additionally, we add a temporal weighting scheme for adaptively discarding some trees based on their out-of-bag-error in given time intervals and consequently growing of new trees. The experiments on common machine learning data sets show that our algorithm converges to the performance of the off-line RF. Additionally, we conduct experiments for visual tracking, where we demonstrate real-time state-of-the-art performance on well known scenarios and show good performance in case of occlusions and appearance changes where we outperform trackers based on on-line boosting. Finally, we demonstrate the usability of on-line RFs on the task of interactive realtime segmentation.

Interactive Texture Segmentation using Random Forests and Total Variation

Jakob Santner, Markus Unger, Thomas Pock, Christian Leistner, Amir Saffari and Horst Bischof
In: *Proceedings of the 20th British Machine Vision Conference* (BMVC),
September 2009, London, UK
(Accepted for poster presentation, 37.9 % acceptance rate)

Abstract: Common methods for interactive texture segmentation rely on probability maps based on low dimensional features such as e.g. intensity or color, that are usually modeled using basic learning algorithms such as histograms or Gaussian Mixture Models. The use of low level features allows for fast generation of these hypotheses but limits applicability to a small class of images. We address this problem by learning complex descriptors with Random Forests and exploiting their inherent parallelism in a GPU implementation. The segmentation itself is based on a convex energy functional that uses weighted Total Variation regularization and a point-wise data term allowing for continuous foreground/background membership hypotheses. Its globally optimal solution is obtained by a fast primal-dual algorithm providing a reasonable convergence criterion. As a result, we present a versatile interactive texture segmentation framework. We show experiments with natural, artificial and medical data and demonstrate superior results compared to two recent approaches.

Bibliography

Adams, R. and Bischof, L. (1994). Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647. (cited on pages 19 and 36)

Allène, C., Audibert, J.-Y., Couprie, M., and Keriven, R. (2010). Some links between extremum spanning forests, watersheds and min-cuts. *Image and Vision Computing*, 28(10):1460–1471. (cited on page 34)

Amit, Y., August, G., and Geman, D. (1996). Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588. (cited on page 71)

Arbelaez, P. and Cohen, L. (2008). Constrained image segmentation from hierarchical boundaries. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 105)

Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2010). Contour detection and hierarchical image segmentation. Technical report, EECS Department, University of California, Berkeley. (cited on pages 12, 100, and 103)

Bai, X. and Sapiro, G. (2007). A geodesic framework for fast interactive image and video segmentation and matting. In *Proceedings 11th International Conference on Computer Vision*. (cited on pages 32, 38, and 99)

Barrett, W. A. and Cheney, A. S. (2002). Object-based image editing. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 777–784. (cited on pages 18 and 99)

Blake, A., Rother, C., Brown, M., Perez, P., and Torr, P. (2004). Interactive image segmentation using an adaptive gmmrf model. In *Proceedings 8th European Conference on Computer Vision*. (cited on page 26)

Bolz, J., Farmer, I., Grinspun, E., and Schroeder, P. (2005). Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 171. (cited on page 29)

Bosch, A., Zisserman, A., and Munoz, X. (2007). Image classification using random forests and ferns. In *Proceedings 11th International Conference on Computer Vision*. (cited on pages 69 and 71)

Boykov, Y. and Kolmogorov, V. (2001). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *EMMCVPR '01: Proceedings of the Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 359–374, London, UK. (cited on page 24)

Boykov, Y. and Kolmogorov, V. (2003). Computing geodesics and minimal surfaces via graph cuts. In *Proceedings 9th International Conference on Computer Vision*. (cited on pages 25 and 30)

Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137. (cited on page 24)

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239. (cited on pages 90, 92, and 97)

Boykov, Y. Y. and Jolly, M. P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proceedings 8th International Conference on Computer Vision*. (cited on pages 23, 37, 38, 39, 84, 89, 97, and 140)

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24:123–140. (cited on page 71)

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32. (cited on pages 13, 45, 69, 71, and 139)

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. Chapman and Hall/CRC. (cited on page 70)

Bresson, X., Esedoglu, S., Vandergheynst, P., Thiran, J.-P., and Osher, S. (2007). Fast global minimization of the active contour/snake model. *Journal of Mathematical Imaging and Vision*, 28(2):151–167. (cited on pages 23, 30, 38, 46, 84, 89, 96, 97, and 140)

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698. (cited on page 8)

Caselles, V., Kimmel, R., and Sapiro, G. (1997). Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79. (cited on pages 22, 36, and 84)

Chan, T. F., Esedoglu, S., and Nikolova, M. (2006). Algorithms for finding global minimizers of image segmentation and denoising models. *SIAM Journal on Applied Mathematics*, 66(5):1632–1648. (cited on page 94)

Cheng, H., Jiang, X., Sun, Y., and Wang, J. (2001). Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259 – 2281. (cited on pages 51 and 52)

Cohen, L. D. and Kimmel, R. (1997). Global minimum for active contour models: A minimal path approach. *International Journal of Computer Vision*, 24(1):57–78. (cited on page 23)

Comaniciu, D. and Meer, P. (1997). Robust analysis of feature spaces: color image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 10)

Couprie, C., Grady, L., Najman, L., and Talbot, H. (2009). Power watersheds: A new image segmentation framework extending graph cuts, random walker and optimal spanning forest. In *Proceedings 12th International Conference on Computer Vision*. (cited on page 33)

Couprie, C., Grady, L., Najman, L., and Talbot, H. (2010). Power watershed: A unifying graph-based optimization framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. pre-print, accepted in August 2010. (cited on pages 33, 39, 130, 131, and 135)

Criminisi, A., Sharp, T., and Blake, A. (2008). Geos: Geodesic image segmentation. In *Proceedings 10th European Conference on Computer Vision*. (cited on pages 33, 38, and 84)

Criminisi, A., Sharp, T., Rother, C., and Perez, P. (2010). Geodesic image and video editing. In *ACM Transactions on Graphics*. to appear. (cited on pages 33 and 84)

Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., and Yannakakis, M. (1992). The complexity of multiway cuts (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. (cited on pages 90, 93, 97, and 98)

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 13 and 45)

Dice, L. R. (1945). Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302. (cited on page 111)

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271. (cited on page 18)

Ding, L. and Yilmaz, A. (2010). Interactive image segmentation using probabilistic hypergraphs. *Pattern Recognition*, 43(5):1863–1873. (cited on page 101)

Donoser, M., Urschler, M., Hirzer, M., and Bischof, H. (2009). Saliency driven total variation segmentation. In *Proceedings 12th International Conference on Computer Vision*. (cited on page 92)

Duchenne, O., Audibert, J.-Y., Keriven, R., Ponce, J., and Segonne, F. (2008). Segmentation by transduction. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 101 and 102)

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338. (cited on pages 13 and 100)

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874. (cited on pages 69 and 75)

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181. (cited on page 11)

Friedland, G., Jantz, K., and Rojas, R. (2005). Siox: Simple interactive object extraction in still images. *International Symposium on Multimedia*, 0:253–260. (cited on pages 28 and 101)

Garcia, V., Debreuve, E., and Barlaud, M. (2008). Fast k-nearest neighbor search using gpu. In *CVPR Workshop on Computer Vision on GPU*, Anchorage, Alaska, USA. (cited on page 75)

Gipp, M., Marcus, G., Harder, N., Suratanee, A., Rohr, K., Koenig, R., and Maenner, R. (2009). Haralick's texture features computed by gpus for biological applications. *International Journal of Computer Science*, 36:587–592. (cited on pages 54 and 58)

Gleicher, M. (1995). Image snapping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 183–190. (cited on page 18)

Goldschlager, L. M., Shaw, R. A., and Staples, J. (1982). The maximum flow problem is log space complete for p. *Theoretical Computer Science*, 21(1):105 – 111. (cited on page 26)

Gonzalez, R. C. and Woods, R. E. (2001). *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (cited on page 49)

Grady, L. (2005). Multilabel random walker image segmentation using prior models. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 29 and 37)

Grady, L. (2006). Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783. (cited on pages 26, 28, and 99)

Grady, L. and Funka-Lea, G. (2004). Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials. In *Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis, ECCV 2004 Workshops CVAMIA and MMBIA*. (cited on pages 28, 29, 37, 39, and 84)

Griesser, A., Roeck, S. D., Neubeck, A., and Gool, L. V. (2005). Gpu-based foreground-background segmentation using an extended colinearity criterion. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2005*. (cited on page 42)

Gulshan, V., Rother, C., Criminisi, A., Blake, A., and Zisserman, A. (2010). Geodesic star convexity for interactive image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 28, 39, 53, 99, 101, and 105)

Hackbusch, W. (1994). *Iterative solution of large sparse systems of equations*, volume 95 of *Applied Mathematical Sciences*. Springer-Verlag, New York. (cited on page 29)

Han, S., Tao, W., Wang, D., Tai, X.-C., and Wu, X. (2009). Image segmentation based on grabcut framework integrating multiscale nonlinear structure tensor. *IEEE Transactions on Image Processing*, 18(10):2289–2302. (cited on pages 28 and 53)

Haralick, R. M., Shanmugam, K., and Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621. (cited on pages 54, 56, and 139)

Hough, P. and Powell, B. (1960). A method for faster analysis of bubble chamber photographs. *Il Nuovo Cimento (1955-1965)*, 18:1184–1191. (cited on page 8)

Illingworth, J. and Kittler, J. (1988). A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116. (cited on page 8)

Ishikawa, H. (2003). Exact optimization for markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1333–1336. (cited on pages 90 and 97)

Jain, A. and Farrokhnia, F. (1990). Unsupervised texture segmentation using gabor filters. In *IEEE International Conference on Systems, Man and Cybernetics*. (cited on page 61)

Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331. (cited on pages 20 and 36)

Labatut, P., Keriven, R., and Pons, J.-P. (2006). A gpu implementation of level set multiview stereo. In *International Conference on Computational Science (4)*. (cited on page 42)

Laws, K. I. (1980). Rapid texture identification. In *Proc. SPIE Conf. Image Processing for Missile Guidance*, pages 376–380. (cited on page 61)

Leistner, C., Saffari, A., Santner, J., and Bischof, H. (2009). Semi-supervised random forests. In *Proceedings 12th International Conference on Computer Vision*. (cited on page 141)

Lellmann, J., Kappes, J., Yuan, J., Becker, F., and Schnörr, C. (2009). Convex multi-class image labeling by simplex-constrained total variation. In *SSVM '09: Proceedings of the Second International Conference on Scale Space and Variational Methods in Computer Vision*. (cited on page 96)

Lempitsky, V., Kohli, P., Rother, C., and Sharp, T. (2009). Image segmentation with a bounding box prior. In *Proceedings 12th International Conference on Computer Vision*. (cited on pages 28 and 101)

Leung, T. and Malik, J. (1999). Recognizing surfaces using three-dimensional textons. In *Proceedings 7th International Conference on Computer Vision*. (cited on page 61)

Leung, T. and Malik, J. (2001). Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44. (cited on pages 59 and 61)

Li, Y., Sun, J., Tang, C. K., and Shum, H. Y. (2004). Lazy snapping. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 303–308. (cited on pages 26, 37, and 99)

Liu, J., Sun, J., and Shum, H. Y. (2009). Paint selection. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*. (cited on pages 27, 39, and 141)

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings 8th International Conference on Computer Vision*. (cited on pages 12, 100, and 103)

McGuinness, K. and O'Connor, N. E. (2010). A comparative evaluation of interactive segmentation algorithms. *Pattern Recognition*, 43(2):434 – 444. (cited on pages 99 and 105)

Meyer, F. (1994). Topographic distance and watershed lines. *Signal Processing*, 38(1):113–125. (cited on page 10)

Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85. (cited on page 40)

Mortensen, E. N. and Barrett, W. A. (1995). Intelligent scissors for image composition. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 191–198. (cited on pages 16 and 36)

Mortensen, E. N., Reese, L. J., and Barrett, W. A. (2000). Intelligent selection tools. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 18)

Mumford, D. and Shah, J. (1989). Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685. (cited on page 10)

Ojala, T. and Pietikainen, M. (1999). Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477 – 486. (cited on page 63)

Ojala, T., Pietikainen, M., and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987. (cited on pages 63, 64, and 139)

Olsson, C., Byröd, M., Overgaard, N. C., and Kahl, F. (2009). Extending continuous cuts: Anisotropic metrics and expansion moves. In *Proceedings 12th International Conference on Computer Vision*. (cited on pages 93 and 97)

Osher, S. J. and Fedkiw, R. P. (2002). *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 1 edition. (cited on page 22)

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66. (cited on pages 6 and 7)

Pietikainen, M., Nieminen, S., Marszalec, E., and Ojala, T. (1996). Accurate color discrimination with classification based on feature distributions. In *Proceedings of 13th International Conference on Pattern Recognition*. (cited on page 51)

Pock, T., Chambolle, A., Cremers, D., and Bischof, H. (2009). A convex relaxation approach for computing minimal partitions. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 46, 94, 96, 97, 98, 115, and 140)

Pock, T., Schoenemann, T., Graber, G., Bischof, H., and Cremers, D. (2008). A convex formulation of continuous multi-label problems. In *Proceedings 10th European Conference on Computer Vision*. (cited on pages 90 and 97)

Price, B. L., Morse, B., and Cohen, S. (2010). Geodesic graph cut for interactive image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 33, 39, and 101)

Protiere, A. and Sapiro, G. (2007). Interactive image segmentation via adaptive weighted distances. *IEEE Transactions on Image Processing*, 16(4):1046–1057. (cited on pages 32, 38, and 53)

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106. (cited on page 71)

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (cited on page 71)

Randen, T. and Husoy, J. (1999). Filtering for texture classification: a comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):291 –310. (cited on page 61)

Randen, T. and John, J. H. k. (1994). Multichannel filtering for image texture segmentation. *Optical Engineering*, 33(8):2617–2625. (cited on page 61)

Reese, L. and Barrett, W. A. (2002). Image editing with intelligent paint. In *Proceedings of Eurographics*. (cited on pages 18 and 99)

Robertson, A. R. (1977). The cie 1976 color-difference formulae. *Color Res.Appl.*, 2:7–11. (cited on page 51)

Rother, C., Kolmogorov, V., and Blake, A. (2004). Grabcut: interactive foreground extraction using iterated graph cuts. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 309–314. (cited on pages 26, 37, 100, 101, and 111)

Russell, B., Torralba, A., Murphy, K., and Freeman, W. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173. (cited on page 100)

Ruzon, M. A. and Tomasi, C. (2000). Alpha estimation in natural images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 19)

Sahoo, P. K., Soltani, S., Wong, A. K., and Chen, Y. C. (1988). A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2):233–260. (cited on page 6)

Santner, J., Pock, T., and Bischof, H. (2010). Interactive multi-label segmentation. In *Proceedings 10th Asian Conference on Computer Vision*. (cited on page 46)

Santner, J., Unger, M., Pock, T., Leistner, C., Saffari, A., and Bischof, H. (2009). Interactive texture segmentation using random forests and total variation. In *Proceedings 20th British Machine Vision Conference*. (cited on pages 45, 53, 69, and 99)

Schmid, C. (2001). Constructing models for content-based image retrieval. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 61)

Schroff, F., Criminisi, A., and Zisserman, A. (2008). Object class segmentation using random forests. In *Proceedings 19th British Machine Vision Conference*. (cited on page 13)

Sethian, J. A. (1999). *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 2 edition. (cited on page 22)

Sharp, T. (2008). Implementing decision trees and forests on a gpu. In *Proceedings 10th European Conference on Computer Vision*. (cited on pages 43, 69, and 72)

Shi, J. and Malik, J. (1997). Normalized cuts and image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 10)

Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 13)

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proceedings 9th European Conference on Computer Vision*. (cited on pages 13, 59, and 101)

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2009). Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23. (cited on page 101)

Sinop, A. K. and Grady, L. (2007). A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *Proceedings 11th International Conference on Computer Vision*. (cited on pages 30, 33, and 99)

Smith, A. R. (1978). Color gamut transform pairs. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 12–19. (cited on pages 50 and 51)

Sonka, M., Hlavac, V., and Boyle, R. (2007). *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering. (cited on pages 5 and 8)

Strandmark, P. and Kahl, F. (2010). Parallel and distributed graph cuts by dual decomposition. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 26)

Tan, K.-H. and Ahuja, N. (2001). Selecting objects with freehand sketches. In *Proceedings 8th International Conference on Computer Vision*. (cited on page 19)

Toivanen, P. J. (1996). New geodesic distance transforms for gray-scale images. *Pattern Recognition Letters*, 17(5):437–450. (cited on page 32)

Unger, M., Pock, T., Trobin, W., Cremers, D., and Bischof, H. (2008). Tvseg - interactive total variation based image segmentation. In *Proceedings 19th British Machine Vision Conference*. (cited on pages 31, 38, and 99)

Varma, M. and Zisserman, A. (2002). Classifying images of materials: Achieving viewpoint and illumination independence. In *Proceedings 7th European Conference on Computer Vision*. (cited on page 62)

Varma, M. and Zisserman, A. (2009). A statistical approach to material classification using image patch exemplars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2032 – 2047. (cited on page 54)

Veksler, O. (2008). Star shape prior for graph-cut image segmentation. In *Proceedings 10th European Conference on Computer Vision*. (cited on pages 27, 38, 39, 53, and 99)

Vicente, S., Kolmogorov, V., and Rother, C. (2009). Joint optimization of segmentation and appearance models. In *Proceedings 12th International Conference on Computer Vision*. (cited on page 26)

Vincent, L. and Soille, P. (1991). Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598. (cited on pages 9, 18, 19, and 26)

Werlberger, M., Pock, T., Unger, M., and Bischof, H. (2009). A variational model for interactive shape prior segmentation and real-time tracking. In *International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*, Voss, Norway. (cited on page 53)

Wu, Z. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101 –1113. (cited on page 10)

Yang, R. and Pollefeys, M. (2003). Multi-resolution real-time stereo on commodity graphics hardware. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 43)

Yatziv, L., Bartesaghi, A., and Sapiro, G. (2006). O(n) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212(2):393 – 399. (cited on page 32)

Zach, C., Gallup, D., Frahm, J.-M., and Niethammer, M. (2008). Fast global labeling for real-time stereo using multiple plane sweeps. In *Proceedings of the Vision, Modeling, and Visualization Conference 2008 (VMV 2008)*. (cited on page 96)

Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime tv-l1 optical flow. In *Proceedings 29th DAGM Pattern Recognition Symposium*. (cited on page 43)

Zhang, J., Zheng, J., and Cai, J. (2010). A diffusion approach to seeded image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. (cited on pages 30, 39, and 101)

Zucker, S. W. (1976). Region growing: Childhood and adolescence. *Computer Graphics and Image Processing*, 5(3):382 – 399. (cited on page 19)