Compression of visual data into symbol-like descriptors in terms of a cognitive real-time vision system

Dissertation

ZUR ERLANGUNG DES MATHEMATISCH-NATURWISSENSCHAFTLICHEN DOKTORGRADES
"DOKTOR RERUM NATURALIUM" DER GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

vorgelegt von

Alexey Abramov

aus Moskau, Russland

Göttingen 2012

Referent: Prof. Dr. Florentin Wörgötter
Koreferent: Prof. Dr. Winfried Kurth
Tag der mündlichen Prüfung: 18/07/2012

Abstract

Humans have five main senses: sight, hearing, touch, smell, and taste. Most of them combine several aspects. For example vision addresses at least three perceptual modalities: motion, color, and luminance. Extraction of these modalities begins in the human eye in the retinal network and the preprocessed signals enter the brain as streams of spatio-temporal patterns. As vision is our main sense, particularly for the perception of the three dimensional structure of the world around us, major efforts have been made to understand and simulate the visual system based on the knowledge collected to date.

The research done over the last decades in fields of *image processing* and *computer vision* coupled with a tremendous step forward in hardware for parallel computing opened the door to building of so-called *cognitive vision systems* and for their incorporation into robots. The goal of any cognitive vision system is to transform visual input information into more descriptive representations than just color, motion, or luminance. Furthermore, in most robotic systems "live" interactions of robots with the environment are required, greatly increasing demands on the system. In such systems all pre-computations of the visual data need to be performed in real-time in order to be able to use the output data in the perception-action loop. Thus, a central goal of this thesis is to provide techniques which are strictly compatible with real-time computation.

In the first part of this thesis we investigate possibilities for the powerful compression of the initial visual input data into symbol-like descriptors, upon which abstract logic or learning schemes can be applied. We introduce a new real-time video segmentation framework performing automatic decomposition of monocular and stereo video streams without use of prior knowledge on data and considering only preceding information. All entities in the scene, representing objects or their parts, are uniquely identified.

In the second part of the thesis we make additional use of stereoscopic visual information and address the problem of establishing correspondences between two views of the scene solved with apparent ease in the human visual system (for images acquired with left and right eye). We exploit these correspondences in the stereo image pairs for the estimation of depth (distance) by proposing a novel disparity measurement technique based on extracted stereo-segments. This technique approximates shape and computes depth information for all entities found in the scene. The most important and novel achievement of this approach is that it produces reliable depth information for objects with weak texture where performance of traditional stereo techniques is very poor.

In the third part of this thesis we employ an active sensor, producing indoors much more precise depth information encoded as range-data than any passive stereo

technique. We perform fusion of image and range data for video segmentation which results in better results. By this we can now even handle fast moving objects, which was not possible so far.

To address the real-time constraint, the proposed segmentation framework was accelerated on a Graphics Processing Unit (GPU) architecture using the parallel programming model of Compute Unified Device Architecture (CUDA). All introduced methods: segmentation of single images, segmentation of monocular and stereo video streams, depth-supported video segmentation, and disparity computation from stereo-segment correspondences run in real-time for middle-size images and close to real-time for higher resolutions.

In summary: The main result of this thesis is a framework which can produce a compact representation of any visual scene where all meaningful entities are uniquely identified, tracked, and important descriptors, such as shape and depth information, are extracted. The ability of the framework was successfully demonstrated in the context of several European projects (PACO-PLUS, Garnics, IntellAct, and Xperience). The developed real-time system is now employed as a robust visual front-end in various real-time robotic systems.

Contents

Citations to Related Publications

Large portion of Chapter 2 has appeared in the following paper:

> Abramov, A., Kulvicius, T., Wörgötter, F., and Dellen, B. (2010). Real-time image segmentation on a GPU. *Facing the Multicore-Challenge, Lecture Notes in Computer Science*, 6310, 131-142.

Most of Chapters 3 and 4 has appeared in the following papers:

> Abramov, A., Aksoy, E. E., Dörr, J., Pauwels, K., Wörgötter, F., and Dellen, B. (2010). 3D semantic representation of actions from efficient stereo-image-sequence segmentation on GPUs, *Fifth International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT).*

> Abramov, A., Pauwels, K., Papon, J., Wörgötter, F., and Dellen, B. (2012). Real-time segmentation of stereo videos on a portable system with a mobile GPU, *IEEE Transactions on Circuits and Systems for Video Technology* (in press).

Most of Chapter 5 has been submitted as

> Abramov, A., Pauwels, K., Kornewald, W., Wörgötter, F., and Dellen, B. Real-time dense disparity from stereo-segment silhouettes for weakly-textured images. Submitted to *International Journal of Computer Vision* in June 2012.

Finally, Chapter 6 appears in its entirety as

> Abramov, A., Papon, J., Pauwels, K., Wörgötter, F., and Dellen, B. (2012). Depth-supported real-time video segmentation with the Kinect. *IEEE workshop on the Applications of Computer Vision (WACV)*, 457-464.

Acknowledgments

This thesis would not have been possible without the support of many friends and colleagues. First of all I would like to thank my supervisors Prof. Dr. Florentin Wörgötter and Dr. Babette Dellen for guiding me through my research by sharing their experiences with me and for many fruitful discussions without which this work would not have been done. I thank Prof. Dr. Florentin Wörgötter for giving me a chance to work on computer vision in his group in Germany, for the opportunity to present my research at conferences, and for the possibility of research visits (Barcelona, Granada, Leuven, Odense, Innsbruck, Jülich). I also thank his wife and secretary Ursula for always being very helpful and friendly. I thank Dr. Babette Dellen for numerous useful advices during all these years and especially for spending so much time on reviewing my thesis and giving such a valuable feedback.

My special thanks go to Dr. Tomas Kulvicius and Dr. Karl Pauwels who made an outstanding contribution to this work. Also I thank all members of our very friendly, creative, and talkative vision group: Eren Erdal Aksoy, Johannes Dörr, Waldemar Kornewald, Jeremie Papon, Simon Reich, Markus Schöler, Johannes Widenka. I would like to thank Prof. Dr. Eduardo Ros and Dr. Javier Díaz from the machine vision group in Granada as well as Dr. Anders Kjær-Nielsen and Dr. Lars Baunegard With Jensen from the computer vision group in Odense.

I am very grateful to Dr. Christoph Kolodziejski, Dr. Tomas Kulvicius, Dr. Irene Markelić, Christian Tetzlaff, and Alexander Wolf for helping me in the everyday life during my first days in Germany. Thank you very much indeed!

I thank all members of Florentin's group. It was a great pleasure to be a part of it which was much more than a research group: Mohamad Javad Aein, Dr. Alejandro Agostini, Martin Biehl, Jan-Matthias Braun, Dr. Markus Butz, Sakyasingha Dasgupta, Faramarz Faghihi, Michael Fauth, Dennis Goldschmidt, Dr. Frank Hesse, Dr. Guoliang Liu, Timo Nachstedt, Dr. KeJun Ning, Dr. Poramate Manoonpong, Chanwit Musika, Vishal Patel, Harm-Friedrich Steinmetz, Dr. Minija Tamosiunaite, Birk Urmersbach, Thomas Wanschik, Xiaofeng Xiong, and Steffen Zenker.

Also I am greatly appreciate everyone who helped me to fill the scientific breaks with sports, traveling, fun, and beer. I thank all guys from our hobby football teams at the Groner Freibad and in the university league of Göttingen, notably Niels Clausen, Christoph Kornitzky, Patrick Mielke, Phillip Oberdorfer, Julian Plagemann, Karsten Thieleking, and Dr. Qui Van. It was always a great fun to play despite the final score. Beyond that, Kicker (tabletop football) games after lunch and in Thanner's including experts such as Felix von Denkowski, Phillip Kroehn, and Timo Reinhold were indeed a very big part of my scientific work in Göttingen too!

Furthermore, I thank all my friends from Russia who never forgot me and were very happy to see me every time in Moscow. The way how you supported me being so far away is fantastic: Sergey Archangelskiy, Tigran Ayrapetyanc, Sergey Blagodurov, Ekaterina Epik, Feodor Ivchenko, Olga Karpova, Stanislav Kolupanskiy, Sofia Mikhailova, Natasha Panteleeva, Olga Pulkina, Yuri Shaykevich, Alexey Teesheen,

Vasily Troshkin, Andrey Yudakov. A special thanks goes to my German teacher Olga Fomina from the Goethe-Institut in Moscow whose excellent professional skills simplified a lot my life in Germany.

Last but not least, I want to thank my family. I am very grateful to my parents Alexander and Liudmila without whom I would not have achieved all that in my life what I have now. It is impossible to put into words how much your support means for me. Also I would like to thank my sister Nadia who never forgot her younger brother. Thank you very much for supporting me in all I am doing and being always by my side no matter what!

"Life did not intend to make us perfect.
Whoever is perfect belongs in a museum."

Erich Maria Remarque (1898 – 1970)

Dedicated to my father Alexander,
my mother Liudmila,
and my sister Nadia.

List of Symbols and Notations

The list below contains the mathematical symbols and notations that are used most frequently throughout the thesis.

$\boldsymbol{q}$ – the number of spin states in the Potts model

$\boldsymbol{g_1, g_2, \ldots, g_N}$ – color vectors in the image of N pixels

$\boldsymbol{\sigma_k}$ – a spin variable

$\boldsymbol{w_1, w_2, \ldots, w_q}$ – spin states

$\boldsymbol{S_1, S_2, \ldots, S_n}$ – spin state configurations

$\boldsymbol{\Omega}$ – the space of all spin state configurations

$\hbar$ – a set of new possible spin state configurations

$\boldsymbol{\Lambda}$ – a graph structure defined on the domain Ω

$\boldsymbol{\Lambda(S)}$ – spin configurations that are neighbors of $S \in \Omega$

$\boldsymbol{H[S]}$ – a global energy function of the spin state configuration $S \in \Omega$

$\boldsymbol{J_{ij}}$ – an interaction strength between two spins i and j

$\boldsymbol{\delta_{ij}}$ – the Kronecker delta

$\boldsymbol{\ell}$ – a constant defining 2D neighborhood

$\boldsymbol{\Delta_{ij}}$ – the color difference between color vectors $\boldsymbol{g_i}$ and $\boldsymbol{g_j}$

$\overline{\boldsymbol{\Delta}}$ – the mean color distance averaged over all neighborhoods in the image

$\boldsymbol{\alpha}$ – a system parameter

$\boldsymbol{r}$ – a control parameter for the global inhibition

$\boldsymbol{T}$ – the system temperature

$\boldsymbol{T_0}$ – a starting temperature in the simulated annealing schedule

$\boldsymbol{\gamma}$ – the simulated annealing factor

$\boldsymbol{n_1}$ – number of the basic Metropolis iterations

$\boldsymbol{n_2}$ – number of the relaxation Metropolis iterations

$\boldsymbol{\alpha_1}$ – the factor used during the basic n_1 spin updates

$\boldsymbol{\alpha_2}$ – the factor used during the relaxation n_2 spin updates

$\boldsymbol{C}$ – segmentation covering

$\boldsymbol{\theta_p}$ – an orientation of the complex Gabor filter

$\boldsymbol{\omega_0}$ – a peak frequency of the Gabor filter

$\boldsymbol{\sigma_G}$ – a spatial extension of the Gabor filter

$\boldsymbol{f_p(x)}$ – the Gabor filter at pixel location $\boldsymbol{x}(x, y)^T$

$\boldsymbol{R_p(x)}$ – responses of the Gabor filter

$\boldsymbol{\rho_p(x)}$ – the amplitude of the quadrature filter pair

$\boldsymbol{\phi_p(x)}$ – the phase component of the quadrature filter pair

$\mathbf{v}$ – the optical flow vector at pixel location $\boldsymbol{x} = (x, y)^T$

$\boldsymbol{\psi}_p(x)$ – the temporal phase gradient

$\boldsymbol{v}_y$ – the vertical component of the optical flow vector

$\boldsymbol{v}_x$ – the horizontal component of the optical flow vector

$\boldsymbol{\delta}_p(x)$ – a disparity estimate at pixel location $\boldsymbol{x} = (x, y)^T$

$\boldsymbol{d}$ – a disparity map estimated by the phase-based technique

$\boldsymbol{\eta}$ – a sparsity level of the disparity map

$\boldsymbol{\mu}$ – an entropy value of the neighborhood around the corresponding pixel

$\boldsymbol{d_C}$ – an estimated disparity map

$\boldsymbol{d_T}$ – a ground truth disparity map

$\boldsymbol{d_A}$ – an average line disparity map

$\boldsymbol{d_E}$ – an edge disparity map

$\boldsymbol{\chi^2}$ – the merit function for the linear last squares

$\boldsymbol{a_1, \ldots, a_M}$ – parameters of the surface model

$\boldsymbol{\vartheta}$ – a measurement error for disparity from stereo-segment correspondences

$\boldsymbol{\varphi_i(x, y)}$ – a basis function

1 Introduction

"Vision is the art of seeing the invisible"

– Jonathan Swift

Visual perception is the ability to interpret information from light reaching the eye. The resulting perception is also known as *vision*. The human visual system is of extreme complexity which is not yet fully understood and whose research can still take many decades. However, it is known that the human visual system has low, middle, and high levels of the visual perception. The low level deals with tasks such as detecting colors, finding edges, locating objects in space. On the middle level detected objects are segregated from the background and object features are determined. Finally, the high level performs recognition of objects in the visual scene. Last achievements in fields of image processing and computer vision in conjunction with an enormous progress over the last decades in hardware for parallel computing opened the door to building a so-called *cognitive vision system* and its incorporation into robots.

Visual perception is a part of the perception-action loop which is the fundamental logic of the nervous system. Perception and action processes are functionally interrelated and feedback to each other in such a way that perception informs action and action informs perception. Many robotic systems try to replicate the perception-action loop with robots where the cognitive vision system does the cognitive visual part to close the loop between sensors and robots. The goal of the vision system is to transform input visual information presented by color, motion, or luminance into some kind of descriptors presenting objects or their parts. Such a symbol-like representation is a compression of the visual input where all entities of the scene are detected, identified, and relations between various objects or their parts are established. This representation of the visual input is quite sufficient and can be used for performing actions aimed at objects.

Due to many sources of noise or uncertainty in the formation and processing of the visual information, the cognitive visual system can erroneously perceive locations, appearances, and motions of detected objects. This effect is known as *an*

uncertainty principle in vision and includes the crucial aperture and correspondence problems (Forsyth and Ponce, 2002). Establishing correspondences between images acquired from different view points or adjacent frames of a video stream is one of the most fundamental problems in computer vision, as information about correspondences concludes about the 3D structure of the scene, its motion, and the state of present objects.

Over the last decades various approaches for the computation of correspondences have been proposed. Generally, correspondences can be classified in the two following categories: *local correspondences* and *region correspondences*. Local correspondences are established between certain pixels or local image features, whereas region correspondences are established between whole regions or segments of input images that need to be matched. Algorithms for the computation of disparity (Scharstein and Szeliski, 2002) and correspondent feature descriptors (Snavely et al., 2008) are the most famous approaches for computation of local correspondences between multiple views, e.g., stereo image pairs. Optical flow algorithms estimate local correspondences between sequential frames t and $t+1$ of a video stream (Wedel et al., 2008; Pauwels et al., 2011; Brox and Malik, 2011). However, in many cases the ambiguity of local descriptors does not allow an assignment of unique correspondences, especially in weakly-textured areas (see Fig. 1.1(A)). This is known as *the correspondence problem*. Region matching techniques, on the contrary, use region-based descriptors instead of pixels or local image features, e.g., starting from an independent segmentation of the images. The obtained segments are then matched based on their region features and structure (Hedau et al., 2008; Brendel and Todorovic, 2009), local geometric relations among regions (Lee and Lei, 1990), or graph-based representations (Wang and Abe, 1995). But if the visual scene undergoes even small changes in perspective, lighting, or when objects in the scene are moving, the structure and shape of corresponding regions might not match anymore, leading to ambiguous or wrong correspondences (see Fig. 1.1(B)). Furthermore, the segmentation method itself might produce different results (robustness problem) from image to image due to illumination or composition changes in the scene. But despite these fundamental ambiguities and the complexity of the problem, the human visual system solves these issues with a performance unreachable by any state-of-the-art computer vision method in terms of the both precision and time.

In this thesis a novel framework based on the combination of both local and region correspondences for establishing matchings between stereo images, frames in monocular video streams, and frames in stereo video streams is proposed. This conjoint framework is automatic, does not use prior knowledge about the data, and considers only preceding information, as future perception is undefined. Local correspondences, found using stereo or optical flow techniques, are used in the framework to find matchings between segments in multiple view images or frame sequences, respectively (see Fig. 1.1(C)). The fusion of both correspondence types helps to improve and accelerate the matching procedure as compared to both approaches applied separately. We

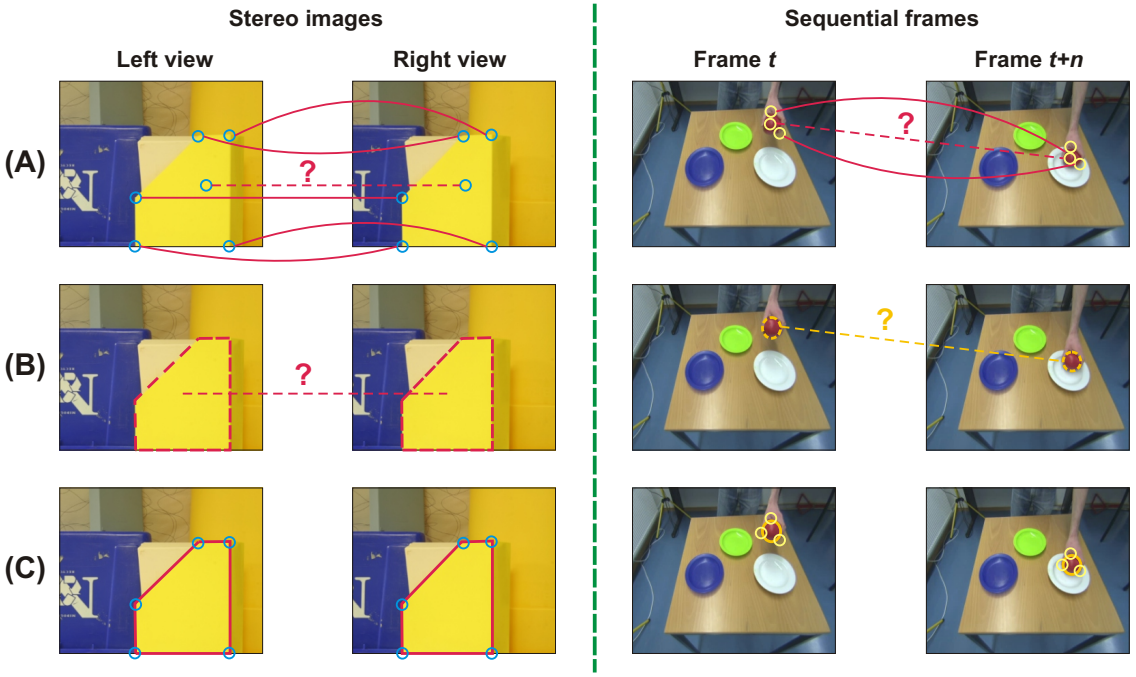

Figure 1.1: Establishing correspondences between two views of the same object in the stereo image and in the video stream. (A) Local correspondences computed mainly at points with high structure without considering object surfaces. (B) Matching of segments obtained via any segmentation technique faces the problem that segments can be deformed between the reference and matching views due to perspective changes or motion leading to the lack of segment matches. (C) Combining point correspondences and image segmentation in a conjoint framework allows consistent segmentation of stereo and sequential views.

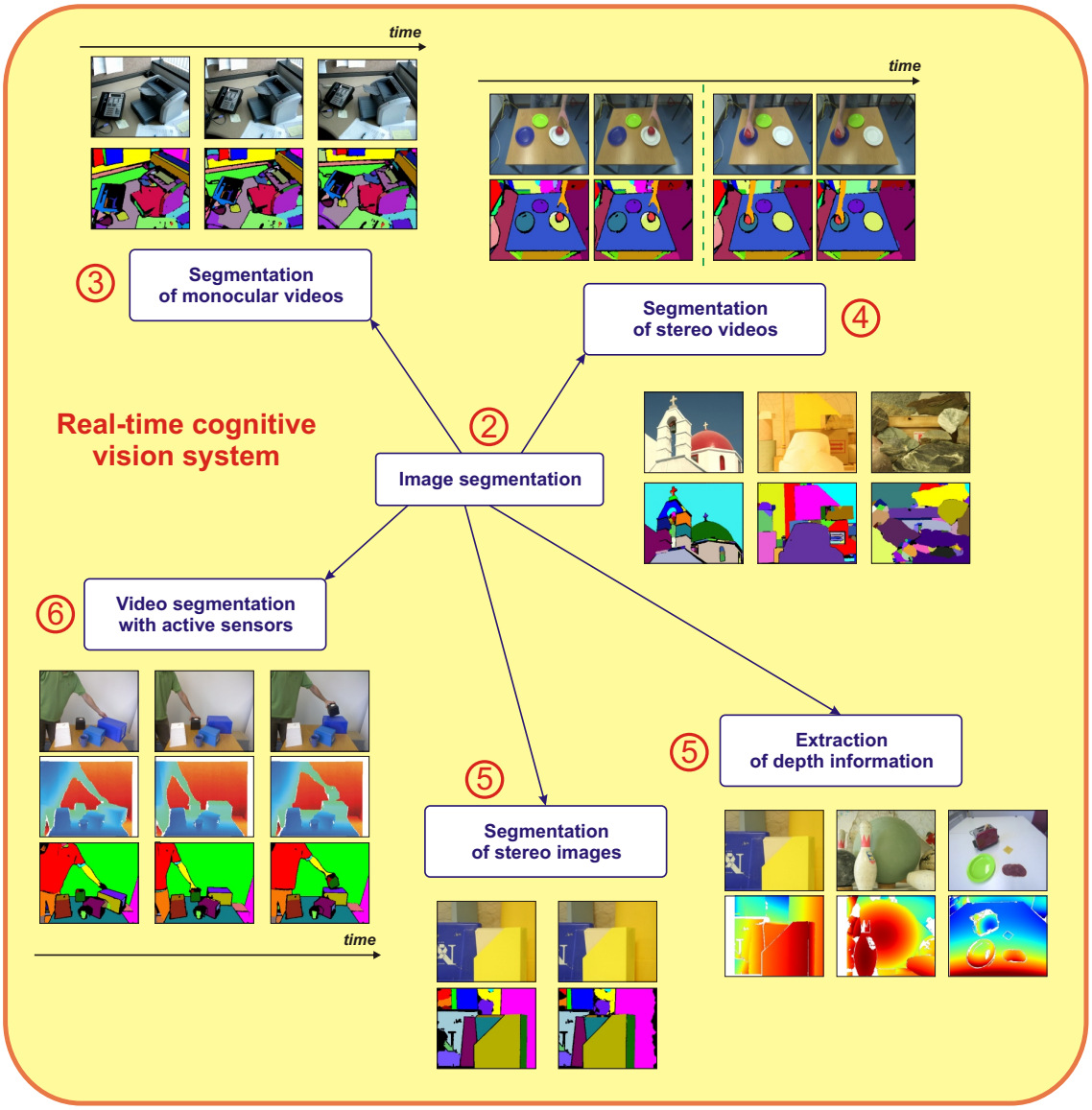

Figure 1.2: A structure of the thesis. Numbers at blocks show the chapter numbers. The chapters 2 and 3 should be read first. Other chapters can be read in an arbitrary order.

present a novel approach for video segmentation based on a very efficient segmentation technique coupled with a mechanism for the transfer of found segments from one view to another one in the case of stereo images and from frame t to frame $t+1$ in the case of frame sequences. In both cases available local correspondences are employed in order to transfer segments between images. Since most of robotic systems require "live" interactions with the environment, demands on the framework in terms of the processing time are extremely high. Therefore, all pre-computations of the visual data need to be performed in real-time to use output data in the perception-action loop.

An overview of the thesis is presented in Fig. 1.2. A novel image segmentation technique based on fundamental principles known from classical physics is introduced in Chapter 2. It is the central part of this work. The method can be used in a very efficient way for the segmentation of monocular and stereo video streams (Chapters 3 and 4). Video segmentation supported by the depth information produced by an active sensor is presented in Chapter 6. Matches between stereo images are used for extraction of 3D information about the scene (Chapter 5). Time performance is a very important issue in this study and only real-time or close to real-time solutions were considered here [1]. All parts of the thesis were integrated into the real-time modular cognitive computer vision system which can serve as a visual front-end for robotic applications (Papon et al., 2012).

Each chapter starts with its own Introduction section, where we discuss the state of the art and our goals in relation to the topic, and ends with Discussion section where our approach is compared to other conventional methods with respect to quality of results and time performance. We will conclude the thesis with Chapter 7 where all findings are summarized and an outlook for future investigations is given.

[1] By real-time we understand processing of a full frame at 25Hz or faster.

2

Real-time Image Segmentation on a GPU

"There are no lines in nature, only
areas of color, one against another"

– Édouard Manet

2.1 Introduction

Image segmentation, i.e., the partitioning of an image into disjoint parts based on some image characteristics, such as color information, intensity, texture or range data, is one of the most fundamental tasks in computer vision and image processing and of large importance for many kinds of applications, e.g., object tracking, classification and recognition (Szeliski, 2010). A formal definition of image segmentation can be given as follows (Pal and Pal, 1993): if $\Phi()$ is a homogeneous predicate defined on groups of connected pixels Ψ, the segmentation is a partition of the set Ψ into connected subsets or regions $(R_1, R_2, \ldots, R_n)$ in such a way that

$$\bigcup_{i=1}^{n} R_i = \Psi \;\; with \;\; R_i \cap R_j = \emptyset \;\; (i \neq j). \tag{2.1}$$

The uniformity predicate $\Phi(R_i) = true$ for all regions R_i, and $\Phi(R_i \cup R_j) = false$, when $i \neq j$ and R_i and R_j are neighbors.

2.1.1 Conventional image segmentation techniques

Finding a suitable splitting of an image into regions is not a trivial task, since it is pretty much unknown how the desired result should look like which depends very often on a specific application. As a consequence, many different approaches for image segmentation have been proposed during the past three decades. Based on the technique used for finding segments all methods can be classified in the following groups: active contours (Blake and Isard, 1998; Mortensen and Barrett, 1999), watershed (Vincent

and Soille, 1991; Beare, 2006), clustering (Ohlander et al., 1978; Brice and Fennema, 1970; Swendsen and Wang, 1987; Wolff, 1989; von Ferber and Wörgötter, 2000), graph-based (Felzenszwalb and Huttenlocher, 2004), mean shift (Comaniciu et al., 2002; Paris and Durand, 2007), graph cuts and energy-based methods (Estrada et al., 2004; Boykov and Funka-Lea, 2006; Lempitsky and Boykov, 2007; Vicente et al., 2008), normalized cuts (Shi and Malik, 2000; Cour et al., 2005), and contour relaxation (Mester et al., 2011).

Active contour methods, also known as snakes, tend to detect and track object boundaries in the image. Active contours are initialized manually by boundary guesses and optimal object boundaries are found iteratively due to minimization of energy associated with initial contours (Blake and Isard, 1998). In some situations erroneous initial boundaries require additional input information from the user to get the desired curve. Mortensen and Barrett (1999) proposed so-called *intelligent scissors* that optimize the contour simultaneously with the user initialization which makes the whole procedure faster and leads to better results. Segments obtained by active contours are represented by areas enclosed by contours.

Watershed computation is one of the oldest image segmentation techniques. It is based on the thresholding of a grayscale image which is considered as a topographic relief. Grayscale values of pixels represent the point elevations in the relief. The segmentation is achieved by flooding water in each relief minimum and applying a threshold to find a watershed line (Vincent and Soille, 1991). Watershed segmentation associates a unique region with each local minimum which can cause over-segmentation. Therefore, watershed segmentation requires the provision of seed locations (specified interactively by a user) determining centers of desired segments (Beare, 2006).

Clustering is another old segmentation approach having various modifications. The input image is divided into regions called *clusters* based on some image characteristics such that any two pixels from the same region are more similar than any two pixels belonging to different regions [1]. Ohlander et al. (1978) proposed a *region splitting* technique that first computes a color histogram for the whole image and then partitions it into regions having bin differences higher than a pre-defined threshold. An opposite procedure is *region growing* that first defines a similarity criterion and then merges pixels and regions fulfilling it (Brice and Fennema, 1970). *K-means clustering* is built upon a natural objective function based on the assumption that the number of clusters k is known and each cluster is assumed to have a center. The algorithm chooses cluster centers randomly and updates iteratively each cluster center location considering pixels that are closest to each center (Bishop, 2006). This process converges eventually to a local minimum of the objective function, but it is not guaranteed to converge to its global minimum. The biggest drawback of this method is that the number of clusters k is an input parameter and a bad choice of k

[1] Due to possible reflections and varying lightness within one object, it is more correct to say "any two neighboring pixels".

may lead to poor clustering results. *Superparamagnetic clustering* methods describe image pixels as interacting granular ferromagnets featured by oriented vectors called *spins*. Depending on the temperature, i.e., disorder introduced to the system, the spin system can be in the paramagnetic, superparamagnetic, or ferromagnetic phase. In the ferromagnetic phase, all spins are aligned, while in the paramagnetic phase the system is in a state of complete disorder. In the superparamagnetic phase regions of aligned spins coexist and correspond to a natural partition of an image (Blatt et al., 1996). Finding the image partition corresponds to the computation of the equilibrium states of the system (Geman and Geman, 1984; von Ferber and Wörgötter, 2000; Swendsen and Wang, 1987; Wolff, 1989).

In *graph-based* methods an image is represented by a weighted undirected graph where nodes define pixels or small groups of pixels and edge weights define similarity between neighbors in the graph. To date the graph-based method proposed by Felzenszwalb and Huttenlocher (2004) based on relative dissimilarities between regions is one of the most powerful and fastest methods for segmentation. This method segments an image by merging regions according to internal and external differences defined for every region. The method produces a segmentation that is neither too fine nor too coarse, i.e., there are no regions that need to be split in multiple regions or merged to one region.

Mean shift techniques associate feature vectors with every pixel of an image (e.g., position, color, texture, range values, etc.). Feature vectors are used as samples for estimation of the probability density function that needs to be segmented. Mean shift computes initially a weighted mean of feature vectors within a local neighborhood in feature space (centered at each pixel's feature vector) and finds peaks in the distribution. Regions of feature space climbing to the same peak tend to belong to one segment. A crucial aspect of this approach is the determination of peaks in the high-dimensional data distribution without computing the distribution function explicitly (Cheng, 1995; Comaniciu et al., 2002; Paris and Durand, 2007).

Graph cuts and energy-based methods formulate the image segmentation task as a binary Markov random field (MRF). In these methods a pixel-based energy function is associated with an image consisting of the region and boundary terms. Despite various existing techniques for MRF energy minimization, the graph-based approach proposed by Boykov and Jolly (2001) is still the most commonly used for solving binary MRF problems. More recent approaches use some knowledge about objects and involve connectivity and shape priors in the segmentation process (Vicente et al., 2008; Lempitsky and Boykov, 2007).

The *normalized-cuts* technique proposed by Shi and Malik (2000) uses a graph-based representation of an image and tries to separate pixels or groups of pixels connected by weak edges (low similarity). The quality of the segmentation results depends on the segmentation measure defining the cut between regions (Cour et al., 2005).

The *contour relaxation* approach introduced by Mester et al. (2011) combines a

target function, called also "energy function", obtained from a statistical region-based image model, and an optimization technique for "contour relaxation". The method is based on the two following assumptions: the feature values (texture, color, motion, etc.) at various *pixel sites* obey the same distribution within a region, they are pairwise statistically independent; the feature values in the different *feature channels* at each pixel site are statistically independent of each other (between the channels) (Mester et al., 2011).

Among all these techniques we can distinguish between parametric (model-driven or nonautomatic) (Vincent and Soille, 1991; Blake and Isard, 1998; Mortensen and Barrett, 1999; Boykov and Jolly, 2001; Boykov and Funka-Lea, 2006; Beare, 2006; Bishop, 2006; Lempitsky and Boykov, 2007; Vicente et al., 2008) and nonparametric (data-driven, automatic, or unsupervised) techniques (Swendsen and Wang, 1987; Wolff, 1989; von Ferber and Wörgötter, 2000; Shi and Malik, 2000; Cour et al., 2005; Comaniciu et al., 2002; Felzenszwalb and Huttenlocher, 2004). Note that some techniques (Mester et al., 2011) can run in both automatic and nonautomatic modes. If little is known about the data being segmented, nonparametric methods have to be applied, while parametric methods require user input or some prior knowledge about objects in the scene.

Since the current study is focused on a condensed representation of the visual scene over time without prior knowledge of the data (see Chapter 1), we are only interested in image segmentation techniques which: (i) run without user input and do not need assumptions about the number of objects present in the scene (i.e., automatic); (ii) can be used for the video segmentation problem; (iii) run in real-time or close to real-time. Although the most famous and efficient image segmentation techniques such as normalized-cuts (Shi and Malik, 2000; Cour et al., 2005), graph-based (Felzenszwalb and Huttenlocher, 2004), and mean shift (Comaniciu et al., 2002) are automatic, they operate on single images and cannot be applied directly to the video segmentation problem because the segmentations of adjacent frames will be incoherent, i.e., segments of the same object carry different labels. As a consequence, some additional region matching techniques will be required to find correspondent segments (Hedau et al., 2008; Brendel and Todorovic, 2009). But such techniques are usually very time consuming which makes their usage in the context of the presented framework almost impossible. Another problem is that the partitioning may vary from one frame to the next due to small variations in lighting or other changes in the scene making the region matching procedure not straightforward. Furthermore, methods based on the normalized cuts do not run in real-time and need some seconds to segment a single frame of size 300×400 pixels. The most efficient graph-based and mean shift segmentation approaches (Felzenszwalb and Huttenlocher, 2004; Comaniciu et al., 2002) can handle more than one image per second having the following frame rates: for image size of 320×256 pixels 28.5 and 40.0 Hz, respectively, and for image size of 640×512 pixels 6.1 and 9.1 Hz, respectively. However, even these frame rates are not enough for the pre-processing step in the real-time cognitive vision

system, because both algorithms require in addition a region matching procedure to find correspondent segments between adjacent frames.

The method of superparamagnetic clustering of data is automatic and does not require any prior knowledge about the visual scene or the number of objects. In contrast to the previously mentioned techniques, it can be easily used for the segmentation of video streams. As the segmentation problem is solved here by finding equilibrium states of a spin system, there are no particular requirements to the initial states of spins and they can take on any available values. The closer the initial spin states are to the equilibrium, the less time the method needs for convergence. Due to this fact temporal coherence in the segmentation of video streams can be achieved just by using the previous segmentation result for the initialization of the current frame and its adjustment to the temporal changes (Dellen et al., 2009). Here only shifts between frames need to be taken into account. In such a way a final segmentation result can be obtained much faster as compared to a complete resegmentation with subsequent region matching, drastically reducing computation time. Note that any other automatic segmentation technique can be used for segmentation of the very first frame and labels of the obtained regions (segments) can be considered later as spin states in the spin system. The superparamagnetic clustering of data has two evident disadvantages: the method does not produce consistent results on very textured images resulting in a variety of tiny segments and all its previous implementations are extremely slow requiring from seconds to minutes for the segmentation of one frame (Swendsen and Wang, 1987; Wolff, 1989; von Ferber and Wörgötter, 2000; Dellen et al., 2009). The former can be resolved by the use of special texture filters that smooth highly-textured areas preserving region boundaries (Forsyth and Ponce, 2002), whereas the latter excludes the usage of the existing implementations in the real-time vision systems despite all their advantages.

The contour relaxation technique in the automatic mode can also be employed for the segmentation of video streams. Similar to the superparamagnetic clustering, the contour relaxation uses prior knowledge obtained during the processing of the previous images and the segmentation results obtained at time $t-1$ can be used as an initialization for the segmentation at time t (Mester et al., 2011). Despite fast processing time, the contour relaxation in the automatic mode typically produces an over-segmentation in the sense of a super-pixel representation of the input image which is of significantly lower quality in comparison to other techniques.

2.1.2 Special hardware for acceleration

The real-time aspect is getting nowadays more and more important in image processing and computer vision mainly for two reasons: first, the research done during the last decades in computer vision and image processing allows transforming visual information into more descriptive but nevertheless quite precise representations of the visual scene for using them in a wide range of robotic applications, e.g., robot

movement, object grasping, and object manipulation (Klingbeil et al., 2011; Kjellström et al., 2011; Aksoy et al., 2011). Second, new hardware architectures and programming models for multi-core computing have been proposed in the last ten years, through which many algorithms could be upgraded to real-time processing.

Currently different hardware platforms are used as accelerators for complex computations in the domain of visual processing, such as multicore processors, Digital Signal Processors (DSP), the Cell Broadband Engine Architecture (CBEA), Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) (Brodtkorb et al., 2010). For cognitive vision systems used by robots interacting with the environment, the real-time computations are of particular importance, since only real-time algorithms can be employed in the perception-action loop. Image segmentation is usually used only as a pre-processing step and hence it needs to run in real-time leaving enough time for subsequent high-level computations (Meribout and Nakanishi, 2005).

In the area of visual processing, the evolution of Graphics Processing Units (GPUs) during the last four years has been of particular importance. GPUs are specialized microprocessors which have been initially invented for image processing and acceleration of 2D and 3D graphics rendering. GPUs are used in workstations, personal computers, mobile phones and embedded systems. At present GPUs are a part of every computer and can be used immediately without any additional hardware upgrades. Over the last four years GPUs have evolved into highly parallel, multi-threaded, multi-core processors with tremendous computational power and very high memory bandwidth. For algorithms of high complexity, their parallel architecture makes them in many cases more efficient than general-purpose CPUs. Therefore, GPUs can be used not only for graphics processing but also for general-purpose parallel computing. Furthermore, the graphics capabilities of GPUs make the visual output of the processed data directly from the microprocessor much simpler compared to other parallel platforms. The parallel programming model of Compute Unified Device Architecture (CUDA) proposed by Nvidia in 2007 makes parallelization of software applications on GPUs quite transparent (Lindholm et al., 2008).

As mentioned above, all previous image segmentation approaches based on the superparamagnetic clustering are very slow and, therefore, cannot be employed for the real-time video segmentation. But all these algorithms have been implemented on traditional CPU architectures without special hardware for acceleration. However, considering all advantages of the superparamagnetic clustering in terms of the video segmentation problem (automatic processing and fast temporal coherence without block matching), a real-time implementation of this technique would be very desirable. In this chapter we investigate opportunities for achieving efficient performance of the superparamagnetic clustering of data and propose a real-time implementation of this technique GPUs.

The chapter is organized in the following way. First we describe the method of the superparamagnetic clustering of data. Then we present in more detail a new

real-time segmentation algorithm belonging to this class of segmentation techniques. Next we introduce the GPU architecture and consider the parallel implementation of the proposed algorithm. Finally, we discuss our results and conclude this chapter with a comparison to conventional image segmentation methods.

2.2 Real-time image segmentation on a GPU

2.2.1 Superparamagnetic clustering of data

In the superparamagnetic clustering of data each pixel of the image is represented by a spin in a Potts model. The Potts model (Potts, 1952), which is a generalization of the Ising model (Ising, 1925), describes a system of granular ferromagnets or spins which interact in such a way that neighboring spins corresponding to similar pixels tend to align. In the Ising model spins can be either aligned or anti-aligned, while in the Potts model spins can be in q different states, characterizing the pointing direction of the respective spin vectors. Segments appear naturally as regions of correlated spins at a given temperature (von Ferber and Wörgötter, 2000).

Depending on the temperature, i.e., disorder introduced to the system, the spin system can be in the paramagnetic, the superparamagnetic, or the ferromagnetic phase. In the paramagnetic phase the temperature is high and the system is in a state of complete disorder. As the temperature is decreased a transition to a superparamagnetic phase is observed and spins become completely aligned in every homogeneous region, while different regions remain uncorrelated. In the ferromagnetic phase all spins are aligned. Blatt et al. (1996) applied the Potts model to the image segmentation problem in a way that in the superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data. Therefore, the segmentation problem can be solved by finding the equilibrium states of the energy function of a ferromagnetic Potts model (without data term) in the superparamagnetic phase (Eckes and Vorbrüggen, 1996; Opara and Wörgötter, 1998; von Ferber and Wörgötter, 2000; Dellen et al., 2009).

By contrast, methods which find solutions by computing the minimum of an energy function require a data term – otherwise only trivial solutions are obtained. A data term puts by definition constraints on the solution which require prior knowledge on the data. Hence, the equilibrium-state approach to the image segmentation problem has to be considered as fundamentally different from approaches which find the minimum energy configuration of energy functions in MRFs (Boykov and Kolmogorov, 2004).

The equilibrium states of the Potts model have been approximated in the past using the Metropolis-Hastings algorithm with annealing (Geman and Geman, 1984) and methods based on cluster updating, which are known to accelerate the equilibration of the system by shortening the correlation times between distant spins. Prominent

algorithms are Swendsen-Wang (Swendsen and Wang, 1987), Wolff (Wolff, 1989), and energy-based cluster updating (ECU) (von Ferber and Wörgötter, 2000). All of these methods obey detailed balance, ensuring convergence of the system to the equilibrium state.

Using the Potts model an input image is represented in a form of color vectors $\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g_N}$ arranged on the $N = L_x L_y$ sites of a two-dimensional (2D) lattice. The segmentation problem consists in finding regions of the similar color. In the Potts model, a spin variable σ_k, which can take on q discrete values ($q > 2$) $w_1, w_2, \ldots, w_q$, called spin states, is assigned to each pixel of the image. We define a spin state configuration by $S = \{\sigma_1, \sigma_2, \ldots, \sigma_N\} \in \Omega$, where Ω is the space of all spin configurations. A global energy function or a cost function of this particular q-state Potts configuration $S \in \Omega$ is the Hamiltonian

$$H[S] = -\sum_{<i,j>} J_{ij}\delta_{\sigma_i\sigma_j} + \frac{r}{N}\sum_{i,j}\delta_{\sigma_i\sigma_j}. \tag{2.2}$$

The segmentation problem is solved by finding regions or clusters of correlated spins in the low temperature equilibrium states of the Hamiltonian $H[S]$. The first term in (2.2) represents the system energy where <i,j> denotes the closest neighborhood of spin i with $||i, j|| \leqslant \ell$, where ℓ is a constant that needs to be set. 2D bonds (i, j) between two pixels with coordinates (x_i, y_i) and (x_j, y_j) are created if

$$\begin{aligned} |(x_i - x_j)| &\leqslant \ell, \\ |(y_i - y_j)| &\leqslant \ell. \end{aligned} \tag{2.3}$$

J_{ij} is an interaction strength or coupling constant and δ_{ij} is the Kronecker delta defined by

$$\delta_{ij} = \begin{cases} 1 & \text{if } \sigma_i = \sigma_j, \\ 0 & \text{otherwise.} \end{cases}, \tag{2.4}$$

where σ_i and σ_j are the respective spin variables of two neighboring pixels i and j, respectively. A coupling constant, determining the interaction strength between two spins i and j, is given by

$$J_{ij} = 1 - \Delta_{ij}/\overline{\Delta}, \tag{2.5}$$

where $\Delta_{ij} = ||\mathbf{g_i} - \mathbf{g_j}||$ is the color difference between respective color vectors $\mathbf{g_i}$ and $\mathbf{g_j}$ of the input image (see Section 2.2.2). $\overline{\Delta}$ is the mean distance averaged over all interaction neighborhoods N in the image. The interaction strength is defined in such a way that regions with similar color values will get positive weights with a maximum value of 1 for equal colors, whereas dissimilar regions get negative weights (Eckes

and Vorbrüggen, 1996). The mean distance $\overline{\Delta}$ represents the intrinsic (short-range) similarity within the whole input image [2]:

$$\overline{\Delta} = \alpha \cdot \left(\frac{1}{N} \frac{1}{(2\ell+1)^2 - 1} \sum_{i=1}^{N} \sum_{<i,j>} ||\mathbf{g_i} - \mathbf{g_j}|| \right), \tag{2.6}$$

where $(2\ell+1)^2 - 1$ is the number of neighbors of one spin. The factor $\alpha \in (0, 10]$ is a system parameter used to increase or decrease the coupling constants.

The second term in (2.2) is introduced in analogy to neural systems, where it is generally called "global inhibition". It is optional and only useful for cluster updating. It serves to favor different spin values for spins in different clusters and r is a control parameter that adjusts the strength of the global inhibition ($r \geqslant 0$). This concept is employed in many neural systems that perform recognition tasks (von Ferber and Wörgötter, 2000). If the global inhibition term was set to zero, the Hamiltonian features the global energy function of the generic Potts model in its usual form.

Various techniques have been proposed in the literature to order spins in the Potts model according to a pre-defined goal, as for example the detection of phase transitions in ferromagnetic systems, or as in the current study, in order to segment images. These algorithms differ mainly in the way how the interaction range between spins is defined and how spins are iteratively updated. The following three approaches are commonly used for the simulation of the Potts model: local update techniques, cluster update algorithm, and the energy-based cluster update.

Local update algorithms (Geman and Geman, 1984; Eckes and Vorbrüggen, 1996) are featured by small interaction ranges and modify only one spin variable per iteration. The algorithm proposed by Metropolis et al. (1953) is the most famous local-update technique. Every iteration it rotates spin variables σ_k and tries to minimize the global energy function employing *simulated annealing*. Simulated annealing operates by simulating the cooling of a system whose possible energies correspond to the values of the objective function being minimized (see the first term in (2.2)). The annealing process starts at a relatively high temperature $T = T_{init}$ and at each step attempts to replace the current solution S_{cur} by a new spin configuration S_{new} chosen according to the employed distribution. A set of potential new solutions $S_1, S_2, \cdots, S_n \in \Omega$ is generated by the Metropolis algorithm (see Section 2.2.3). Note that the Metropolis algorithm is highly local and generates new spin configurations proposing individual moves of spin variables. The temperature is a parameter that controls the acceptance probability of new solutions and it is gradually decreased after each iteration or after a group of iterations. At high temperatures almost all new solutions are accepted, while at low temperatures only "downhill" solutions leading to the energy minimization are considered. In the limit $T = 0$, only the lowest energy states

[2]Note that (2.5) is ill-defined in the case of $\overline{\Delta} = 0$. But in this case only a single uniform surface exists and segmentation is not necessary.

have nonzero probability. System perturbations at high temperatures are needed to save the method from being trapped in local minima. The name of the method originates from annealing in metals where the heating and controlled slow cooling increase crystal sizes and reduce their defects (Salamon et al., 2010). It explains why the method is called sometimes "simulated cooling" [3]. The Metropolis local update algorithm with simulated annealing solves the segmentation problem by propagating a certain modification of the spin state configuration through the lattice step by step, which makes it very slow. Furthermore, due to slowing down at low temperatures the local update becomes very time consuming. Hence the original Metropolis algorithm running on traditional CPU architectures is inapplicable to the real-time tasks. Even optimizing the annealing schedule cannot accelerate the method, since an extremely slow rate is needed to find the final spin state configuration S_{final}.

Cluster update algorithms (Swendsen and Wang, 1987; Wolff, 1989; Blatt et al., 1996) introduce larger interaction ranges and at every iteration groups of spins, called *clusters*, are updated simultaneously. The first widely used cluster update algorithm was proposed by Swendsen and Wang (1987). In this algorithm, "satisfied" bonds, i.e., those that connect nearest neighbor pairs of identical spins $\sigma_i = \sigma_j$, are identified first. The satisfied bonds (i, j) are then "frozen" with some probability p_{ij}. Sites of the lattice connected by frozen bonds define the clusters $c_1, c_2, \ldots, c_M$. Each cluster is then updated by assigning to all its spins the same new value. This is done independently for each cluster and the external bonds connecting the clusters are "deleted". Here the temperature remains fixed and no annealing takes place between the iterations. Since a change in the current spin configuration can affect many spin variables at the same time, cluster update algorithms running on traditional CPU platforms are much faster compared to local update techniques. However, updating of complete spin clusters often leads to undesired cluster fusions when regions that should get different labels form one segment.

The energy-based cluster update (*ECU algorithm*) proposed by Opara and Wörgötter (1998) combines the advantages of both local and global update techniques. Here the same new value is assigned to all spins inside one cluster in consideration of the energy gain calculated for a neighborhood of the regarded cluster. Similar to the Swendsen and Wang cluster update algorithm (Swendsen and Wang, 1987), the temperature in the ECU method remains fixed and no annealing takes place between the iterations. Once the clusters of spins connected by frozen bonds are defined, a Metropolis update is performed that updates all spins of each cluster simultaneously to a new spin value. The new spin value for a cluster c is computed considering the energy gain obtained from a cluster update to a new spin value w_k, where the index k denotes the possible spin value between 1 and q, respectively. Updating the respective cluster to the new value results in a new spin configuration S_k^c. The probability for the choosing the new

[3] *Webster's Revised Unabridged Dictionary* defines *anneal* as "to subject to great heat and then to cool slowly".

spin value w_k for the cluster c is computed by taking into account the interactions of all spins in the cluster c with those outside the cluster, assuming that all spins of the cluster are updated to the new spin value w_k with the Hamiltonian

$$H[S_k^c] = -\sum_{\substack{<i,j> \\ c_i \neq c_j}} \varepsilon J_{ij} \delta_{\sigma_i \sigma_j} + \frac{r}{N} \sum_{i,j} \delta_{\sigma_i \sigma_j}, \tag{2.7}$$

where <i,j>, $c_i \neq c_j$ is a noncluster neighborhood of spin i and ε is a parameter which allows us to "share" the interaction energy between the clustering and updating steps (von Ferber and Wörgötter, 2000). Similar to a Gibbs sampler, the probability $P(S_k^c)$ of selecting the new spin value w_k for the cluster c is given by

$$P(S_k^c) = \frac{\exp(H[S_k^c]/T)}{\sum_{i=1}^{q} \exp(H[S_i^c])}. \tag{2.8}$$

All mentioned update techniques define segments as groups of correlated spins. As was mentioned before, the spin states σ_i in the Potts model can take values between 1 and q, where q is a parameter of the system. The number of segments is not constrained by the parameter q. Note that spins belonging to the same segment are always in the same spin state, while the reverse is not necessarily true.

Local update algorithms are extremely slow requiring minutes to segment an image of size 320×256 pixels on traditional CPU platforms. Cluster updates are much faster then local updates and need seconds instead of minutes to segment an image of the same size. However, this time performance is not enough for the segmentation technique to be employed for the real-time video segmentation. In terms of parallelization on special hardware, local updates are more preferable, since each spin update involves only local information about its closest neighborhood and, thus, many updating operations can be done simultaneously. Furthermore, local updates fit very well to the GPU architecture which does not require tremendous resources and is commonly used in robotic systems. Cluster updates, on the contrary, cannot be parallelized easily due to the very global spin update procedure of arbitrary shaped clusters. Although cluster updates do not depend on each other and can be done in parallel, one cluster update is sequential because its shape before update is unknown. Sequential updates within each cluster are a bottleneck in parallelization of cluster updates and their latency can be reduced only on very powerful computer systems. Since our goal is an image segmentation technique applicable for the real-time video segmentation running on common and not very expensive hardware, only local update techniques for the simulation of the Potts model are considered in this study (Abramov et al., 2010b).

2.2.2 Computation of coupling constants

In the homogeneous Potts model, all spins are interacting with the same strength ($J_{ij} = const$). In the inhomogeneous Potts model, the interaction strength is changing over space ($J_{ij} \neq const$). For image segmentation we use the inhomogeneous Potts model and the interaction strengths J_{ij} between the neighboring spins (see (2.5)) are defined as the feature similarity of the respective pixels. Spins representing similar image parts (same objects or their parts) interact strongly, while spins of nonsimilar image parts will interact only weakly (Opara and Wörgötter, 1998).

Essentially three parameters R (red), G (green), and B (blue), called *tristimulus values*, describe human color sensation. Red, green and blue color values are the brightness values of the scene derived by integrating the responses of three distinct color filters on the incoming light S_R, S_G, and S_B according to

$$R = \int_\lambda E(\lambda) S_R(\lambda) d\lambda, \quad G = \int_\lambda E(\lambda) S_G(\lambda) d\lambda, \quad B = \int_\lambda E(\lambda) S_B(\lambda) d\lambda, \tag{2.9}$$

where $E(\lambda)$ is a spectral power distribution and λ is the wavelength.

RGB color space

The RGB color space is a linear color space where a broad range of colors is derived by adding R, G, and B components together in diverse ways. Geometrically the RGB color space can be represented as a 3-dimensional cube where the coordinates of each point inside the cube represent the values of red, green and blue components, respectively.

Other color representations (spaces) can be derived from the RGB representation by using either linear or nonlinear transformations (Cheng et al., 2001). Besides the RGB color space, various other color spaces, such as HSV (hue, saturation, value) and CIE [4] are frequently utilized in image processing. However, there is no superior color space and the choice of the proper color space depends on the specifics of the concrete problem.

Although RGB is a widely used color space, it is not ideally suitable for color scene segmentation and analysis because of the high correlation between the R, G and B components (Forsyth and Ponce, 2002). In the RGB space changes in intensity lead to changes in the values of all three color components. The difference between two color vectors $\mathbf{g_i} = (r_i, g_i, b_i)^T$ and $\mathbf{g_j} = (r_j, g_j, b_j)^T$ in the RGB space is given by the Euclidean distance in the RGB cube

$$||\mathbf{g_i} - \mathbf{g_j}|| = \sqrt{(r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2}. \tag{2.10}$$

[4] The "CIE XYZ color space" created by the International Commission on Illumination (CIE) in 1931 is one of the first mathematically defined color spaces.

The representation of color distances in the RGB cube is not perceptually uniform and, therefore, it is impossible to evaluate the similarity of two colors from their distance in the RGB space. Furthermore, linear color spaces do not capture human intuitions about the topology of colors. A common intuition is that hues form a circle, in the sense that hue changes from red through orange to yellow and then green and from there to cyan, blue, purple, and then red again. This means that no individual coordinate of a linear color space can model hue, since that coordinate has a maximum value which is far from the minimum value (Forsyth and Ponce, 2002).

In order to deal with the mentioned problems a color space is needed that reflects these relations. By applying a nonlinear transformation to the RGB space, other, more suitable color spaces can be created. CIE and HSV are the most commonly used nonlinear color spaces in the image processing.

HSV color space

The HSV color space separates color information of an image from its intensity information. Color information is represented by hue and saturation values, while intensity (also called *lightness*, *brightness* or *value*) is determined by the amount of light. Hue represents basic colors and saturation color purity, i.e., the amount of white light mixed in with the hue. For example, if we want to check whether a color lies in a particular range of reds, we can encode the hue of the color directly. Geometrically the HSV color space can be represented by a cone where hue is described by the angle on the circle with the range of values from 0° to 360°. The saturation component represents the radial distance from the center of the circle, which by definition has zero saturation. The closer the point is to the center, the lighter is the color. Value is the vertical axis of the cone and colors toward the point of the cone are dark (low value), while colors further out are brighter (higher value). The conversion from the RGB to the HSV color space is a well-defined procedure and images can be converted without loss of information. The known color vector $\mathbf{g_i} = (r_i, g_i, b_i)^T$ in the RGB color space is converted to the vector $\overline{\mathbf{g_i}} = (h_i, s_i, v_i)^T$ in the HSV color space through the following equations (Kyriakoulis and Gasteratos, 2010):

$$v_i = \max(r_i, g_i, b_i), \quad s_i = \begin{cases} (v_i - \min(r_i, g_i, b_i))/v_i & \text{if } v_i \neq 0, \\ 0 & \text{if } v_i = 0. \end{cases} \tag{2.11}$$

If $s_i = 0$ then $h_i = 0$. If $r_i = v_i$ then

$$h_i = \begin{cases} 60° \cdot (g_i - b_i)/(v_i - \min(r_i, g_i, b_i)) & \text{if } g_i \geqslant b_i, \\ 360° + 60° \cdot (g_i - b_i)/(v_i - \min(r_i, g_i, b_i)) & \text{if } g_i < b_i. \end{cases} \tag{2.12}$$

In the case of $g_i = v_i$, we have

$$h_i = 120° + \frac{60° \cdot (b_i - r_i)}{v_i - min(r_i, g_i, b_i)}. \tag{2.13}$$

If $b_i = v_i$, then

$$h_i = 240^\circ + \frac{60^\circ \cdot (r_i - b_i)}{v_i - min(r_i, g_i, b_i)}. \tag{2.14}$$

Note that gray tones, from black to white, have undefined hue and 0 saturation. Also the saturation is undefined when the intensity is zero. In order to segment objects with different colors in the HSV space the segmentation algorithm can be applied to the hue component only. Different thresholds can be set on the range of hues that separate different objects easily, but it is difficult to transform these thresholds into RGB values, since hue, saturation and intensity values are all encoded as RGB values. Hue is especially useful in the cases where the illumination level varies from pixel to pixel or from frame to frame in the video. It is very often the case in regions with non-uniform illumination such as shadows, since hue is independent on intensity values.

For two color vectors $\mathbf{g_i} = (h_i, s_i, v_i)^T$ and $\mathbf{g_j} = (h_j, s_j, v_j)^T$ in the HSV color space, the color difference between them is determined by Koschan and Abidi (2008)

$$||\mathbf{g_i} - \mathbf{g_j}|| = \sqrt{(\Delta V)^2 + (\Delta C)^2}, \tag{2.15}$$

where

$$\Delta V = |v_1 - v_2|, \quad \Delta C = \sqrt{s_1^2 + s_2^2 + 2 s_1 s_2 cos\theta}, \tag{2.16}$$

$$\theta = \begin{cases} |h_1 - h_2| & \text{if } |h_1 - h_2| \leqslant \pi, \\ 2\pi - |h_1 - h_2| & \text{if } |h_1 - h_2| > \pi \end{cases}. \tag{2.17}$$

CIE color space

The CIE color system is a three dimensional space and contains all colors that can be perceived by the human eye. Thereby this color space is very often called *the perceptual color space*. The CIE color space is based on the evidence that the human eye has three types of cone cells. The first type responds mostly to large wavelengths which correspond to yellowish colors, the second type responds mostly to medium wavelengths which correspond to greenish colors, the third type responds mostly to small wavelengths which correspond to bluish colors. The types of cone cells are abbreviated due to the wavelength value as L for long, M for medium, and S for short (Wyszecki and Stiles, 2000). In the $CIE\ XYZ$ color space, the tristimulus values are not L, M and S responses of the human eye, but rather a set of tristimulus values X, Y, Z which are roughly red, green and blue. Note that X, Y, Z are not physically observed red, green and blue colors. They rather can be thought of as "obtained" parameters from the red, green and blue colors. Any color can be represented by the combination of X, Y, and Z values. The values of X, Y, and

Z can be computed by a linear transformation from RGB tristimulus coordinates. The transformation matrix for the $NTSC$ [5] receiver primary system is determined as (Cheng et al., 2001):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \tag{2.18}$$

There are several CIE color spaces that can be established once the XYZ tristimulus coordinates are known. CIE $(L^*a^*b^*)$ and CIE $(L^*u^*v^*)$ are the most commonly used CIE color spaces. They can be obtained through nonlinear transformations of X, Y, and Z values. In the current study we will consider only the CIE $(L^*a^*b^*)$ space which is defined as

$$\begin{aligned} L^* &= 116 \cdot \left(\sqrt[3]{\frac{Y}{Y_0}} \right) - 16, \\ a^* &= 500 \cdot \left[\sqrt[3]{\frac{X}{X_0}} - \sqrt[3]{\frac{Y}{Y_0}} \right], \\ b^* &= 200 \cdot \left[\sqrt[3]{\frac{Y}{Y_0}} - \sqrt[3]{\frac{Z}{Z_0}} \right], \end{aligned} \tag{2.19}$$

where $X/X_0 > 0.01$, $Y/Y_0 > 0.01$, and $Z/Z_0 > 0.01$ and (X_0, Y_0, Z_0) are X, Y, and Z values for the standard white. The CIE $(L^*a^*b^*)$ is substantially uniform. In many cases it is important to know how different two colors are for a human observer and differences in the $L^*a^*b^*$ space give a good hint for that (Cheng et al., 2001). A bunch of metrics have been proposed for the computation of color differences in the CIE $(L^*a^*b^*)$ color space. Perceptual non-uniformities caused refinement of metrics due to the fact that the human eye perceives certain colors better than others and a good metric should take this into account. In the current work we use the $CIE94$ metric (CIE-Publication-116-1995, 1995) which defines the difference between two colors (L_1^*, a_1^*, b_1^*) and (L_2^*, a_2^*, b_2^*) as

$$\Delta E_{94}^* = \left[\left(\frac{\Delta L^*}{K_L} \right)^2 + \left(\frac{\Delta C_{ab}^*}{1 + K_1 C_1^*} \right)^2 + \left(\frac{\Delta H_{ab}^*}{1 + K_2 C_2^*} \right)^2 \right]^{1/2}, \tag{2.20}$$

where

$$\Delta L^* = L_1^* - L_2^*, \tag{2.21}$$

[5] National Television System Commission, United States.

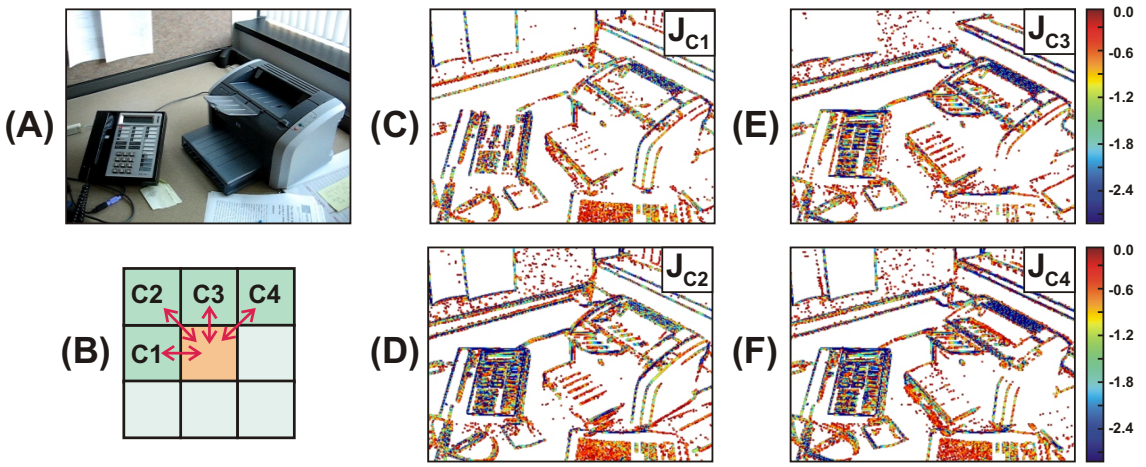

Figure 2.1: Coupling constants for the 8-connectivity case in the CIE $(L^*a^*b^*)$ color space. (A) Original image. (B) Mask for eight-connected connectivity. (C - F) Matrices with coupling constants computed for horizontal, left diagonal, vertical and right diagonal directions. Note that only coupling constants leading to the formation of segments are shown here $(J < 0)$.

$$C_1^* = \sqrt{(a_1^{*2} + b_1^{*2})}, \quad C_2^* = \sqrt{(a_2^{*2} + b_2^{*2})}, \quad \Delta C_{ab}^* = C_1^* - C_2^*, \tag{2.22}$$

$$\Delta H_{ab}^* = \sqrt{\Delta E_{ab}^{*2} - \Delta L^{*2} - \Delta C_{ab}^{*2}} = \sqrt{\Delta a^{*2} + \Delta b^{*2} - \Delta C_{ab}^{*2}}, \tag{2.23}$$

$$\Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2}, \tag{2.24}$$

$$\Delta a^* = a_1^* - a_2^*, \quad \Delta b^* = b_1^* - b_2^*, \tag{2.25}$$

where K_L, K_1, and K_2 are the weighting factors. Since the CIE $(L^*a^*b^*)$ color space is approximately uniform chromatic scale, it matches the sensitivity of the human eyes with computer processing, while the RGB or XYZ color spaces do not have this property (Tseng and Chang, 1992). Therefore, the perceptual color attributes such as intensity, hue and saturation can be derived easily. CIE spaces control color and intensity information more independently and simply than RGB primary colors. Direct color comparison in CIE spaces is especially efficient in the measurement of small color differences.

Interaction strengths computed in the CIE $(L^*a^*b^*)$ color space for one testing image are shown in Fig. 2.1. Since in the current study we use 8-connectivity of pixels, interaction strengths for each pixel need to be computed for the 8 closest neighbors in four different directions: horizontal, left diagonal, vertical, right diagonal. In order

to compute interaction strengths between all neighboring pixels once, only bonds connecting the current pixel with the previous pixels (in terms of the image scan from left to right and from top to bottom) are considered per pixel (see $C1$, $C2$, $C3$, and $C4$ bonds in Fig. 2.1(B)). Remaining interactions are covered by the following pixels resulting in four matrices containing coupling constants (see Fig. 2.1(C-F) where only coupling constants affecting the formation of segments are shown, i.e., $J < 0$).

2.2.3 Metropolis algorithm with simulated annealing

As was already mentioned, simulated annealing is a heuristic algorithm based on analogy to physical systems. The following three components are needed to define a simulated annealing problem:

1. A configuration space Ω that is the domain of our objective function. The elements $S \in \Omega$, i.e., spin configurations in the image segmentation problem, are called states of the system. Note that Ω is a discrete state space which is large but finite.

2. The objective function H defined on this domain. For any state $S \in \Omega$, $H[S]$ is called the energy of S.

3. A graph structure Λ defined on this domain that specifies which spin configurations are to be one move apart; Λ: $\Omega \rightarrow 2^{\Omega}$ is the collection of subsets of Ω. For any spin configuration $S \in \Omega$, the states in $\Lambda(S)$ are called the neighbors of S.

We can say that a simulated annealing problem is an ordered triple $(\Omega, H[\cdot], \Lambda(\cdot))$ consisting of a set, a real valued function on the set, and a graph structure on the set [6] (Salamon et al., 2010).

The Metropolis algorithm with simulated annealing works by simulating a random walk on the set of spin states Ω looking for low-energy states. Primarily, spin variables of the initial configuration S_{init} are initialized randomly which results in a quite high energy value. According to the Metropolis algorithm, one update iteration consists of the following steps (Metropolis et al., 1953):

1. The system energy $H[S_{cur}]$ of the current spin configuration S_{cur} is computed as the global energy function of the generic Potts model (see 2.2) without the inhibition term by

$$H[S_{cur}] = -\sum_{<i,j>} J_{ij}\delta_{\sigma_i\sigma_j}. \tag{2.26}$$

[6] Sometimes in the literature this triple is called a Markov kernel (Azencott, 1992).

2. For each pixel i, a set of n (number of neighbors) new possible spin configurations $\hbar = S'_1, S'_2, \cdots, S'_n \in \Lambda(S_{cur})$ is created by changing the spin state of pixel i to the spin states of the neighbors. The number of new possible spin configurations n does not depend on q.

3. Every spin configuration $S'_i \in \hbar$ is considered as a potential new configuration of the system. Therefore, energy values of all configurations from the set $\hbar$ need to be computed according to (2.26).

4. Among all new possible configurations from the set $\hbar$ the spin configuration with the minimum energy value is selected according to

$$H[S_{new}] = min(H[S_1], H[S_2], \cdots, H[S_n]). \tag{2.27}$$

   The respective change in energy between the current configuration S_{cur} and the selected configuration $S_{new} \in \hbar$ is defined as $\Delta H \equiv H[S_{new}] - H[S_{cur}]$. According to $\Delta H > 0$ or $\Delta H \leqslant 0$, moves can be classified as uphill or downhill, respectively.

5. To effect a bias in favor of moves that decrease the energy, downhill moves are always accepted, whereas uphill moves are accepted only sometimes in order to avoid getting trapped in local minima. Uphill moves with an energy gain ΔH are accepted with probability $x^{\Delta H}$, where $x \in [0, 1]$ is a control parameter. Note that for $x = 1$ all moves are accepted, while for $x = 0$ only downhill moves are accepted. For intermediate values of x the probability of accepting an uphill move decreases as x decreases. Simulated annealing proceeds by a random walk through the configuration space Ω decreasing x from an initial value near one to a final value close to zero. It means that progressively less time is spent moving uphill as the algorithm proceeds. In the Metropolis algorithm the parameter x is expressed in terms of the temperature T by $x = e^{-1/T}$. Therefore, the probability that the proposed move leading to increase in energy will be accepted is given by

$$P(S_{cur} \to S_{new}) = \exp\left(-\frac{|\Delta H|}{T_n}\right). \tag{2.28}$$

   A number ξ is drawn randomly from a uniform distribution in the range of $[0, 1]$. If $\xi < P(S_{cur} \to S_{new})$, the move is accepted.

6. The temperature is gradually reduced after every iteration or after a group of iterations according to the pre-defined annealing schedule (see further).

Metropolis updates (1) - (6) run until convergence, i.e., when no more spin flips towards a lower energy state are being observed. The equilibrium state of the system, achieved after several Metropolis iterations, corresponds to the image partition or segmentation from which the final segments larger than a pre-defined threshold can be extracted. The full update of a spin configuration, i.e., when all spins have been updated, is usually referred to as a *global Metropolis iteration*. However, since we will be interested further only in updates of all spins in the image, we use the term *iteration* for the full Metropolis update.

The Metropolis algorithm is useful for finding average properties of the physical system that moves in its states space Ω spending some time in each state. The portion of time spent in any spin configuration S for a temperature T is proportional to $\exp(-H[S]/T)$. The Metropolis algorithm visits spin configurations with exactly these frequencies. The set of frequencies forms a distribution which is called the *Boltzmann distribution*, given by (Salamon et al., 2010)

$$P_T(S) = \frac{e^{-H[S]/T}}{\sum_{\bar{S}\in\Omega} e^{-H[\bar{S}]/T}}. \tag{2.29}$$

The Boltzmann distribution contains only one parameter: the temperature T. The lower the temperature, the more this distribution favors low-energy states. At infinite temperature all spin configurations are equiprobable, while for $T = 0$ only the configurations of lowest energy (corresponding to ideally segmented images) have a nonzero probability. The Boltzmann distribution has two important properties for global optimization tasks:

1. The system needs to be cooled through the full range of temperatures.

2. The Boltzmann distribution is uniform in terms of energy values, i.e., if two spin configurations have the same energy, then they have the same probability.

Selecting the annealing schedule

The annealing schedule is usually a decreasing sequence of temperatures $T(n)$ successively employed in the transition probabilities of the Metropolis algorithm (see (2.28)). In some cases the performance of the annealing does not depend on the form of $T(n)$ (Salamon et al., 1988; Johnson and McGeogh, 1997) and the number-one exponential cooling schedule is employed (Kirkpatrick et al., 1983; Černý, 1985). But there are problems for which the choice of annealing schedules makes a significant difference (Mosegaard and Vestergaard, 1991).

Most annealing schedules give the user an opportunity to determine the overall rate of cooling. A fast cooling leads to suppressing of the system's degrees of freedom, whereas a slower cooling allows more of the spin configuration space Ω to be explored. Consequently, the slower we cool, the better the final spin configuration

can be expected to be. The question is how fast to cool up the system with respect to the time performance and eventually how much we care for the quality of the final solution. A trade-off between acceptable time performance and quality of results needs to be found. Independent from the chosen schedule, the start and stop criteria of the annealing schedule need to be determined. The starting and ending temperatures T_0 and T_{final}, respectively, need to be set according to the considered problem. T_0 needs to be chosen so that the system should initially be hot enough to allow large fluctuations in the energy so that many uphill moves (see Section 2.2.3) leading to increase in energy are accepted (White, 1984). The scale of these fluctuations is given by the standard deviation of the energy at infinite temperature

$$\sigma_{H|T=\infty} = \sqrt{\langle H^2 \rangle_{T=\infty} - \langle H \rangle^2_{T=\infty}}. \tag{2.30}$$

If a priory is not known, this quantity can be estimated by randomly sampling the configuration space. The same procedure gives a good random starting configuration (Salamon et al., 2010). Practically T_0 should be moderately larger than σ_H, e.g., twice as large. Having either a finite or an infinite number of copies of the system, called *ensemble*, a set of random initial configurations is needed rather than just one. It can be easily achieved by saving an appropriate number of spin configurations obtained in the random sampling.

A simpler way is to set the starting temperature T_0 so that most moves (downhill as well as uphill) are accepted initially. Parameterizing the temperature by the reciprocal temperature $\beta = 1/T$, it is reasonable to start with $\beta = 0$ where all spin moves will be accepted. A more popular strategy is to set T_0 so that exactly half of the spin moves will be accepted. But it is not critical anyway, since relaxation is very fast at high temperatures. The ending temperature T_{final} is usually set as follows. If the energy does not change during the last N_{final} updates, which denotes convergence, then it is time to stop the annealing procedure.

The choosing of schedules has both theoretical and practical aspects. Geman and Geman (1984) introduced a schedule that guarantees convergence to the optimal solution. If H_{best} is the best energy observed during the walk of length t, the annealing schedule has the property that

$$\lim_{t \to \infty} \text{prob}\{H_{best} = H_{min}\} = 1, \tag{2.31}$$

where H_{min} is the minimum energy. The schedule itself is determined by

$$T(t) = \Delta H^{max}_{activation} / \ln(t+1), \tag{2.32}$$

where $\Delta H^{max}_{activation}$ is the largest activation energy, i.e., the largest energy difference that must be overcome along the paths leading out from the bottom of any suboptimal basin (Salamon et al., 2010). The probability of not finding the minimum goes down with time as $P(H_{best} > H_{min}) \propto t^{-x}$, where x is a suitable exponent (Azencott, 1992).

However, it is obvious that with such slow logarithmic decay of the temperature and a large $\Delta H^{max}_{activation}$, the time required for the annealing procedure to stop is enormous and inapplicable to problems with high claims to time performance. Instead of this a schedule of the form (2.32) but having a smaller constant in the numerator can be used:

$$T(t) = d/\ln(t+1). \tag{2.33}$$

Such a schedule will drain the acceptance probability out of all basins with depth less than d. The value of d needs to be chosen in relation to the total length t_{max} of the simulation. As was already mentioned, an *exponential schedule* [7] given by

$$T(t) = T_0 \cdot \gamma^t, \tag{2.34}$$

is the most commonly used annealing schedule. The annealing factor γ is usually set to a number close to 1, such as 0.999. The temperature is updated after every iteration or after a group of k iterations. The value of k is chosen empirically making the actual functional form

$$T(t) = T_0 \cdot \gamma_2^{[t/k]}. \tag{2.35}$$

This schedule allows partial equilibration at each temperature and corresponds to a step function approximation to the exponential function in (2.34) with $\gamma = \gamma_2^{1/k}$. Note that γ can be determined if T_0, T_{final} and t_{final} are known.

Also many other simple functional forms have been employed for the annealing procedure. It is quite common when $\beta = 1/T$ is linearly increased as a function of time (Salamon et al., 1988)

$$\beta_{t+1} = \beta_t + m, \quad m > 0. \tag{2.36}$$

Similar to γ in (2.34), m can be determined from the values of T_0, T_{final} and t_{final}. T can also be decreased linearly in time or as an inverse power $T_0 \cdot t^{-x}$ with almost any $x > 0$.

Another type of annealing schedule is so called *adaptive cooling*. The main idea of this approach is to collect data about the system and employ it to make sophisticated assumptions how the temperature should be decreased. The more one uses the system, the better is the feeling for the proper schedule. Adaptive schedules can be implemented *on-line* or *off-line*. Off-line implementations consider data collected during previous runs to find the desired schedule $T(t)$. Here multiple coolings can be executed concurrently, since no runtime communication between them is needed. The decision about the most suitable schedule is made once all runs finished. Off-line implementations tend to be more robust than on-line which cannot change previously

[7] Sometimes also referred to as a *geometrical schedule*.

	$T_{n+1} = \gamma \cdot T_n$		
Schedule	Starting temperature T_0	Annealing factor γ	Required iterations N_{conv}
(1)	9.0	0.9999	16×10^3
(2)	5.0	0.9999	10×10^3
(3)	3.0	0.9999	5×10^3
(4)	3.0	0.999	10^3
(5)	9.0	0.999	15×10^3
(6)	5.0	0.999	12×10^3
(7)	9.0	0.99	4×10^3
(8)	5.0	0.99	3×10^3
(9)	1.0	0.9999	200
(10)	1.0	0.99	150
(11)	0.5	0.9999	250
(12)	9.0	0.9	100

Table 2.1: Simulated annealing schedules of the on-line adaptive cooling employed in the Metropolis algorithm for the image segmentation problem.

made decisions. Hoffmann et al. (1991) proposed an adaptive annealing schedule which is very simple to implement. The temperature is always decreased by a fixed factor γ, but the time t spent at each temperature T, i.e., a number of performed Metropolis updates, can vary in an adaptive way. Ideally, we would like to reach the equilibrium at each temperature. Such a schedule is determined by $T_{n+1} = \gamma \cdot T_n$.

In this work we employ the on-line adaptive annealing schedule due to the following reasons: (i) this schedule is quite fast comparing to some other introduced schedules; (ii) it is possible to find a schedule suitable for various images after some experimental runs; (iii) as our aim is the real-time segmentation of video streams, there is no opportunity to change previously made decisions, i.e., previously obtained segmentation results. However, we perform only one Metropolis update at each temperature without reaching the equilibrium on each cooling phase. It arises from strict demands for the time performance. The starting temperature T_0 is selected so that most uphill moves are initially accepted. Note that there is no need to specify the ending temperature T_{final}, since Metropolis updates run until convergence when no more moves towards a lower energy state are observed. Thereby only a number of iterations N_{conv} sufficient for convergence of the system needs to be determined.

Various schedules of the on-line adaptive simulated annealing employed in the Metropolis algorithm for the image segmentation problem are shown in Table 2.1.

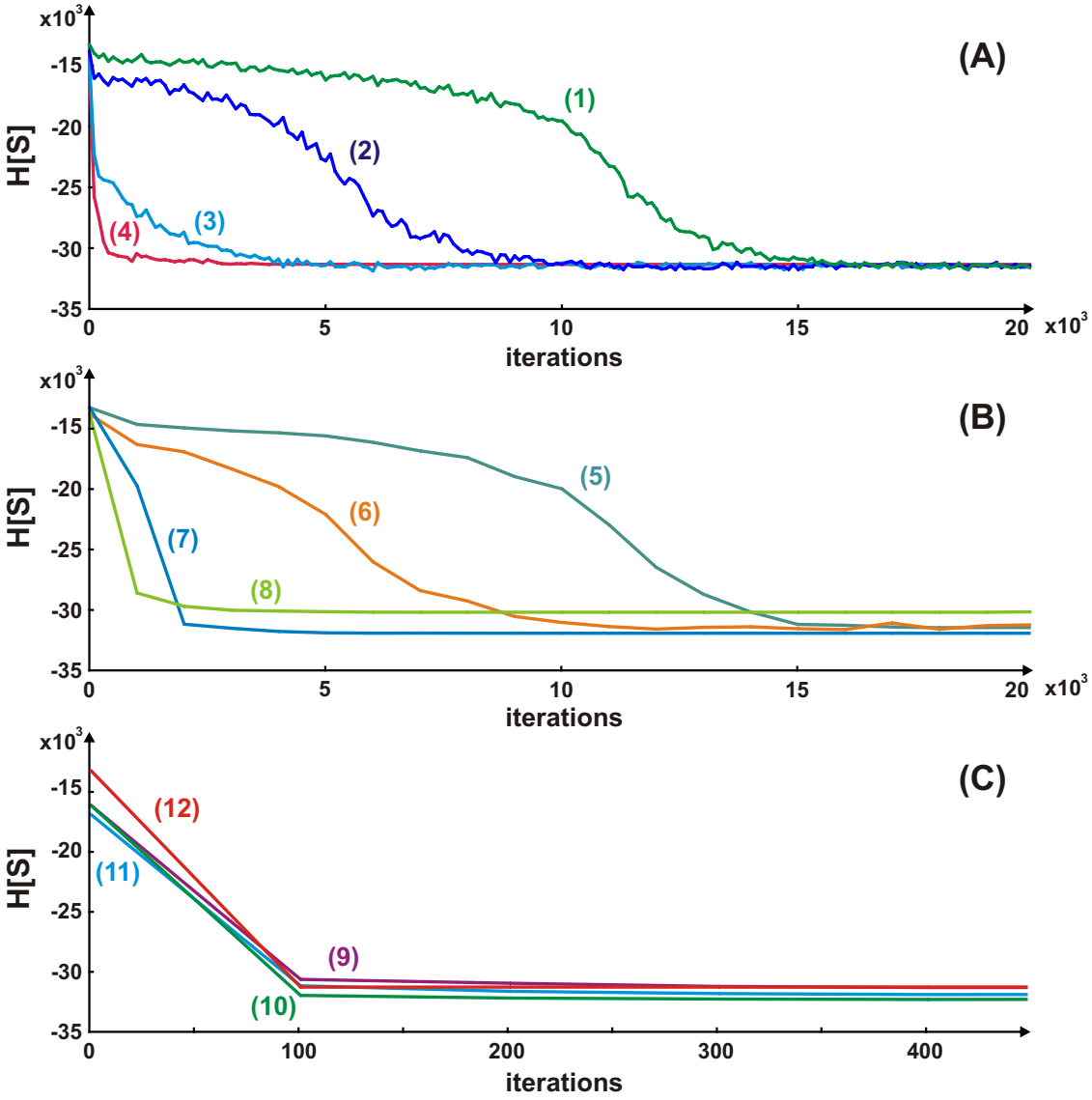

Figure 2.2: The effect of various schedules of the on-line adaptive simulated annealing employed in the Metropolis algorithm for the image segmentation problem. Each panel shows one group of annealing schedules from Table 2.1. See the text for more details.

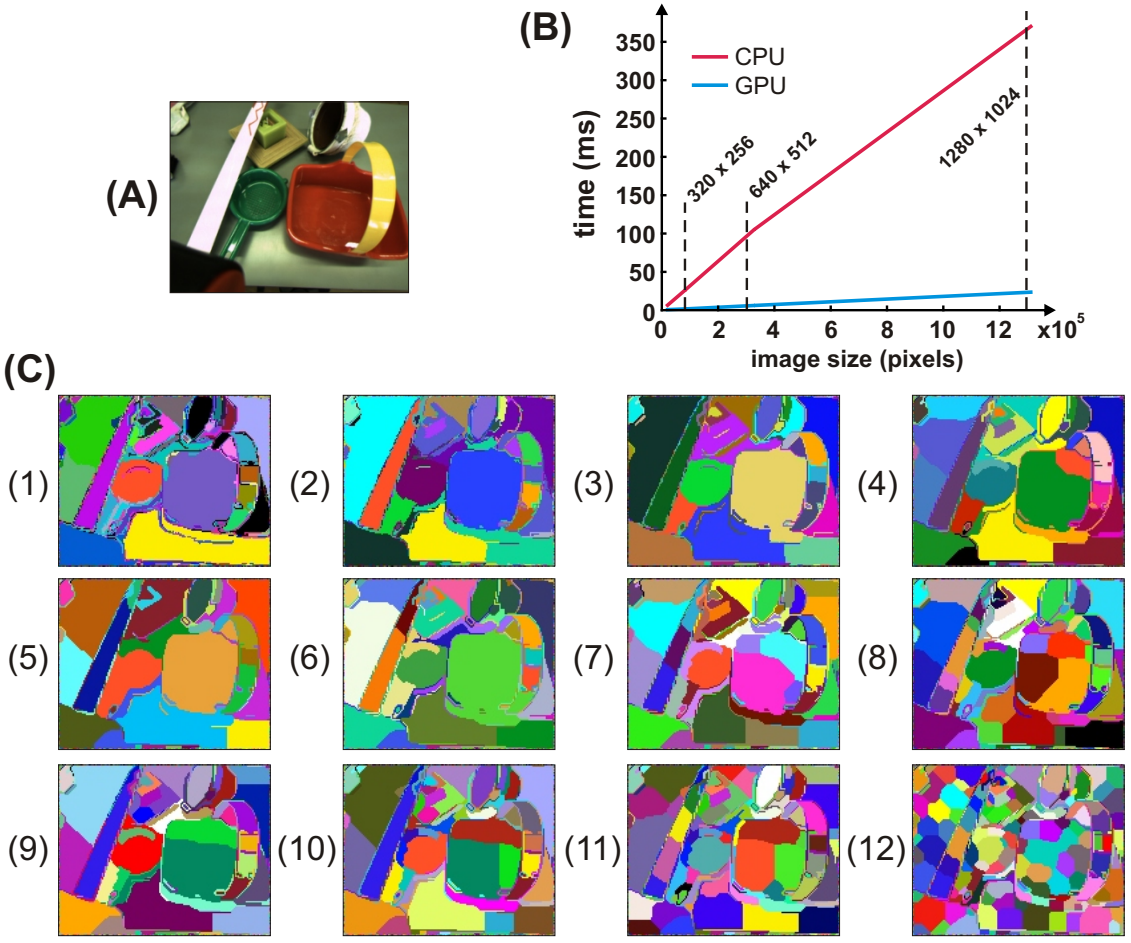

Figure 2.3: Segmentation results obtained by the Metropolis algorithm employing various schedules of the on-line adaptive simulated annealing (see Table 2.1 and Fig. 2.2). (A) Original image of size 160×128 pixels. (B) Processing times of one Metropolis iteration on CPU and GPU architectures as a function of the total number of pixels in the image. (C) Segmentation results for various simulated annealing schedules derived with $\alpha = 2.5$. See the text for more details.

The schedules differ in values of the starting temperature T_0 and the annealing factor γ. The number of iterations N_{conv} required for convergence, i.e., complete image segmentation, has been determined experimentally for each schedule. The effect of each presented schedule used in the Metropolis algorithm to segment an image of size 160×128 pixels is shown in Fig. 2.2 as a function of the iteration number where each panel shows one group of annealing schedules from Table 2.1. Segmentation results corresponding to all introduced annealing schedules obtained on one testing image are presented in Fig. 2.3(A,C). We can see that only schedule (1) leads to a complete segmentation of an image where all homogeneous image regions are represented by a single segment. Note that the basket handle cannot be segmented in one segment due to the light reflections on it. This schedule starts at a very high temperature $T_0 = 9.0$ and cools down very slowly ($\gamma = 0.9999$) allowing many uphill moves during two thirds of the whole annealing walk. Namely these moves help the system to explore more of the spin configuration space Ω resulting in a consistent segmentation result. A bit faster schedule (5) ($\gamma = 0.999$) starting with the same temperature cannot already resolve big regions (parts of the green background) dividing them into multiple segments. The faster the cooling is, the less of the spin configuration space Ω can be explored resulting in inconsistent segmentation results despite the reached equilibrium state.

The next two best segmentation results are obtained by schedules (2) and (3). Both schedules are slow ($\gamma = 0.9999$) and start with still relatively high temperatures $T_0 = 5.0$ and $T_0 = 3.0$, respectively, allowing many uphill moves at the beginning of cooling. Derived segmentation results are consistent except one part of the green background divided into two segments. Schedules (4), (6), (7), (9) are either too fast or start with very low temperatures making the exploration of the spin configuration space Ω very limited. Such schedules lead to "overfrozen" spin configuration where the low temperature excludes uphill moves completely, while homogeneous regions are divided into multiple segments. Schedules (10), (11), (12) are even more critical, since the spin configuration is "overfrozen" already after about 100 iterations and only a few homogeneous regions are segmented properly. Although schedule (12) starts at a high temperature $T_0 = 9.0$, it is so fast ($\gamma = 0.9$) that no region can be segmented properly and the final spin configuration represents a set of superpixels.

Drawing a conclusion, the best annealing schedule of the on-line adaptive cooling employed in the Metropolis algorithm for the image segmentation is schedule (1) with $T_0 = 9.0$ and $\gamma = 0.9999$. All other schedules do not lead to complete segmentation leaving some homogeneous regions divided into multiple segments. The Metropolis algorithm with the simulated annealing schedule (1) applied to the image segmentation problem is demonstrated in more detail in Fig. 2.4. Here both the synthetic image "Colored blocks" and the real image "Various objects" of size 160×128 pixels are segmented employing the on-line adaptive cooling with the same fixed factor $\gamma = 0.9999$ and the starting temperature $T_0 = 9.0$. The factor α in the Metropolis algorithm equals 2.5 for both images. The following spin configurations are shown

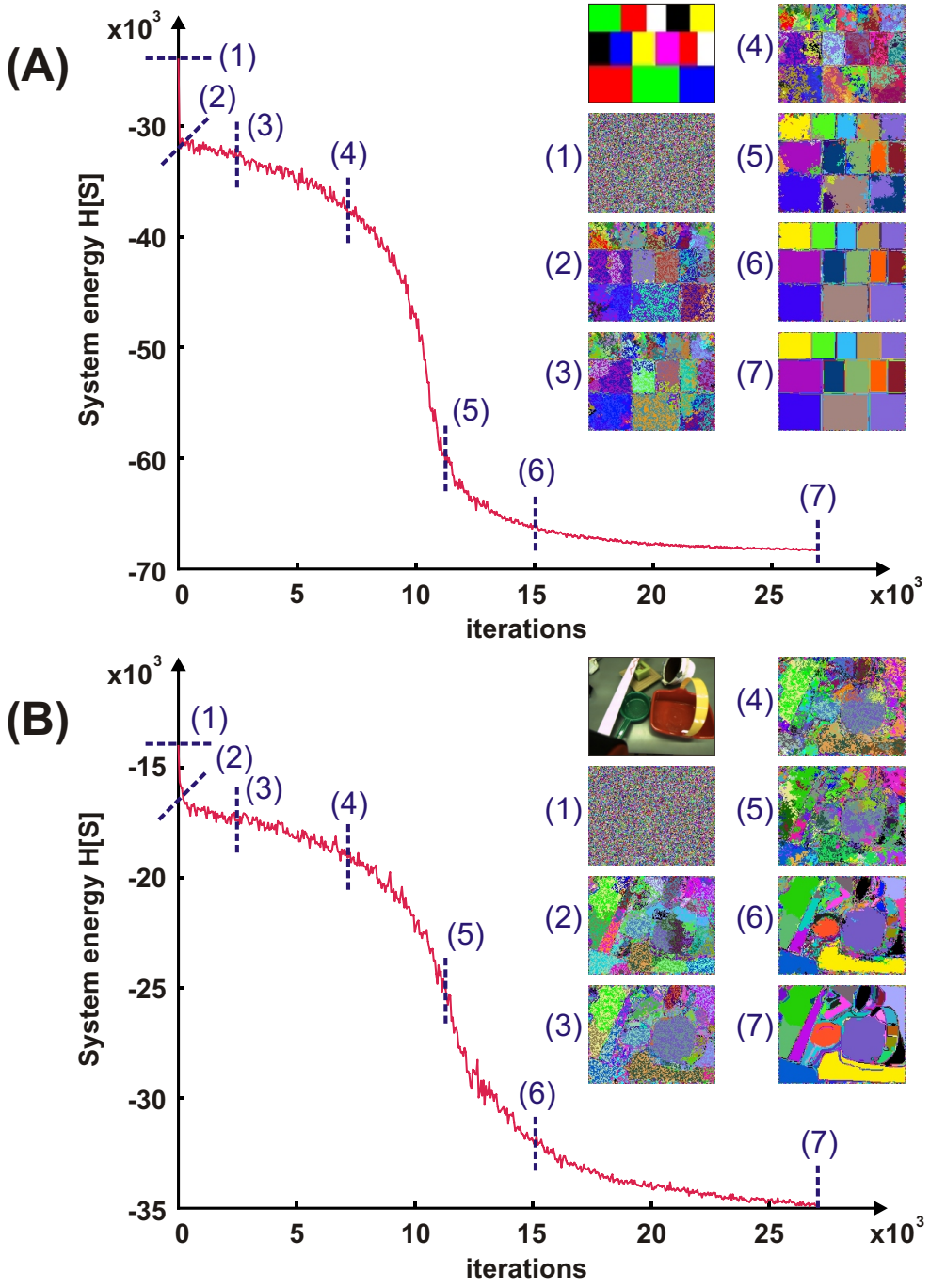

Figure 2.4: The Metropolis algorithm with on-line adaptive cooling applied to the image segmentation problem is shown for two testing images of size 160×128 pixels: "Colored blocks" (A) and "Various objects" (B). For both images $\alpha = 2.5$, the cooling factor $\gamma = 0.9999$, and the starting temperature $T_0 = 9.0$.

	Processing time (ms)	
	Metropolis on CPU	Metropolis on GPU
160×128 (px)	6.5	0.7
320×256 (px)	26.0	1.6
640×512 (px)	105.0	6.1
1280×1024 (px)	370.0	23.5

Table 2.2: Processing times of one Metropolis iteration for four various image sizes on both CPU and GPU platforms.

on the annealing process: (1) initial configuration, when all spin variables are initialized randomly, (2) - (6) intermediate configurations, and (7) final spin configuration after convergence. We can see that configurations (1) - (4) for the "Colored blocks" image and (1) - (5) for the "Various objects" are featured by a high disorder of spins due to the high temperature values allowing many uphill moves. However, namely extensive spin updates at high temperatures ensure convergence of the system to the equilibrium state [8]. The local nature of the Metropolis algorithm results in thousands of updating iterations needed for convergence. As a result about 20×10^3 iterations are needed for segmentation of the "Colored blocks" and "Various objects" images, respectively.

Processing time of one Metropolis iteration on the traditional CPU architecture as a function of the total number of pixels is shown in Fig. 2.3(B). Processing times for four commonly used resolutions are shown in Table 2.2. We can see that 20×10^3 update iterations for an image of size 160×128 pixels result in about 130 seconds on the traditional CPU architecture. The same number of iterations applied to an image of size 640×512 pixels will already result in 35 minutes. But 20×10^3 iterations are definitely not enough to segment completely an image of this size. The given time performance excludes the use of the Metropolis algorithm with the chosen annealing schedule running on common computers for the real-time video segmentation. However, in the meanwhile the local nature of the Metropolis updates is the key to its acceleration on parallel hardware like GPU. Time performance of one Metropolis iteration on the GPU GeForce GTX 580 as a function of the total number of pixels is shown for comparison in Fig. 2.3(B) as well. Processing times for four commonly used resolutions are given in Table 2.2. We can see that 20×10^3 update iterations for an image of size 160×128 pixels require near 14 seconds on the GPU which is 9.3 times faster as compared to the CPU. The same number of iterations applied to an image of size 640×512 pixels require 32 seconds on the GPU which is 65.6 times

[8] In crystal physics it is denoted as a "steady state".

faster. But even the runtime obtained on the GPU is not sufficient for the real-time processing and additional solutions are needed for our aims.

2.2.4 Graphics processing unit architecture

A Graphics Processing Unit (GPU) is a symmetric architecture featured by a highly parallel, multithreaded, manycore processors with enormous computational power and very high memory bandwidth. GPUs are usually connected to the CPU via the PCI Express bus resulting in a heterogeneous system where the GPU, called *the device*, operates asynchronously from the CPU, referred as *the host*, enabling simultaneous execution and data transfer. Parallel computations running on the GPU or device are initiated and controlled by the CPU or host (Brodtkorb et al., 2010). AMD, Intel, and Nvidia are the main GPU vendors, where AMD and Nvidia dominate these days the market of the 2D and 3D graphics rendering. The parallel programming model of Compute Unified Device Architecture (CUDA) proposed by Nvidia in 2006 (Lindholm et al., 2008) makes software parallelization on Nvidia's graphics cards quite straightforward which turns Nvidia to the leader on the market of general-purpose parallel computing with the GPUs. In the following, we will give an overview of the up-to-date Nvidia GPUs architecture and the CUDA programming model.

General-purpose parallel computing architecture

The most recent GPUs of Nvidia perform more than 1500 giga floating-point operations per second (GFLOP/s) with single precision (GeForce GTX 580) and more than 500 GFLOP/s with double precision (Tesla C2050) having a memory bandwidth of 190 GB/s. In the sense of processing power and memory transfers GPUs considerably outperform traditional CPUs which makes them very attractive for general purpose programming. The reason behind the extremely high processing capabilities of the GPU is that the GPU is designed for intensive, highly parallel computations devoting more transistors to data processing rather than data cashing and flow control (NVIDIA-Corporation, 2011). Here we introduce the GPU architecture on the first Fermi based GPU, the GeForce GTX 580 (NVIDIA-Corporation, 2009).

GeForce GTX 580 with compute capability 2.0, implemented with 3.0 billion transistors, features up to 512 CUDA cores. A CUDA processor core executes a floating point or integer instruction per clock cycle for a thread. The architecture of the GTX 580 card is shown in Fig. 2.5. The 512 CUDA cores are organized in 16 streaming multiprocessors (SMs) of 32 cores each. The GPU has six 64-bit memory partitions, for a 384-bit memory interface, supporting up to a total of 6 GB of GDDR5 DRAM memory. The GPU is connected to the CPU via PCI Express interface.

The GTX 580 is based on the third generation of SMs which are the result of several architectural innovations making the GPU easier to program and more ef-

ficient. Each SM contains 32 CUDA processor cores - four times more comparing to the previous generation. Each core has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU). Each SM has 16 load/store units which make it possible to calculate source and destination addresses for sixteen threads per clock cycle. Supporting units perform loads and stores of the data at each address to cache or global memory. There are four special-function units (SFUs) that execute transcendental instructions such as as *sine*, *cosine*, *reciprocal*, and *square root*. Each SFU performs one instruction per thread, per clock cycle.

The SM schedules and executes threads in groups of 32 parallel threads, called *warps*. Each SM has two warp schedules and two instruction dispatch units, allowing two warps to be issued and executed at the same time. The dual warp scheduler of the Fermi card selects two warps and sends one instruction from each warp to a group of 16 cores, 16 load/store units, or 4 SFUs. Since warps execute independently, the scheduler does not need to check for dependencies within the instruction stream.

On-chip shared memory is one of the key innovations that greatly improved the programmability and performance of GPU applications. Shared memory allows threads within the same block (see Section 2.2.4) to cooperate, greatly reducing the off-chip data traffic. Usage of on-chip shared memory is a key for many high-performance CUDA applications, since it is faster than local and global memory spaces (DRAM). To achieve high read/write bandwidth, shared memory is organized as a set of equally-sized memory modules, called *banks*, which can be accessed simultaneously. Any shared memory request made of n addresses that accesses n distinct memory banks can, therefore, be serviced simultaneously leading to a bandwidth that is n times higher than a bandwidth of a single module. G80 and GT200 architectures have 16 KB of shared memory per SM. In the Fermi architecture each SM has 64 KB of on-chip memory that can be configured as 48 KB of shared memory plus 16 KB of L1 cache or as 16 KB of shared memory plus 48 KB L1 cache. For applications that make extensive use of shared memory, this flexibility leads to significant performance improvements.

Besides on-chip L1 cache the Fermi card features a 768 KB unified L2 cache that services all load, store and texture accesses. L2 cache enables efficient and high speed data sharing across the GPU. Algorithms for which data addresses are not known in advance especially benefit from the cache hierarchy introduced in the Fermi cards. Commonly used in computer vision and image processing convolution and filter kernels, requiring multiple SMs for reading the same data, also benefit.

One of the most important innovations of the Fermi architecture is its two-level, distributed thread scheduler. On the card level, the global scheduler, called *GigaThread*, distributes thread blocks to various SMs, whereas on the SM level each warp scheduler distributes warps of 32 threads to its processing units. The first generation GigaThread engine introduced in G80 cards can manage up to $12,288$ threads in real-time. Furthermore, the Fermi architecture provides higher thread throughput, significantly faster context switching, concurrent kernel execution and improved

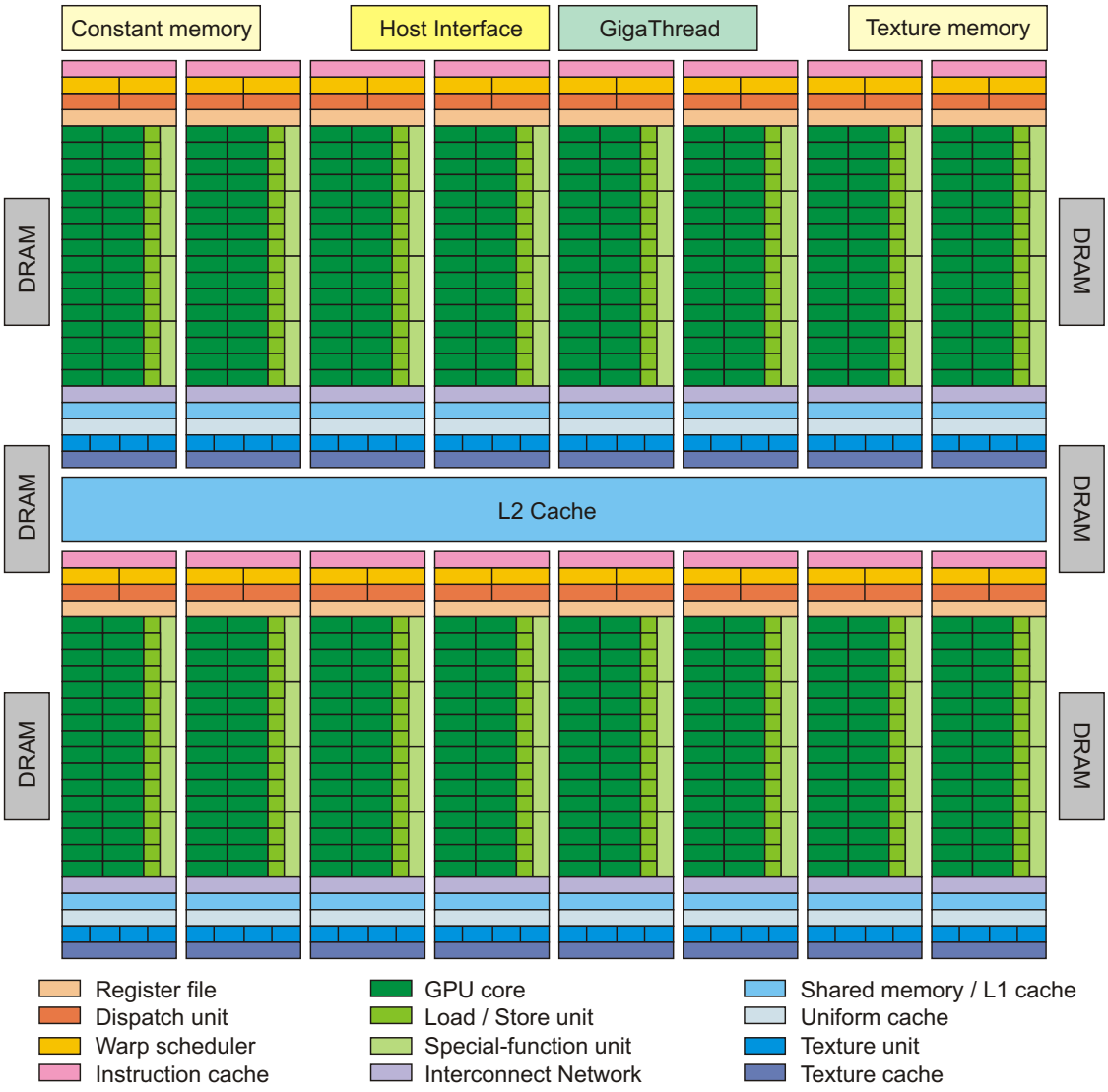

Figure 2.5: Architecture of the first Fermi based GPU device GeForce GTX 580. 16 streaming multiprocessors (SMs) are located around a common L2 cache. Each SM is shown as a vertical rectangular strip.

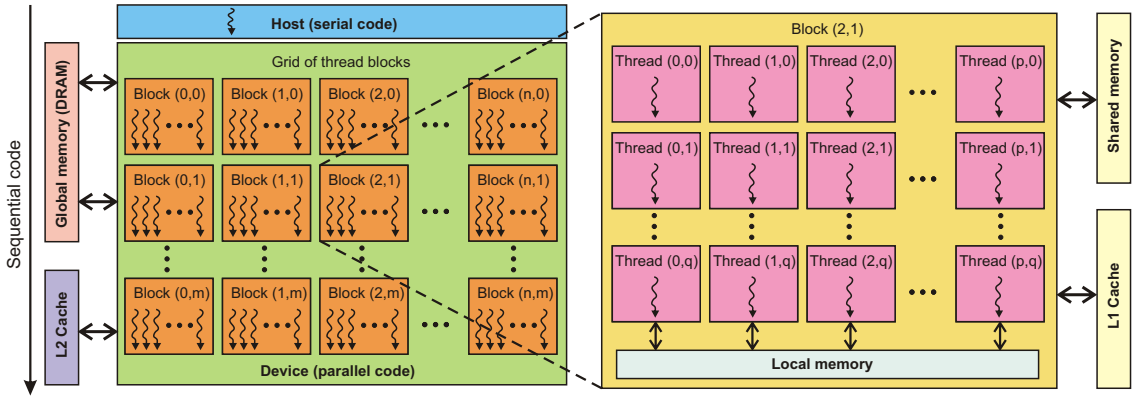

Figure 2.6: Heterogeneous programming with the CUDA programming model. Grid of thread blocks and memory hierarchy.

thread block scheduling.

Highly parallel systems consisting of multicore CPUs and manycore GPUs follow Moore's law according to which the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years (Moore, 1965). The main issue is to develop a program that transparently scales its parallelism in order to exploit the increasing number of processing cores similar to 3D graphics applications. For this purpose the CUDA programming model was designed in order to solve this issue avoiding excessive demands on developer's programming skills.

CUDA programming model

First attempts to exploit the GPU for non-graphical applications due to its high arithmetic throughput and memory bandwidth have been done in 2003. Various data parallel algorithms have been moved to the GPU using high-level shading languages such as DirectX, OpenGL, and Cg which led to remarkable performance speedups comparing to implementations on traditional CPUs. The first efforts of using the GPU for the general purpose programming are known as *GPGPU programming* (Brodtkorb et al., 2010). Whereas the GPGPU programming demonstrated great speedups, there were several crucial drawbacks. First, it required from developers very deep knowledge of graphics APIs and GPU architecture. Second, programming tasks had to be expressed in terms of vertex coordinates, textures and shader programs, which resulted in a high program complexity. Third, basic programming operations such as random reads and writes to memory were not supported, restricting the programming model. Finally, the lack of double precision support (until last four years) excluded implementation of some scientific applications.

To overcome these problems, Nvidia introduced the software and hardware ar-

chitecture CUDA that makes the GPU available for programming with a variety of high level programming languages. CUDA represents a new way of using the GPU. Instead of low level programming of graphics units with graphics APIs, developers can write now C programs with CUDA extensions for massively parallel processor. Obviously only algorithms having parallel independent computations, which can be implemented in a multithreaded mode, can be ported to the GPU. According to the parallel CUDA model, a multithreaded program is partitioned into blocks of threads running independently of each other. Therefore, a GPU with more cores needs less time to finish the program than a GPU with fewer cores.

The CUDA programming model with a hierarchy of blocks and threads is shown in Fig. 2.6. All computations, running in parallel on the GPU, are initiated by the host CPU. Since the GPU cannot access CPU's RAM memory directly, all input data needs to be loaded first to the GPU's global memory. Data transfers between CPU and GPU are organized via the PCI Express bus. Once output data is ready, it needs to be copied from GPU's global memory back to the main CPU program. All blocks run on the GPU the same program, called *kernel*, and threads within one block can synchronize and communicate using shared memory. Blocks are organized into a grid and the number of blocks depends on both the size of the data being processed and resources of SMs.

When the main CUDA program running on the CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to SMs with available card's execution *occupancy*. Occupancy is the ratio of active warps to the maximum number of supported warps (Brodtkorb et al., 2010). Threads of a thread block run concurrently on one SM, while multiple thread blocks can execute at the same time on one SM. Once previous blocks terminated, new thread blocks start on the free SM. Since SM was designed to execute simultaneously hundreds of threads, it employs a unique architecture called *SIMT* (Single Instruction, Multiple Thread). The SIMT architecture is descended from the *SIMD* (Single Instruction, Multiple Data) organization in that a single instruction operates on multiple data elements. A difference is that SIMD is related to the software, while SIMT instructions specify the execution and branching of a single thread (NVIDIA-Corporation, 2011). Each GPU architecture has a predefined maximum number of resident blocks and a number of resident warps. The total number of warps W_{block} in a block is defined as

$$W_{\text{block}} = ceil\left(\frac{N_T}{W_{\text{size}}}, 1\right), \tag{2.37}$$

where N_T is the number of threads per block, W_{size} is the warp size, which is equal to 32, and the function $ceil(x, y)$ is equal to x rounded up to the nearest multiple of y. The total number of registers R_{block} allocated for a block for devices with a compute capability 1.x is given by

$$R_{\text{block}} = ceil(ceil(W_{\text{block}}, G_W) \times W_{\text{size}} \times R_K, G_T), \tag{2.38}$$

and for devices with compute capability 2.x by

$$R_{\text{block}} = ceil(R_K \times W_{\text{size}}, G_T) \times W_{\text{block}}, \tag{2.39}$$

where G_W is the warp allocation granularity, equal to 2 (compute capability 1.x only), R_K is the number of registers used by the kernel and G_T is the thread allocation granularity. G_T equals to 256 for devices of compute capability 1.0 and 1.1, to 512 for devices of compute capability 1.2 and 1.3, and to 64 for devices of compute capability 2.0. The total amount of shared memory S_{block} (in bytes) allocated by a block is given by

$$S_{\text{block}} = ceil(S_K, G_S), \tag{2.40}$$

where S_K is the amount of shared memory used by the kernel (in bytes) and G_S is the shared memory allocation granularity, which is equal to 512 for devices of compute capability 1.x, and to 128 for devices of compute capability 2.x.

Memory hierarchy

Parallel threads running on CUDA processor cores can access data from multiple memory spaces as shown in Fig. 2.6. Each thread has its private local memory (register file) located on the chip which has the same lifetime as a thread. Each thread block has on-chip shared memory visible to all threads of the block and with the same lifetime as a block. All threads have access to the same global memory (see Fig. 2.5). There are also two additional read-only memory spaces available for all threads: the constant and texture memory spaces.

As was already mentioned, shared memory is on-chip memory and, therefore, it is much faster than the local and global memory spaces. Shared memory of the Fermi card has 32 banks that are organized such that successive 32-bit words are assigned to successive banks. Each bank is featured by a bandwidth of 32 bits per two clock cycles. A *bank conflict* only occurs if two or more threads try to access any bytes within different 32-bit words belonging to the same bank. An access of any bytes within the same 32-bit word by two or more threads does not lead to bank conflicts. Unlike for devices of lower compute capability, in devices of compute capability 2.x bank conflicts can occur between a thread belonging to the first half of a warp and a thread belonging to the second half of a warp.

Device or global memory can be accessed from SMs directly as *linear memory* or through the texture units that use the texture cache as *CUDA arrays* (see Fig. 2.5). Textures are a computer graphics data concept representing data as a read-only 2D image. Thus, texture access is optimized for 2D access keeping small 2D neighborhoods in the cache as opposed to the linear caching of traditional CPUs. Global memory access has a high latency and is optimized for linear access. Full bandwidth is achieved only when the memory requests are *coalesced* into the read or write of

the full memory segment. Global memory can be accessed via 32-, 64-, or 128-byte memory transactions which must be naturally aligned. Only 32-, 64-, or 128-byte segments of global memory that are aligned to their size (i.e., whose first address is a multiple of their size) can be read or written by coalesced memory transactions. For devices of compute capability 2.x, the memory transactions are cached exploiting data locality to reduce impact on throughput (NVIDIA-Corporation, 2011).

2.2.5 Parallel Metropolis algorithm

As was mentioned above, the Metropolis algorithm is highly local and, therefore, fits perfectly to parallel hardware architectures. Barkema and MacFarland (1994) proposed two parallel implementations of Ising model simulations with the Metropolis algorithm on the KSR-1 parallel computer. According to their first implementation, called *a straightforward parallel Metropolis algorithm*, the lattice is divided into domains and each processor, operating on one domain, selects sites accepting or rejecting spin flips on these lattice sites. Due to the nature of the Metropolis procedure it is desirable to select sites on which moves are proposed in a random way, since it helps to avoid biases caused by the *pseudorandom* nature of the random number generation (Compagner and Hoogland, 1987). However, a straightforward simultaneous update of lattice sites leads to a conflict when two different processors select adjacent sites ϱ_i and ϱ_j at the same time. In this situation the spin values at sites ϱ_i and ϱ_j might be updated both based on old spin values. Such a "combined" spin update does not fulfill detailed balance and produces biased results. The correct way to deal with this situation is to update spin variables of adjacent lattice sites sequentially, which in its turn requires additional constraints on the spin updates and slows down the algorithm. In the straightforward parallel Metropolis algorithm by Barkema and MacFarland (1994) such situations are avoided by using the same sequence of random sites on each processor. Synchronization between processors must take place if a border site is selected which means that we have to synchronize B times during one Metropolis iteration if the border of each domain has B sites. According to this updating strategy processors select sites that are periodic images of each other which, therefore, are never adjacent. Although the lattice sites are not selected strictly at random, this selection method causes no bias in the equilibrium properties.

The second parallel implementation of the Metropolis algorithm, called *a smart parallel Metropolis algorithm*, limits considerably the number of times at which processor synchronization has to take place. Each processor operates on lattice sites from its own domain and has copies of the spin values on the adjacent sites of the neighboring domains. As in the first implementation, the domain sites on which a spin move is proposed are determined by a generated random pattern which is equal for all processors. Spins of internal sites can be updated without an additional communication between processors. If a border site is selected, the copy of that site located on the adjacent neighboring processor may be out of date. Since the site selection is

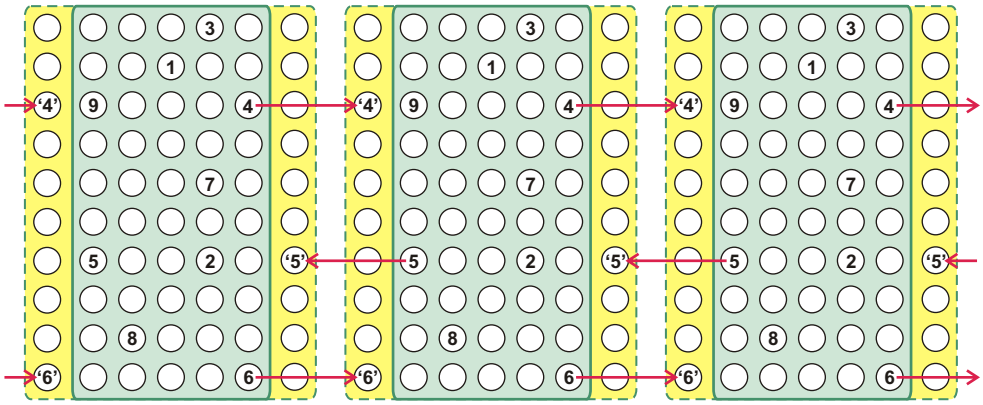

Figure 2.7: Update of a spin state configuration by the "smart" parallel Metropolis algorithm proposed by Barkema and MacFarland (1994). The dotted sites indicate copies of spin values located on the adjacent border of the neighboring domains. See the text for more details.

synchronized between processors, each processor can control which copies of border sites are still up-to-date. Synchronization between all processors takes place only when information needed for accepting or rejecting a move is not up-to-date. Reducing the frequency of synchronization operations results in a faster runtime. The "smart" parallel Metropolis algorithm is illustrated in Fig. 2.7. Each processor selects sequentially the lattice sites $1-9$. When the site number 4 is selected, its spin value is updated based among others on the copy of the spin value taken from the neighboring domain. At the same time the site denoted by "4" is marked to be out of date in the next processor. Spin updates continue and no synchronization is needed till site 9 is selected. Here synchronization between all processors must take place, since the copy of the spin value at the site marked "4" may have changed. After synchronization copies of all border sites are current. This decreases the number of synchronizations during one Metropolis iteration to $\sqrt{2B/\pi}$ instead of B (Barkema, 1992).

Now we present a new implementation of the parallel Metropolis algorithm using the GPU architecture for acceleration. The parallel Metropolis algorithm applied to the image segmentation problem represents a very intensive computational process where all data items (spin variables of pixels) repeatedly undergo the same procedure (update of spin states). Such computations are exactly what GPUs with the SIMT architecture (see Section 2.2.4) have been designed for. Similar to the implementations considered before, an input image, represented by a 2D lattice with $N = L_x L_y$ sites, is divided into a number of domains where each domain is a rectangle containing one image part. According to the CUDA programming model all domains are handled on the GPU by thread blocks organized a thread grid. Before the CUDA kernel, performing Metropolis updates, can be launched, pre-computed interaction strengths

and an initial spin state configuration are stored in the global memory of the GPU (see Section 2.2.4). Due to the fact that global memory accesses are very time consuming, we need to reduce its usage in the kernel as much as possible. Thereby each thread block first loads input data from the corresponding domain to the shared memory which is then repeatedly accessed during spin updates. Regarding read and write accesses to the global memory there are some memory issues that need to be taken into account for a better performance.

First, since each thread block can access only its own shared memory, information about border pixels from the neighboring domains needs to be copied to the shared memory of each block as well. Therefore, each block loads both a corresponding domain and a so-called "read-only" overlap of border pixels from adjacent domains. Second, global memory accesses need to be coalesced in order to achieve high throughput (see Section 2.2.4). For this purpose memory transactions need to be aligned which requires a rearrangement of the input data for the CUDA architecture and a decomposition of the output data for the further processing on the host side. A flow diagram demonstrating an execution of the parallel Metropolis algorithm on the GPU with rearrangements of the input / output data is shown in Fig. 2.8(A). In the proposed implementation a block of 16×16 threads is used and each thread loads and updates four pixels. Such a configuration makes it possible to avoid idle threads (only loading data from overlaps without performing any spin updates) and to use the resources of the GPU in a very efficient way. Once all spin updates are complete, the global memory is used again for saving the final spin state configuration which is sent later back to the host.

The rearrangement of data in the device global memory is shown schematically in Fig. 2.8(B-D). Note that the data rearrangement is needed only for maximization of global memory throughput and both on the host side and within thread blocks (in the shared memory) data is saved in a common image format in compliance with the 8-connectivity of pixels (see Fig. 2.8(B,D)). For an image of size $width \times height$ pixels we need to load coupling constants (a byte each) pre-computed for four different directions (see Section 2.2.2) and a current spin value (one byte) as input data for each pixel to the device global memory. It results in five matrices of size $width \times height$ bytes. Since according to our loading strategy one thread loads and updates four pixels, the simplest parallel implementation of the Metropolis algorithm is when one thread loads and updates a square of 2×2 pixels (see Fig. 2.8(D)). Thereby each thread needs to load four bytes from each matrix to the shared memory. In order to coalesce accesses of the global memory, the best and obvious solution is a rearrangement of 2×2 pixel groups (in all five matrices) in segments of 4 consecutive pixels (see Fig. 2.8(C)). Such segments meet the size and alignment requirements leading to efficient global memory accesses.

The proposed CUDA kernel performing updates of spin configurations due to the introduced loading strategy is shown schematically in Fig. 2.9. We can see that each vertical overlap contains two pixel columns and each horizontal overlap contains two

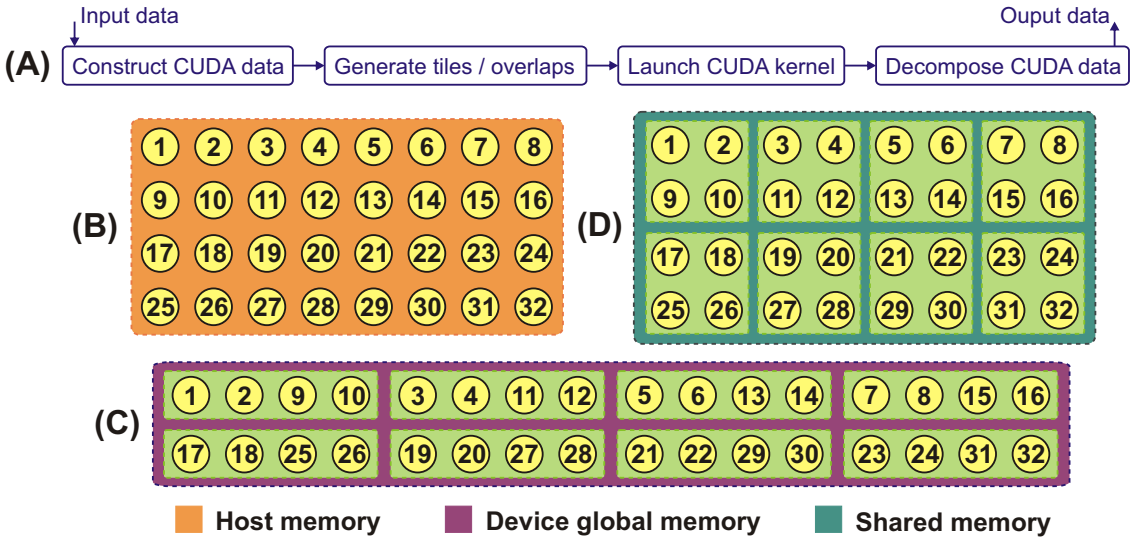

Figure 2.8: The rearrangement of input / output data in the device global memory shown schematically for a group of 32 pixels from one matrix. (A) Flow diagram of the kernel function performing parallel Metropolis updates on the GPU architecture with rearrangements of input / output data. (B) An input data in the regular image format in the host memory. (C) A reorganization of the input data in the device global memory for coalesced memory accesses. (D) An input data in the regular image format loaded from the global memory to the shared memory by a thread block.

pixel rows. Each thread block updates only inner pixels of the overlaps, whereas outer pixels are updated by neighboring blocks. As an update of one spin involves only its nearest neighbors from the 3×3 mask, spin variables that are not neighbors of each other can be updated simultaneously. It results in four parallel updates performed in a sequence. Synchronization between thread blocks takes place once all spin variables are updated, i.e., when four parallel updates finished and outer pixels of all overlaps need to be reloaded before the next iteration can start. Here we do not involve any random pattern for updates like in the "smart" parallel Metropolis algorithm by Barkema and MacFarland (1994) and update as many spins as possible at each step. Despite some biases caused by this updating strategy, Metropolis updates are performed extremely fast without checking which spin values are out-of-date. All border spins are out-of-date after one full iteration and synchronization between thread blocks must take place. Synchronization runs over the device global memory and takes place n times where n is the number of full Metropolis iterations. GPU occupancy data and physical limits of the proposed CUDA kernel are summarized in Appendix A.1.

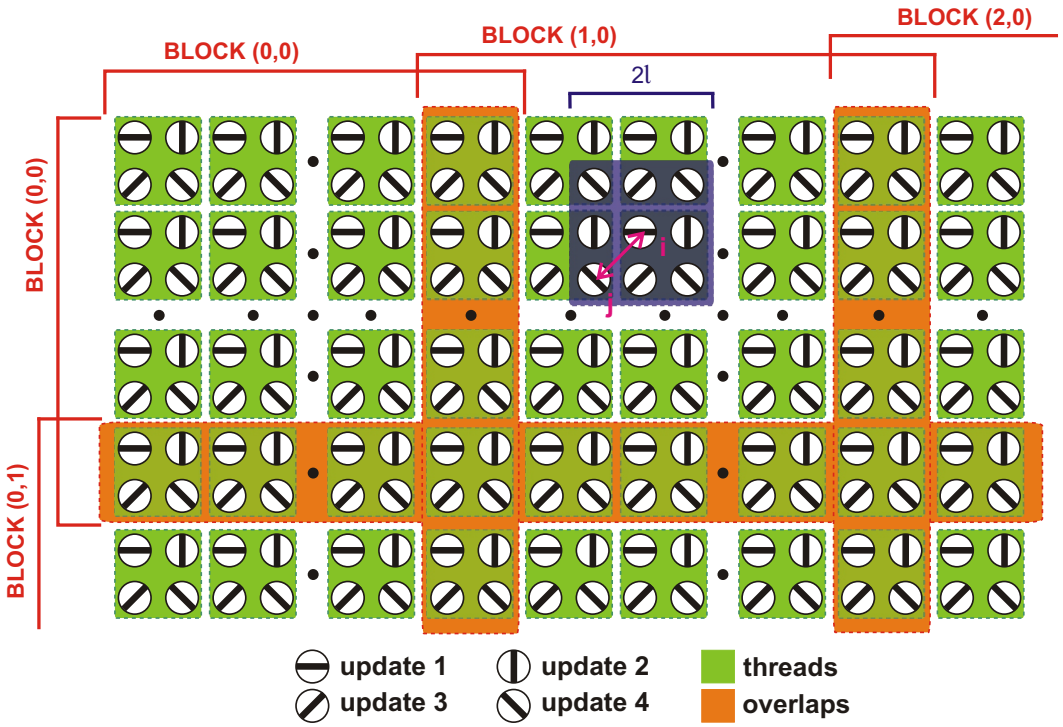

Figure 2.9: Update of a spin state configuration of the input image by the proposed parallel Metropolis algorithm on the GPU. Pixels depicted by the same pattern are updated simultaneously. The dark blue region includes $(2\ell+1)^2-1$ pixels from the closest neighborhood of the pixel i ($\ell=1$). The arrow shows an interaction between pixels i and j – one of eight interactions of the pixel i in its neighborhood (see (2.5)).

Time performance of one Metropolis iteration on the GPU has been already given in Section 2.2.3 (see Fig. 2.3(B) and Table 2.2) for comparison with the sequential implementation on the traditional CPU. The presented GPU runtimes have been obtained on the GeForce GTX 580. The parallel Metropolis update running in parallel on the GPU is significantly faster as compared to its sequential version running on the CPU. Thus, for image sizes of 160×128, 320×256, and 640×512 pixels the rates of acceleration are 9.3, 16.3, and 17.2, respectively. However, even these accelerations are not enough for the real-time image segmentation by the parallel Metropolis algorithm with the chosen simulated annealing schedule and, as a consequence, some additional optimizations are required. While slow annealing leads to an undesired increase in computation time, fast cooling faces the problem of fragmentation of homogeneous regions into multiple segments (see Fig. 2.3 in Section 2.2.3). In the upcoming section we introduce the parallel Metropolis algorithm that makes use of the "overfrozen" spin configurations, produced by fast annealing schedules, applying a special short-

cut procedure for the real-time performance.

2.2.6 Parallel Metropolis algorithm with a short-cut

The flow diagram of the parallel Metropolis algorithm with a short-cut is presented in Fig. 2.10(A). As in the full Metropolis procedure, at the beginning all spin states are initialized by random values. After that the first group of $n_1 = 10$ parallel Metropolis iterations with the on-line adaptive annealing is applied to the image. Note that here the updating procedure starts at a very low temperature ($T_0 = 0.5$) and a relatively fast annealing schedule is employed ($T_{n+1} = 0.999 \cdot T_n$). It leads to the frozen state when an image is not segmented but all regions are featured by well observable superpixels which emphasize region borders. Output results of all processing steps are shown for the "Cluttered scene" image in Fig. 2.10(B).

This effect is known as *domain fragmentation* and describes the fact that large uniform areas are being split into sub-segments despite high attractive forces within them (Eckes and Vorbrüggen, 1996). It happens when the starting temperature T_0 is very low or decreases rapidly (too fast annealing process) and the system arrives too early at the "frozen" state. However, in the meanwhile the fragmented domains carry all the required information needed to resolve this problem due to the property that domain-fragment boundaries are unstable and clear-cut, whereas true segment boundaries are stable and characterized by a noisy local neighborhood. It allows us to distinguish true segment boundaries from those caused by domain fragmentation and find potential segments as regions featured by closed contours. Since border detection based on the described border properties of superpixels cannot resolve the image segmentation problem completely giving inaccurate and unidentifiable segments, we need to come back to the spin representation of the image for final processing. For this purpose all connected components are identified and their labels are turned to spin variables. Therefore, the short-cut consists of the following three steps: border detection based on superpixels, labeling of connected components, and reassignment of spins. Once the short-cut is over, the second group of $n_2 = 10$ Metropolis iterations with the same annealing schedule is applied to the image resulting in more accurate segments. This procedure is also called *system relaxation* and runs until convergence. After that final segments larger than a pre-defined threshold can be extracted.

The system convergence of the employed on-line adaptive cooling with the short-cut is shown in Fig. 2.10(C). We can see that during the first group of Metropolis iterations the system energy drops down very fast. However, the potential convergence here means only that the spin configuration is frozen and no uphill moves are taking place. Thus, the system convergence does not necessarily mean a solution of the image segmentation problem. Right away after the short-cut the system energy slightly increases due to disorder introduced into the system by randomly initialized spin variables that fill gaps between found connected components. Additional Metropolis relaxation iterations lead to convergence which in this case corresponds to the final

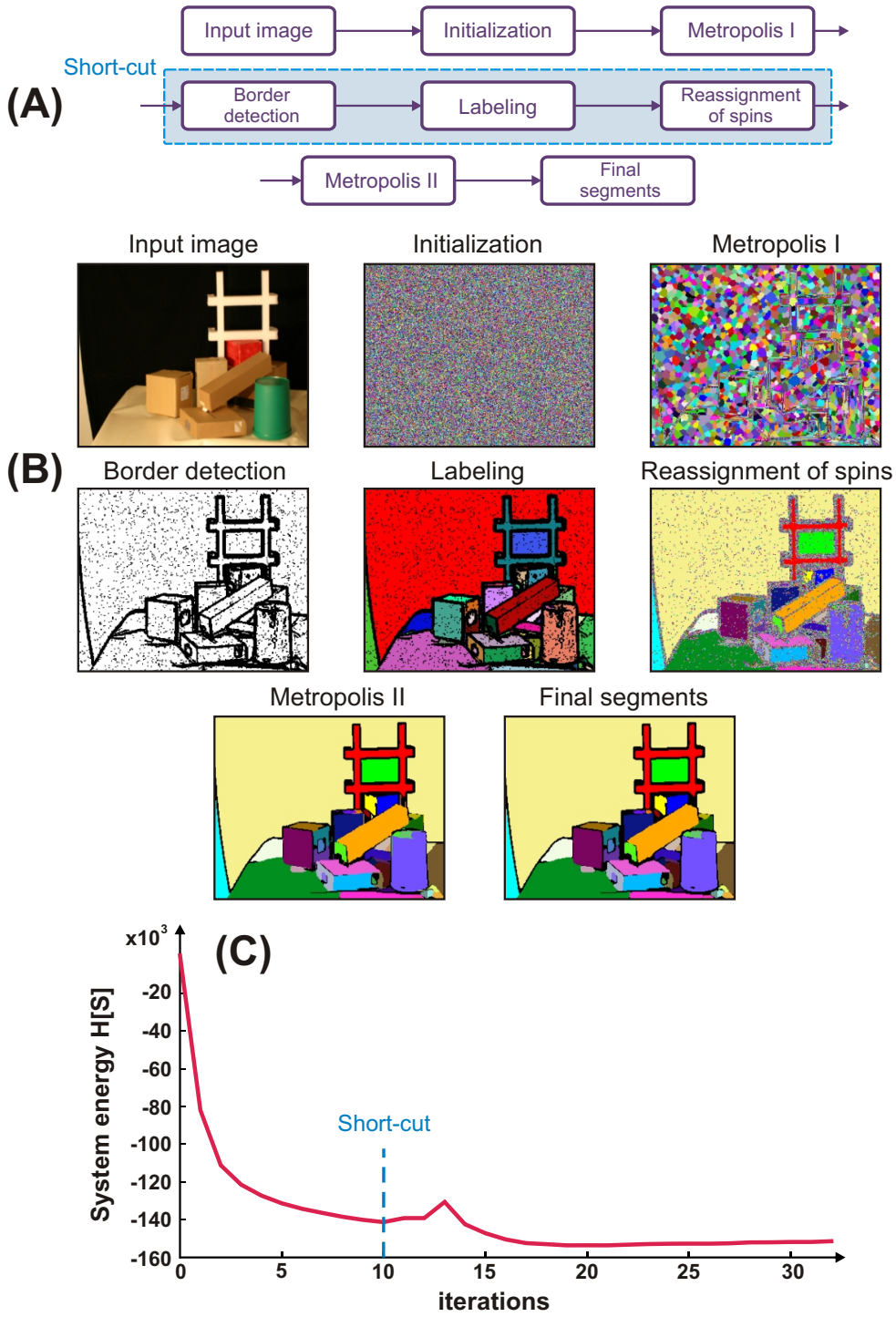

Figure 2.10: The parallel Metropolis algorithm with the proposed short-cut. (A) Flow diagram of the algorithm. (B) Results obtained on each stage. (C) Convergence of the Metropolis algorithm with the on-line adaptive cooling using the short-cut against the number of iterations for the "Cluttered scene" image of size 320×256 pixels, $\alpha_1 = 0.9$, $\alpha_2 = 4.0$, $n_1 = 10$, $n_2 = 20$, $T_0 = 0.5$, and $\gamma = 0.999$.

segmentation result.

Note that in the short Metropolis algorithm we use different values of the factor α for two groups of Metropolis updates (see (2.6)). The factor α_1 used during the first n_1 updates influences a number of connected components, i.e., found segments, therefore, it should be quite low in order to avoid undesired merges. The factor α_2 is used on the relaxation stage during the second n_2 updates when all main segments have been already found. In this respect values of α_2 can be higher than α_1 which allows the system to arrive faster at the final segmentation solution getting rid of noise in the spin configuration.

Resolving domain fragmentation

For illustration of the domain fragmentation problem the spin configuration with $q = 6$ after $n_1 = 10$ Metropolis iterations is presented for an example image in Fig. 2.11(A,B). Spin updates are performed at a very low temperature $T_0 = 0.5$ which remains constant during all iterations. Large interaction forces within the apple and the background lead to the creation of domains that try to cover each other. This effect has its origin in the finite interaction range and local dynamics of the Metropolis algorithm (Eckes and Vorbrüggen, 1996). As already mentioned above, the information about superpixels is used to resolve the domain fragmentation due to the fact that domain-fragment boundaries are unstable and clear-cut, whereas true segment boundaries are stable and characterized by a noisy local neighborhood (Fig. 2.11(B)). This holds true for real images due to their finite image gradient at true boundaries and it allows us to distinguish true segment boundaries from those caused by domain fragmentation. For this we consider the spin configuration S obtained after an initial fast cooling phase consisting of $n_1 = 10$ Metropolis iterations only.

The procedure works as follows. After the first group of Metropolis iterations we compute the spatial derivatives along the x and y direction of the spin state configuration $S(x, y)$ according to

$$S'_x = \frac{\Delta S(x,y)}{\Delta x} \qquad \text{and} \qquad S'_y = \frac{\Delta S(x,y)}{\Delta y}. \tag{2.41}$$

In Fig. 2.11(C,D) functions S_x and S'_x are depicted for one row of the original image. Each peak of S'_x represents a change in the spin state. Here we are interested only in the number of peaks rather than in the derivative values, because the Potts model does not penalize differences between certain spin states stronger than others (see (2.4)). We can see that the frequency of peaks increases significantly at real boundaries (depicted by dashed lines). Considering couples of pixels in parallel we

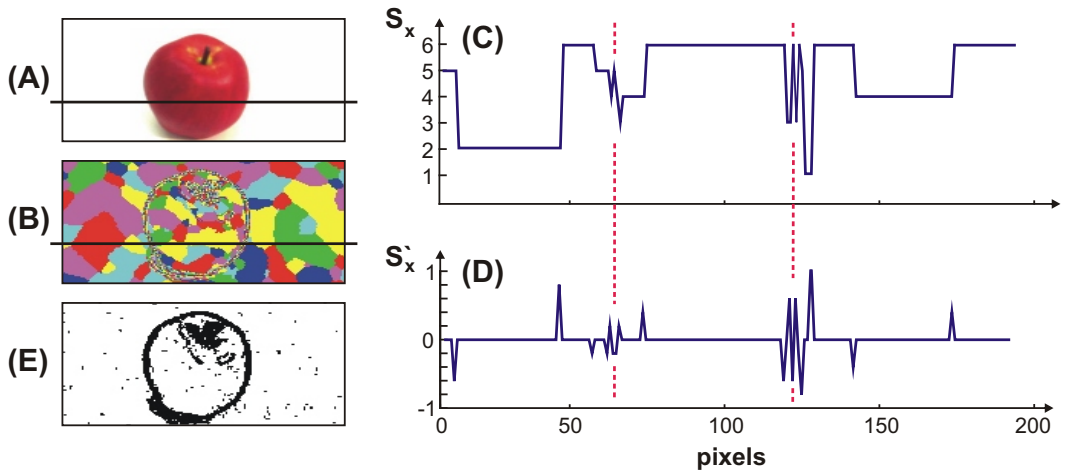

Figure 2.11: Detection of real boundaries after using the Metropolis algorithm with a very low temperature $T_0 = 0.5$. (A) Original input image. (B) Configuration of spin states after $n_1 = 10$ Metropolis iterations, $\alpha_1 = 0.9$. (C) Function of spin states for one image row as marked by a horizontal line in panels (A) and (B). (D) Changes of spin state for the same row where each peak represents a changing spin state. (E) Detected object boundaries.

find boundaries

$$B(x_i, y_j) = \begin{cases} 1 & \text{if } S'(x_i, y_j) \neq 0 \text{ and } S'(x_{i-1}, y_j) \neq 0, \\ 1 & \text{if } S'(x_i, y_j) \neq 0 \text{ and } S'(x_i, y_{j-1}) \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.42}$$

The result of this procedure is a binary image (see Fig. 2.11(E)). Since spatial derivatives can be computed for all pixels at the same time, this processing step has been implemented on the GPU as well. Erroneous noisy speckles arising from this procedure are corrected by assigning again spin variables to all found segments and applying additional Metropolis update iterations to a newly established spin configuration for the system relaxation (see further).

Note that one cannot easily use a conventional edge detector for this. An edge detector would indeed find many segment boundaries, but it would also find boundaries between superpixels which are unrelated to the segments and come out from the Metropolis procedure. In order to reach the equilibrium state of the system, i.e., final image segmentation solution, as soon as possible, we should use only "correct" segments. Otherwise the system relaxation would have to undo all wrong segments performing a very long cooling which cannot be accomplished in real-time. Furthermore, the proposed procedure yields closed object boundaries while many edge detectors produce borders having gaps. The method of using the noisiness to distinguish

Figure 2.12: Fast connected-component labeling. (A) Computed region boundaries. (B) Provisional labels assigned after the first image scan. (C) Representative labels assigned after the second image scan.

real edges from domain edges is consistent within our algorithmic framework and, as a consequence, allows continuation of the Metropolis procedure without problems.

Labeling of connected components

Once the domain fragmentation is resolved, all potential segments have closed contours. However, a binary image obtained after the border detection can contain noisy speckles, especially at object boundaries or in the textured areas. Furthermore, the binary image contains only two values and does not identify all segments in a unique way, i.e., it cannot be a solution for the image segmentation problem. To identify uniquely all connected components, i.e., areas having a closed boundary, we employ a procedure called *labeling of connected components*.

Since the considered segmentation algorithm has to be sufficient for real-time applications, we decided to use for this purpose fast connected-component labeling proposed by He et al. (2009) which is, to our knowledge, the fastest labeling algorithm at the moment. It is a fast two-scan technique that performs labeling of connected components in a binary image. The method has the following advantages: (i) it is easy to implement (having less than 50 lines in the C language); (ii) it is faster than all conventional labeling algorithms; (iii) it is featured by the ideal linearity property versus image size having complexity $O(N^2)$ for images of size $N \times N$ pixels.

All steps of the employed labeling procedure are shown in Fig. 2.12. The chosen method completes labeling in two scans of an image: during the first scan it assigns provisional labels to pixels belonging to regions, i.e., potential segments, (see Fig. 2.12(B)) and records label equivalences for labels belonging to the same region. Label equivalences are being resolved during the first scan choosing one of the equivalent labels as a representative label for each region. All representative labels are stored in the representative label table where provisional labels represent indices. During the second scan, all provisional equivalent labels are replaced by the corresponding representative label obtained from the representative label table (see Fig. 2.12(C)). Detailed evaluations of the method with testing images can be found in the study of He et al. (2009). Both image scans run on the CPU and are very fast for image sizes that are being used in our work. However, the second scan can be accelerated

on the GPU architecture, since representative labels can be assigned simultaneously to all pixels by independent parallel threads.

Employment of Metropolis for final relaxation

An image produced by the labeling of connected components could be already considered as a final segmentation solution. However, it can still contain some inaccuracies caused by arbitrary shapes of superpixels resulting in false positives of the edge detection. To get rid of noise within segments and to arrive at the final segmentation result, assigned unique labels need to be converted back to spin variables. Then the second group of Metropolis update iterations can be applied to the image leading to smoother segments and ensuring convergence of the system to the equilibrium state.

Region labels can be used either directly in the Potts model as spin states or converted to spin states if their number q is pre-defined and lower than the highest region label. In the latter case spin states are assigned to all pixels based on region labels according to

$$\sigma(x_i, y_j) = L(x_i, y_j) \mod q, \tag{2.43}$$

where mod means that the segment label $L(x_i, y_j)$ of the pixel is divided by the number of possible spin states q and the new spin state σ is the remainder of the division. For consistent Metropolis updates spin states have to be assigned to all pixels in the image even those belonging to region borders. As there are no pre-requirements for spin values on region borders, randomly chosen values between 1 and q are assigned to them. After the image has again a spin state representation, the second group of Metropolis update iterations $n_2 = 10$ is applied to obtain the final spin configuration after which final segments can be extracted (see Fig. 2.10(B)).

2.3 Segmentation results and time performance

2.3.1 Evaluation of the segmentation

Due to the fact that visual perception is not the same for all humans, each subject has its own intuitions which parts of the scene are meaningful and which are not, which parts are similar and represent one entity (segment) and which ones are dissimilar and represent different entities (segments). For example, looking at a keyboard one subject perceives it as one single object (or entity) and other subject sees all keys on the keyboard as separate objects (entities). It depends largely on the aim of the subject towards the object, e.g., whether the subject wants to move the keyboard on the table a bit further or wants to type text with it. Another example could be a green cup with stripes of slightly different color on it. This makes the problem of image

segmentation subjective and image segmentation techniques difficult to evaluate, since there are multiple correct solutions.

In computer vision there are two main strategies for evaluation of vision algorithms. The first strategy suggests that vision algorithms should be evaluated in the context of particular task or application (Borra and Sarkar, 1997). In the case of image segmentation for low- and mid-level vision tasks, it consists in evaluation how much a particular algorithm contributes to the success of higher-level procedures performing, for example, object recognition, tracking, or manipulation tasks (Estrada and Jepson, 2009). Due to this strategy, in the case of the keyboard object, a segmentation result where each key is represented by a segment is correct for text typing tasks and incorrect for actions operating on the keyboard as on a single object. However, this strategy can only give an evaluation related to a specific application and, therefore, the applicability of the method for other tasks remains unclear.

According to the second strategy vision algorithms can be evaluated in relation to some properly defined ground truth data (Martin et al., 2001). Because of the mentioned above ambiguities in the human visual perception, there is no uniquely defined ground truth data for image segmentation as in stereo vision (Scharstein and Szeliski, 2002). The ground truth segmentation S', called also *human* or *target segmentation*, is defined as an average of manual annotations S'_i of an image produced by different subjects. The ground-truth data for a large collection of natural images shows how humans perceive the visual scene and, therefore, can be considered as the most suitable segmentation of the image data. In this chapter we use the second strategy for evaluation of the proposed segmentation algorithm and present the evaluation results on a large database of natural images for which the ground truth is known. The image database, used in the current work consists of well-known real world indoors and outdoors images from the Berkeley Segmentation Database (BSD) (Martin et al., 2001) [9], Middlebury dataset (Scharstein and Szeliski, 2008) [10], and some robotic vision databases. Some sample images from the database with three different segmentations for each image are shown in Fig. 2.13 and 2.14. Each image in the database has been segmented by 3 different people using a tablet computer. In total 25 people produced the segmentation data. The ground truth data in the database has been created based on color information only ignoring additional knowledge about visible objects. In such a way an object consisting of differently colored parts had to be divided into multiple regions due to color dissimilarities.

Note that the segments produced by different humans are not identical (see especially the segmentations of the green cover in Fig. 2.13 and the plant in Fig. 2.14). But all of them are consistent, as each human observer has its own level of granularity. Due to this fact we need only those measures for evaluation of image segmentation

[9] available under http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

[10] available under http://vision.middlebury.edu/stereo/

results that do not penalize such differences. Therefore, both the quantitative and qualitative evaluations are needed to judge and compare various image segmentation techniques. The quantitative evaluation gives a numerical valuation of the machine segmentation results for the finite number of natural images taking the known ground truth data into account. The qualitative evaluation shows outputs of different segmentation methods on the same image dataset giving a chance to a user to judge the techniques and select the most appropriate one.

Quantitative evaluation

Generally, there are two possible quantitative measures of segmentation quality: those that compare the overlap between regions in multiple segmentations of the same image and those that evaluate agreement between corresponding boundaries of found segments (Estrada and Jepson, 2009). For the complete quantitative evaluation of segmentation results we use here both measures: *segmentation covering* belonging to the former and *precision* and *recall* belonging to the latter.

Segmentation covering

The idea of the *segmentation covering* metric introduced by Arbelaez et al. (2009) evaluates the covering of a machine segmentation S by a human or ground truth segmentation S'. A machine segmentation is the result of the segmentation algorithm under consideration. The covering of a segmentation S by a segmentation S' is defined as

$$C(S' \to S) = \frac{1}{N} \sum_{R \in S} |R| \cdot \max_{R' \in S'} O(R, R'), \tag{2.44}$$

where N denotes the total number of pixels in the image, $|R|$ is the number of pixels in region R, and $O(R, R')$ is the overlap between regions R and R' defined as

$$O(R, R') = \frac{|R \cap R'|}{|R \cup R'|}. \tag{2.45}$$

The covering of a machine segmentation S by a family of ground truth segmentations $\{S'_i\}$ is computed by covering S separately with each human map from $\{S'_i\}$ and then averaging over the different humans. In this way the perfect covering of the machine segmentation is achieved.

Precision and recall

The *precision and recall* evaluations have been proposed by Martin et al. (2002). *Precision* measures the percentage of boundary pixels in the machine segmentation that correspond to boundary pixels in the ground truth segmentation and, therefore,

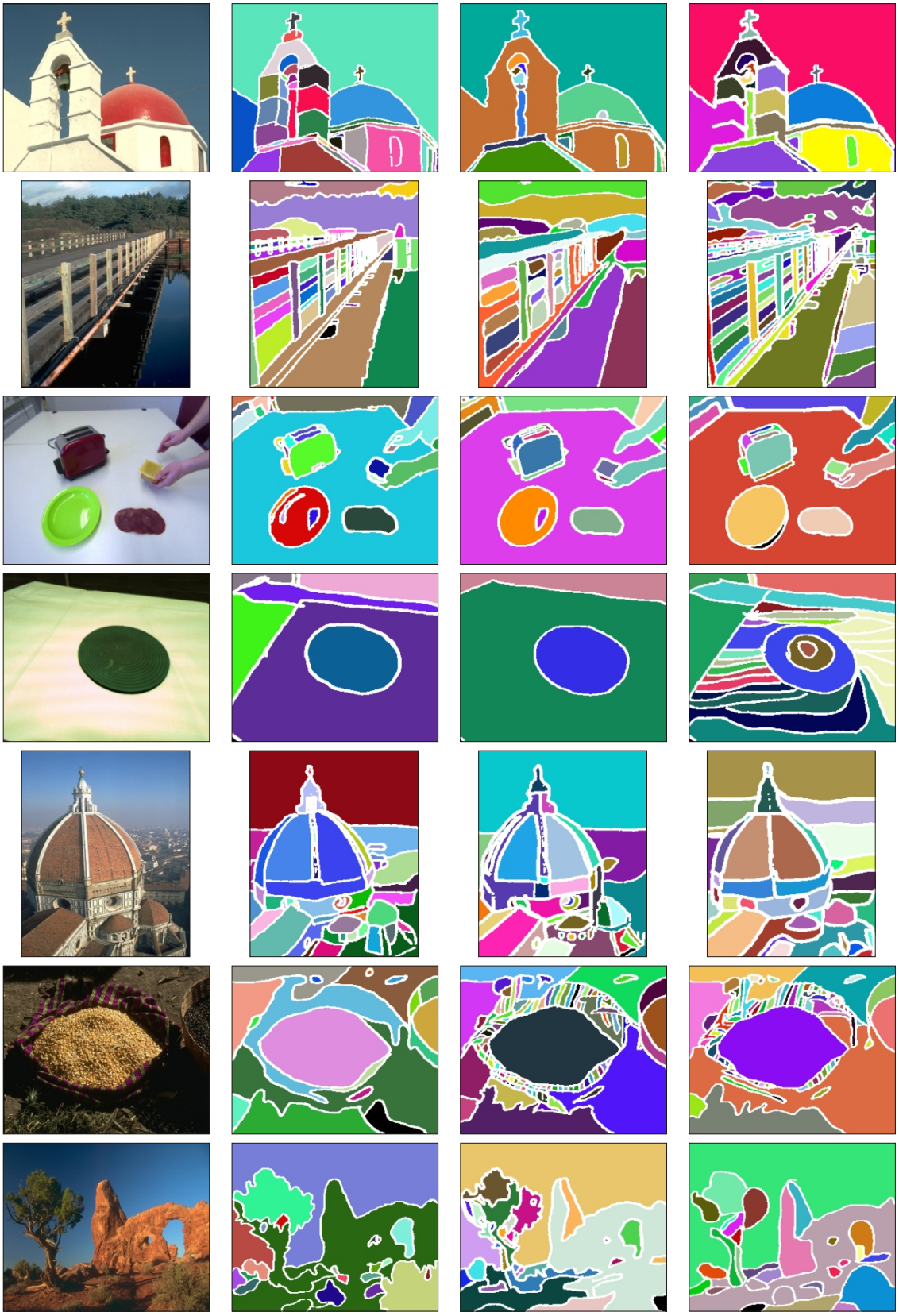

Figure 2.13: Some samples from the image segmentation database (part I).

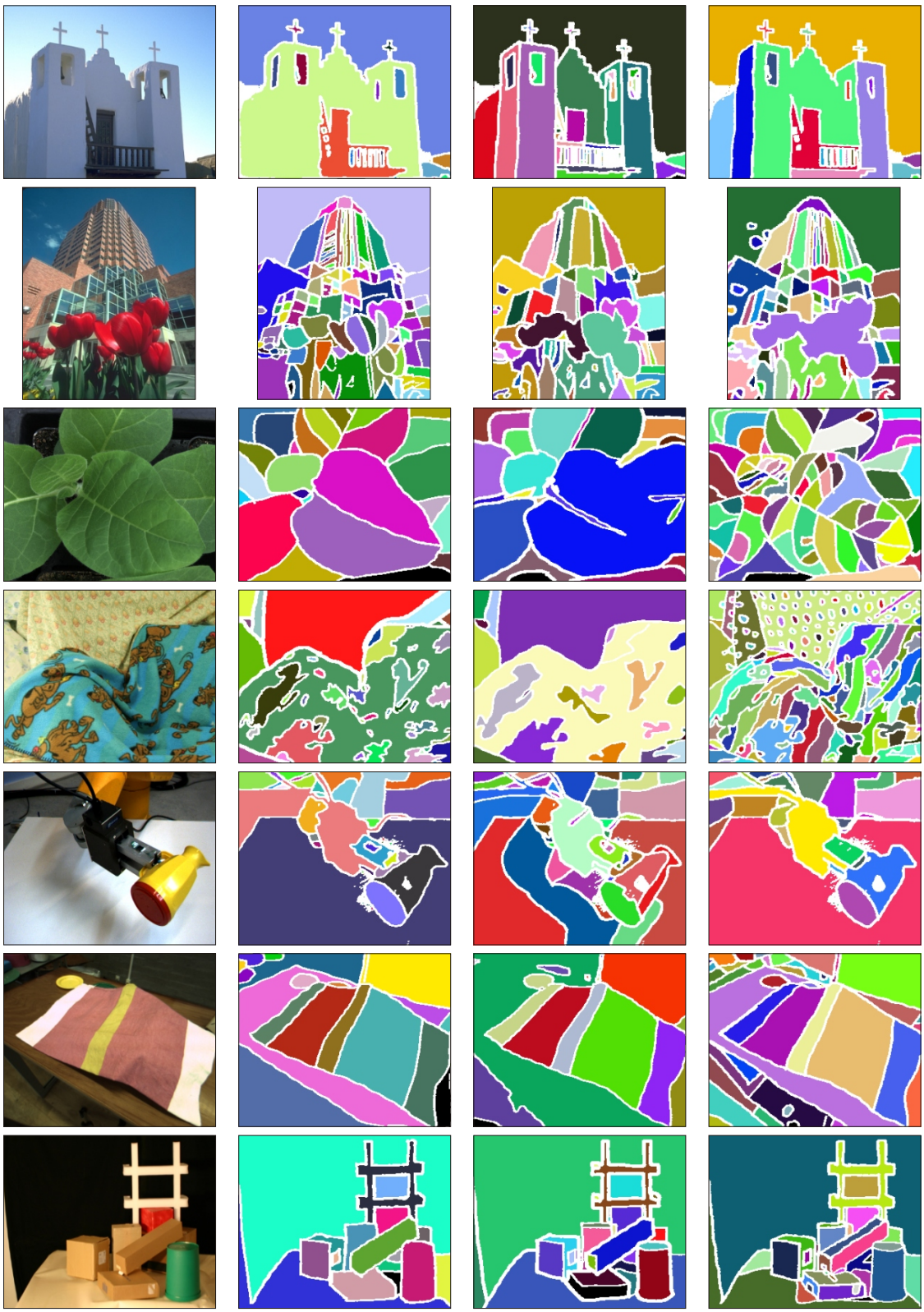

Figure 2.14: Some samples from the image segmentation database (part II).

is sensitive to *over-segmentation*. We say that an image is over-segmented if it is split into too many regions versus the ground truth data. *Recall* measures the percentage of boundary pixels in the ground truth segmentation that correspond to boundary pixels in the machine segmentation and, therefore, is sensitive to *under-segmentation*. We say that an image is under-segmented if it is split into too few regions as opposed to the ground truth. Given a machine segmentation S, and a ground truth segmentation S', precision is defined as the proportion of boundary pixels in S for which a matching boundary pixel in S' can be found and recall is defined as the proportion of boundary pixels in S' for which a matching boundary pixel in S can be found according to

$$Precision = \frac{Matched(S, S')}{|S|}, \quad Recall = \frac{Matched(S', S)}{|S'|}, \tag{2.46}$$

where $|S|$ and $|S'|$ are the total numbers of boundary pixels in the machine and ground truth segmentations, respectively. To compute precision and recall a method for determining correspondences between boundary pixels in the machine and ground truth segmentations is needed. In the current study we use the procedure proposed by Estrada and Jepson (2009) which is an improvement upon the original formulation given by Martin et al. (2002).

2.3.2 Segmentation results

Here we present and evaluate segmentation results obtained by the proposed segmentation technique and compare them to segmentations given by conventional segmentation approaches. As conventional methods we chose the most efficient to date segmentation techniques: graph-based segmentation [11] by Felzenszwalb and Huttenlocher (2004) and mean shift segmentation [12] by Comaniciu et al. (2002); Paris and Durand (2007). In the current study we aim to provide objective and fair performance evaluation of all three segmentation techniques under the following conditions: (i) the used image segmentation database consists of different image types acquired indoors / outdoors under various light conditions having both weakly and sufficiently textured regions; (ii) for each segmentation technique the best values of segmentation parameters are defined (those leading to the best segmentation results) and all images from the database are segmented using these values excluding in this fashion their tuning for each image. Note that this is very important, since we are interested in the use of the image segmentation for automatic segmentation of video streams where any interaction with a user is impossible and bad segments cannot be fixed by the manual parameter tuning. (iii) Both qualitative and quantitative evaluations of segmentation results are given. The qualitative evaluation shows outputs of the

[11] available under http://www.cs.brown.edu/~pff/segment/

[12] available under http://people.csail.mit.edu/sparis/

considered segmentation techniques for the wide range of input images giving a user a chance to select the most appropriate method for the given application. Quantitative evaluation gives *Covering*, *Precision*, and *Recall* values for all images in the database (see Section 2.3.1).

For the graph-based segmentation by Felzenszwalb and Huttenlocher (2004) we have three parameters: σ used to smooth an input image before segmenting it, value for the threshold function k, and minimum component size min enforced by post-processing. In our experiments we use the combination of parameters recommended by authors for segmentation of arbitrary images: $\sigma = 0.5$, $k = 500$, $min = 20$.

The mean shift segmentation by Paris and Durand (2007) takes three input parameters: the Gaussian parameter σ_{range} used on the color axes, the Gaussian σ_{space} used on the x and y axes, and persistence threshold τ_p. The authors recommend the following value ranges: $\sigma_{range} \in [5, 10]$, $\sigma_{space} \in [4, 256]$, $\tau_p \in [0, 5]$. We determined experimentally that the following combination of three parameters gives the best results on the used segmentation database: $\sigma_{range} = 2$, $\sigma_{space} = 8$, $\tau_p = 1$, thus, we use these values in our experiments.

For the proposed segmentation technique we have four parameters: number n_1 of the first Metropolis iterations, number n_2 of the relaxation Metropolis iterations, the factor α_1 used during the first n_1 spin updates, and the factor α_2 used during the second n_2 updates. We tested the algorithm for the following value ranges: $n_1 \in [5, 20]$, $n_2 \in [5, 105]$, $\alpha_1 \in [0.6, 2.2]$, and $\alpha_2 \in [1.0, 9.0]$. Within these value ranges meaningful segmentation results can be obtained and our goal is to find a combination of four values that gives the best segmentation results. For this purpose we must test our algorithm for different combinations of its four input parameters. Thereby the complete database has been segmented by the algorithm for each combination of parameters and average values of *Covering* and *Precision*/*Recall* scores have been computed with respect to the available human segmentations. By the *Precision*/*Recall* score of the algorithm for a particular combination of input parameters we understand the median of the precision and recall scores obtained for the individual images. The median is chosen because the distribution of precision/recall values is typically non-Gaussian. But using the average instead of the median yields similar results (Estrada and Jepson, 2009). Due to the computationally intensive nature of this procedure we have carried out the evaluation on images of size 320×256 and 256×320 pixels for horizontal and vertical images, respectively. The human segmentations contained in the segmentation database have the same size as the original images.

The median precision and recall values obtained for different combinations of input parameters in all considered color spaces result in tuning curves that fully describe the performance of the algorithm. Tuning curves of the parallel Metropolis algorithm with the short-cut for various combinations of input parameters α_1, α_2, n_1, and n_2 are shown in Fig. 2.15 for three color spaces: CIE $(L^*a^*b^*)$(A,B), HSV(C,D), and RGB(E,F). Changing only one input parameter results in a single tuning curve, whereas changing of more parameters result in multiple curves each of which corre-

sponds to *Precision/Recall* scores obtained by changing one input parameter while holding all other parameters fixed. The left column shows scores obtained by changing factor values α_1 and α_2 holding iteration numbers n_1 and n_2 fixed. We set $n_1 = 20$ and $n_2 = 100$, since it is priori clear that 20 and 100 Metropolis iterations are more than enough for the short-cut and system relaxation, respectively. In each color space we chose the most favorable curve or curves. For the CIE $(L^*a^*b^*)$ space we selected the curves with $\alpha_2 = 2.0$ and $\alpha_2 = 4.0$ (see Fig. 2.15(A)), for the HSV space the curves with $\alpha_2 = 2.0$ and $\alpha_2 = 3.0$ (see Fig. 2.15(C)), and for the RGB space the curves with $\alpha_2 = 2.0$, $\alpha_2 = 3.0$, and $\alpha_2 = 5.0$ (see Fig. 2.15(E)). Scores for the graph-based and mean shift segmentations are shown as well. Note that between two curves one upon the other we always choose the upper one like in the CIE $(L^*a^*b^*)$ space between curves with $\alpha_2 = 4.0$ and $\alpha_2 = 6.0$ we choose the former. However, it is not the case if two curves have both diverse recall and precision values. For example we cannot say that between two curves with $\alpha_2 = 2.0$ and $\alpha_2 = 4.0$ the former is preferable, since we cannot say that a slight over-segmentation is better than a slight under-segmentation. However, it is obvious that between the curves with $\alpha_2 = 1.0$ and $\alpha_2 = 2.0$ the latter is preferable, as the former represents considerably over-segmented images having precision lower than 0.65.

The second column, on the contrary, shows scores obtained by changing iteration numbers of the first and second Metropolis updates n_1 and n_2, respectively, taking values for α_1 and α_2 from the best curves of the left column and holding them fixed. For the CIE $(L^*a^*b^*)$ space we set $\alpha_1 = 1.0$ and $\alpha_2 = 4.0$ (see Fig. 2.15(B)), for the HSV space $\alpha_1 = 1.2$ and $\alpha_2 = 3.0$ (see Fig. 2.15(D)), and for the RGB space $\alpha_1 = 1.4$ and $\alpha_2 = 3.0$ (see Fig. 2.15(F)). We can see that the choice of the factor α affects segmentation results more than the choice of iteration numbers resulting in minor differences of precision and recall values. But nevertheless, segmentation results obtained after $n_2 = 5$ relaxation iterations preserve small segments and are more over-segmented than results after $n_2 = 105$ iterations. This is observed for all considered color spaces.

The best tuning curves for the proposed image segmentation algorithm in all considered color spaces are shown in Fig. 2.16. The left column shows scores obtained by changing factor values α_1 and α_2 holding iteration numbers n_1 and n_2 fixed, where again $n_1 = 20$ and $n_2 = 100$. The following curves are displayed: $\alpha_2 = 1.0$, $\alpha_2 = 2.0$, and $\alpha_2 = 4.0$ for the CIE $(L^*a^*b^*)$ space, $\alpha_2 = 1.0$, $\alpha_2 = 2.0$, and $\alpha_2 = 3.0$ for the HSV space, $\alpha_2 = 1.0$, $\alpha_2 = 2.0$, and $\alpha_2 = 3.0$ for the RGB space. The right column shows the best scores obtained after $n_2 = 55$ iterations by changing the number of first Metropolis updates n_1 and holding the values of α_1 and α_2, taken from the best curves of the left column, fixed. We can see that the proposed segmentation has the highest *Precision/Recall* scores in the CIE $(L^*a^*b^*)$ space with $\alpha_2 = 2.0$ and $\alpha_2 = 4.0$. In terms of iteration numbers, $n_1 = 10$ and $n_2 = 20$ are enough to obtain consistent segmentation results. Less iterations will lead to over-segmentation, while more relaxation iterations n_2 lead to under-segmentation eliminating small regions.

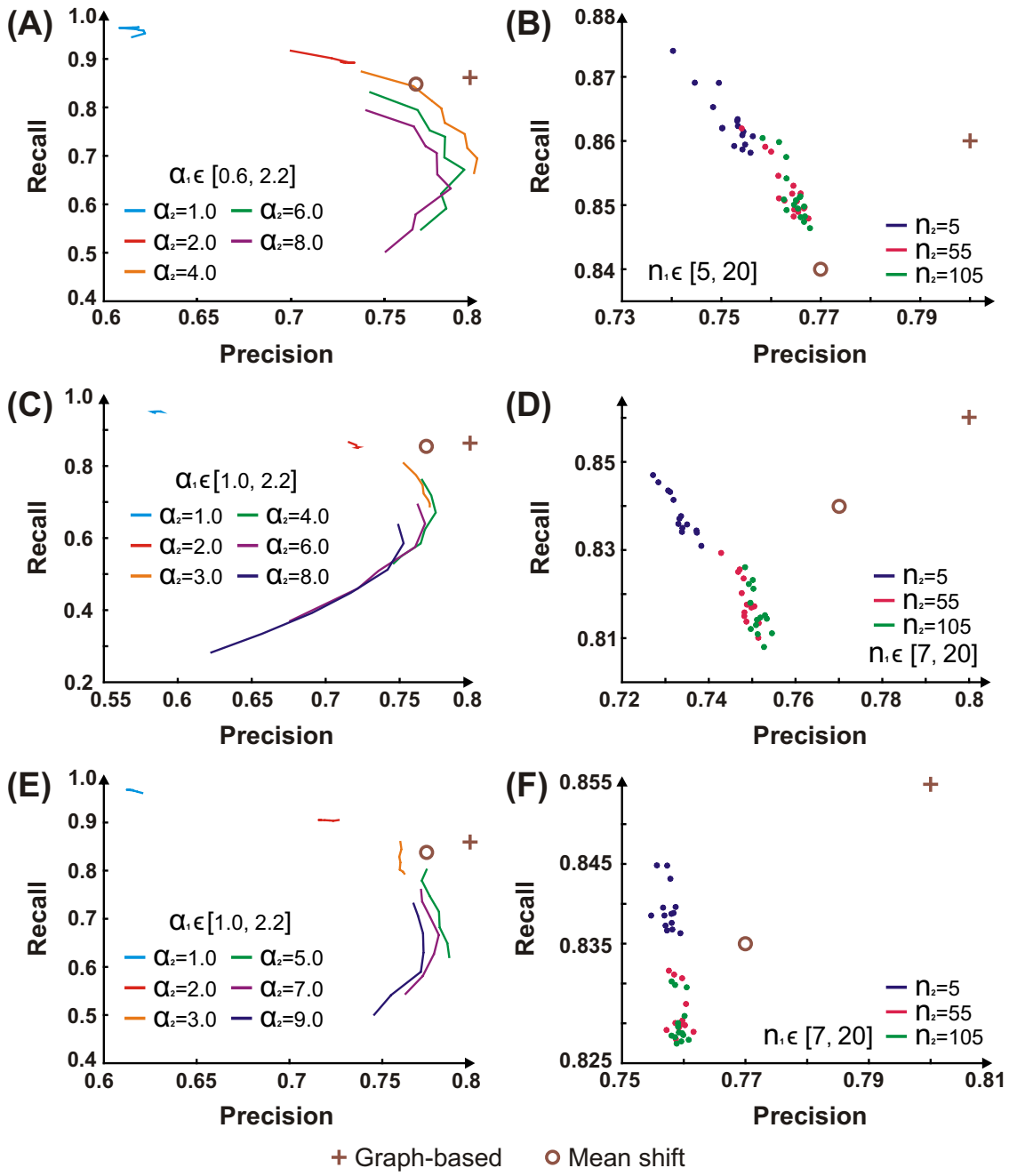

Figure 2.15: Tuning curves for the proposed image segmentation algorithm shown for CIE $(L^*a^*b^*)$(A,B), HSV(C,D), and RGB(E,F) color spaces. The left column shows *Precision* / *Recall* scores obtained by changing factor values α_1 and α_2 while holding n_1 and n_2 fixed. The right column, on the contrary, shows *Precision* / *Recall* scores obtained by changing iteration numbers n_1 and n_2 holding α_1 and α_2 fixed. Scores for the graph-based and mean shift segmentations are shown as well.

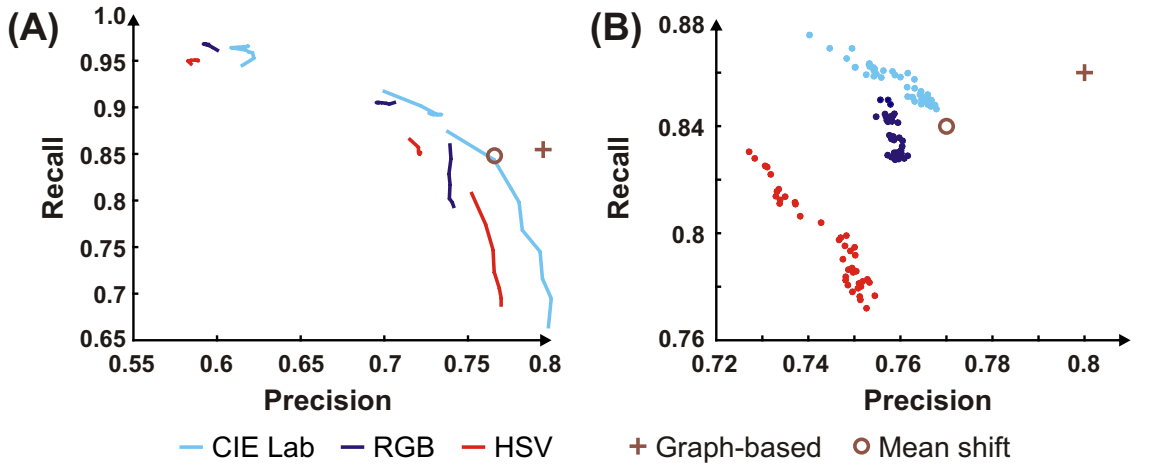

Figure 2.16: The best tuning curves for the proposed image segmentation algorithm shown for CIE $(L^*a^*b^*)$, HSV, and RGB color spaces. The left column shows *Precision* / *Recall* scores obtained by changing factor values α_1 and α_2 while holding n_1 and n_2 fixed. The right column, on the contrary, shows *Precision* / *Recall* scores obtained by changing the iteration number n_1 holding α_1, α_2, and n_2 fixed. Scores for the graph-based and mean shift segmentations are shown as well.

With these parameters the method has the performance similar to the mean shift while being slightly outperformed by the graph-based algorithm.

The highest values of the *Segmentation covering* measure obtained on the images from the database with corresponding *Precision*/*Recall* scores and input parameters are presented in Table 2.3 for all considered color spaces. The comparison of covering values for the graph-based, mean shift, and proposed segmentation algorithms are given in Table 2.4.

The proposed segmentation method has the highest covering value of 0.54 in the CIE $(L^*a^*b^*)$ space. In terms of the segmentation covering measure our technique outperforms the mean shift and yields slightly to the graph-based segmentation. Consequently, taking into account both *Precision/Recall* and *Segmentation covering* measures, in our experiments we work in the CIE $(L^*a^*b^*)$ space and use the following combination of input parameters: $\alpha_1 = 1.0$, $\alpha_2 = 4.0$, $n_1 = 10$, and $n_2 = 20$.

Image segmentation results obtained on the employed image database for the mean shift segmentation, graph-based segmentation, and segmentation based on parallel Metropolis updates with the short-cut are shown for various image types in four parts in Fig. 2.17, 2.18, 2.19, and 2.20. All images from the database have been segmented automatically with the same fixed set of input parameters. Each segmentation output is attended with the corresponding *Precision* (P), *Recall* (R), and *Covering* (C) values. We can see that some segmentation outputs are not re-

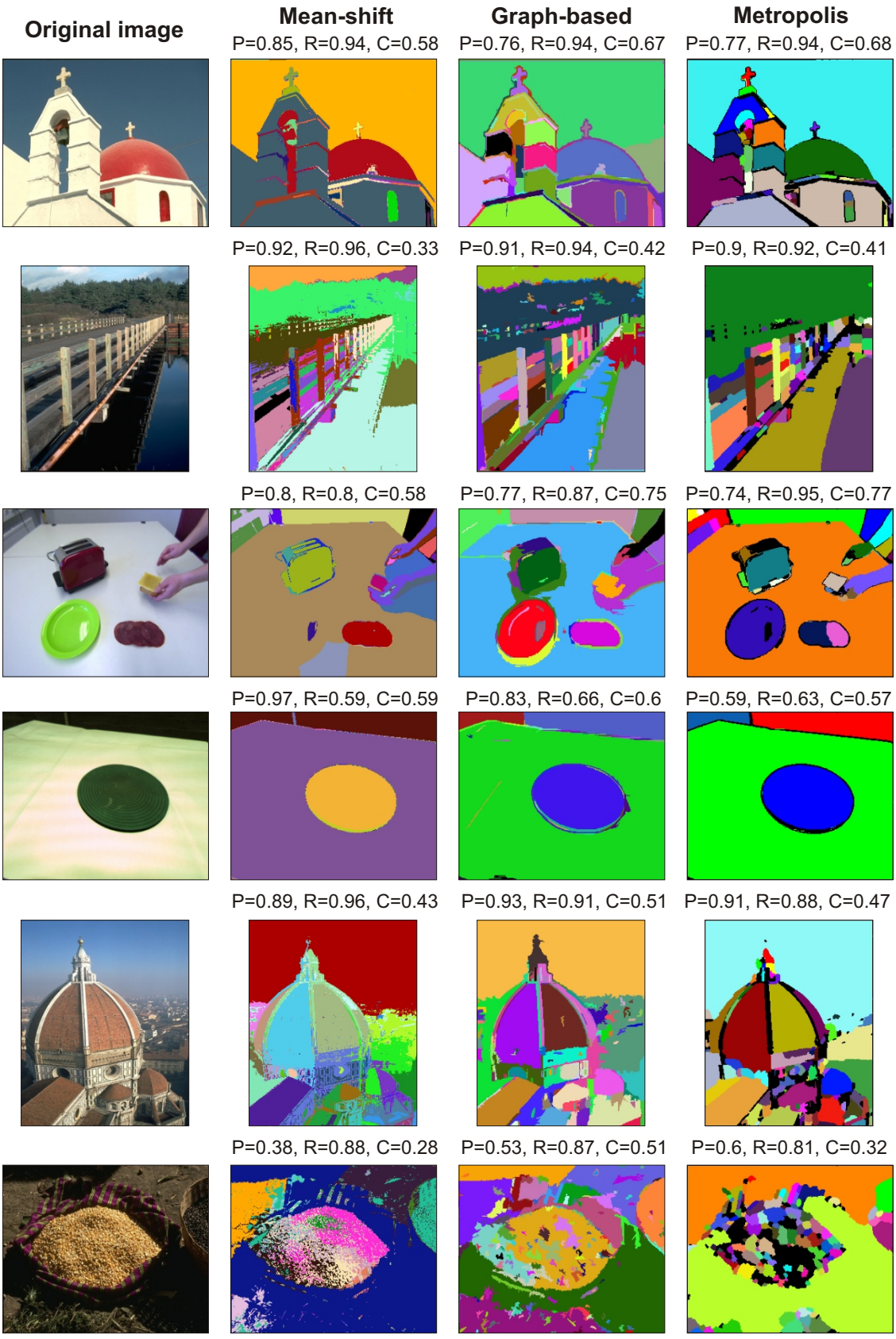

Figure 2.17: Segmentation results on the employed image database (part I).

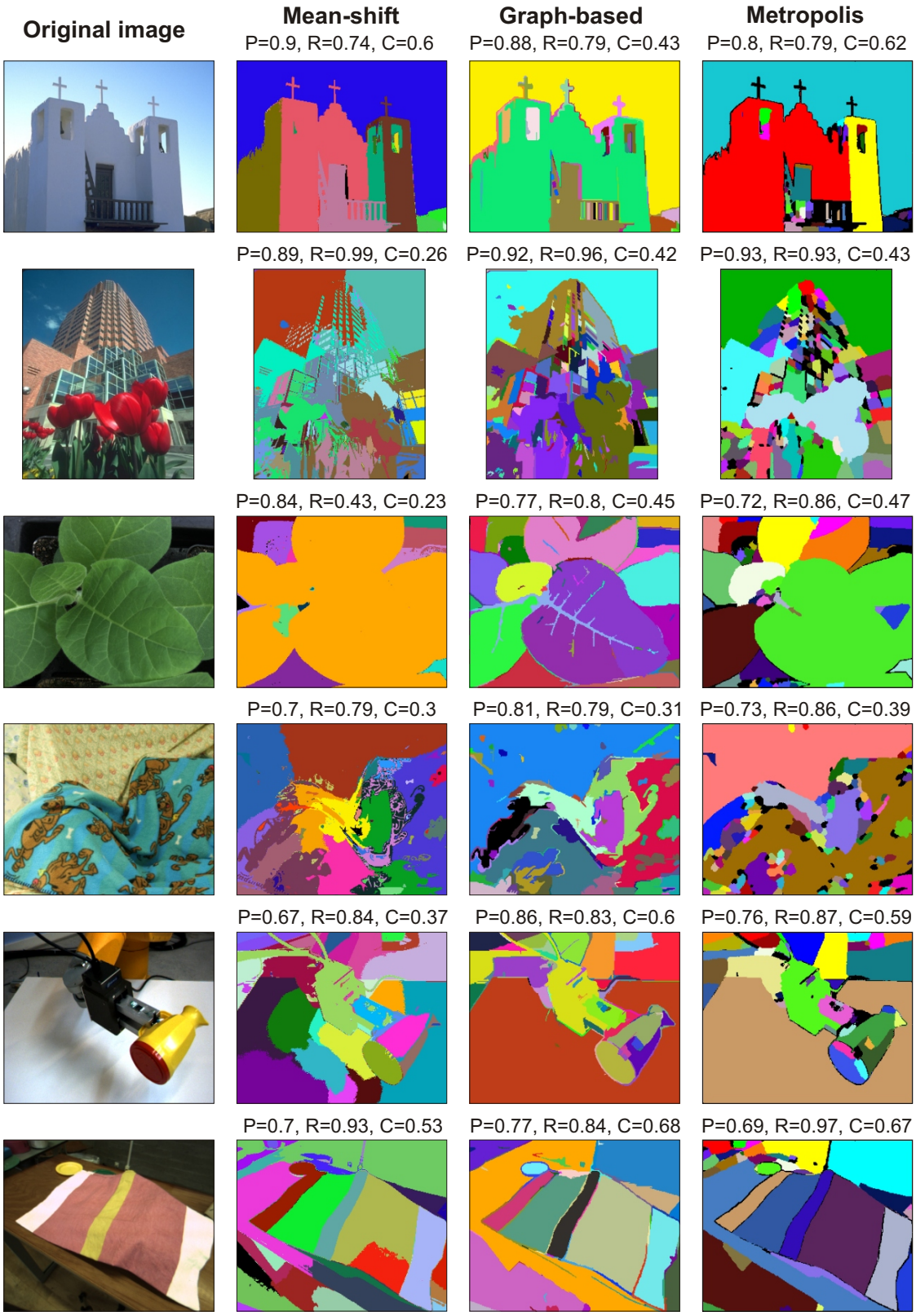

Figure 2.18: Segmentation results on the employed image database (part II).

Figure 2.19: Segmentation results on the employed image database (part III).

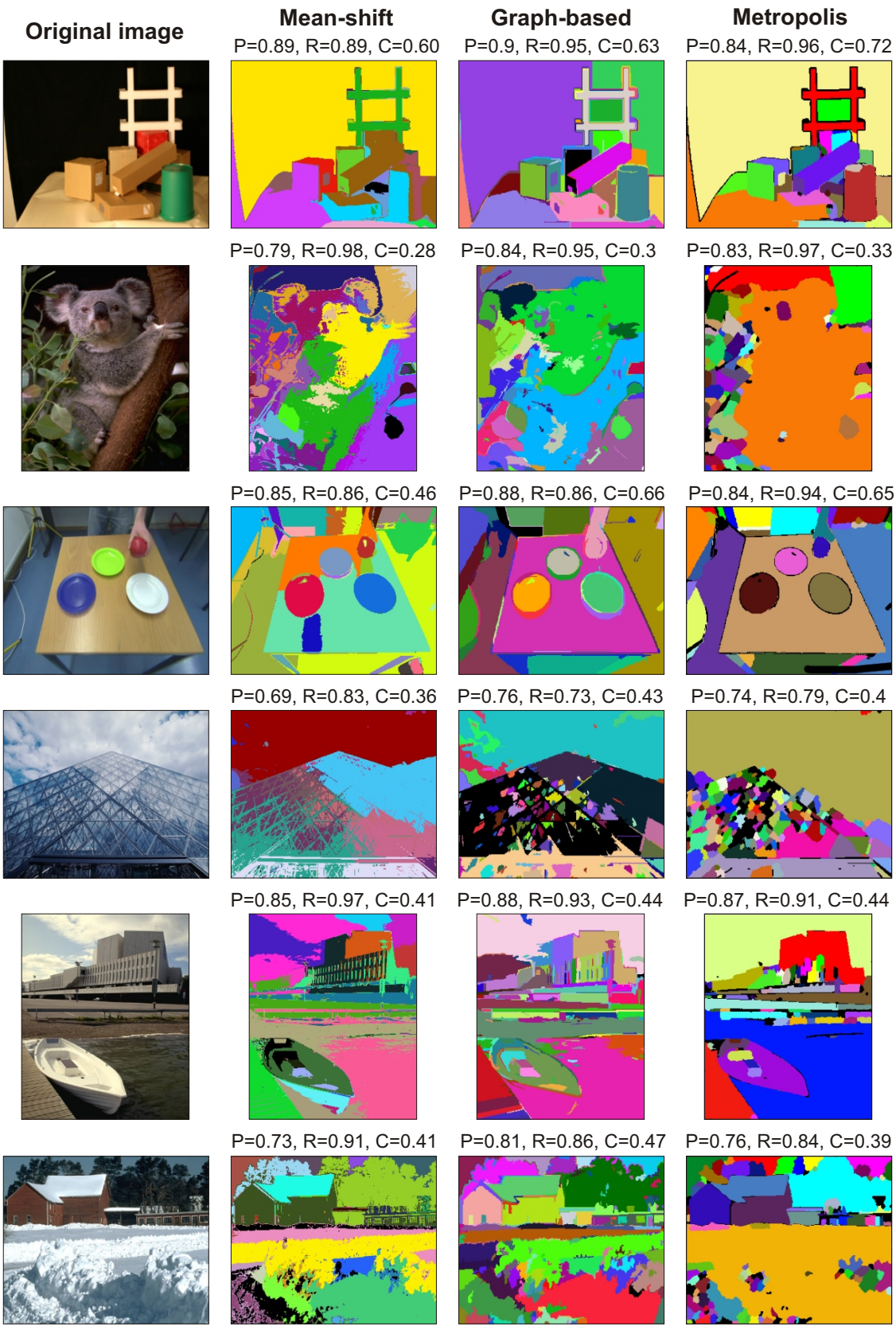

Figure 2.20: Segmentation results on the employed image database (part IV).

Color space	n_1	n_2	α_1	α_2	*Precision*	*Recall*	*Covering*
CIE ($L^*a^*b^*$)	10	20	0.6	2.0	0.7	0.91	0.5
CIE ($L^*a^*b^*$)	10	20	0.6	3.0	0.72	0.9	0.5
CIE ($L^*a^*b^*$)	10	20	0.8	5.0	0.78	0.78	0.54
CIE ($L^*a^*b^*$)	10	20	1.0	1.0	0.62	0.96	0.5
HSV	10	20	1.0	3.0	0.73	0.85	0.50
HSV	10	20	1.0	4.0	0.76	0.85	0.50
HSV	10	20	1.0	5.0	0.76	0.84	0.50
HSV	10	20	1.0	6.0	0.74	0.83	0.50
RGB	10	20	1.0	3.0	0.76	0.86	0.53
RGB	10	20	1.0	4.0	0.77	0.82	0.53
RGB	10	20	1.0	5.0	0.77	0.8	0.53
RGB	10	20	1.2	5.0	0.77	0.78	0.52

Table 2.3: Input parameters of the proposed segmentation technique resulting in the highest *Segmentation covering* values in each color space.

Technique	*Precision*	*Recall*	*Covering*
Graph-based segmentation	0.8	0.86	0.58
Mean shift segmentation	0.77	0.84	0.48
Metropolis with the short-cut	0.78	0.78	0.54

Table 2.4: *Segmentation covering* values with the corresponding *Precision*/*Recall* scores for the graph-based, mean shift, and proposed segmentation techniques.

ally reasonable containing either over- or under-segmented parts. Note that for each testing image very high *Precision*/*Recall* scores and covering values can be derived via adjustment of input parameters. However, it was not the goal of our study. Our aim is to compare and evaluate the considered segmentation techniques in terms of automatic, unsupervised segmentation excluding any interaction with a user. We can see that our segmentation approach outperforms in many cases the mean shift technique which has sometimes dramatic region splits and merges (see the third row in Fig. 2.17, the second and fifth rows in Fig. 2.18, the first row in Fig. 2.19, and the third row in Fig. 2.20). But the graph-based approach outperforms our method on images featured by a high texture level (see the last row in Fig. 2.17, the third row in Fig. 2.19, and the fourth row in Fig. 2.20). In some cases our approach suffers from under-segmentation, whereas mean shift and graph-based produce more consistent

segments (see the second row in Fig. 2.17, the fifth row in Fig. 2.19, and the second and sixth rows in Fig. 2.20). But on other images the graph-based technique suffers from over-segmentation (see the third row in Fig. 2.17, and the third row in Fig. 2.20). Furthermore, sometimes it produces unexplainable region splits (see the first row in Fig. 2.17). But the biggest disadvantage of the graph-based technique is that almost all region contours or small shadows are identified as segments (it is particularly visible in the third row in Fig. 2.17, the last row in Fig. 2.19, and the third row in Fig. 2.20). Bad news are that such artifacts do not influence quantitative measures of the segmentation performance and can be detected only by the qualitative evaluation of the segmentation output.

2.3.3 Processing time

The processing times obtained for all steps of the proposed image segmentation algorithm (see Section 2.2.6) are shown in Table 2.5 for the following resolutions: 160×128, 320×256, and 640×512 pixels. The total processing times with frame rates are shown as well. The computation times and frame rates have been measured by processing all images from the database and averaging the results using the following experimental environment: CPU 3.40 GHz Intel(R) Core(TM) i7-2600K (using a single core) with 15.6 GB RAM and GPU GeForce GTX 580 (with 1.5 GB device memory) consisting of 16 SMs each having 32 cores, so 512 processor cores in total. For image sizes of 160×128 and 320×256 pixels the real-time performance has been obtained, whereas for image size of 640×512 pixels the frame rate is 10.9 which is still pretty fast.

	Processing time (ms)		
Algorithmic step	160×128 (px)	320×256 (px)	640×512 (px)
Metropolis I ($n_1 = 10$)	2.8	7.5	30.0
Border detection	0.2	0.83	5.2
Labeling	0.15	0.6	2.3
Metropolis II ($n_2 = 20$)	5.7	17.2	54.0
Total	8.8	26.1	91.5
Frames per second	113.6	38.3	10.9

Table 2.5: Processing times of all stages of the proposed image segmentation algorithm for multiple image sizes.

Among all algorithmic steps only the runtime of the labeling depends on the structure of the input image, i.e., shapes and amount of connected components, but deviations are in the range of two milliseconds for images up to 320×256 pixels

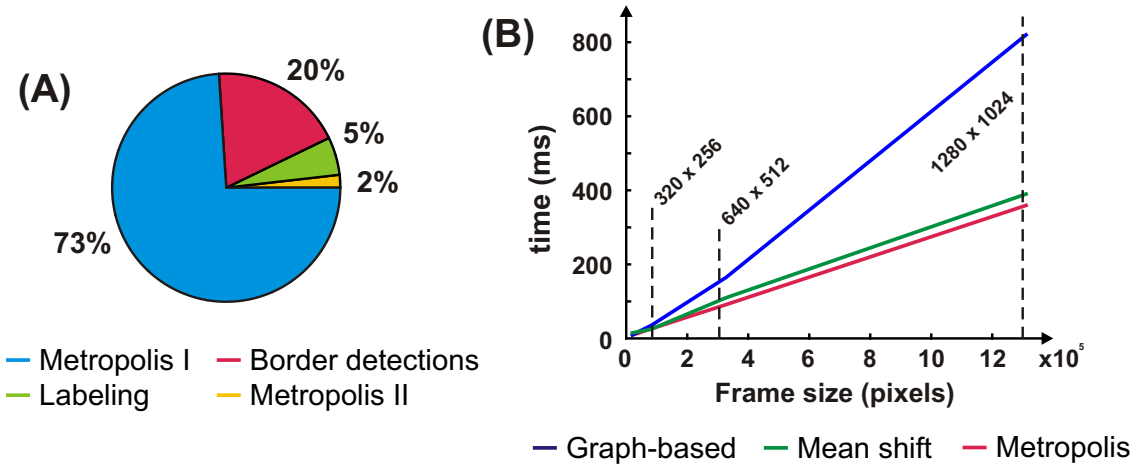

Figure 2.21: Time performance. (A) Computation time of all steps in the parallel Metropolis algorithm with the short-cut in percentage of the total runtime. (B) Runtime comparison for the proposed approach and the conventional segmentation techniques.

and of ten milliseconds for images up to 1280×1024 pixels. Labeling of connected components takes more time for very textured images containing many small arbitrary shaped regions, since more provisional labels are required and more time is needed to resolve label equivalences before representative labels can be assigned on the second image scan. The most time consuming step in the method is the Metropolis update procedure taking for both iteration groups more than 93% of the total runtime (see Fig. 2.21(A)).

	total (ms) / frames per second			
Technique	160×128 (px)	320×256 (px)	640×512 (px)	1280×1024 (px)
Graph-based	10.0 / 100.0	35.0 / 28.5	165.0 / 6.1	820.0 / 1.2
Mean shift	15.0 / 66.7	25.0 / 40.0	110.0 / 9.1	390.0 / 2.6
Metropolis	8.8 / 113.6	26.1 / 38.3	91.5 / 10.9	359.3 / 2.8

Table 2.6: Comparison of computation times obtained for the graph-based, mean shift, and proposed segmentation techniques.

In Table 2.6 time performance of the proposed algorithm is compared to the graph-based and mean shift segmentation techniques. Note that in this work we use CPU implementations of the graph-based and mean shift algorithms provided by their au-

thors and did not consider their potential accelerations on the GPU. Therefore, only our method makes use of the parallel hardware here. The computation times and frame rates have been measured by processing all images from the database and averaging the results within the same experimental environment. The processing time as a function of the total number of pixels in the image is shown for all three approaches in Fig. 2.21(B). All three algorithms depend linearly on the number of pixels and we can see that the proposed method is faster (especially for high resolutions like 640×512 and 1280×1024 pixels) than the graph-based segmentation and slightly faster than the mean shift technique.

2.4 Discussion

In this chapter we introduced the novel automatic image segmentation algorithm based on the method of superparamagnetic clustering of data. The superparamagnetic clustering has been chosen as a technique for the image segmentation due to its following advantages:

1. It is fully automatic allowing us to segment arbitrary images without any prior knowledge about the visual scene and the number of objects.

2. The method can be applied to the segmentation of video streams in a very efficient way, since the temporal coherence within a video stream can be achieved just by warping the previous spin configuration to the current frame avoiding a very expensive region matching procedure.

Segmentation techniques based on the superparamagnetic clustering have not be applied before to the real-time segmentation of video streams because of their very low processing speed (Opara and Wörgötter, 1998; von Ferber and Wörgötter, 2000; Dellen et al., 2009). All made attempts toward the acceleration of the method resolved into parallel implementations of local-update techniques on parallel hardware (Barkema and MacFarland, 1994) or switching from local-update algorithms to cluster updates on traditional CPU platforms. Cluster updates allow the method to update much more pixels per one iteration based on pre-defined clusters. Despite a significant acceleration of the processing time as opposed to implementations on common computers, accelerated local-update methods still remained extremely slow requiring minutes for segmentation of even quite small images with size of 160×128 pixels. Such computational speed excludes the use of these implementations in terms of real-time video processing. Although the cluster update approach has been successfully applied to segmentation of image sequences (Dellen et al., 2009), its processing speed is still far from the real-time performance requiring some seconds to segment one frame.

Since a real-time implementation of image segmentation based on the superparamagnetic clustering of data is of high importance for the real-time video processing, various update techniques for approximation of the equilibrium states of the Potts model (used for the image representation) and their potential accelerations on the special hardware have been considered in this study. Despite the cluster update algorithms are much faster than local-update techniques, they do not reach the real-time performance on traditional CPU platforms and their latency can be overcome only by the use of very powerful computer systems that are very expensive and massive. However, such platforms cannot be employed in many robotic systems which have strict requirements to the system size and power consumption. And, as a consequence, easier solutions are desired. In this chapter we investigated the local-update technique proposed by Metropolis et al. (1953) with simulated annealing for approximation of the equilibrium states of the Potts model. The highly parallel, multi-threaded, and multi-core GPU architecture has been considered as a parallel hardware for acceleration because of the following reasons:

1. Powerful GPUs are currently a part of almost all computers, have a moderate price, and can be used for general-purpose parallel computing without any additional hardware upgrades. Also graphics capabilities of GPUs make the visual output of the processed data directly from the card much simpler compared to other parallel platforms.

2. Modern GPUs are featured by tremendous computational power and very high memory bandwidth which makes them more efficient than traditional CPUs in the case of very intensive parallel computations.

3. The CUDA parallel programming model makes parallelization of software applications on GPUs quite transparent drastically decreasing design time.

4. Metropolis updates, being very intensive and highly local, ideally fit to the GPU architecture.

The simulated annealing procedure running on the GPU has an enormous acceleration as compared to traditional CPUs, especially for relatively big images like 640×512 and 1280×1024 pixels. But the analysis of various simulated annealing schedules has demonstrated that schedules performing the complete segmentation of even very small images with size of 160×128 pixels running in parallel on the GPU require some seconds per image. Therefore, it still remains impossible to segment images in real-time by the Metropolis algorithm with the simulated annealing only, even on the parallel hardware. For that reason we introduced a short-cut procedure into the annealing process. The proposed short-cut is based on superpixels obtained after a very quick system cooling at the low temperature and allows us to detect all regions (segments) accelerating the system convergence without performing the full

annealing. Some parts of the short-cut have been implemented on the GPU as well. For the proposed image segmentation algorithm based on parallel Metropolis updates with the short-cut we obtained processing times which are sufficient for the real-time video processing. The real-time performance is derived for image sizes of 160×128, 320×256, and 640×512 pixels, whereas for image size of 1280×1024 pixels only close to real-time performance can be achieved.

The proposed image segmentation technique has been applied to various real images from the employed image segmentation database and compared to the conventional segmentation techniques such as graph-based (Felzenszwalb and Huttenlocher, 2004) and mean shift (Comaniciu et al., 2002; Paris and Durand, 2007) segmentations both qualitatively and quantitatively. Since in our work we pursued the aim of objective and fair performance evaluation of all three segmentation techniques, each segmentation algorithm has been tested on the same images without manual tuning of input parameters for each image. First, the best set of input parameters has been defined for each algorithm and then results obtained using those values have been evaluated. For the quantitative evaluation of segmentation results we used two different measures: segmentation covering and precision/recall. Both of them compare the machine segmentation with the ground truth segmentation given by humans. In compliance with both measures the proposed segmentation produces the best results in the CIE $(L^*a^*b^*)$ color space achieving the performance of the mean shift technique. Segmentation results produced by the graph-based have slightly higher covering and precision/recall values. Algorithms based on the superparamagnetic clustering of data suffer generally a lot from image areas featured by a high level of texture resulting in a variety of tiny segments. The graph-based and mean shift methods incorporate a pre-processing of texture producing more meaningful results in very textured regions. However, the quality of the proposed segmentation technique can be improved on very textured images by the use of special texture filters that smooth highly-textured areas preserving boundaries between diverse regions.

Concerning the time performance, our algorithm is two times faster as compared to the graph-based technique and a bit faster than the mean shift. Also it is necessary to point out that the runtime of the parallel Metropolis with the short-cut is almost independent of image structure, number of segments, and image density, i.e., the relation between object and background pixels (He et al., 2009). The slowest part of the proposed algorithm is the Metropolis update consisting of two iteration groups (basic and relaxation iterations), whereas the short-cut itself is very fast. Unlike the graph-based and mean shift algorithms, in terms of the video segmentation it is not necessary to segment each frame from scratch resolving afterwards a region matching for adjacent frames. The temporal coherence within a video sequence can be achieved by using a spin configuration obtained for the previous frame as an initial state for the current frame. Then the current frame just needs to undergo a short Metropolis relaxation procedure after which final segments can be extracted. The whole procedure of the video segmentation based on the superparamagnetic clustering

of data is considered in detail in the next chapter.

3

Real-time Segmentation of Monocular Video Streams

"The only reason for time is so that everything doesn't happen at once"
– Albert Einstein

3.1 Introduction

Real-time cognitive vision systems have to process and structure abundant dynamic visual information for enabling the robots to interact with the environment in a meaningful way. For example, the understanding of the visual scene in terms of object and object-action relations (Aksoy et al., 2011) requires objects to be detected, segmented, tracked (Salembier and Marqués, 1999), and important descriptors, e.g., shape information, to be extracted. This process corresponds to a dramatic compression of the initial visual data into symbol-like descriptors, upon which abstract logic or learning schemes can be applied, e.g., for the execution of a grasping action (Klingbeil et al., 2011; Kjellström et al., 2011). Finding this reduced symbol-like representation without prior knowledge on the data (model free), thus, represents a major challenge in cognitive-vision applications – this problem is also known as the signal-symbol gap (König and Krüger, 2006). Furthermore, in most of robotic systems "live" interactions of robots with the environment make this task even more challenging. In such systems all pre-computations of the visual data need to be performed in real-time which limits the applicability of many vision algorithms.

The video segmentation problem is generally formulated as the grouping of pixels into *spatio-temporal* volumes where each found object or object part is uniquely identified and satisfies *temporal coherence*, i.e., carries the same label along the whole video stream (Grundmann et al., 2010; Reina et al., 2010). Several approaches for the video segmentation problem have been proposed over the last two decades. They can be summarized shortly as follows.

On-line and off-line methods. *On-line* video segmentation techniques use only preceding information and do not need future data. Such methods can segment video sequences of arbitrarily length in a continuous, sequential manner (Liu et al., 2008b,a; Paris, 2008; Hedau et al., 2008; Wang et al., 2009; Breitenstein et al., 2009; Dellen et al., 2009; Reina et al., 2010). However, those methods usually either perform segmentation of all frames independently of each other applying a block matching procedure at a time for temporal coherence (see Fig. 3.1(A)), or track segment labels through the video stream considering preceding frames (see Fig. 3.1(B-D)). As was mentioned in the previous chapter, block matching is a very slow operation almost excluding the real-time performance. Methods that consider only two frames at a time (see Fig. 3.1(B)) are sensitive to segmentation errors that gradually accumulate over time. Taking into account the whole history (see Fig. 3.1(D)) leads to more robust spatio-temporal volumes but it is very time and memory consuming. The longer the frame sequence is, the more time and memory resources are required. For that reason such approaches are efficient only for short sequences and cannot be applied to arbitrary long videos. Considering only several preceding frames at a time (see Fig. 3.1(C)) could be a trade-off solution, but it is still time consuming and runs on the order of seconds per frame (Reina et al., 2010). *Off-line* methods, on the contrary, require future data or even the entire video sequence as input (see Fig. 3.1(E,F)) (Unger et al., 2009; Brendel and Todorovic, 2009; Huang et al., 2009; Grundmann et al., 2010). Off-line techniques are more robust in terms of temporal coherence but they cannot be involved in perception-action loops, since future perception is unknown.

Dense and sparse techniques. A video segmentation method is *dense* if it treats all objects visible in the scene trying to assign each pixel in every frame to a proper spatio-temporal volume (Liu et al., 2008b,a; Paris, 2008; Hedau et al., 2008; Brendel and Todorovic, 2009; Huang et al., 2009; Dellen et al., 2009; Reina et al., 2010; Grundmann et al., 2010). Techniques that perform segmentation of pre-selected objects only are *sparse* (Wang et al., 2009; Unger et al., 2009; Breitenstein et al., 2009). Focusing only on the tracking of pre-selected objects excludes an estimation of object position relative to the environment which, as a consequence, excludes robot movements aimed at objects.

Automatic and nonautomatic approaches. The method is *automatic* or *unsupervised* if it runs without interaction with a user and does not need any prior knowledge about objects in the scene (Liu et al., 2008b; Paris, 2008; Hedau et al., 2008; Brendel and Todorovic, 2009; Dellen et al., 2009; Reina et al., 2010). *Nonautomatic* or *supervised* techniques are very often driven by user input, use some prior knowledge about the visual scene and make assumptions about the number of objects present (Liu et al., 2008a; Unger et al., 2009; Huang et al., 2009; Wang et al., 2009; Breitenstein et al., 2009). Some segmentation techniques, e.g., the hierarchical graph-based video segmentation proposed by Grundmann et al. (2010), can run in both automatic and nonautomatic modes.

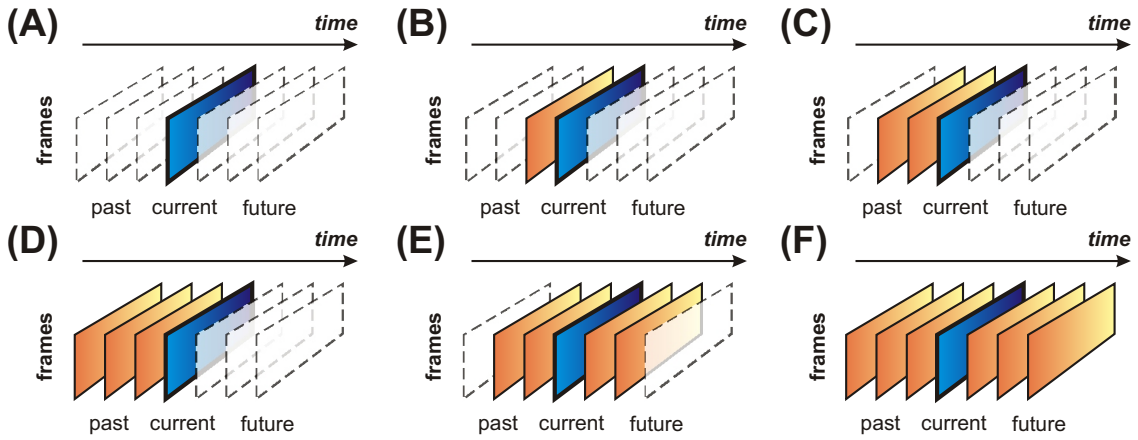

Figure 3.1: Various strategies for the video segmentation problem in terms of the used input data. (A-D) On-line methods using only the current and preceding frames to find a solution. (E,F) Off-line methods requiring some future data or the whole sequence.

Since robots are usually autonomous systems that interact with the environment, only on-line automatic video segmentation techniques can be employed in the perception-action loop. Furthermore, complete information about the visual scene and relations between present objects can be derived only by the use of dense methods. The following techniques are the most famous and up-to-date on-line dense automatic video segmentation approaches:

The mean-shift video segmentation, proposed by Paris (2008), is based on the popular image segmentation technique introduced by Comaniciu et al. (2002) and discussed in Chapter 2. The temporal coherence is achieved by estimating the density of feature points, associated with all pixels, with a Gaussian kernel using data from all preceding frames (see Fig. 3.1(D)). The method has a real-time performance on gray-level videos of size 640×360 pixels.

Multiple hypothesis video segmentation (MHVS) from superpixel flows by Reina et al. (2010) generates multiple pre-segmentations per frame considering only a few preceding frames (see Fig. 3.1(C)). For each pre-segmentation it finds sequences of time consistent superpixels, called *superpixel flows* or *hypotheses*. Each hypothesis is considered as a potential solution and a hypothesis leading to the best spatio-temporal coherence. In this approach the segmentation decision is postponed until evidence has been collected across several frames. Despite quite accurate segmentation results the MHVS needs seconds to process one frame which makes it inapplicable in real-time robotic applications.

Video segmentation based on propagation, validation and aggregation of a preceding graph by Liu et al. (2008b) exploits inter-frame correlations to propagate reliable

groupings from the previous frame to the current (see Fig. 3.1(B)). A preceding graph is built and labeled for the previous frame and temporally propagated to the current frame using a global motion estimation, followed by validation based on similarity measures. Pixels remaining unlabeled after the propagation are grouped into subgraphs by a simple color clustering. Although the method gives results of a very high quality, it runs at frame rates inapplicable to real-time utilization.

Matching images under unstable segmentations by Hedau et al. (2008) is based on the fact that object regions obtained by existing segmentation methods do not always produce perceptually meaningful regions. In this approach the current frame is segmented independently of preceding frames and the temporal coherence is achieved by region matching between the current and previous frames (see Fig. 3.1(B)) using *the Partial Match Cost* which allows fragments belonging to the same region to have low match cost with the original region. However, the method cannot run in real-time due to very slow region matching procedure.

The three last approaches provide very accurate spatio-temporal volumes and can segment arbitrary long video sequences, but these methods do not run in real-time and, as a consequence, cannot be employed in the real-time cognitive vision system. The mean shift video segmentation approach, on the contrary, runs in real-time but works only on gray-scale videos and needs all past data to achieve satisfactory temporal coherence. But it is not always possible to keep all past data in the memory, especially in mobile robotic systems having a very limited memory space.

Dellen et al. (2009) proposed a video segmentation technique based on the superparamagnetic clustering of data using the energy-based cluster update (ECU) for ordering spins in the Potts model according to the image data (see Section 2.2.1). Being on-line, dense, and automatic, the method considers only the current frame coupled with the very last previous frame at a time to reach the spatio-temporal synchronization (see Fig. 3.1(B)). Segmentation of a frame sequence is performed as follows. A sequence is split in pairs of two frames at a time, where the last frame of the previous pair is identical with the first frame in the current pair. In such a manner the spin states of each pair are initialized with the spin states of the previous pair. Spin updates are applied to each pair considering both 2D bonds, i.e., the closest neighbors of each pixel within the first frame, and 3D bonds, i.e., the neighbors of each pixel in the last frame. 3D bonds are determined through the recovery of visual motion or *optical flow* from a sequence of images. Despite the efficient linking between segments of adjacent frames, cluster updates running on CPU still need some seconds to process one frame which makes the real-time performance impracticable.

In this chapter we present a novel visual front-end for real-time spatio-temporal segmentation of monocular videos which overcomes limitations of the considered approaches. The proposed visual front-end is on-line, automatic, dense, and solves the following problems:

1. Video frames are segmented using the parallel Metropolis algorithm introduced

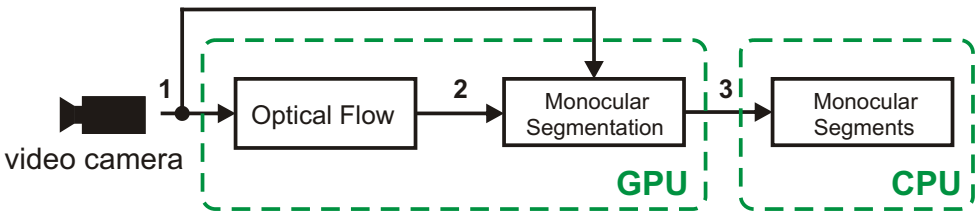

Figure 3.2: The architecture of the framework for segmentation of monocular videos on the heterogeneous computing system consisting of one CPU and one GPU.

in Chapter 2, avoiding time consuming region matching or ECU. Segmentation is carried out in a consistent model-free way.

2. The temporal coherence in a video stream is achieved using a label-transfer strategy based on estimated pixel motion, resulting in a consistent partitioning of neighboring frames together with a consistent labeling. Only the results obtained on the very last frame are employed at a time in order to guarantee spatio-temporal coherence for the current frame (see Fig. 3.1(B)).

3. All computations run in real-time allowing the framework to be used in the perception-action loop.

The chapter is organized as follows. First we introduce the framework for segmentation of monocular videos. Then we present in detail a new real-time video segmentation technique based on the parallel Metropolis algorithm presented in Chapter 2. Finally, we present the results of an extensive experimental evaluation and conclude the chapter.

3.2 Real-time segmentation of monocular videos

The architecture of the framework representing the proposed visual front-end for segmentation of monocular videos is shown in Fig. 3.2. It consists of a video camera, a computer with a GPU, and various processing components that are connected by channels in the framework. Each component can access the output data of all other components in the framework. The processing flow is described as follows. Images are captured by a video camera and undistorted before they enter the framework. *Optical flow* is pre-computed for each frame on the GPU in real-time and the results are accessible from channel 2 for segmentation (see Section 3.2.1).

Segmentation of all frames is performed as follows. The very first frame is segmented completely from scratch using the parallel Metropolis algorithm with the

short-cut introduced in Section 2.2.6. Segmentation of each next frame relies on segments obtained for the previous frame. Thus, similar to the method by Dellen et al. (2009), a pair of two adjacent frames is considered at a time where segments obtained for frame t are used as an initialization of frame $t+1$. However, as opposed to this algorithm, we do not need to incorporate 3D bonds. Instead, spins from the previous frame, residing already in the equilibrium state, are warped to the current frame taking shifts from the optical flow vector field into account. This new spin configuration is much closer to the equilibrium state than a random initialization. Since no cluster updating is performed, labels can be preserved, unlike in the method of Dellen et al. (2009). To complete the segmentation of the current frame, i.e., to arrive at the equilibrium state, initial spin states of frame $t+1$ need to be adjusted to the current image data by the parallel Metropolis running on the GPU. The adjustment of initial spins to the current frame will be referred to as *relaxation process* and the Metropolis updates in the relaxation mode as the *image segmentation core*. This way the required time for segmentation of sequential frames can be reduced, and, even more importantly, a temporally coherent labeling of the frames can be achieved, i.e., segments describing the same object or its part are likely to carry the same spin. The final spin configuration (after convergence) is sent to the main program on the CPU (channel 3) where segments larger than a pre-defined threshold are extracted. After all these processing steps each object or each object part is represented by a uniquely identified segment. This information can be exploited directly by a robot. The framework guarantees that every found segment carries a spin value which is unique within the whole image, therefore, the terms *spin* and *label* are equivalent in this and next chapters. In the upcoming sections we will consider all processing components in more detail.

3.2.1 Phase-based optical flow

Since fast processing is a very important issue in this study, the real-time optical flow algorithm, proposed by Pauwels et al. (2011), is used to find pixel correspondences between adjacent frames in a monocular video stream. The algorithm runs on the GPU and belongs to the class of phase-based techniques, which are highly robust to changes in contrast, orientation, and speed. According to this optical flow can be obtained from the evolution of phase in time (Fleet and Jepson, 1990). The method operates on the responses of a filterbank of quadrature pair Gabor filters tuned to different orientations and different scales. The used filterbank consists of $N = 8$ oriented complex Gabor filters (Sabatini et al., 2010). The different orientations, θ_p, are evenly distributed and equal to $\frac{p\pi}{N}$, with p ranging from 0 to $N-1$. For a specific orientation θ_p the 2D complex Gabor filter at pixel location $\mathbf{x} = (x, y)^T$ equals:

$$f_p(\mathbf{x}) = e^{-\frac{x^2+y^2}{2\sigma_G^2}} e^{j\omega_0(x\cos\theta_p + y\sin\theta_p)}, \tag{3.1}$$

with peak frequency ω_0 and spatial extension σ_G. The filtebank relies on 11×11 separable spatial filters that are applied to an image pyramid (Burt et al., 1983). The peak frequency is doubled from one scale to the next. At the highest frequency a four pixel period is used. The filters are separable and by exploiting symmetry considerations, all 16 responses can be obtained on the basis of only 24 1D convolutions with 11 tap filters (Fleet and Jepson, 1990). The filter responses, obtained by convolving the image $I(\mathbf{x})$, with the oriented filter (3.1) can be written as:

$$R_p(\mathbf{x}) = (I * f_p)(\mathbf{x}) = \rho_p(\mathbf{x}) e^{j\phi_p(\mathbf{x})} = C_p(\mathbf{x}) + jS_p(\mathbf{x}). \tag{3.2}$$

Here $\rho_p(\mathbf{x}) = \sqrt{C_p(\mathbf{x})^2 + jS_p(\mathbf{x})^2}$ and $\phi_p(\mathbf{x}) = \text{atan2}(S_p(\mathbf{x}), C_p(\mathbf{x}))$ are the amplitude and phase components, and $C_p(\mathbf{x})$ and $S_p(\mathbf{x})$ are the real and imaginary responses of the quadrature filter pair. The $*$ operator depicts convolution. The use of atan2 as opposed to atan doubles the range of the phase angle. As a result, correspondences can be found over larger distances (Pauwels et al., 2011).

Phase-based techniques rely on the assumption that constant phase surfaces evolve according to the motion field and points on an equi-phase contour satisfy $\phi(\mathbf{x}, t) = c$, where c is a constant. Differentiation with respect to time gives

$$\nabla\phi \cdot \mathbf{v} + \psi = 0, \tag{3.3}$$

where

$$\nabla\phi = \left(\frac{\delta\psi}{\delta x}, \frac{\delta\psi}{\delta y}\right)^T \tag{3.4}$$

is the spatial phase gradient, $\mathbf{v} = (v_x, v_y)^T$ the optical flow vector, and $\psi = \delta\phi/\delta t$ the temporal phase gradient. Due to the aperture problem, only the velocity component along the spatial phase gradient can be computed (normal flow). Under a linear phase model, the spatial phase gradient can be substituted by the radial frequency vector, $\omega_0(\cos\theta_p, \sin\theta_p)$. Therefore, the component velocity, $\mathbf{c}_p(\mathbf{x})$, can be estimated directly from the temporal phase gradient, $\psi_p(\mathbf{x})$:

$$\mathbf{c}_p(\mathbf{x}) = -\frac{\psi_p(\mathbf{x})}{\omega_0}(\cos\theta_p, \sin\theta_p). \tag{3.5}$$

At each location, the temporal phase gradient is obtained from a linear least-squares fit to the model

$$\hat{\phi}_p(\mathbf{x}, t) = a + \psi_p(\mathbf{x})t, \tag{3.6}$$

where $\hat{\phi}_p(\mathbf{x}, t)$ is the unwrapped phase. Five subsequent frames are used in this estimation. The intercept a is discarded. Each component velocity $\mathbf{c}_p(\mathbf{x})$ provides the linear constraint (3.3) on the full velocity

$$v_x(\mathbf{x}) \cdot \omega_0 \cos\theta_p + v_y(\mathbf{x}) \cdot \omega_0 \sin\theta_p + \psi_p(\mathbf{x}) = 0. \tag{3.7}$$

The constraints given by several component velocities need to be combined to estimate the full velocity. Provided a minimal number of component velocities at pixel $\mathbf{x}$ are reliable (their mean squared error is below the phase linearity threshold), they are integrated into a full velocity by solving the over-determined system of (3.7) in the least-squares sense. A 3×3 spatial median filter is applied (separately to each optical flow component) to regularize the estimates. To integrate the estimates over the different pyramid levels a coarse-to-fine control scheme is employed (Pauwels and Hulle, 2009). Starting from the coarsest level k, the optical flow field $\mathbf{v}^k(\mathbf{x})$ is computed, median-filtered, expanded, and used to warp the phase at the next level, $\phi^{k+1}(\mathbf{x}', t)$, as follows:

$$\mathbf{x}' = \mathbf{x} - 2 \cdot \mathbf{v}^k(\mathbf{x}) \cdot (3 - t). \tag{3.8}$$

This effectively warps all pixels in the five frame sequence to their respective locations in the center frame, i.e., frame three.

Although any other optical flow estimation technique can be used in the proposed framework (Wedel et al., 2008), we decided on the mentioned phase-based approach since it combines high accuracy with computational efficiency. A comparable qualitative evaluation of the method including test sequences from the Middlebury benchmark and implementation details with performance analyses can be found in studies of Gautama and Van Hulle (2002) and Pauwels et al. (2011).

3.2.2 Monocular video segmentation

In the current framework optical flow is computed for the input video stream. The algorithm provides a vector field

$$\mathbf{v}(\mathbf{x}) = (v_x, v_y)^T, \tag{3.9}$$

which indicates the motion of pixels in textured region. Segmentation of a monocular video stream using the parallel Metropolis algorithm with optical flow is shown in Fig. 3.3 on two adjacent frames out of the "Toy" sequence acquired with a moving camera. This sequence is taken from the motion annotation benchmark [1]. An optical flow vector field estimated for two adjacent frames t and $t + 1$ is presented in Fig. 3.3(A - C). Since the employed optical flow algorithm belongs to the class of local methods, optical flow cannot be estimated everywhere (for example not in the very weakly-textured black regions of the panda toy or on the white background). For pixels in these regions, vertical and horizontal flows, i.e., v_y and v_x, do not exist. As was mentioned above, the very first frame in the sequence is segmented from scratch by the parallel Metropolis algorithm with the short-cut (see Section 2.2.6), while segmentation of the following frames relies on segments obtained up to this point using

[1] available under http://people.csail.mit.edu/celiu/motionAnnotation/

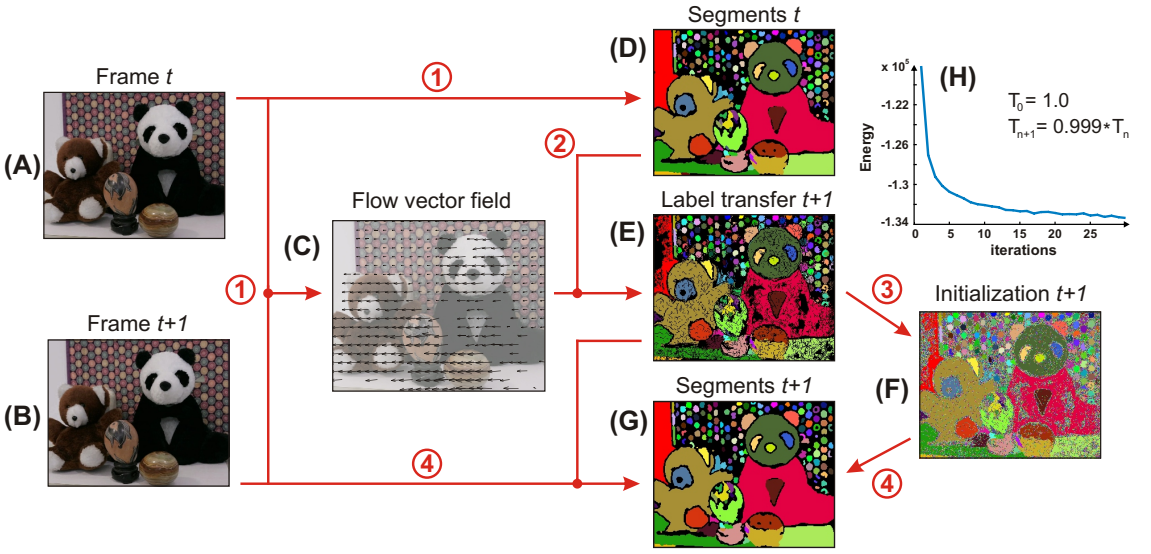

Figure 3.3: Segmentation of two adjacent frames in a sequence using $n_2 = 30$ Metropolis relaxation iterations and $\alpha_2 = 2.5$. Numbers at arrows show the sequence of computations. (A) Original frame t. (B) Original frame $t+1$. (C) Estimated optical flow vector field from the phase-based method (sub-sampled 13 times and scaled 6 times) (step 1). (D) Extracted segments S_t for frame t (step 1). (E) Label transfer from frame t to frame $t+1$ (step 2). (F) Initialization of frame $t+1$ for the image segmentation core (step 3). (G) Extracted segments S_{t+1} for frame $t+1$ (step 4). (H) Convergence of the Metropolis algorithm for frame $t+1$.

the procedure described below. Note that in the current example frame t cannot be the first frame in the sequence, since the considered optical flow algorithm requires five subsequent frames for the estimation and for this reason the framework does not give any output for the first four frames. Furthermore, to avoid usage of future data, optical flow vectors are warped from the center frame in the sequence of five frames to the last frame.

Let us suppose frame t is segmented and S_t is its final label configuration, i.e., obtained segments (see Fig. 3.3(D)). An initial label configuration for frame $t+1$ is found by warping all labels from frame t taking estimations from the optical flow vector field into account as (see Fig. 3.3(E))

$$S_{t+1}(x_{t+1}, y_{t+1}) = S_t(x_t, y_t), \tag{3.10}$$

$$x_t = x_{t+1} - v_x(x_{t+1}, y_{t+1}), \quad y_t = y_{t+1} - v_y(x_{t+1}, y_{t+1}), \tag{3.11}$$

where $\mathbf{v}(\mathbf{x}) = (v_x, v_y)^T$ is the flow at time $t+1$. Since there is only one flow vector per pixel, there will only be one label transferred per pixel. Note that it is not the

case if the flow at time t is used for linking, since there can be multiple flow vectors pointing to the same pixel in frame $t+1$. Pixels which did not obtain an initialization via (3.10) are then given a label which is not occupied by any of the found segments (see Fig. 3.3(F)). Once frame $t+1$ is initialized, it needs to be adjusted to the current image data by the image segmentation core (see Section 3.2). This adjustment is needed in order to fix erroneous bonds that can take place during the transfer of spin states from frame t. Flow interpolations for weakly-textured regions are not considered in this work because of the following reasons:

1. The image segmentation core inherently incorporates the data from all pixel neighborhoods in the image during spin relaxation and, therefore, performs interpolation.

2. An interpolation based on a camera motion estimation is only useful in static scenes (with moving cameras), but cannot help when dealing with moving objects.

The relaxation process performed by the image segmentation core runs until convergence and only after that the final segments are extracted (see Fig. 3.3(G) where corresponding segments between frames t and $t+1$ are labeled with identical colors). Convergence of the relaxation process against a number of iterations is shown in Fig. 3.3(H). For the relaxation process we use an on-line adaptive simulated annealing (see Section 2.2.3) with the schedule determined by both the starting temperature $T_0 = 1.0$ and the simulated annealing factor $\gamma = 0.999$. As we can see the annealing process with this schedule converges after $25 - 30$ iterations making it possible to segment monocular video streams with a frame size of 320×256 pixels in real-time. Longer annealing schedules can lead to better segmentation results but at the cost of processing time.

3.3 Experimental results

Similar to the image segmentation evaluation (see Section 2.3.1) both the quantitative and qualitative evaluations are needed to judge and compare video segmentation techniques. The quantitative evaluation gives a numerical valuation of the machine segmentation results taking the known ground truth data into account. The qualitative evaluation shows outputs of different video segmentation algorithms on the same frame sequence or the same set of sequences giving a user a chance to judge the techniques and select the most appropriate one.

Quantitative evaluation

The quality of video segmentation is measured based on the *segmentation covering* metric, introduced in Section 2.3.1, which evaluates the covering of a human segmen-

tation, called also *ground-truth segmentation*, by a machine segmentation produced by an algorithm under consideration. In the case of video streams ground-truth segmentation is a manual annotation of a video with preserved temporal coherence. In the current study the human-assisted motion annotation tool proposed by Liu et al. (2008a) is used which allows a user to annotate video sequences very efficiently ensuring the spatio-temporal synchronization [2]. The covering of a machine segmentation S by a human segmentation S' for a video stream is defined as

$$C(S' \to S) = \frac{1}{N} \sum_{V \in S} |V| \cdot \max_{V' \in S'} d(V, V'), \tag{3.12}$$

where N denotes the total number of pixels in the video, $|V|$ is the number of pixels in the spatio-temporal volume V and $d(V, V')$ is the Dice coefficient in 3D between the labeled spatio-temporal volumes V and V' within S and S', respectively (Reina et al., 2010). The Dice coefficient between the compared spatio-temporal volumes V and V' is defined as

$$d(V, V') = \frac{2|V \cap V'|}{|V| + |V'|}. \tag{3.13}$$

The covering of a machine segmentation S by a family of ground truth segmentations $\{S'_i\}$ is defined by covering S separately with each human map from $\{S'_i\}$ and then averaging over the different humans. In this way the perfect covering of the machine segmentation is achieved (Arbelaez et al., 2009).

Qualitative evaluation

In Fig. 3.4 video segmentation results for the "Toy" video sequence (see Fig. 3.4(A)) acquired with a moving camera are presented. The ground truth segmentation created with the human-assisted motion annotation tool is shown for some frames in Fig. 3.4(B). Note that the ground truth segmentations provided on the web page of the motion annotation benchmark cannot be used for the comparison in this work, since they show layer segmentation based on motion only without considering color differences. The video segmentation results for both the RGB and CIE $(L^*a^*b^*)$ color spaces are shown in Fig. 3.4(C) and in Fig. 3.4(D), respectively. In both cases the same segmentation parameters and the same annealing schedule have been used. As we can see, results obtained in the CIE $(L^*a^*b^*)$ color space are more accurate which is confirmed by the comparison of the segmentation covering values computed for both color spaces and shown against the system parameter α_2 in Fig. 3.4(E). Furthermore, the image segmentation core in the CIE $(L^*a^*b^*)$ space needs less time to converge. Fig. 3.4(F) shows how the segmentation covering values are changing for both color spaces depending on the number of iterations in the relaxation process.

[2] available under http://people.csail.mit.edu/celiu/motionAnnotation/

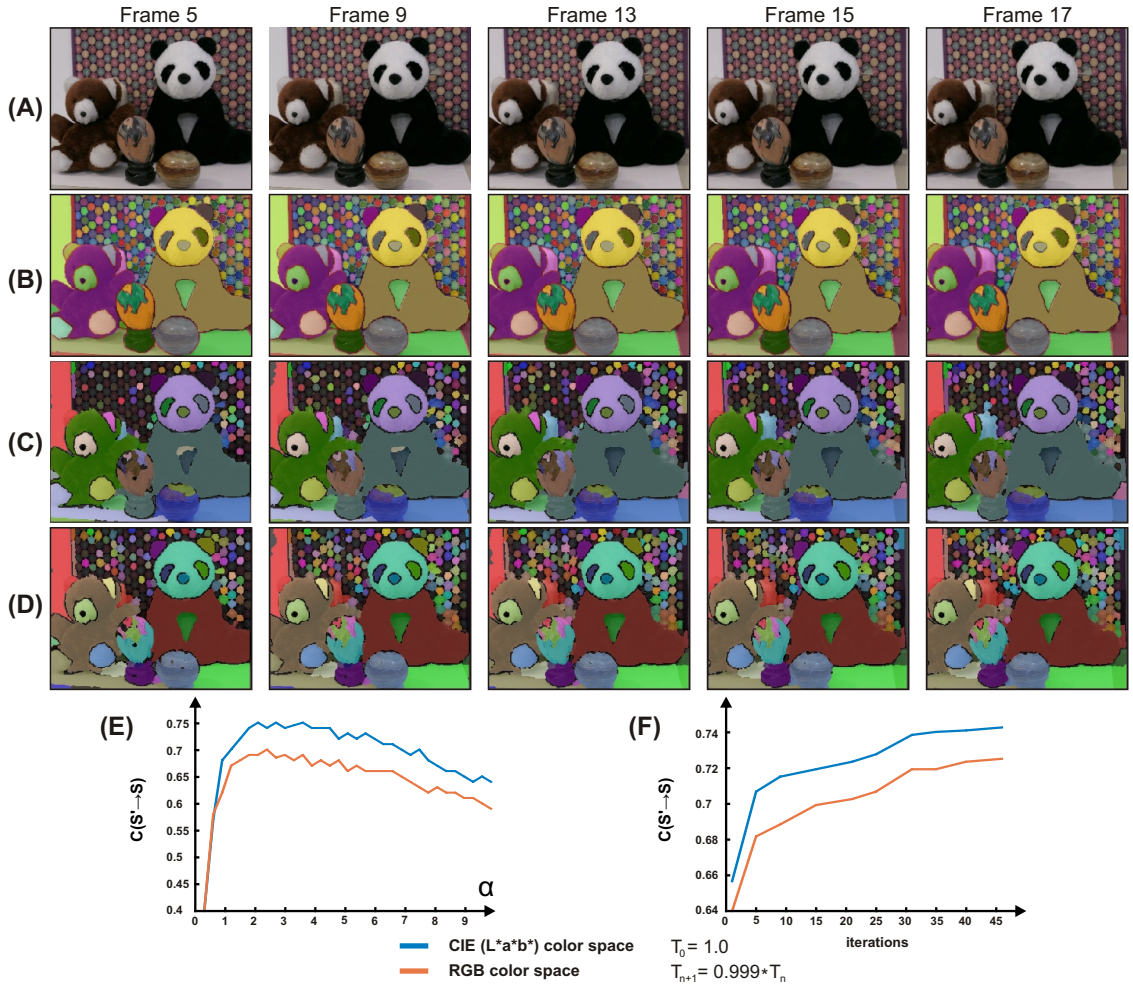

Figure 3.4: Segmentation results for the "Toy" monocular video sequence with a moving camera. (A) Original frames. (B) Ground-truth segmentation created by the human-assisted annotation. (C) Machine segmentation performed in the input *RGB* color space ($n_2 = 30$ iterations, $\alpha_2 = 2.5$). (D) Machine segmentation performed in the perceptual color space *CIE* ($L^*a^*b^*$) ($n_2 = 30$ iterations, $\alpha_2 = 2.5$). (E,F) The segmentation covering shown for both color spaces against the system parameter α_2 and the number of relaxation iterations n_2.

Figure 3.5: Segmentation results for monocular video sequences "Toy" (see Fig. 3.4(A)) and "Phone" (C) with a moving camera and "Women" (F) with moving objects. (A,D,G) Graph-based video segmentation results obtained at 70% (A,D) and 50% (G) of highest hierarchy level. (B,E,H) Segmentation results from the proposed method derived after $n_2 = 30$ iterations with $\alpha_2 = 1.5$ (B,E) and $\alpha_2 = 2.0$ (H), respectively.

More segmentation results in the CIE $(L^*a^*b^*)$ color space are shown in Fig. 3.5. Besides the already considered "Toy" sequence, segmentation results are presented for two more videos: "Phone" (see Fig. 3.5(C)) from the same benchmark and the well-known "Women" sequence containing moving objects (see Fig. 3.5(F)). Results for all sequences obtained by the proposed framework are shown in Fig. 3.5(B,E,H). Although all types of sequences can be successfully segmented using the same set of parameters determined in Section 2.3.2, in the case of the video segmentation it is advisable to use lower values for the system factor α_2 as compared to segmentation of single images. Lower values of α_2 preserve first of all small segments which otherwise can be absorbed by larger segments because of erroneous label transfers. The recommended input parameters for the automatic segmentation of video streams are: $n_2 = 30$ iterations for the relaxation, $\alpha_2 = 2.5$, and $T_{n+1} = 0.999 \cdot T_n$ starting with $T_0 = 1.0$. For the presented segmentation, α_2 was slightly tuned for each sequence to get the best possible segmentation results.

The proposed video segmentation technique is compared here to the hierarchical graph-based video segmentation, proposed by Grundmann et al. (2010), which is to our knowledge the most efficient spatio-temporal segmentation technique to date. Since the publicly available implementation of the graph-based approach uses future data for segmentation and our framework not, both methods cannot be compared entirely and here we only show that our approach gives output comparable to results of the conventional video segmentation methods. From three hierarchy levels available on the web page [3] for the graph-based segmentation, the best segmentation result for each sequence was chosen (see Fig. 3.5(A,D,G)). We can see that the graph-based method leads sometimes to dramatic merges of segments or oversegmentations which is not the case in the proposed approach (see both spherical objects in front of the bears in the "Toy" sequence in Fig. 3.5(A), and a part of the fax machine and the lying on the table papers in Fig. 3.5(D)). However, similar to the graph-based image segmentation, the graph-based video segmentation deals in some situations better with very textured objects (e.g., the background in the "Toy" sequence in Fig. 3.5(A), or the phone in the "Phone" sequence in Fig. 3.5(D)). Also note that the gray-scale "Women" sequence is an extremely difficult case for both techniques due to the lack of color information. Time performance of the framework for various resolutions will be given in Section 4.4.2.

3.4 Discussion

In this chapter we presented a novel framework for real-time spatio-temporal segmentation of monocular video streams based on the parallel Metropolis algorithm introduced in Chapter 2. The proposed visual front-end is on-line, automatic and

[3] available under http://neumann.cc.gt.atl.ga.us/segmentation/

dense. The performance of the framework has been demonstrated on real-world sequences acquired with moving cameras and containing arbitrary moving objects. The GPU architecture is used as an accelerator for highly-parallel computations of the system such as optical flow, and image segmentation core. For the frame resolutions of 160×128 and 320×256 pixels we achieved a processing time sufficient for many real-time robotic applications. The framework manages to process bigger frames as well, but not in real-time mode.

The following problems have been solved by the visual front-end: images from monocular videos are segmented in a consistent model-free way (without prior knowledge of data), the temporal coherence in a monocular video stream is achieved resulting in a consistent labeling of the original frames. However, consistent labeling for a long video sequence can be obtained by the proposed framework only under the following conditions:

1. Objects should not get entirely occluded along the action, since the current method can deal only with partial occlusions. If an object is occluded by any other object, it will not be recognized when it reappears. In order to properly track occluded objects, additional mechanisms are needed that perform high-level analysis of objects (Nummiaro et al., 2002; Wang et al., 1994). It is not possible to resolve such kind of problems on the pixel domain.

2. Objects should not move too fast. The phase-based optical flow used in the current system has a speed limit of 2 pixels per scale, so using 4 scales, the limit is $2^4 = 16$ pixels (Pauwels et al., 2010). In the case of a very fast movement more than 50% of the label transfers can be erroneous. This leads to a completely erroneous initialization of the current frame, which cannot be resolved by the relaxation process in the image segmentation core. The segmentation covering value for such a segment will be dramatically low, which signals inaccurate video segmentation. For the tracking of fast moving objects large displacement optical flow is needed (Brox and Malik, 2011).

3. No disjoint parts of physically the same object should be joined during the action. If two large parts of the same object represented by different segments are merged, we face again the domain fragmentation problem (see Section 2.2.6). In the current framework the domain fragmentation problem can be resolved only by a very long annealing schedule (see Section 2.2.3) which cannot be achieved in real-time.

An important goal of this work has been the improvement of the computational speed of the system, since a low latency in the perception-action loop is a crucial requirement of systems where a visual front-end is needed. Consequently, since the proposed framework is running in real-time, it can be used in a wide range of robotic applications such as object manipulation, visual servoing, and robot navigation. All

these applications require object detection and tracking along with the extraction of meaningful object descriptors as a pre-processing step.

In the future, the mentioned limitations need to be overcome. For very complex scenarios where objects are getting occluded all the time, some high-level knowledge about objects needs to be accumulated during that part of the sequence where objects are present and visible.

4

Real-time Segmentation of Stereo Video Streams

"Great things are done by a series
of small things brought together"
– Vincent Van Gogh

4.1 Introduction

In this chapter we present a novel visual front-end for real-time spatio-temporal segmentation of stereo videos. Although stereo data has recently been employed for segmentation (Ladický et al., 2010; Mutto et al., 2011), there is no method that performs real-time spatio-temporal segmentation of stereo videos while simultaneously establishing correspondences between left and right segments. The segmentation of stereo videos is of high importance in computer vision, since segmented stereo videos provide an additional information about the scene and allow us to derive 3D relations between objects (Aksoy et al., 2011). Furthermore, the obtained correspondences between segments in the left and right video streams can be used for depth computation (Dellen and Wörgötter, 2009).

The visual front-end proposed here is on-line, automatic, dense, and solves the following problems (Abramov et al., 2012c):

1. Stereo images are segmented in a consistent model-free way using the image segmentation core applied to segmentation of monocular video streams in Chapter 3.

2. The temporal coherence in a stereo video stream is achieved using a label-transfer strategy based on estimated motion within left and right video streams and disparity data, showing the amount of horizontal motion between two views, resulting in a consistent partitioning of neighboring frames together with a consistent labeling. Only the results obtained on the very last left and right

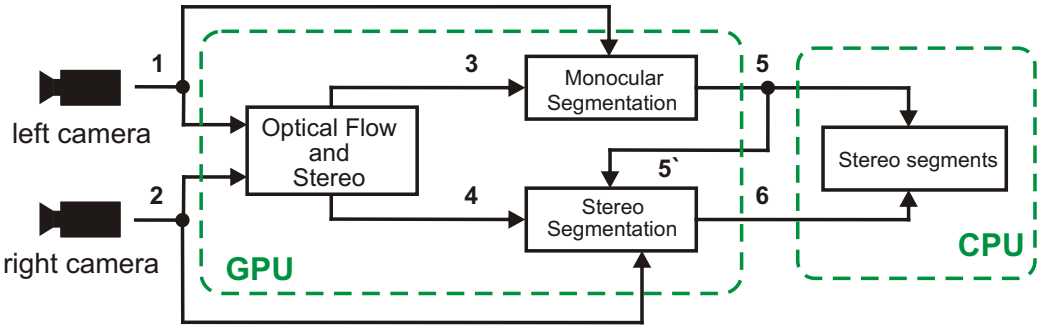

Figure 4.1: The architecture of the framework for segmentation of stereo videos on the heterogeneous computing system consisting of one CPU and one GPU.

frames are employed at a time in order to guarantee spatio-temporal coherence for the current left and right frames, respectively.

3. All computations run in real-time which allows the framework to be used in the perception-action loop.

The chapter is organized as follows. First we introduce the framework for the segmentation of stereo videos and extend the segmentation method for monocular video streams to stereo video streams. We further present an implementation of the framework on a portable system with a mobile GPU. Finally, we perform an extensive experimental evaluation and discuss the results.

4.2 Real-time segmentation of stereo videos

The architecture of the framework for segmentation of stereo videos is shown in Fig. 4.1. It consists of a stereo camera, a computer with a GPU, and various processing components that are connected by channels in the framework. Each component in the framework can access the output data of all other components in the framework. The processing flow is as follows. Stereo images (synchronized left and right frames) are captured by a stereo camera. The acquired images are undistorted and rectified (in real-time with a fixed stereo-geometry (Bradski, 2000)) before they enter the framework (channels 1 and 2). Optical flow is computed for the current left and right frames together with the disparity map on the GPU using real-time algorithms and the results are accessible from channels 3 and 4.

Segmentation of both left and right streams is performed as follows. Similar to the segmentation of monocular video streams considered in the previous chapter, only the very first frame of the left stream is segmented completely from scratch by the parallel Metropolis algorithm with the short-cut introduced in Section 2.2.6. Segmentation of

the right stream relies on segments obtained for the left stream, while segmentation of the left video stream is equal to the segmentation of monocular video streams introduced in Chapter 3. The segmentation results of left frames can be accessed from channel 5.

A label initialization of the current right frame is created by warping of both the current left (channel 5′) and previous right segments using the optical flow and disparity information (channel 4) (see Section 3.2.1 and Section 4.2.1). Similar to the segmentation of the left stream, the initial labels are adjusted to the image data of the current right frame by the relaxation process of the image segmentation core. The segmentation results of the right frame, which is now consistently labeled with respect to its corresponding left frame, are stored in channel 6. Once segmentation for both left and right frames is achieved, the final spin configuration (after convergence) is sent to the main program on the CPU (channels 5 and 6) where segments larger than a pre-defined threshold are extracted. After all these processing steps each object or object part is represented by uniquely identified left and right segments.

4.2.1 Phase-based stereo

Since fast processing is a very important issue in the present study, the real-time stereo algorithm, proposed by Pauwels et al. (2011), is used to find pixel correspondences between left and right frames in a stereo video stream. The algorithm runs on a GPU and belongs to the class of phase-based techniques, which are highly robust to changes in contrast, orientation and speed. According to this stereo disparity estimates can be efficiently obtained from the phase difference between the left and the right image (Fleet and Jepson, 1990). For oriented filters (see Section 3.2.1), the phase difference has to be projected on the epipolar line. Since in the current study we work with rectified images, this is equal to the horizontal. For a filter at orientation θ_p, a disparity estimate is obtained as follows:

$$\delta_p(\mathbf{x}) = \frac{[\phi_p^L(\mathbf{x}) - \phi_p^R(\mathbf{x})]_{2\pi}}{\omega_0 \cos\theta_p}, \tag{4.1}$$

where the $[\,]_{2\pi}$ operator depicts reduction to the $]-\pi;\pi]$ interval. These different estimates are robustly combined using the median. As in the case of optical flow, to reduce noise, a subsequent 3×3 median filtering is performed that gives the median as an output if the majority of its inputs are valid, otherwise it signals an invalid estimate. Because of phase periodicity, the phase difference approach can only detect shifts up to half the filter wavelength. To compute larger disparities, the estimates obtained at the different pyramid levels are integrated by means of coarse-to-fine control strategy (Bergen et al., 1992). A disparity map $\delta^k(\mathbf{x})$ is first computed at the coarsest level k. It is upsampled to be compatible with the next level, using an

expansion operator χ, and multiplied by two:

$$d^k(\mathbf{x}) = 2 \cdot \chi(\delta^k(\mathbf{x})). \tag{4.2}$$

This map is then used to reduce the disparity at level $k+1$, by warping the right filter responses before computing the phase difference

$$\delta_p^{k+1}(\mathbf{x}) = \frac{[\phi_p^L(\mathbf{x}) - \phi_p^R(\mathbf{x}')]_{2\pi}}{\omega_0 \cos\theta_p} + d^k(\mathbf{x}), \tag{4.3}$$

where

$$\mathbf{x}' = (x + d^k(\mathbf{x}), y)^T. \tag{4.4}$$

Consequently, the remaining disparity is guaranteed to lie within the filter range. This procedure is repeated until the finest level is reached. The median filter is applied at each scale of the pyramid.

Although any other stereo technique can be used in the proposed framework (Scharstein and Szeliski, 2002), we decided to use phase-based approach since it combines high accuracy with computational efficiency. Furthermore, the used implementation combines both the phase-based optical flow, employed in segmentation of monocular video streams (see Section 3.2.1), and stereo in a very efficient manner. A comparable qualitative evaluation of the method including test stereo pairs from the Middlebury benchmark and implementation details with performance analyses can be found in studies of Gautama and Van Hulle (2002) and Pauwels et al. (2011).

4.2.2 Stereo video segmentation

In the proposed framework disparity is computed for each input stereo pair. Segmentation of a stereo video stream using the parallel Metropolis algorithm with optical flow and stereo is shown in Fig. 4.2 on one stereo pair consisting of left and right frame. The procedure is very similar to the segmentation of a monocular video stream. Here, an initial label configuration for the right frame at time t is obtained by warping the labels from both the corresponding left frame t and the previous right frame $t-1$. Labels from the left frame are transferred using the disparity map d (see Fig. 4.2(A - C)) and labels from the previous right frame are transferred using the optical flow vector field (see Fig. 4.2(E)). Since the stereo algorithm relies on phase (and not magnitude), it can find correct matches even in weakly-textured regions. Also, ambiguous matches are avoided by the use of a coarse-to-fine control mechanism. However, reliable information cannot be found under drastically changing light conditions (see the reflection shift over the table).

Suppose the left frame L_t is segmented and S_L is its final label configuration (see Fig. 4.2(D)). Labels from the previous right frame R_{t-1} are warped according to the

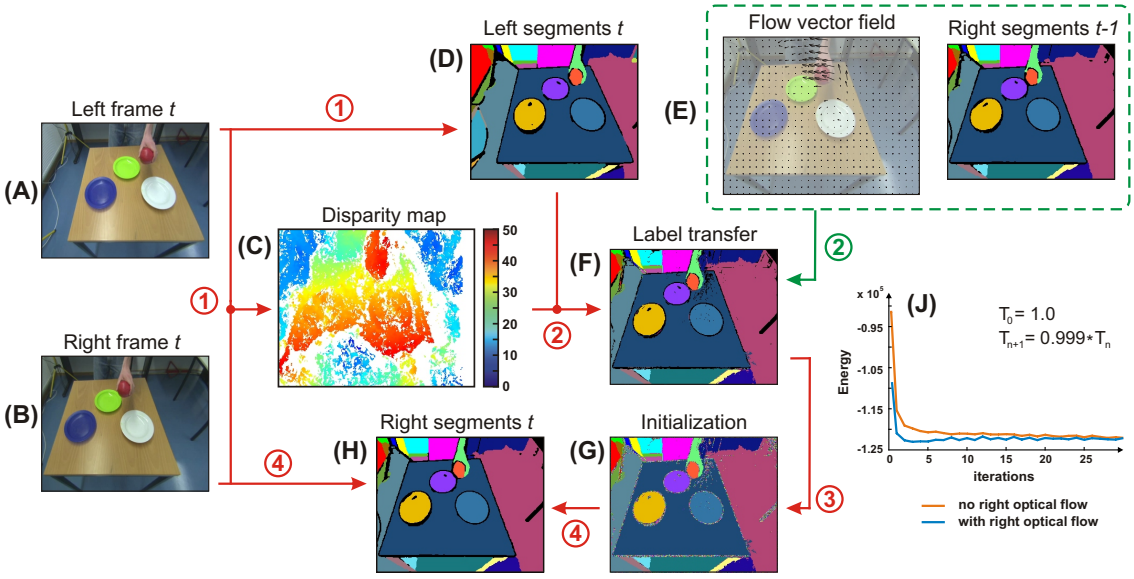

Figure 4.2: Segmentation of a stereo pair for the time moment t. Numbers at arrows indicate the order of the computations. (A) Original left frame L_t. (B) Original right frame R_t. (C) Disparity map estimated by the phase-based method (step 1). (D) Extracted segments S_L for frame L_t after $n_2 = 30$ iterations with $\alpha_2 = 2.5$ (step 1). (E) Segments and estimated optical flow vector field for right frame $t-1$ (sub-sampled 13 times and scaled 6 times). (F) Label transfer from frames L_t and R_{t-1} to frame R_t (step 2). (G) Initialization of frame R_t for the image segmentation core (step 3). (H) Extracted segments S_R for frame R_t after $n_2 = 10$ iterations with $\alpha_2 = 2.5$ (step 4). (J) Convergence of the Metropolis algorithm for frame R_t.

procedure described in Section 3.2.2, whereas labels from the current left frame L_t are warped to the right frame as follows

$$S_R(x_R, y_R) = S_L(x_L, y_L), \tag{4.5}$$

$$x_L = x_R + \delta_p(x_R, y_R), \quad y_L = y_R. \tag{4.6}$$

The disparity map δ_p is computed relative to the right frame which guarantees that there will only be one label transferred per pixel from the left frame. Both warpings are performed at the same time (see Fig. 4.2(F)). In the case of multiple correspondences, i.e., if a pixel in frame R_t has label candidates in frames L_t and R_{t-1}, there are no preferences and we select randomly either the flow or the stereo. In this way they can both contribute without bias and the segmentation core can make the final decision. Pixels that did not obtain a label initialization via (4.5) are given a label which is not occupied by any of the found segments (see Fig. 4.2(G)). Once frame R_t is initialized, it needs to be adjusted to the current image data by the image segmentation core (see Section 3.2). This adjustment is needed in order to fix erroneous bonds that can take place during the transfer of spins. The relaxation process runs again until it converges and only after that the final right segments S_R at time t are extracted (see Fig. 4.2(H) where correspondent segments between frames L_t and R_t are labeled with identical colors). Convergence of the relaxation process against a number of iterations is shown in Fig. 4.2(J) for the combined label transfer and for the label transfer based only on disparity shifts without the use of optical flow for the right stream. We can see that the use of the previous right labels drastically reduces a number of iterations needed for convergence and already after $5 - 10$ iterations the final right segments can be extracted. It makes it possible to segment stereo video streams with a frame size of 320×256 pixels in real-time. Using only stereo information about $25 - 30$ iterations are needed in order to reach the equilibrium state. This is because occlusions in stereo images are significantly larger than occlusions between adjacent frames in one video stream if disparities are large. For the relaxation process we use an on-line adaptive simulated annealing (see Section 2.2.3) with the same schedule as for the segmentation of monocular video stream with parameters $T_0 = 1.0$ and $\gamma = 0.999$. Note that longer annealing schedules can lead to better segmentation results but at the cost of processing time.

4.3 Experimental results

To evaluate segmentation results of stereo videos again both the quantitative and qualitative measures, presented in Section 3.3, are used. In Fig. 4.3 segmentation results for two stereo sequences are shown. Since the sequences are quite long, only stereo pairs at a few key points of actions can be shown. In the first sequence,

Figure 4.3: Segmentation results for stereo frame sequences of the sample actions "Moving an apple over plates" with moving objects (A) and "Cluttered scene" with a moving stereo camera (B). Results are obtained using the following parameters: $n_2 = 30$ and $n_2 = 15$ iterations are applied for the relaxation of left and right frames, respectively, $\alpha_2 = 2.5$ for both the left and right streams, the annealing schedule is $T_{n+1} = 0.999 \cdot T_n$ starting with $T_0 = 1.0$.

called "Moving an apple over plates", a hand moves an apple around the table and places it on a plate (see Fig. 4.3(A)). In the second scenario, "Cluttered scene", the scene is static but the stereo camera moves (see Fig. 4.3(B)). As we can see the spatial-temporal coherence is achieved in the segmentation of both stereo sequences and the determined stereo segments correspond to the natural partitioning of the original stereo pairs. Too small segments are completely removed from the final label configuration.

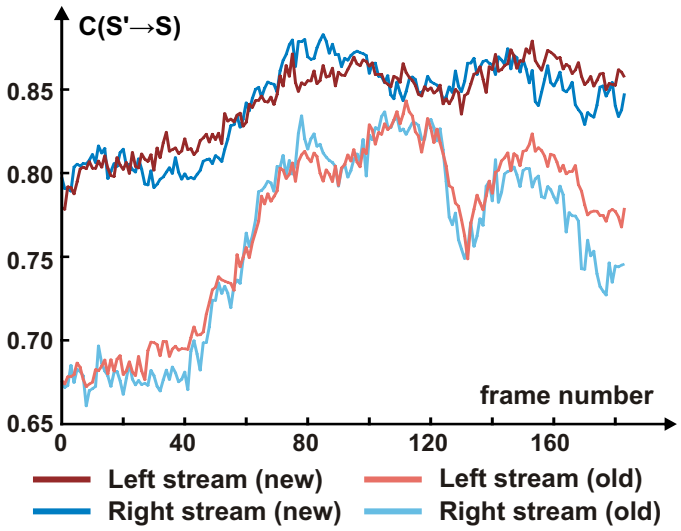

Figure 4.4: Segmentation covering for the stereo sequence "Moving an apple over plates" shown for the previous and current framework versions. The average values are 0.77 (left stream) and 0.76 (right stream) for the previous version and 0.84 (for left and right streams) for the current version, respectively.

The performance comparison of the proposed framework with its previous version using the input *RGB* color space and optical flow for the left stream only (Abramov et al., 2010a) is shown in Fig. 4.4 as the segmentation covering against the current frame number. As we can see, in the proposed framework the left and right sequences are segmented with higher accuracy (the average segmentation covering value is 0.84 for both streams as opposed to 0.77 in the previous version). Furthermore, the current approach is more robust, having significantly smaller deviations of the segmentation covering values along the whole sequence. Time performance of the framework for various resolutions is given in Section 4.4.2.

4.4 Implementation on a portable system

Processing power, memory bandwidth and number of cores are not the only important parameters in robotic systems. Since robots are dynamic, movable and very often wireless systems, huge processing platforms with high power consumption (mostly for cooling) are not practicable despite their high processing efficiency. Because of this, mobile parallel systems running on portable devices are of growing interest for computer-controlled robots. Nowadays mobile GPUs from the Nvidia G8X series are supported by CUDA and can be used very easily for general-purpose parallel computing. In Fig. 4.5, the dynamics of development for desktop and mobile GPUs from the Nvidia G8X series until today are shown, demonstrating that desktop GPUs are three times more powerful and have three times faster memory bandwidths than mobile ones. However, powerful desktop GPUs consume so much power that it is almost impossible to use them in small computer-controlled robots, while even the most powerful mobile GPUs integrated into mobile PCs do not need an extra power supply. Taking this fact into account we consider in the current study a mobile PC with an integrated mobile GPU from Nvidia supported by CUDA as a portable system. Such a system can run for up to three hours in autonomous mode being supplied by the laptop battery.

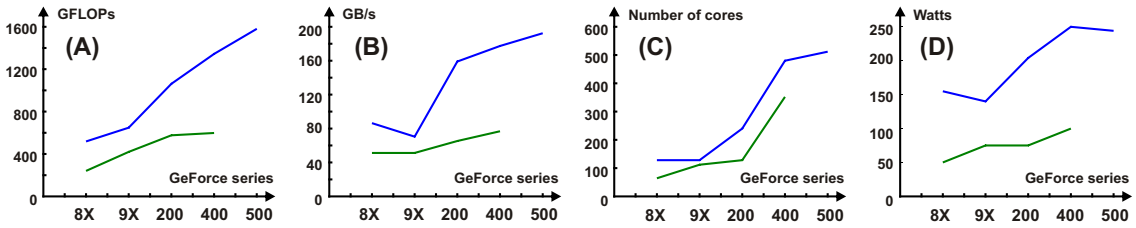

Figure 4.5: Comparison of desktop (blue) and mobile (green) graphics cards for Nvidia GeForce 8X, 9X, 100, 200, 400, 500-series GPUs with a minimum of 256 MB of local graphics memory. The following parameters are compared: (A) Processing power in floating point operations per second, (B) Maximum theoretical memory bandwidth, (C) Number of CUDA cores, and (D) Graphics card power.

Here we present an implementation of the proposed framework for real-time spatio-temporal segmentation of stereo videos on a mobile PC with an integrated mobile GPU. The architecture of the mobile framework for segmentation of stereo videos is shown in Fig. 4.6(A). The only difference to the framework introduced in Section 4.2 is that the segmentation core, the phase-based optical flow, and the stereo algorithm run on the mobile GPU instead of the common desktop GPU and the main program runs on the portable system. Uniquely identified left and right segments can be exploited directly by a mobile robot. A prototype of a movable robot steered by a

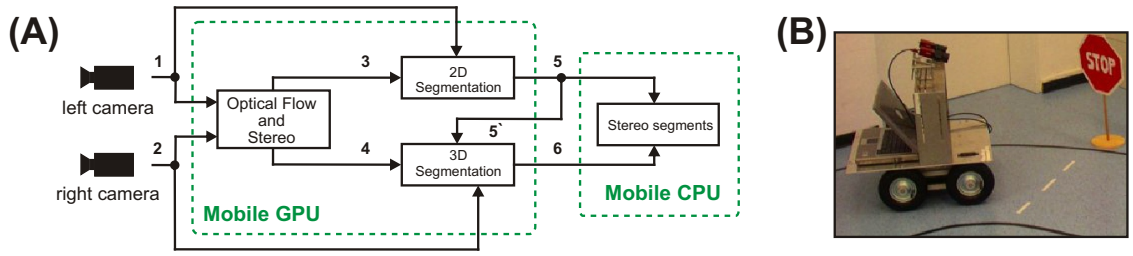

Figure 4.6: (A) The architecture of the framework for segmentation of stereo videos on the portable system with a mobile GPU. (B) A movable robot steered by a mobile system with stereo cameras and a laptop with an integrated mobile GPU.

mobile system including stereo cameras and a laptop with an integrated mobile GPU is shown in Fig. 4.6(B).

4.4.1 Experimental environment

The proposed framework runs on a laptop with mobile Intel Core 2 Duo CPU with 2.2 GHz and 4 GB RAM. The mobile GPU used in the laptop is Nvidia GeForce GT 240M (with 1 GB device memory). This card has 6 multiprocessors and 48 processor cores in total and belongs to the 200-series of mobile Nvidia GPUs. The card is shared by all the framework components running on the GPU. As a desktop GPU (used for the comparison of processing times) we use here Nvidia GeForce GTX 295 (with 896 MB device memory) consisting of two GPUs, each of which has 30 multiprocessors and 240 processor cores in total. In this study we use only one GPU of this card.

4.4.2 Time performance

Time performance of all components of the proposed framework is shown as a function of frame size in Fig. 4.7. Image resolutions 160×128, 320×256, and 640×512 pixels are marked by black dashed lines. The processing times of components running on the mobile GPU are compared to the respective runtime on the desktop GPU (Fig. 4.7(A - C)). Runtimes of the video segmentation are shown for monocular as well as stereo video streams. For segmentation of monocular video streams $n_2 = 30$ Metropolis relaxation iterations are used, whereas for stereo video streams besides the same $n_2 = 30$ iterations required for the left stream additional $n_2 = 15$ iterations are needed for relaxation of the right stream resulting in $n_2 = 45$ iterations in total (see Fig. 4.7(B)). Note that the relaxation process takes about 60% of the whole runtime.

Although all computations on the mobile card are significantly slower (the speed up factors derived on the desktop card in relation to the mobile one for optical flow /

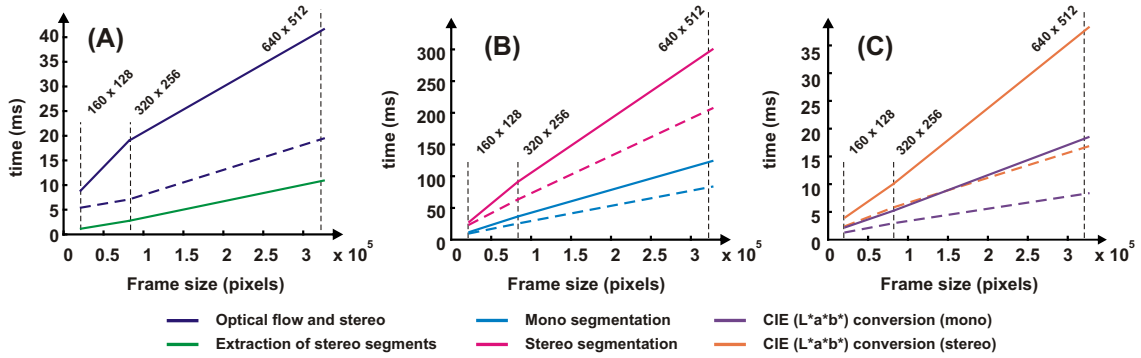

Figure 4.7: Processing times of all stages of the framework for segmentation of monocular and stereo videos for both the mobile and desktop platforms. For computations running on the mobile GPU, processing times derived on the desktop GPU are shown for comparison (by dashed lines). (A) Runtime for optical flow with stereo and extraction of stereo segments. (B) Processing time of monocular segmentation ($n_2 = 30$ iterations) and stereo segmentation ($n_2 = 45$ iterations). (C) Runtime for conversion from the input RGB color space to the CIE ($L^*a^*b^*$) space for both monocular and stereo streams.

stereo and image segmentation core are 2.1 and 2.4, respectively), it is still possible to process several frames per second for all considered resolutions as shown in Table 4.1.

Resolution (px)	CPU sec (Hz)	GTX 295 msec (Hz)	GT 240M msec (Hz)
160×128	0.8 (1.2)	40.0 (25.0)	47.4 (21.1)
320×256	3.4 (0.3)	75.0 (13.3)	117.0 (8.5)
620×512	13.9 (0.1)	230.0 (4.3)	376.0 (2.7)

Table 4.1: Processing times per frame and frame rates for the framework performing segmentation of stereo videos on the CPU, desktop GPU, and mobile GPU platforms.

4.5 Discussion

In this chapter we presented a novel framework for real-time spatio-temporal segmentation of stereo video streams and its implementation on a portable system with an integrated mobile GPU. The proposed visual front-end is on-line, automatic and dense. The performance of the framework has been demonstrated on real-world se-

quences acquired with moving cameras and containing arbitrary moving objects. A trade-off between processing time and hardware configuration exists. Since robotic systems are usually dynamic, movable, and very often wireless autonomous systems, huge computers with high power consumption not always can be considered as a proper hardware architecture. As the most suitable platform for this task we chose a mobile PC with an integrated mobile GPU. Being supplied by the laptop battery such a system can run in autonomous mode up to three hours. A GPU is used as an accelerator for highly-parallel computations of the system such as optical flow, stereo, and image segmentation core. For the frame resolutions of 160×128 and 320×256 pixels we achieved a processing time which is sufficient for many real-time robotic applications. For the resolution 640×512 pixels only close to real-time performance can be achieved. The system can process bigger frames as well, but not in real-time.

5

Disparity from Stereo-segment Correspondences

"The art of painting can never reproduce space because painting lacks the relief of objects in space"
– Leonardo da Vinci

5.1 Introduction

Stereoscopic images are an important cue for computing depth. So closing each eye alternately we can see that objects jump to left and right relative to background and objects in front have larger displacements comparing to objects behind. In the rectified stereo vision geometry, when both cameras are looking straight ahead, this effect results in the amount of horizontal motion or *disparity*. Disparity is inversely proportional to the distance from the observation point. The process of measuring disparity by establishing pixel correspondences between left and right images is known as *stereo matching* and is a widely studied topic in computer vision. Despite a significant progress made during the past two decades it still remains a very active research area because some problems could still not be resolved, e.g., correct depth estimation in weakly-structured image areas. Accurate dense stereo is important for many computer vision applications as 3D robotic navigation and manipulation, 3D modeling, object tracking and image rendering (Szeliski, 2010).

5.1.1 Conventional stereo algorithms

Existing stereo matching algorithms can be classified in two major groups: global and local methods (Scharstein and Szeliski, 2002; Szeliski, 2010). Global methods are featured by a global cost function associated with an input stereo pair. The goal is to find disparities minimizing a global energy (Geiger et al., 1995; Bobick and Intille,

1999; Boykov and Jolly, 2001; Sun et al., 2003; Felzenszwalb and Huttenlocher, 2006; Heo et al., 2011). Local methods are window-based and only use image information in a finite shiftable window surrounding each pixel or a group of pixels (Veksler, 2003; Yoon et al., 2006; Tombari et al., 2007; Hirschmüller, 2008). Local methods are easier to implement efficiently using parallel architectures and, as a consequence, multiple real-time local techniques on various platforms have been proposed over the last decades (Bradski, 2000; Díaz et al., 2007; Lu et al., 2009; Pauwels et al., 2010). Global methods are more accurate, but they are more difficult to parallelize and real-time implementations exist only for low resolutions or for considerably simplified algorithms (Brunton et al., 2006; MacLean et al., 2010; Liang et al., 2011). In the meantime there are some techniques in between of global and local methods that are commonly applied in real-time implementations. These methods are known as coarse-to-fine algorithms operating on an image pyramid where estimations made at coarser levels constrain a more local search at finer levels (Zitnick et al., 1999; Pauwels et al., 2010; Sizintsev et al., 2010).

Global methods can be classified depending on the used computation technique into following two categories: dynamic programming-based (Geiger et al., 1995; Bobick and Intille, 1999) or Markov Random Fields (MRFs)-based (Boykov and Jolly, 2001; Sun et al., 2003; Felzenszwalb and Huttenlocher, 2006; Heo et al., 2011). At the present time global optimization techniques achieve the highest ranking on the Middlebury stereo dataset (Scharstein and Szeliski, 2008), therefore, we consider them in the current study as the most up-to-date and efficient stereo matching techniques. Original energy minimization methods, such as iterated conditional modes (ICM) (Besag, 1986) or simulated annealing (Barnard, 1989) are extremely slow and not very efficient. During the last few years new powerful optimization algorithms such as graph cuts (Boykov and Jolly, 2001; Kolmogorov and Zabih, 2004) and loopy belief propagation (LBP) (Yedidia et al., 2000) have been proposed. These methods provide more accurate results comparing to other stereo approaches and currently almost all top-performing stereo methods rely on graph cuts or LBP (Scharstein and Szeliski, 2008).

Nowadays it is almost impossible to test and evaluate all existing stereo approaches due to the following reasons: first, some techniques are not open source projects and cannot be tested easily; second, too many approaches have been proposed during the last two decades and some of them are not supplied with sufficient description of system parameters that can dramatically affect disparity estimation. This makes the comparison with these techniques unfair, since best results of the methods cannot be derived for the arbitrary testing dataset. In the current study we will consider the following well-known and widely used stereo matching algorithms as conventional stereo methods: *block matching* (BM) (Hirschmüller, 2008), *iterated conditional mode* (ICM) (Besag, 1986), *swap-move* (Swap) and *expansion-move* (Expansion) graph cuts algorithms (Boykov and Jolly, 2001), *sequential tree-reweighted message passing* (TRW-S) (Kolmogorov, 2006), *belief propagation* (BP) (Felzenszwalb and Hutten-

locher, 2006), *contrast space belief propagation* (CSBP) (Yang et al., 2010), *the max-product loopy belief propagation* (BP-M) (Tappen and Freeman, 2003) and *sequential loopy belief propagation* derived from the TRW-S (BP-S), *phase-based stereo* (Pauwels et al., 2010). Implementations of ICM, BP-M, BP-S, Expansion, Swap, TRW-S have been taken from the Middlebury webpage [1]. BM, BP and CSBP are fast stereo methods from the open computer vision library accelerated recently on the GPU [2]. Phase-based stereo is a real-time stereo technique where phase differences between the left and the right images are pooled across different orientations and propagated from coarser to finer scales (see Section 4.2.1).

5.1.2 Performance evaluation

Performance evaluation and comparison of stereo algorithms is not a straightforward procedure due to many factors that need to be taken into account. One important evaluation criterion is the accuracy of a computed disparity map (Scharstein and Szeliski, 2002; Brown et al., 2003; Seitz et al., 2006). It can be judged in two following ways. The estimated disparity values can be either compared with a ground truth disparity map, obtained using tools such as a laser range finder, or an original right image is compared with a synthetic image rendered by warping a left image by a computed disparity map. Scharstein and Szeliski (2008) created the Middlebury stereo benchmark containing a set of stereo images with acquired ground truth disparity maps which is currently the most famous and widely used testing dataset in stereo vision. It gives all scientists an opportunity to compare their own methods with the others on the same set of data.

Unfortunately, the Middlebury dataset does not perform a complete comparison and evaluation of stereo algorithms because of the following reasons. First, it does not take time performance into account, nor the ability of the method to evaluate the quality of its own estimates. The former determines how many stereo frames can be processed by the algorithm per second, while the latter determines the density of the computed disparity map. Second, available testing stereo pairs are very limited and do not represent all variety of input images that stereo matching methods should be able to deal with. Almost all stereo pairs in the dataset are featured by a high level of texture which makes the matching procedure easier, while a lot of images used in industry are weakly-textured. Therefore, some methods highly evaluated by the Middlebury stereo benchmark might be very slow or provide only inaccurate and very sparse disparity maps for weakly-textured scenes.

[1] available under http://vision.middlebury.edu/stereo/code/

[2] see http://opencv.itseez.com/modules/gpu/doc/gpu.html

5.1.3 Motivation and scope

The most common reasons for obtaining bad depth estimates in stereo matching can be summarized as follows: lack of texture or repetitive texture, object boundaries, half-occlusions, i.e., regions visible only in one of two images, changing light conditions, reflections, and image noise. In the current study we use a texture measurer to classify an input image as weakly- or significantly-textured.

Most stereo algorithms perform well in textured image areas, but often fail when there is only weak texture, due to the correspondence problem. Here local matching fails, and, as a consequence, global methods do not deliver correct disparities either, simply because the energy functions used in global methods remain under-constrained. However, stereo from weakly-textured images is important for many applications, which take place in urban or industrial settings, where little texture exists and active techniques based on the structured-light suffer from the problems such as multiple or glossy reflection, ambient light or light absorption (Zhu et al., 2011). Therefore, novel solutions are required to the stereo problem.

While being ill-suited for stereo analysis, weakly-textured image parts can easily be used for color-based segmentation and, in addition, it is often also possible to find unique segment correspondences between two views in the stereo image. Stereo segments provide an additional information limiting the search area for stereo techniques.

5.1.4 Related work

In the past, color image segmentation (Yang et al., 2008; Dellen and Wörgötter, 2009) has been used to improve disparity estimation in weakly-textured scenes. The method proposed by Yang et al. (2008) uses image segmentation to recover disparity in textureless regions by fitting plane surfaces to the weak disparity information found for these segments. Since this method also depends on texture, it fails in untextured areas.

Dellen and Wörgötter (2009) proposed another method that obtains disparity for weakly-textured images from found stereo segments. The method uses interpolation algorithm based on a spring-mass model. It was tested on the Middlebury stereo dataset including such poorly-textured images like *Plastic* and *Lampshade* (see Fig. 5.1). The method can compute disparity also for completely untextured regions because information from the segment boundaries is used as well. Computed disparity maps have a density about 90% and are of acceptable quality. However, the method has a number of drawbacks. Sparse disparity computed inside stereo segments by a window-based matching algorithm can contain some inaccurate estimations that dramatically affect interpolation results. Reliable disparity data cannot be derived for background / foreground segments whose boundaries are out of the image and image edges are partly interpreted as their boundaries. Lastly, the method is extremely slow

requiring minutes to process a frame with a size of 320×256 pixels.

The goal of the present study is to recover disparity in weakly-textured image parts in real-time using the stereo image segmentation (Abramov et al., 2012b). Establishing unique correspondences between left and right segments gives an additional information about objects present in the scene. A sparse disparity output of any conventional stereo technique combined with an additional information and constraints derived from found stereo regions make it possible to compute missing disparity data based on pre-defined surface models associated with segments. Occlusions and related problems such as segment-boundary ownership are considered during this procedure, without which the method would not provide accurate results. This way we can regenerate rather accurate disparity information in regions that are usually quite resistant to stereo analysis, such as certain images from the Middlebury stereo dataset, which are, for this reason, rarely being used for stereo algorithm benchmarking, and other images containing little texture. The method should run in real-time giving a dense disparity for all objects in the scene.

The chapter is organized in the following way. First we will present the texture quantification and evaluate performance of the conventional stereo algorithms on images featured by diverse levels of texture. Then we will give a description of the proposed real-time dense stereo approach. Afterwards the experimental results, quantitative analysis, and time performance are given. Finally we will conclude the chapter and discuss our results.

5.2 Texture as a crucial point

5.2.1 Texture quantification

First of all we need to ask us the following questions: *What is texture? How can we measure texture?* Actually there is no general definition of texture in the literature and all existing texture detectors are based on their own views what texture is. Tuceryan and Jain (1998) say that *"We recognize texture when we see it but it is very difficult to define"* and give six different definitions of texture. The main purpose of the texture analysis is to quantify intuitive qualities as rough, rugged, smooth, or grainy as a function of the spatial variation in pixel intensities. For the texture quantification in this work we use an approach based on *entropy calculation* of a grayscale image (Hong et al., 2008). Entropy is a statistical measure of randomness defined for a region of n pixels as

$$\mu = -\sum_{i=0}^{n} p(x_i) \cdot log_2 p(x_i), \tag{5.1}$$

where $p(x_i)$ is the grayscale value of x_i. The entropy function μ characterizes the texture in such a way that smooth regions are featured by a small range of values in

the neighborhood around a pixel, whereas in textured areas a range is larger [3]. The texture quantification assigns to each pixel an entropy value μ of the neighborhood around the corresponding pixel where low and high values correspond to weakly and sufficiently textured regions, respectively.

5.2.2 Testing dataset

For performance evaluation of conventional stereo methods and for testing purposes we created our own testing stereo dataset. It consists of various stereo images with diverse entropy values: images from the Middlebury stereo benchmark, images from object manipulation scenarios, and images of plants. Some images from the dataset with related outputs of the texture quantification and computed entropies are shown in Fig. 5.1. In our dataset entropy values μ averaged over the image are in the range of $[1.0, 5.5]$ which represents a wide spectrum of images from weakly-textured *Cups* ($\mu = 1.54$) and *Plastic* ($\mu = 2.2$) to sufficiently textured *Aloe* ($\mu = 5.12$) and *Cones* ($\mu = 4.83$). Note that on the Middlebury webpage only highly-textured images with $\mu > 4.0$ are involved in the stereo evaluation [4]. In our dataset images containing numerous weakly-textured regions like *Lampshade* ($\mu = 2.86$) or *Plastic* ($\mu = 2.2$) are considered as well.

There are two ways how to evaluate the performance of various stereo methods on the current dataset. For images supplied with ground truth data (all pairs taken from the Middlebury dataset) the accuracy of estimated disparity maps can be evaluated with respect to the ground truth map. The root-mean-squared (RMS) error based on known ground truth data is used in this work as a quality measure for these images (Scharstein and Szeliski, 2002):

$$R = \left(\frac{1}{N} \sum_{(x,y)} |d_C(x,y) - d_T(x,y)|^2 \right)^{\frac{1}{2}}, \tag{5.2}$$

where N is the total number of pixels having estimates, d_C and d_T are the computed and ground truth disparity maps, respectively. Amount of estimated pixels in percentage is computed to evaluate the density of a computed disparity map.

In the case of stereo pairs without ground truth data (the rest of the dataset) the synthetic images obtained by warping the left images by the computed disparity map need to be evaluated. It is done by the RMS warping error that compares a new

[3] Here we use the *entropyfilt* utility from the Matlab toolbox for the texture analysis (Gonzalez et al., 2003).

[4] available under http://vision.middlebury.edu/stereo/eval/

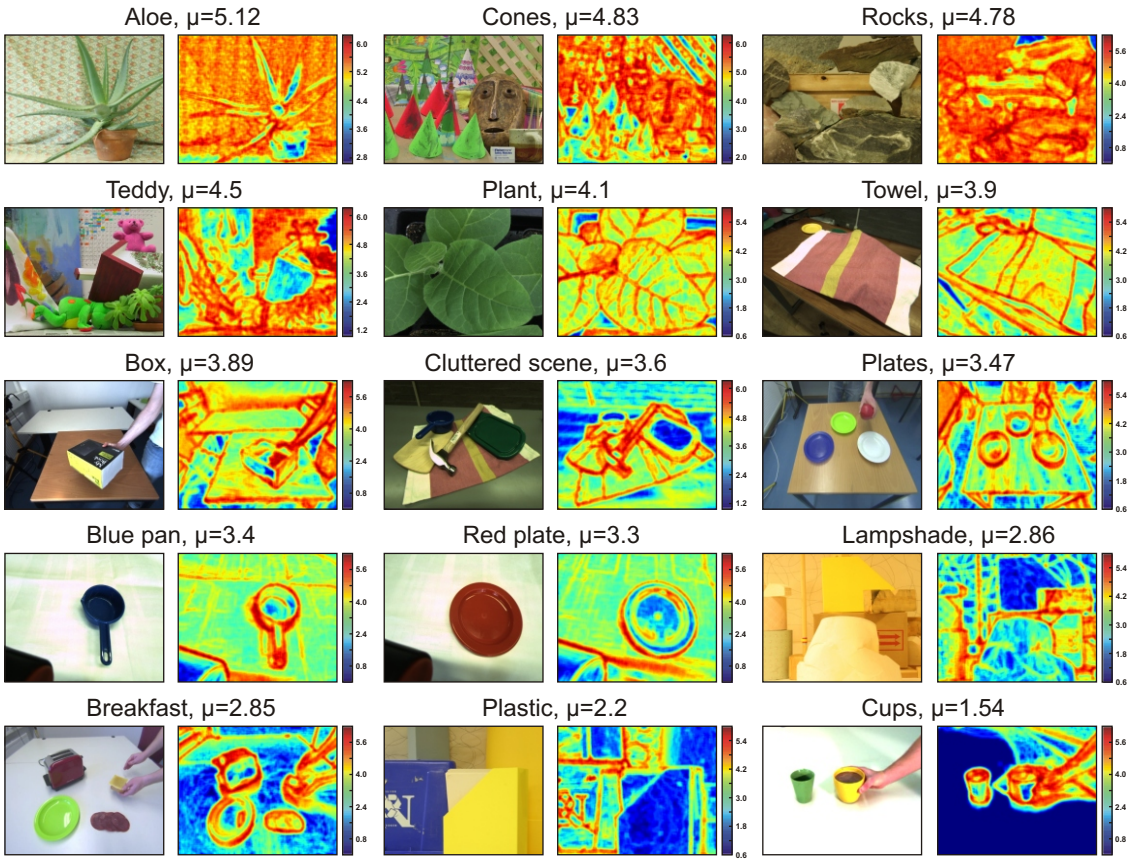

Figure 5.1: Texture quantification for several sample images from the proposed testing stereo dataset with related entropy values. For convenience only original left images are shown here.

rendered image with an original right image according to

$$R_w = \left(\frac{1}{N} \sum_{(x,y)} |I(x,y) - I'(x,y)|^2 \right)^{\frac{1}{2}}, \tag{5.3}$$

where I and I' are original and synthetic images, respectively. Here the percentage of rendered pixels is used as an evaluation of disparity density.

Fig. 5.2 shows the performance of the chosen ten conventional stereo algorithms on the proposed dataset as a function of image entropy. Evaluations obtained for the Middlebury images are shown separately from other samples due to differences in the quality measures of computed disparity maps. RMS errors based on known ground truth data and rates of found matchings for the Middlebury images are shown in Fig. 5.2(A,B). RMS warping errors and rates of rendered pixels for stereo pairs

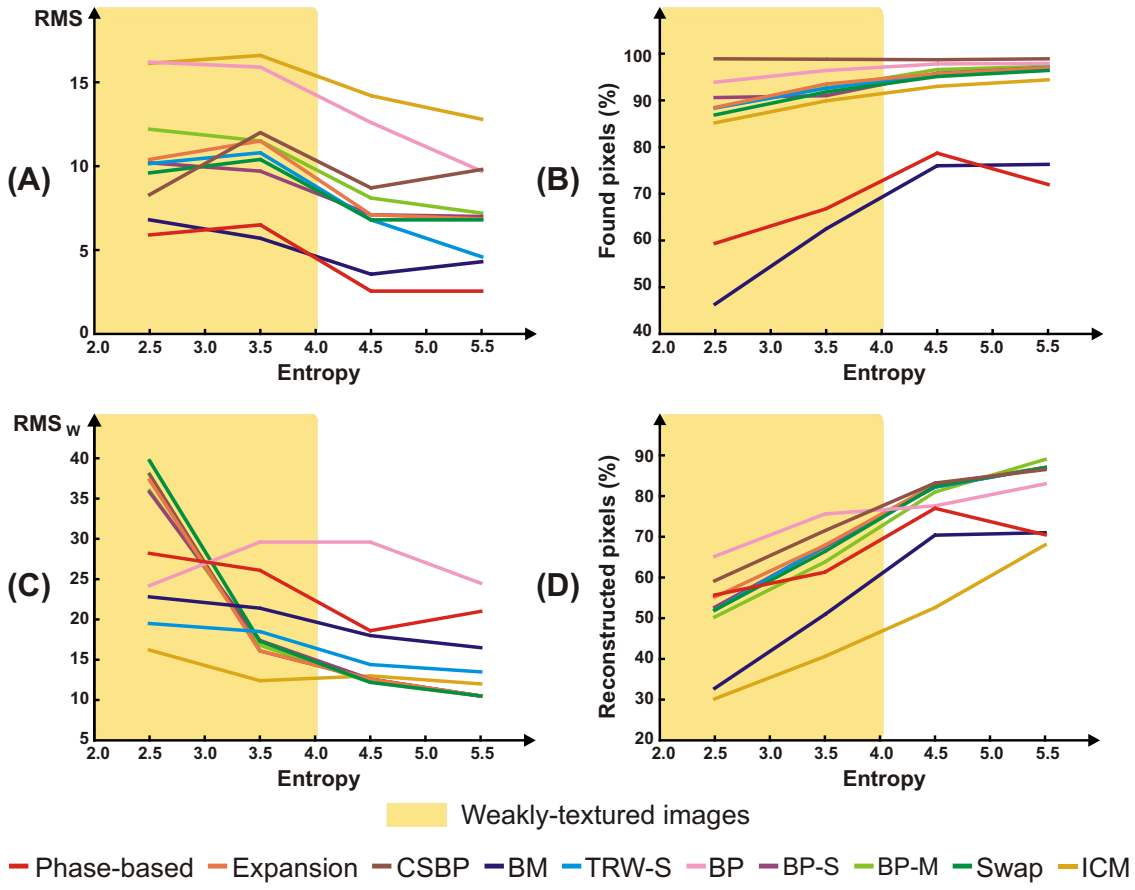

Figure 5.2: Performance of the chosen conventional stereo algorithms as a function of image entropy. (A) RMS error based on known ground truth data for images from the Middlebury stereo dataset. (B) Percentage of pixels with found stereo matchings for images from the Middlebury stereo dataset. (C) RMS warping error for stereo pairs without ground truth data. (D) Percentage of rendered pixels for stereo pairs without ground truth data.

without ground truth data are shown in Fig. 5.2(C,D). Performance of all conventional methods and density of computed disparity maps tend to increase with increase of textureness. Some techniques, e.g., phase-based and BM for the Middlebury images, provide quite accurate disparity estimations but having very sparse maps. Therefore, we need to take both the accuracy and density into account for the performance evaluation of stereo methods. Also you can note that performance of some methods like ICM, CSBP and BM is dissimilar on two groups of images. It can be explained by a nature of texture in both image groups. So texture in the most Middlebury images is caused by many tiny objects or differently looking object parts (see images *Aloe*, *Cones*, *Teddy* in Fig. 5.1) that make the life of stereo methods easier. Texture in other images is caused mostly by changes in lighting (see images *Plant*, *Towel*, *Blue pan* in Fig. 5.1) which leads to more confusions in the process of stereo matching. In the meanwhile we can see that techniques such as Expansion, Swap, TRW-S, BP-M and BP-S seem to be quite robust for all kinds of images.

Decrease in performance with decrease of the image entropy is shown for eight conventional stereo methods in Fig. 5.3. All methods have hard times estimating disparity in poorly-textured areas like in *Cluttered scene* ($\mu = 3.6$) and *Plastic* ($\mu = 2.2$) images. In extremely textureless regions as the green cover in *Cluttered scene* and the yellow box in *Plastic* estimations either wrong or not available. However, even very sparse disparity maps given by the phase-based technique contain estimations uniformly distributed over the image, while estimations made by other methods have large regions either without information at all or filled with considerably inaccurate data (see the *Plastic* image in Fig. 5.3). The phase-based algorithm shows some drastic differences to other methods. Firstly, the employed coarse-to-fine search gives the algorithm the ability to use low frequency structures for matching (when operating on low resolution scales the algorithm is using much larger windows than the other techniques). Secondly, it uses phase as opposed to intensity which is entirely independent of contrast differences between the images. The phase is also sensitive to small intensity changes within the image, so very faint structures can also be exploited (Pauwels et al., 2011).

Now we are ready to involve the classification of images based on amount of texture in relation to performance of conventional stereo methods. Images having the entropy value $\mu > 4.0$ are treated pretty well by all methods and for such images disparity maps with a density about $70 - 80\%$ can be obtained. Images with the entropy value $\mu \leqslant 4.0$ are tough and only a few methods can provide reliable disparity data for some of their parts. As a result, obtained disparity maps are extremely sparse having a density lower than 50%. Furthermore, some of their estimations can be very inaccurate. It is eventually the reason why images having $\mu \leqslant 4.0$ are not used as testing data in the most stereo approaches.

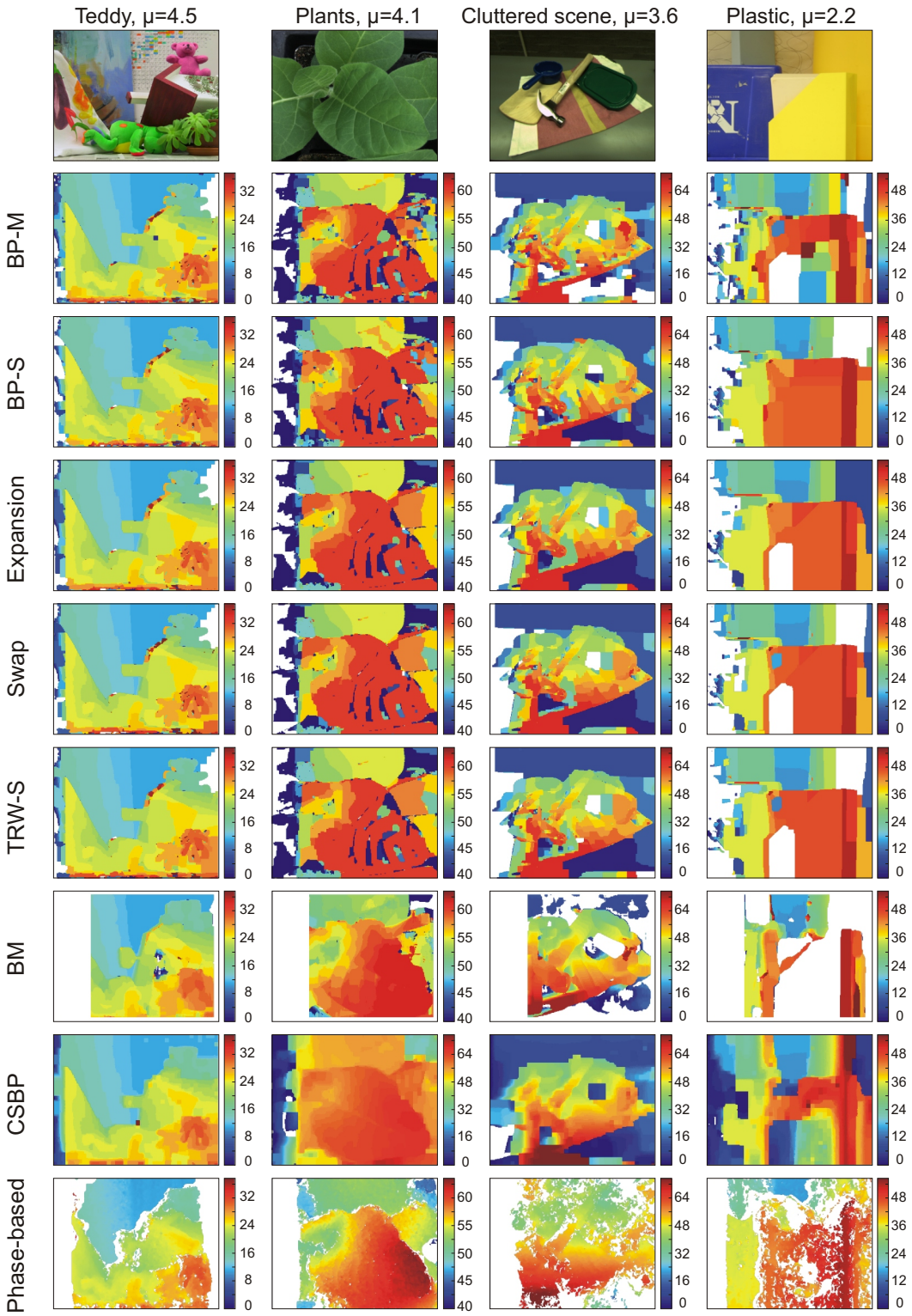

Figure 5.3: Decrease in performance with decrease of image entropy for the most efficient stereo matching methods shown on some images from the testing stereo dataset.

5.3 Dense disparity from stereo-segment silhouettes

The proposed method is based on the assumption that regional correspondences between stereo images provide additional and robust information about 3D structure of the visual scene. This information can be used together with a sparse disparity derived by any conventional stereo technique to obtain dense stereo information in all image regions including weakly-textured ones. The block diagram of the method is presented in Fig. 5.4.

The method takes a rectified stereo image as input. Disparity information is estimated in relation to the left image, i.e., the left image is *the reference image* and the right image is *the matching image*. Two disparity maps with different level of sparsity η are obtained first for the input stereo pair by the phase-based stereo algorithm (see Section 4.2.1 and step 1 in Fig. 5.4). This method runs in real-time and provides quite reliable estimations even in textureless regions as compared to other stereo techniques (see Fig. 5.3). The stereo pair is decomposed then into corresponding regions called *stereo segments* (see step 2 in Fig. 5.4). Once stereo segments are extracted and sparse disparity information is pre-computed, a map combining sparse disparity with additional information from matched stereo regions can be built. We will call it *weighted initial disparity map*. It includes sparse estimations from the map with the lower sparsity ($\eta = 0.4$) and disparity values estimated for segment edges. Disparity values from the sparse map that are close to the segment boundaries or located in potential half-occlusions are excluded from the map. Potential half-occlusions are detected by the use of the average line disparity computed for bunches of lines within each segment. The average line disparity is also used for computation of edge disparity mask used for exclusion of "imaginary edge disparity values" (see step 3 in Fig. 5.4). Once the edge disparity map is ready (see step 4 in Fig. 5.4), the weighted initial disparity map is created (see step 5 in Fig. 5.4). Due to the assumption that edge values are more reliable than values from the binocular disparity, edge values receive larger weights in the initial disparity map. On the last step the built weighted initial disparity is used for the surface fitting and disparity recovery in each segment (see step 6 in Fig. 5.4). In the next sections all steps are described in more detail.

5.3.1 Co-segmentation of stereo pairs

Since in the current study we are interested in the estimation of dense disparity for every single object or its parts, input reference and matching images need to be divided first into homogeneous regions, i.e., segments. Thereby a region-based segmentation technique is needed. Furthermore, in order to extract an additional information from segmented images useful for stereo, correspondences between found left and right segments need to be established. To obtain correspondent left and right segments

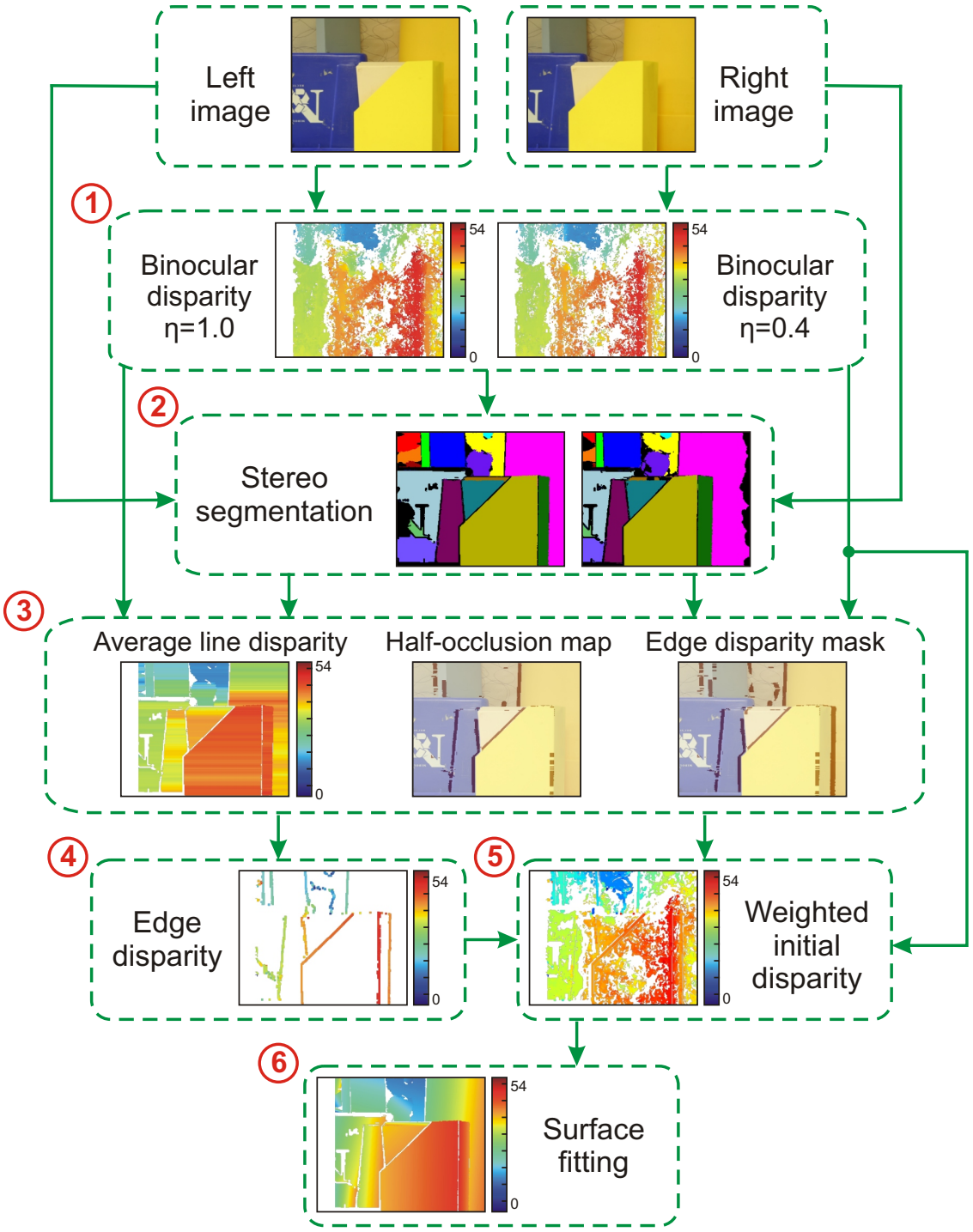

Figure 5.4: The proposed real-time dense stereo algorithm for weakly-textured images. Numbers at arrows show the sequence of computations.

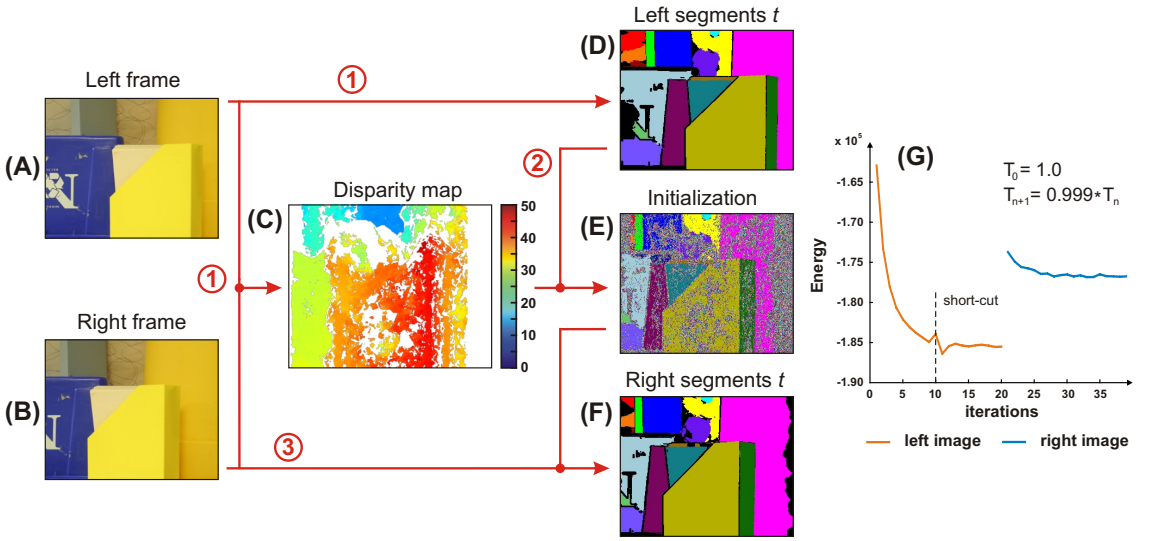

Figure 5.5: Segmentation of a stereo pair. Numbers at arrows show the sequence of computations. (A) Original left or reference image. (B) Original right or matching image. (C) Disparity map estimated by the phase-based stereo technique. (D) Extracted segments for the left image after $n_1 = 10$ and $n_2 = 10$ Metropolis iterations. (E) Label transfer from the left image to the right image. (F) Extracted segments for the right image after $n_2 = 15$ Metropolis iterations. (G) Convergence of the Metropolis algorithm for both the left and right images.

we use the stereo segmentation technique based on the superparamagnetic clustering of data performed by the parallel Metropolis on the GPU introduced in Chapter 2 and used for the segmentation of stereo video streams in Chapter 4. As was already shown in the previous chapters, this method performs the real-time segmentation of middle-size images without prior knowledge on the data (model-free).

The segmentation procedure for weakly-textured *Plastic* stereo pair ($\mu = 2.2$) from the Middlebury dataset is shown in Fig. 5.5. First the left image is segmented completely by the parallel Metropolis algorithm with the short-cut (see Section 2.2.6) and segments larger than a pre-defined threshold are extracted (see Fig. 5.5(A,D)). In the meanwhile a sparse disparity map is obtained for the rectified stereo pair by the phase-based stereo algorithm (see Fig. 5.5(A - C)). Since the current stereo pair is poorly-textured, a consistency check in the stereo method runs with a pretty high threshold ($\eta = 1.0$) giving so many disparity values as possible (see step 1 of Fig. 5.4). It makes an estimated map almost dense containing estimations up to 85% of pixels in unoccluded regions, i.e., those visible in the both left and right images. Pixels that did not obtain a label initialization are given a label which is not occupied by any of the found segments (see Fig. 5.5(E)). Once the right image is initialized, it

needs to be adjusted to the current image data by the image segmentation core (see Section 3.2). This adjustment is needed in order to fix erroneous bonds that can occur during the warping of spins. The relaxation process runs until it converges and only after that the final right segments consistent with left ones can be extracted (see Fig. 5.5(F)). Since in the current chapter we are interested in stereo properties of segments, segments which do not have correspondences in one of two images are eliminated as well. Convergence of the relaxation process for the left and right images as a function of the iteration number is shown in Fig. 5.5(G).

For segmentation of the reference image $n_1 = 10$ and $n_2 = 10$ iterations are required, while $n_2 = 15$ iterations are enough to reach the equilibrium state for the matching image after the label warping. Therefore, 35 Metropolis iterations are needed in total for segmentation of one stereo pair. Note that less iterations are needed here as compared to the segmentation of stereo video streams (see Section 4.2.2). The reason is that we deal here with only one single stereo pair without motion. Thus, only spatial synchronization between left and right frames must be reached and there is no temporal synchronization. For the relaxation process we use an on-line adaptive simulated annealing (see Section 2.2.3) with the same schedule as for the segmentation of monocular and stereo video streams presented in the previous chapters: the starting temperature $T_0 = 1.0$ and the simulated annealing factor $\gamma = 0.999$. Note that longer annealing schedules can lead to better segmentation results (for example on the border of the matching image where no label transfers are possible due to the lack of information in the disparity map) but at the cost of processing time.

5.3.2 Average line disparity

Detection of potential half-occlusions (see Section 5.3.3) and exclusion of imaginary segment boundaries (see Section 5.3.4) are done according to the pre-computed average line disparity map d_A (see step 3 in Fig. 5.4). d_A is computed for bunches of lines within each segment based on the sparse disparity map with $\eta = 0.4$. The mean value of sparse disparity values for a group of lines belonging to the segment S is computed as

$$\overline{d} = \frac{1}{N_S} \sum_{d_i \in d_\eta} \delta_{S_i,S} \cdot d_i \tag{5.4}$$

where N_S is a number of pixels of the segment S in the current group of lines having values in the sparse disparity map $d_{\eta=0.4}$, and $\delta_{S_i,S}$ is the Kronecker delta producing 1 only if the current pixel i belongs to the segment S, i.e., $S_i = S$, and 0 otherwise. The value $\overline{d}$ is assigned eventually to all pixels of the segment S within the current group of lines. The average line disparity d_A obtained for the sample stereo pair *Plastic* is shown in Fig. 5.6(A - C).

Note that the computation of the average disparity based on windows sliding

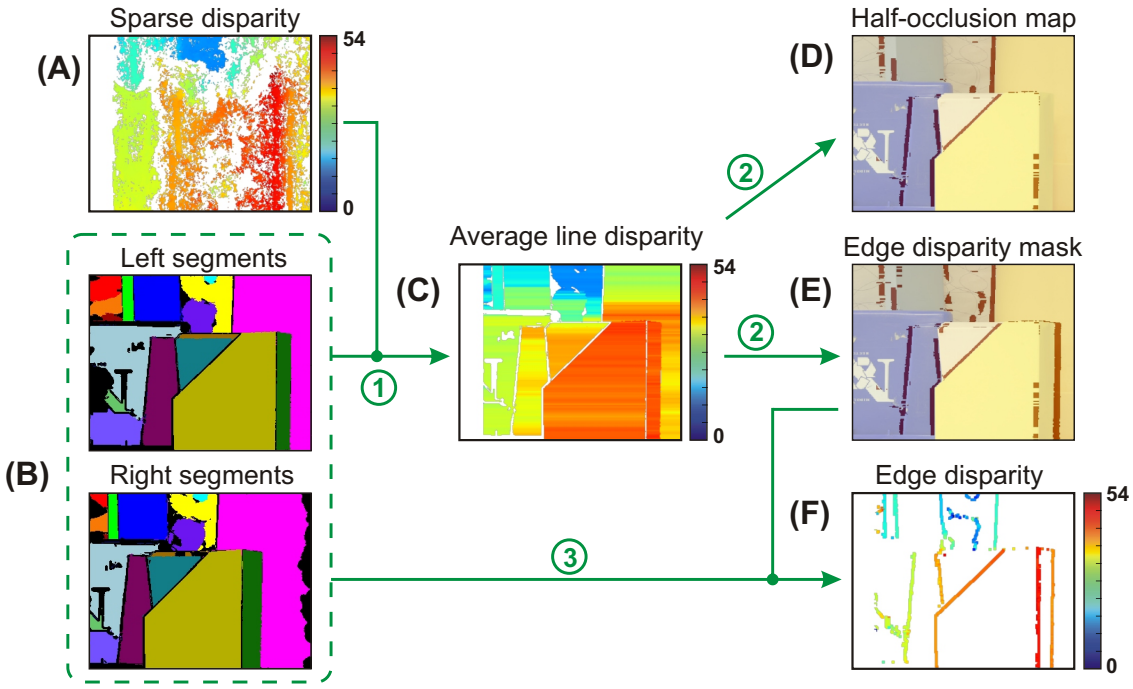

Figure 5.6: Computation of the average line and segment-edge disparities with the occlusion map and the edge disparity mask for the *Plastic* stereo pair. Numbers at arrows show the sequence of computations. (A) Sparse disparity map $d_{\eta=0.4}$ estimated by the phase-based stereo algorithm. (B) Spatially coherent left and right segments. (C) Average line disparity map d_A derived based on the found stereo segments and sparse disparity. (D) Approximate half-occlusion map. (E) Approximate edge disparity mask. (F) Edge disparity map.

within each segment is naturally more correct, but working with sparse disparities obtained for weakly-textured regions it is not really the case, since inaccuracies within a window can be very dramatic leading to wrong reasonings about half-occlusions and edges. You can see how inaccurate estimations could be in textureless areas in Fig. 5.3 on sample images *Cluttered scene* and *Plastic*. Although estimations provided by the phase-based technique are quite reliable, values at boundaries between poorly-textured objects are very erroneous (see edges of the yellow box in *Plastic* image in Fig. 5.6(A)).

5.3.3 Detecting half-occlusions

Due to the stereo vision geometry certain scene points are visible only in one view of the stereo pair. As a result, no stereo correspondences can be found for these points. Areas formed by such points are called *half-occlusions*, the image of half-

occlusions is the *half-occlusion map*. Half-occlusions usually occur around object edges and other scene discontinuities. These points are of high importance for the stereo vision, since they can aid in the matching process. There are five major approaches for half-occlusion reasoning: Bimodality (Wildes, 1991), Match Goodness Jumps (Anderson and Nakayama, 1994), Left-Right Checking (Trapp et al., 1998), Ordering (Belhumeur, 1996), and Occlusion constraint (Geiger et al., 1995). However, none of these approaches is superior and each technique has its pros and cons (Egnal and Wildes, 2002).

The purpose of the half-occlusion map in this work is to determine pixels whose disparity values taken from the sparse map should not appear on the final surface fitting step (see step 3 in Fig. 5.4). The half-occlusion map marks these pixels and removes them from the weighted initial disparity map. For detection of half-occlusions we use the ordering constraints approach operating on the level of segments. Since the left image is considered in this work as the reference image, half-occlusions can occur only at left object borders if an object on the left side from the border is further away, i.e., has lower disparity values, than an object on the right side from the border. Half-occlusions for a synthetic stereo pair containing some objects located on the table are shown in Fig. 5.7(A,B). Pixels belonging to the half-occlusions are marked by red there. These pixels are not visible in the correspondent right image, thereby no stereo matchings can be obtained for them.

Suppose a stereo pair is segmented (see Fig. 5.6(B)) and an average line disparity is computed (see Fig. 5.6(C)). Similar to the computation of the average line disparity map, we compute a half-occlusion map considering groups of lines. For example, in the case of two neighboring segments S_i and S_j (see Fig. 5.7(C)), where the segment S_j is located on the left side from their mutual border, pixels, potentially occluded by the segment S_i in the right image, can be determined as follows. For all pixels $k \in S_j$ such that $k \in N_l$, where N_l is the close neighborhood along the x-axis to the left of pixel i (see Fig. 5.7(C)), an occlusion mask O is defined according to

$$O_k = \theta(\overline{d}_{S_i} - \overline{d}_{S_j} - \tau) \tag{5.5}$$

where θ is a step function defined as

$$\theta(n) = \begin{cases} 1 & \text{if } n > 0, \\ 0 & \text{if } n \leqslant 0. \end{cases} . \tag{5.6}$$

A number of neighboring pixels in N_l considered as half-occlusions is a system parameter and needs to be tuned according to the currently used stereo setup. $\overline{d}_{S_i}$ and $\overline{d}_{S_j}$ are average line disparity values of segments S_i and S_j, respectively, computed according to (5.4), τ is a threshold for the minimum disparity difference between two segments. In most cases $\tau = 5$ provides quite accurate results (see Fig. 5.6(D)). However, even employment of τ cannot avoid some inaccuracies in the average line disparity causing erroneous half-occlusions (see the junction of two sides of the yellow

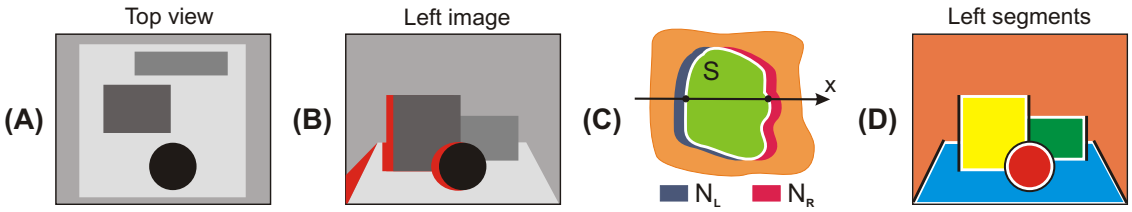

Figure 5.7: Registration of half-occlusions and imaginary object edges shown on a synthetic image. (A) Top view of the scene. (B) Left image of a stereo pair with marked half-occlusions. (C) Detection of half-occlusions and imaginary segment edges caused by the segment S_i. (D) Found left segments where imaginary object edges are shown in black.

box). But it is not a big problem for the method, since disparity values excluded due to occasionally detected half-occlusions will be recovered on the surface fitting stage. It is much more important to eliminate disparity values from the real half-occlusions.

5.3.4 Edge disparity

Stereo-segment silhouettes correspondences provide important additional information about depth. Given two corresponding segments, disparities of segment-silhouette points can be found as long as the pixel does not belong to a segment boundary oriented parallel to x-axis, i.e., the scanline. A pixel i having a segment label S_i and position x_i is considered to be a left segment boundary pixel i_l if there is no pixel j that $x_j < x_i \wedge S_j = S_i$. A pixel i with segment label S_i is considered to be a right segment boundary pixel i_r if there is no pixel j that $x_j > x_i \wedge S_j = S_i$.

For each scanline and for each segment we find the boundary pixels i_l and i_r in the left image and the boundary pixels j_l and j_r in the right image. The left and right edge disparity values are then defined as

$$d_E^l = i_l - j_l,\ d_E^r = i_r - j_r, \tag{5.7}$$

and merged into the edge disparity map d_E. However, it can contain information about so called *imaginary edges* which are not necessarily object borders and need to be excluded from the edge disparity map. What is meant by imaginary edges we will explain again on the synthetic images shown in Fig. 5.7. Here the original left frame (see Fig. 5.7(B)) is segmented (see Fig. 5.7(D)) and every object is represented by a segment. In the image segmentation result each border between two regions consists of two edges: one from each segment. But if two objects are not close to each other, only edge of the segment in front corresponds obviously to the real object border, while we cannot say anything about the edge of the segment behind (see for instance the border between the red and green objects). In Fig. 5.7(D) all imaginary borders

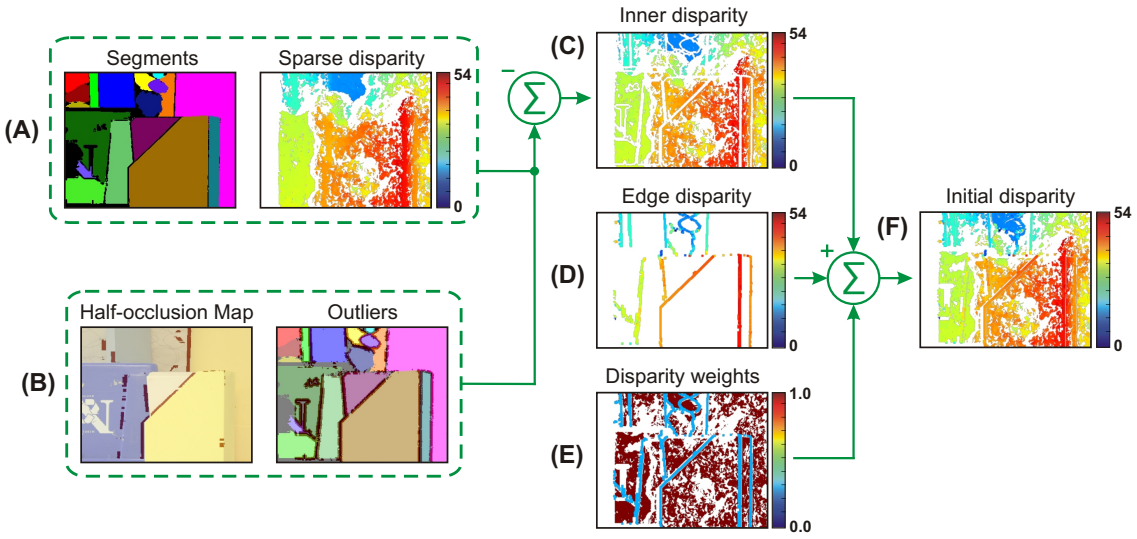

Figure 5.8: Formation of the weighted initial disparity map for the *Plastic* stereo pair. (A) Extracted left segments and sparse disparity map ($\eta = 0.4$) estimated for the left image. (B) Half-occlusion map and marked disparity outliers. (C) Inner disparity map. (D) Edge disparity map. (E) Measurement errors associated with edge and inner disparities. (F) Initial disparity map used for the surface fitting.

are marked by black. Image areas that can potentially contain imaginary edges are determined similar to the half-occlusions based on the average line disparity with the difference that imaginary edges can occur at both left and right object borders. Therefore, the close neighborhood N_r along the x-axis to the right of border pixels needs to be considered as well (see Fig. 5.7(D)). The edge disparity map d_E computed for the *Plastic* stereo pair is shown in Fig. 5.6(F).

5.3.5 Initial disparity

The initial disparity map combines the results obtained from all previous computational steps and can be used for the surface fitting (see step 5 in Fig. 5.4). The whole procedure is shown in Fig. 5.8 for the *Plastic* stereo pair. A disparity map containing values taken from the sparse disparity map ($\eta = 0.4$) for each segment avoiding half-occlusions and outliers we call the *inner disparity map* (see Fig. 5.8(A-C)). As outliers we consider disparity values that are too close to segment boundaries. Note that the inner and edge disparity maps are created independently of each other.

Finally the inner and edge disparity maps are combined into the weighted initial disparity map. Due to the assumption that edge disparity values are more reliable than inner disparity, edge values are featured in the initial disparity map by lower measurement error as compared to inner values (see Fig. 5.8(C-E)). The measurement

errors for edge disparity and inner disparity values are set to the values $\vartheta = 0.3$ and $\vartheta = 1.0$, respectively.

5.3.6 Surface fitting

The created weighted initial disparity map is a sparse set of N data points (x_i, y_i, z_i), $i = 1, \ldots, N$ (see Fig. 5.8(F)). The goal of the surface fitting (see step 6 in Fig. 5.4) is to fit a set of data points to a pre-defined surface model. Fitting of range data to surface models is a well-known approach used for various tasks in computer vision (Besl and Jain, 1988; Bab-Hadiashar and Gheissari, 2006; Dellen et al., 2011). In this work we fit a sparse set of data points within each segment to a surface model. The resulting fitted model is used to compute unavailable disparity values and to fix erroneous estimations from the inner disparity map.

Surface types

In the current work we use two types of surfaces as surface models: planes and a general quadratic function which describes many kinds of curved surfaces including cylinders, cones, and spheres. Planar surfaces are described by three parameters a_1, a_2, and a_3, where the disparity z can be expressed as a function of x and y through:

$$z = a_1 x + a_2 y + a_3. \tag{5.8}$$

Curved surfaces are described by five parameters a_1, a_2, a_3, a_4, and a_5, where the disparity z can be expressed as a function of x and y through:

$$z = a_1 x^2 + a_2 y^2 + a_3 x + a_4 y + a_5. \tag{5.9}$$

The general form of these surface models is a polynomial

$$z(x, y) = \sum_{k=1}^{M} a_k \cdot \varphi_k(x, y), \tag{5.10}$$

where $\varphi_1(x, y), \ldots, \varphi_M(x, y)$ are arbitrary fixed functions of x and y, called the *basis functions*. To solve the surface fitting problem parameters a_k for which the model function $z(x, y)$ looks like the data need to be determined. Once model parameters for each segment are known, disparity values can be easily computed for all segment pixels. To determine which parameter vectors give the best fit to the data for both surface models we use the linear least squares fitting and fitting based on the Nelder-Mead simplex algorithm due to their high time performance as compared to other optimization techniques.

Linear least squares

The basis functions $\varphi_k(x, y)$ in (5.10) can be nonlinear functions of x and y, while the dependence of the model on its parameters a_k is linear. Therefore, "linear" refers only to the model's dependence on its parameters a_k. The model parameters can be found by the minimization of the merit function

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{z_i(x_i, y_i) - \sum_{k=1}^{M} a_k \cdot \varphi_k(x_i, y_i)}{\vartheta_i} \right]^2, \tag{5.11}$$

where N is the number of measurements, z_i are disparity estimations taken from the weighted initial disparity map, and ϑ_i is the measurement error of the i-th data point, supposed to be known (see Fig. 5.8(E)). Parameters that minimize χ^2 are picked as the best model parameters. There are several approaches for finding this minimum. In the current work we use the normal equations technique due to its high time performance. The solution by use of the normal equations is given in Appendix A.2.

Nelder-Mead

The Nelder-Mead method or downhill simplex method minimizes an objective function (see (5.10)) in a multidimensional search space without the need for calculating derivatives. The method requires only function evaluations which makes it quite fast. The method uses a multidimensional shape called a simplex. At every iteration all simplex vertices are ordered according to the values at each test point:

$$z(x_1, y_1) \leqslant z(x_2, y_2) \leqslant \ldots \leqslant z(x_{N+1}, y_{N+1}). \tag{5.12}$$

With each iteration the simplex moves through the search space in a predefined manner and replaces its worst vertex with a vertex which is better than any of its other vertices. The new point is found using a set of pre-defined steps (see Appendix A.3). After some iterations the method is guaranteed to find a minimum. The problem of the Nelder-Mead method as compared to other optimization techniques is that it usually finds a local minimum instead of the global minimum. It can be overcome by choosing the correct size for the starting simplex so that local minima are skipped.

5.4 Experimental results

5.4.1 Proposed method

Disparity maps, estimated using the proposed stereo algorithm for some weakly-textured images ($\mu \leqslant 4.0$) from the testing stereo dataset are shown in Fig. 5.9. The results have been obtained for the planar and curved surface models using both

the linear least squares (LSQ) and Nelder-Mead simplex algorithm (NMD) for the surface fitting. For each output the RMS error value (for stereo pairs having ground truth data) or RMS warping error value (for stereo pairs without ground truth) is given with the percentage of found matchings. None of the four versions (LSQ with the planar surface model (*LSQ planar*), LSQ with the curved surface model (*LSQ quadric*), NMD with the planar surface model (*NMD planar*), and NMD with the curved surface model (*NMD quadric*)) is superior and the quality of the final disparity map depends to a high degree on the concrete scene, i.e., objects and their shapes.

LSQ planar produces the best results for images *Breakfast* ($\mu = 2.3$), *Lampshade* ($\mu = 2.9$), and *Box* ($\mu = 3.9$), *LSQ quadric* has the lowest error values for images *Bowling* ($\mu = 3.9$) and *Table* ($\mu = 3.9$), *NMD planar* has the highest score on image *Plastic* ($\mu = 2.2$), while *NMD quadric* performs best on image *Baby* ($\mu = 3.9$). The surface fitting using the planar model gives better results for images featured by a large number of plane surfaces like *Breakfast*, *Lampshade*, *Box*, and *Plastic*. Quadric surfaces can be recovered with a curved surface model (see the big white pillow in *Lampshade* image, the book in *Baby* image, the green ball and the white bowling pin in *Bowling* image). Furthermore, the quadric model allows deriving the curvature (see the green plate in *Breakfast* image). However, in some cases fitting of curved surfaces can lead to dramatic inaccuracies in final disparity maps (see for example the background in *Plastic* image given by *LSQ quadric*, the yellow box in *Plastic* image given by *NMD quadric*, the background in *Lampshade* image given by *NMD quadric*, and the white plate in *Table* image given by *LSQ quadric*), because more surface parameters need to be computed as compared to the planar case. The inaccuracies in the final disparity maps are largely caused by objects with extremely low texture (see the backgrounds in *Lampshade* and *Box* images and the floor and the background in *Table* image). The problem is that estimations in the sparse disparity map $d_{\eta=0.4}$ propagate out from more textured objects and their borders in front to less textured objects in the background and even removal of outliers (see Section 5.3.5) in the initial disparity map cannot prevent that.

The number of found pixels is about the same for all modes, since it depends on the found segments and is independent of the minimization technique or the chosen surface model. The more pixels in the image are covered by segments, the more dense the final disparity map is. Also note that for some pixels at the left image border no estimations can be made, since the left image is the reference image and those pixels are not visible in the right image, i.e., the matching image, due to the stereo shift. Stereo pairs in the used database have different baselines which results in various stereo shifts. For example shifts in the Middlebury stereo pairs are larger than in the rest of the dataset.

The quantitative comparison of four versions of the proposed stereo algorithm is presented in Fig. 5.10. Again, evaluations obtained for stereo pairs from the Middlebury dataset are shown separately from the rest because of differences in the quality measures of computed disparity maps (see Section 5.2.2). We can see that the al-

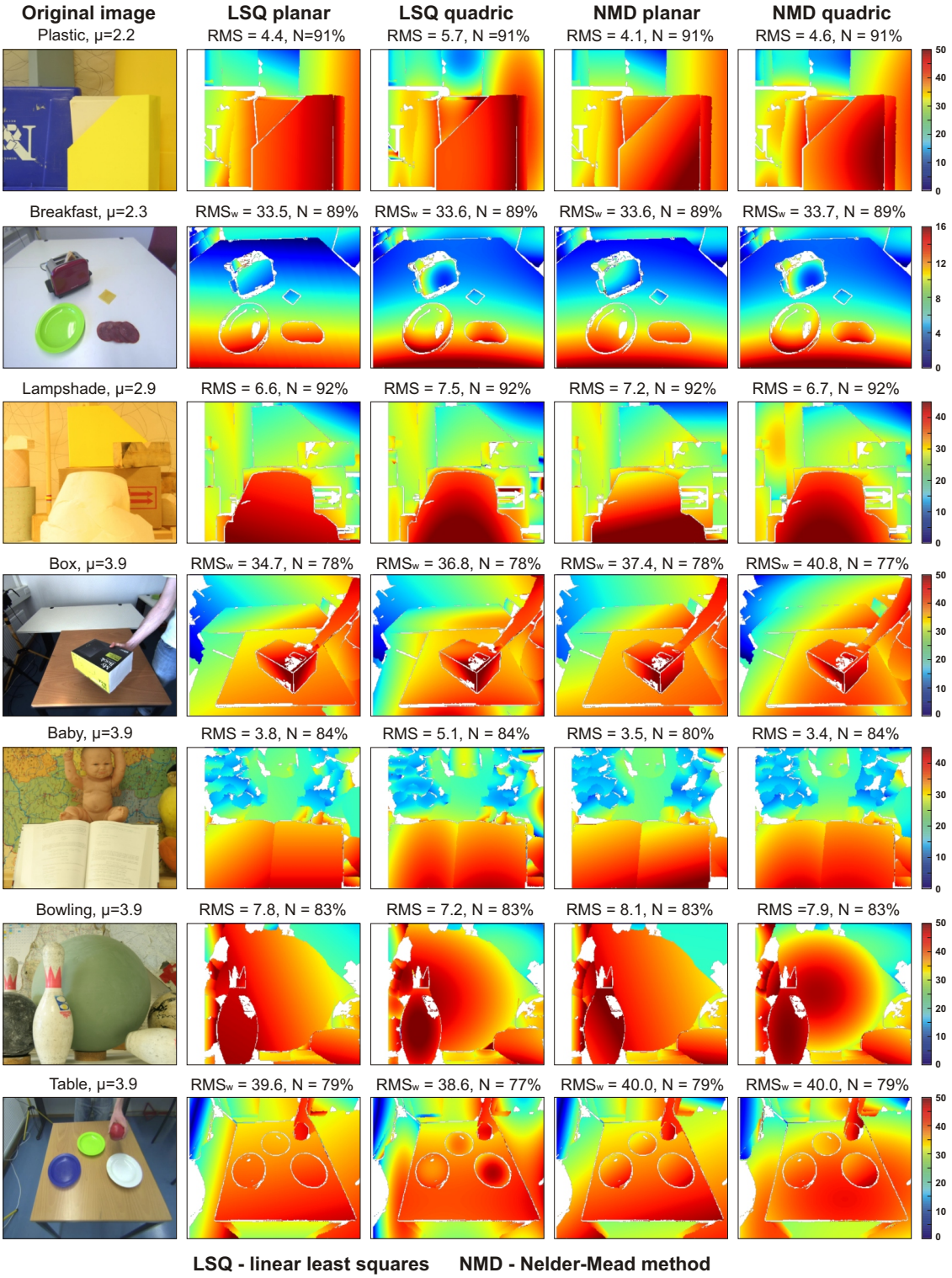

Figure 5.9: Results of the proposed algorithm for weakly-textured stereo pairs.

gorithm performances on the Middlebury images and on the rest of the dataset are different. On the Middlebury images *NMD planar* performs best (see Fig. 5.10(A)), while on other images *LSQ quadric* and *NMD quadric* have the highest scores (see Fig. 5.10(C)). The reason is that the other images contain more curved objects than the Middlebury dataset. Hence, modes using the curved surface model produce more precise outputs. Also for the considered two classes of images we observe different behaviors on images with sufficient texture ($\mu > 4.5$). For stereo pairs from the Middlebury dataset the RMS error values decrease, whereas the RMS warping error values for the rest of the dataset increase. It happens because of various texture natures in both image groups. As was already mentioned in Section 5.2.2, the texture in the most Middlebury images is caused by many tiny objects or differently looking object parts which leads to more precise estimations in the sparse disparity map $d_{\eta=0.4}$. In the rest of the dataset texture is caused mostly by changes in lighting which leads to high entropy values but does not give more clues for estimation of sparse disparity.

5.4.2 Comparison to conventional techniques

We compare the proposed stereo algorithm with the conventional stereo approaches from Section 5.2.2 in Fig. 5.11. For both the Middlebury images and the rest of the dataset only the performance of the best mode is shown: *NMD planar* for the former and *LSQ quadric* for the latter. For the weakly-textured Middlebury images ($\mu \leqslant 4.0$) the proposed algorithm finds matchings for about 85% of pixels having the RMS value significantly lower than other techniques (see Fig. 5.11(A,B)). Even though the phase-based and BM techniques are featured by quite low RMS error values, their disparity maps are much sparser producing less than 65% of matchings for images with $\mu < 3.0$ and less than 75% of matchings for images with $\mu \leqslant 4.0$. For weakly-textured images from the rest of the dataset a number of techniques produce more precise disparities than the proposed approach, but all of them contain less than 65% of matchings for images with $\mu < 3.0$ and less than 75% of matchings for images with $\mu \leqslant 4.0$. The proposed approach, on the contrary, produces about 85% of matchings for images with $\mu < 3.0$ and about 75% for images with $\mu \leqslant 4.0$.

Disparity maps estimated by the BM, Swap, and phase-based techniques, having on the Middlebury stereo pairs the highest scores among all the considered conventional methods (see Fig. 5.11(A)), are compared to the results of the proposed method in Fig. 5.12 for *Plastic* ($\mu = 2.2$), *Lampshade* ($\mu = 2.9$), *Baby* ($\mu = 3.9$), and *Bowling* ($\mu = 3.9$). For each output the RMS error value is given together with the percentage of found matchings. Although disparity maps given by the BM technique are extremely sparse, the available estimations are quite precise. So the BM has the highest score for the image *Bowling*. The maps produced by the Swap algorithm are featured by a very high density (giving even about 98% of matchings for images *Lampshade*, *Baby*, and *Bowling*), but they have the highest RMS error values among all four techniques. The outputs of the phase-based algorithm are quite sparse but

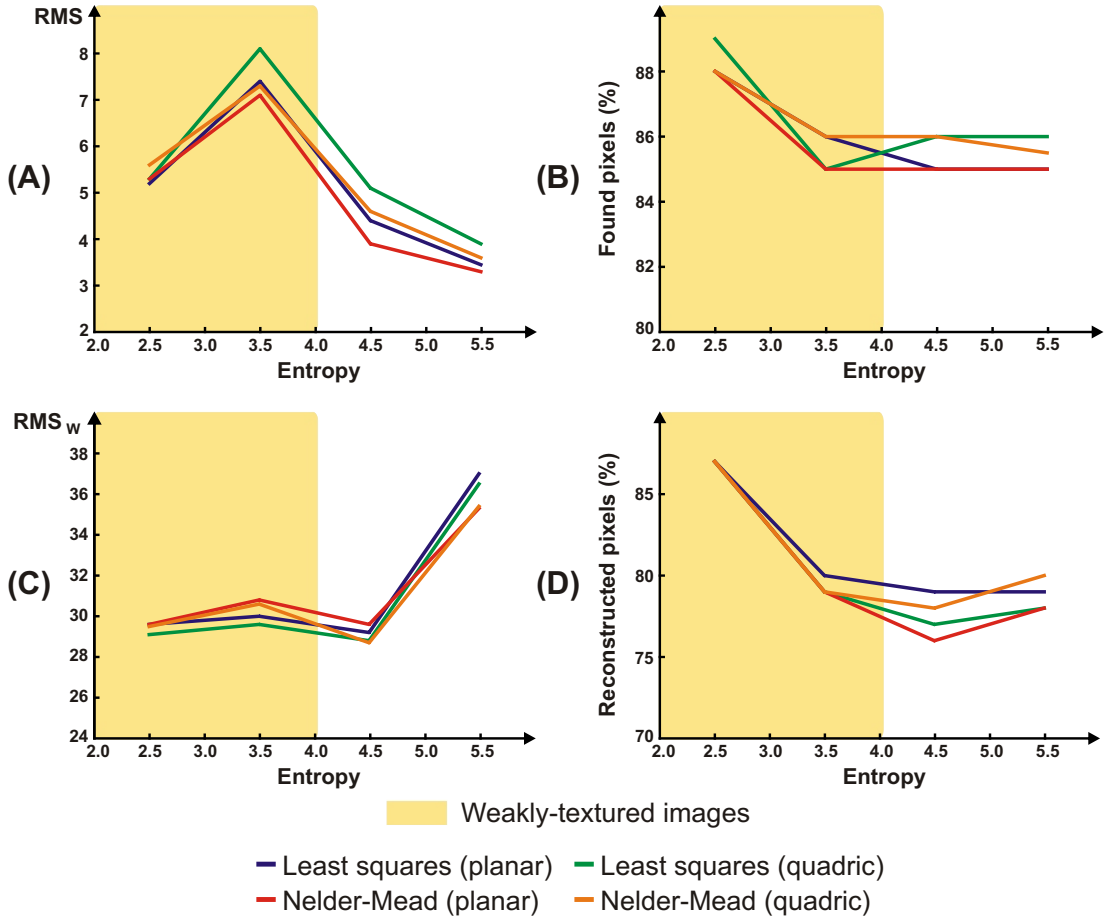

Figure 5.10: Performance of the proposed stereo algorithm as a function of image entropy shown for planar and curved surfaces using the linear least squares and Nelder-Mead optimization techniques. (A) RMS error based on known ground truth data for images from the Middlebury stereo dataset. (B) Percentage of pixels with found stereo matchings for images from the Middlebury stereo dataset. (C) RMS warping error for stereo pairs without ground truth data. (D) Percentage of rendered pixels for stereo pairs without ground truth data.

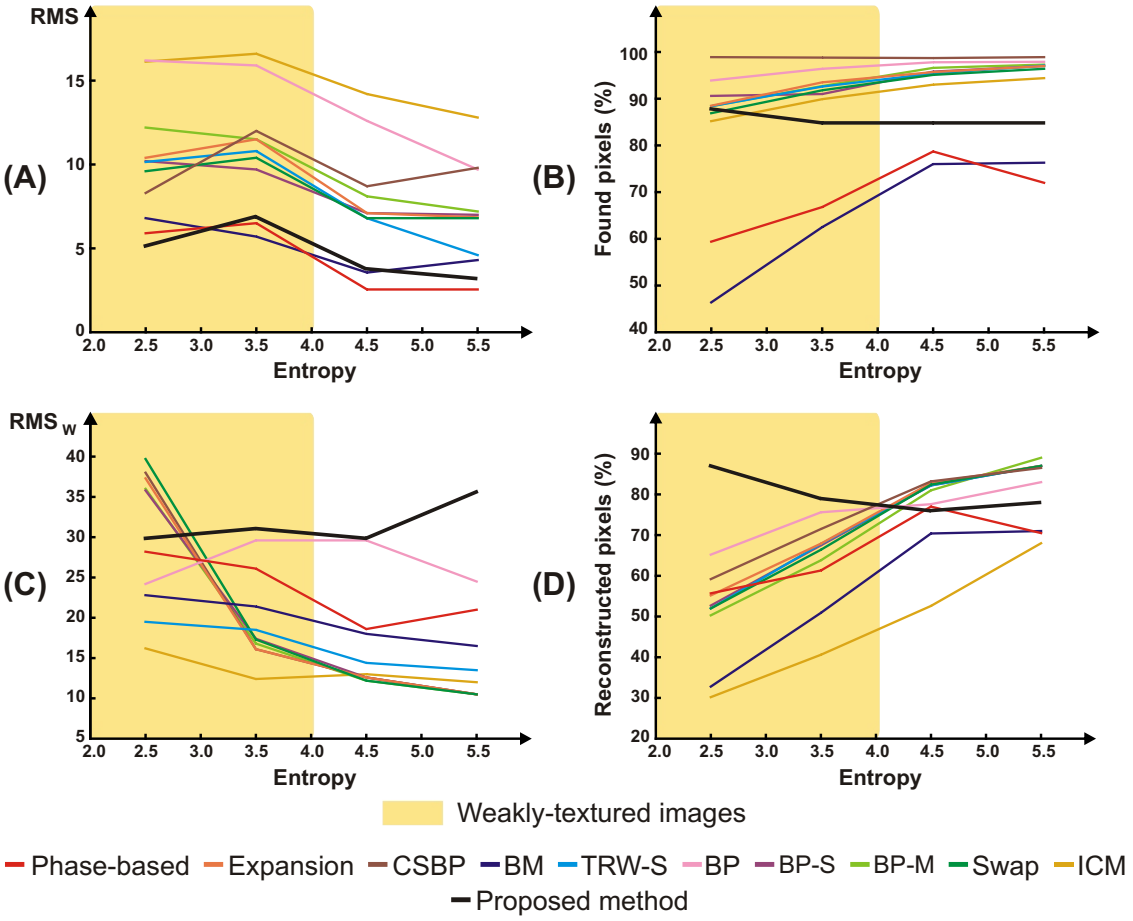

Figure 5.11: Comparison of the proposed approach with the conventional stereo algorithms as a function of image entropy. (A) RMS error based on known ground truth data for images from the Middlebury stereo dataset. (B) Percentage of pixels with found stereo matchings for images from the Middlebury stereo dataset. (C) RMS warping error for stereo pairs without ground truth data. (D) Percentage of rendered pixels for stereo pairs without ground truth data.

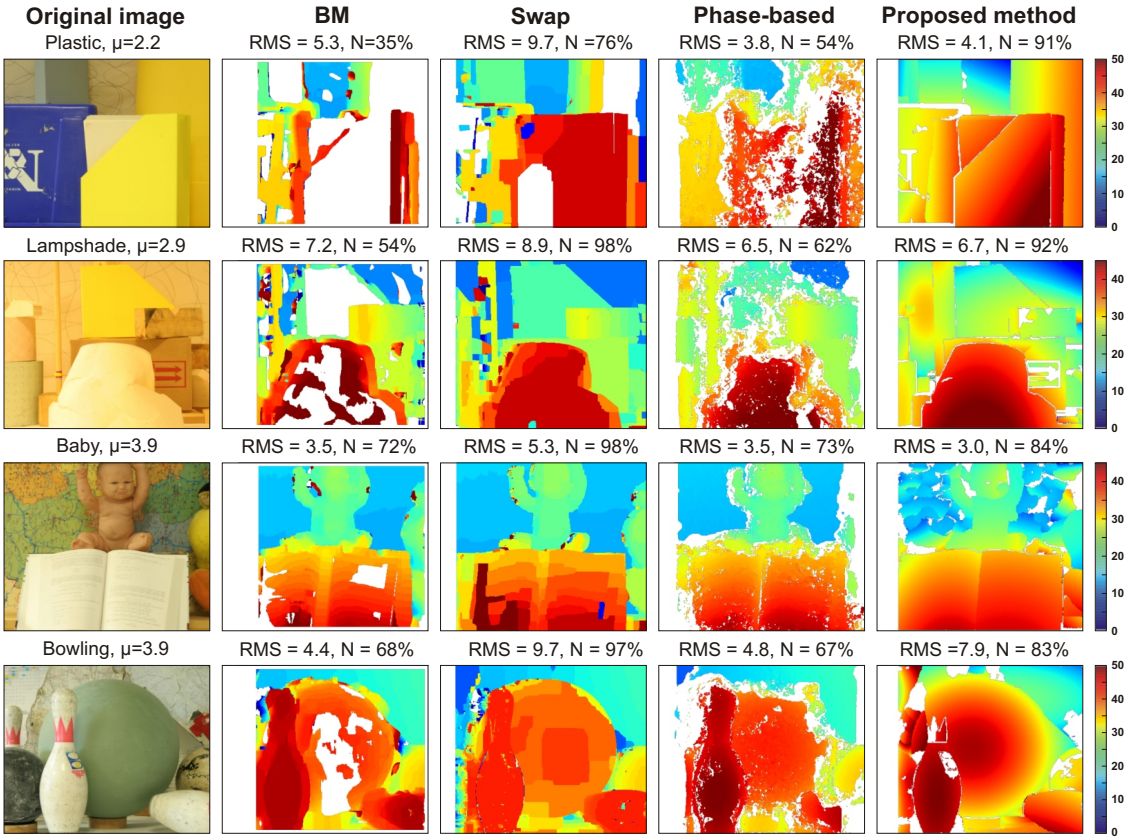

Figure 5.12: Disparities for weakly-textured stereo pairs selected from the Middlebury stereo dataset estimated by the block matching (BM), swap-move (Swap), phase-based, and proposed stereo techniques with given RMS error values and percentage of found matchings.

on the other hand produce the most accurate results for images *Plastic* and *Lampshade*. However, the sparsity of estimations and their instability at object borders (especially at borders of weakly-textured areas (see image *Plastic*)) are the major drawback. The proposed method, producing about $85 - 90\%$ of matchings and having quite low RMS error values (with the best score on *Baby* image), seems to be an ideal trade-off between precision and density.

The disparity maps estimated by the BM, ICM, and TRW-S techniques, having the highest scores among all the considered conventional methods on the rest of the dataset (see Fig. 5.11(C)), are compared to the output of the proposed method in Fig. 5.13. The results are shown for images *Breakfast* ($\mu = 2.3$), *Box* ($\mu = 3.9$), and *Table* ($\mu = 3.9$). For each output the RMS warping error value is given with a percentage of found matchings. The disparity maps produced by the BM are again very sparse and partially quite erroneous. But at the average available estimations

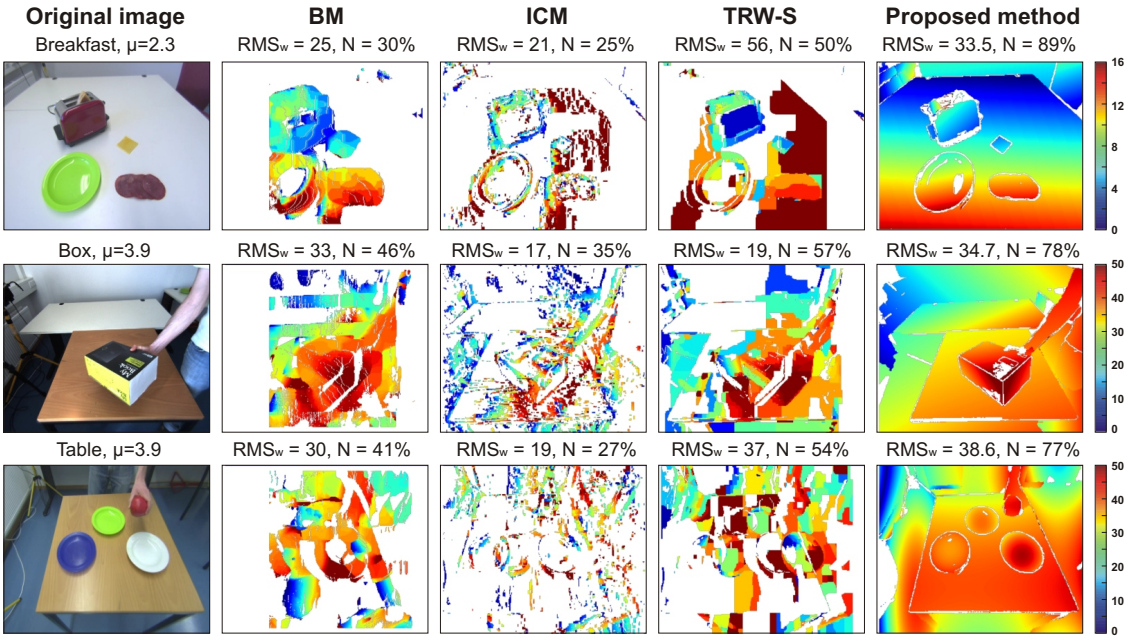

Figure 5.13: Disparities for weakly-textured stereo pairs selected from the testing stereo dataset estimated by the block matching (BM), iterated conditional mode (ICM), sequential tree-reweighted message passing (TRW-S), and proposed stereo techniques with given RMS warping error values and percentage of found matchings.

are still quite accurate. The ICM technique, having the poorest performance on the weakly-textured Middlebury images, has the lowest RMS warping error among all four techniques. Although the final disparity maps given by the ICM look quite disordered, they contain a large number of correct matchings. As compared to the BM and ICM approaches, the TRW-S technique produces more dense disparity maps but with less accuracy. The sparsity of estimations given by the BM, ICM, and TRW-S techniques and their instability at object borders (especially at borders of weakly-textured areas) make the usage of such maps in robotic applications problematic. The proposed method, producing about $77 - 90\%$ of matchings and having quite low RMS warping error values (with the best score on *Breakfast* image), again seems to be an ideal trade-off between precision and density.

5.5 Time performance

The processing times obtained for the segmentation of an input stereo pair and the disparity estimation from stereo-segment correspondences (see Section 5.3) are given

in Table 5.1 for image resolutions of 160×128, 320×256, and 640×512 pixels. The total processing times with frame rates are shown in Table 5.2. The computation times and frame rates have been measured by processing all images from the used stereo dataset and averaging the results using the following experimental environment: CPU 3.40 GHz Intel(R) Core(TM) i-2600S (using a single core) with 15.6 GB RAM and GPU Ge Force GT 580 (with 1.5 GB device memory) consisting of 16 Ms each having 32 cores, so 512 processor cores in total.

	Processing time (ms)		
Algorithmic step	160×128 (px)	320×256 (px)	640×512 (px)
Segmentation	15.8	55.8	179.0
LSQ planar	1.4	11.3	46.9
LSQ quadric	1.5	11.5	47.8
NMD planar	12.3	175.4	591.5
NMD quadric	13.5	189.4	681.7

Table 5.1: Processing times of the proposed stereo algorithm for the segmentation and disparity estimation obtained for planar and curved surfaces using both the least squares and Nelder-Mead optimization techniques..

For image size of 160×128 pixels the real-time performance is obtained for all four algorithmic modes (planar and curved surface models using both the least squares and Nelder-Mead optimization techniques). The choice of the surface model does not affect the runtime much and finding the parameters of curved surfaces (see (5.9)) takes only a bit longer as compared to planar surfaces (see (5.8)) for both function minimization techniques. The choice of the optimization technique influences the runtime quite a lot. As we can see, the Nelder-Mead simplex algorithm needs much more time to find a minimum as opposed to the least squares. For an image size of 320×256 pixels we obtained the frame rate of about 14 fps and 4 fps for the least squares method and Nelder-Mead method, respectively.

The most time consuming step in *LSQ planar* and *LSQ quadric* modes is the segmentation of a stereo pair requiring more than 80% of the processing time. Also note that the left and right images are segmented sequentially which means that the relaxation procedure for the right image can start only when the left image is completely segmented (see Section 5.3.1). In *NMD planar* and *NMD quadric* modes the approximation by the Nelder-Mead simplex algorithm is the bottleneck taking more than 70% of the runtime for image sizes of 320×256 and 640×512 pixels.

In the current study we do not compare the time performance of the proposed stereo algorithm with the considered conventional approaches due to the following reason. None of those methods can handle weakly-textured images ($\mu \leqslant 4.0$) better

	total (ms) / frames per second		
Techniques	160 × 128 (px)	320 × 256 (px)	640 × 512 (px)
LSQ planar	17.2 / 58.1	67.1 / 14.9	225.9 / 4.4
LSQ quadric	17.3 / 57.8	67.3 / 14.9	226.8 / 4.4
NMD planar	28.1 / 35.6	231.2 / 4.3	770.5 / 1.3
NMD quadric	29.3 / 34.1	245.2 / 4.1	860.7 / 1.2

Table 5.2: Total computation times obtained for planar and curved surfaces using both the least squares and Nelder-Mead optimization techniques.

in terms of the both estimation accuracy and density. Due to this reason it is not important whether they are faster or slower, since they do not solve the problem that our approach is aimed to. Among all implementations of the considered conventional stereo techniques used in this chapter, only the phase-based and BM techniques run in real-time, while other methods are much slower and require some seconds to process one stereo pair. However, real-time implementations of BP are also currently available (Brunton et al., 2006; Yang et al., 2006).

5.6 Discussion

In this chapter we presented a new stereo approach aimed at the recovery of disparity information in weakly-textured images. For the texture quantification we used an entropy calculation on the grayscale image. We classified input images having the entropy value $\mu \leqslant 4.0$ as weakly-textured (see Section 5.2.1 for details). Conventional stereo approaches, such as *block matching*, *iterated conditional mode*, *graph cuts*, *tree-reweighted message passing*, *belief propagation*, and *phase-based*, have been used for comparison with the proposed technique. Due to the fact that most of the images in the Middlebury dataset contain texture (having the entropy value $\mu > 4.0$), we extended the Middlebury dataset by images having a little texture ($\mu \leqslant 4.0$) which is required for a fair evaluation of stereo methods.

The proposed method is based on the co-segmentation of an input stereo pair which establishes correspondences between segments in the left and right image. The co-segmentation of stereo images is based on the real-time segmentation algorithm based on the Metropolis updates with the short-cut introduced in Chapter 2. In order to assign disparity values to all pixels of found stereo segments, the algorithm tries to find the most suitable surface for each segment. An initial disparity map consisting of sparse disparity values, estimated by the real-time phase-based technique, and an additional stereo data provided by segment correspondences between the left and

right image are used as input values for the surface approximation. Outliers, such as disparity values which are too close to segment boundaries or located in potential half-occlusions, are eliminated from the initial disparity map before the surface fitting step (see Section 5.3.5). We investigated the least squares and Nelder-Mead simplex optimization techniques for the surface approximation using the planar and curved surface models.

The experiments have shown that the proposed stereo algorithm represents an ideal trade-off between precision and density for images with $\mu \leqslant 4.0$. In weakly-textured environments it produces quite accurate estimations with $80 - 90\%$ and $75 - 90\%$ density of matches for the Middlebury images and the rest of the dataset, respectively. The block matching and phase-based techniques, having a similar accuracy on the Middlebury images, are much sparser. Other conventional methods are outperformed by the proposed algorithm on this data. On the rest of the dataset, the *iterated conditional mode*, *tree-reweighted message passing*, *block matching*, *belief propagation*, and *phase-based* approaches are featured by a higher precision but produce much less matchings as opposed to the introduced method. Despite the high accuracy of some conventional techniques, the sparsity of estimations and their instability at object borders in weakly-textured areas make the usage of such disparity maps in robotic applications very limited. Our method, on the contrary, gives a dense disparity map for each object or object part (identified as a segment on the segmentation stage) together with the model describing its surface. Note that although our algorithm produces quite accurate disparity maps for images with $\mu > 4.0$ as well, traditional approaches are more efficient in sufficiently textured regions producing more accurate and dense results.

For the frame size of 160×128 pixels we achieved a processing time adequate for many real-time robotic applications using both the least squares and Nelder-Mead optimization techniques with planar and curved surface models. For a frame size of 320×256 pixels the real-time performance was obtained only for the approximation with the least squares, while the Nelder-Mead algorithm required much more time for finding a minimum. For the frame size of 640×512 pixels the algorithm can still process a few frames per second for all four modes. This is not sufficient in terms of the real-time cognitive vision system but still can be employed by applications having lower demands on the processing time. The presented stereo algorithm has the following limitations:

1. The method depends on the co-segmentation of stereo pairs, the final disparity maps can drastically suffer from inconsistent stereo segments.

2. Computed disparity values based on surface fitting can be quite inaccurate for objects or object parts which are homogeneous in terms of the color but consist of various surfaces, e.g., it can happen in image *Plastic* shown in Fig. 5.9 if all yellow objects featured by different plane surfaces, are identified by one segment.

3. The performance of the method is very poor for objects with extremely low texture ($\mu < 1.0$). For such objects only a few estimations are available in the sparse disparity map which are not enough for approximation of segment surfaces.

The listed limitations will be addressed in the future work, especially the first two points. Unfortunately, passive stereo techniques, considered in this chapter, cannot recover disparity information in extremely untextured regions, and it is worth to employ active methods for depth data acquisition in robotic applications operating on scenes having $\mu < 1.0$.

6

Depth-supported Real-time Video Segmentation with the Kinect

"Jet engines may not be how nature did it, but it works - and does so far better than flapping wings"

– Jeff Hawkins

6.1 Introduction

Video segmentation aims at representing image sequences through homogeneous regions (segments), where according to the spatio-temporal coherence the same object or object part should carry the same unique label along the whole video stream (see Chapter 3). The segmented visual data can be used for higher-level vision tasks which require spatial and temporal relations between objects to be established (Kjellström et al., 2011; Rao et al., 2010; Aksoy et al., 2011). The major challenges faced in the video segmentation problem are processing time, temporal coherence, and robustness.

The conventional video segmentation techniques considered in Section 3.1 as well as the proposed real-time approach are based on color cues alone. Only color information is incorporated into the computation of interaction strengths in the segmentation core leading to the formation of segments (see Section 2.2.1). In the current chapter, we will show that the inclusion of depth information improves the video segmentation results. We extend the framework for segmentation of monocular videos (see Section 3.2) by including depth information into the segmentation core. The spatio-temporal synchronization along the video stream is achieved through the label transfer from one frame to the next using warping based on both the real-time optical flow (see Section 3.2.2) and the depth information. This way improves the efficiency of the method significantly and allows a soft tracking of segments to be carried out.

There are various ways how scene depth can be obtained. We distinguish *passive methods* and *active methods*. The most well-known passive approach is stereo vision

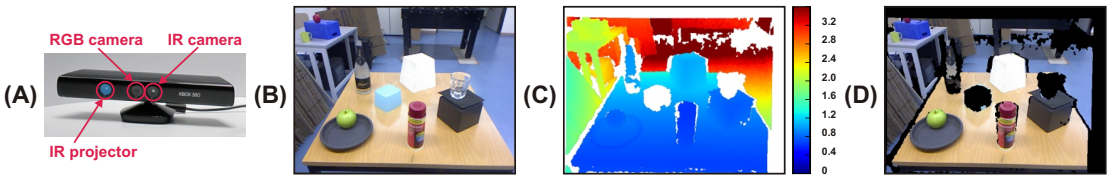

Figure 6.1: (A) The Kinect device. (B) Original frame acquired by RGB camera. (C) Depth data derived from IR image (in meters). White patches in the image denote pixels for which no depth information can be acquired. (D) Color pixels having depth values.

which is least expensive and widely used (Scharstein and Szeliski, 2002). An overview of existing stereo methods was given in Section 5.1.1. Despite significant progress made over last few years in the domain of stereo vision, the fundamental problems of all stereo approaches such as occlusion, lack of texture, and repetitive patterns remain unsolved. Typical sensors are the time-of-flight (ToF) sensors, 3D scanning, structured coded light approaches. Active approaches provide real-time or close to real-time depth estimates under conditions where passive stereo techniques do not work well, for example on white walls (Chen et al., 2008). However, the sensors are noisy and perform poorly on the textured scenes where stereo is very robust. Furthermore, passive techniques perform badly or do not work at all outdoors because of the light interference.

The Microsoft Kinect device, released in the fall of 2010 for the XBox videogame platform [1], is an active approach which is used in this study for depth acquisition. Since the current work is aimed at robots operating indoors and manipulating weakly-textured objects, the mentioned drawbacks of active methods are not crucial and the Kinect, producing input images coupled with depth data for resolution of 640×480 pixels in real-time, seems to be a perfect solution for our task. The new device has immediately attracted the attention of the computer vision society because of its technical capabilities and its very low cost compared to ToF sensors. The Kinect device features an IR projector for generating infrared images and two cameras: an RGB camera for capturing color images and an IR camera for capturing infrared images under various light conditions (see Fig. 6.1(A)). The IR camera is based on a monochrome CMOS sensor used in some ToF cameras (Zhu et al., 2011). However, some ToF cameras can work outdoors, while the Kinect's depth sensor performs well only in shady regions being useless in sunlight, since the IR structured lighting pattern, emitted by the projector, gets completely lost in ambient IR light. For indoor environments the Microsoft Kinect proves to be an inexpensive and suitable device for acquiring depth/color videos in real-time.

[1] Kinect for XBox 360: http://www.xbox.com/en-US/kinect

However, it needs to be considered that indoors the depth data obtained by the Kinect can suffer from the following side effects: multiple or glossy reflection, ambient light, light absorption by objects in the scene, object boundaries. Some of mentioned side effects are shown in Fig. 6.1(B,C); see white patches in panel C. The depth data cannot be derived at all for a glass (multiple reflections), a light blue box on the table (light absorption) and some object boundaries (color differences). A bit better data can be obtained for a bottle of frosted glass that has less reflections. Note that the RGB camera has a larger angle of view than the IR camera, it is the reason why depth cannot be derived for pixels located close to image borders (see Fig. 6.1(D)). Furthermore, in order to relate color and depth images a calibration of the Kinect is required (see Appendix A.4).

In this chapter we present a novel real-time video segmentation algorithm based on the superparamagnetic clustering of data which performs fusion of image and range data acquired by the Kinect device. To our knowledge, the presented approach is the first method combining both depth and color information derived directly from the Kinect device for the on-line, dense, and automatic segmentation of video streams (Abramov et al., 2012a).

The chapter is organized in the following way. First we present a real-time video segmentation technique based on the parallel Metropolis algorithm introduced in Chapters 2 and 3 supplemented by the depth information. Then we present experimental results and time performance of the method. Finally we discuss our results and conclude the chapter.

6.2 Depth-supported video segmentation

The presented in the current chapter depth-supported real-time video segmentation is based on the segmentation core used for segmentation of monocular video streams in Chapter 3. Depth information is incorporated into the Potts model and into the label transfer procedure in a manner which is consistent with the color information, giving an additional cue for segmentation. The inclusion of depth provides important additional information about object boundaries which improves video segmentation.

6.2.1 Extended image segmentation core

In order to employ depth information for segmentation of video streams, the segmentation core needs to be extended by depth data. In the parallel Metropolis algorithm for image segmentation (see Section 2.2.1), interaction strengths between adjacent pixels, leading to formation of segments, take only color information into account (see 2.5).

Depth information, produced by the Kinect device and available for each video frame, is incorporated into the considered image segmentation technique applying

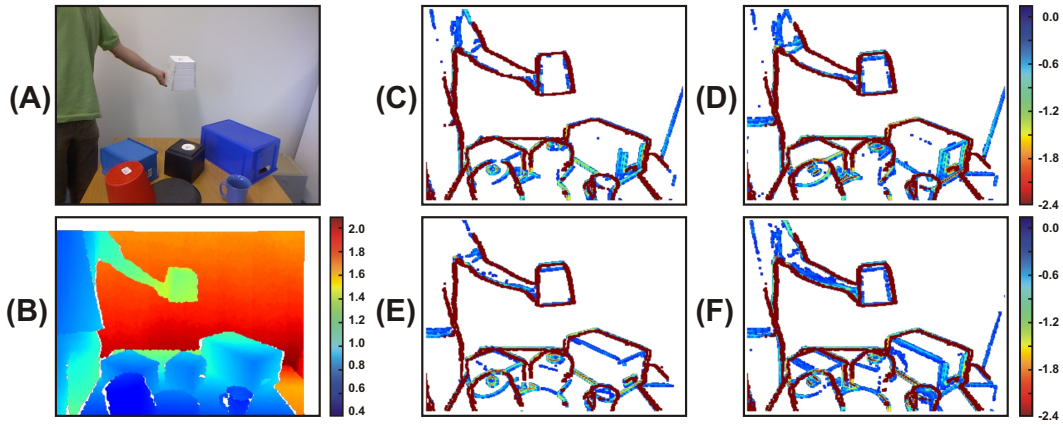

Figure 6.2: Color differences for the 8-connectivity case in the CIE ($L^*a^*b^*$) color space. (A) Original frame. (B) Depth data (in meters). (C - F) Matrices with coupling constants computed for horizontal, left diagonal, vertical and right diagonal directions (here $\tau = 30$ cm). Note that only coupling constants leading to the formation of segments are shown ($J < 0$).

constraints to interaction ranges of pixels. Thus, the depth data acquired along with the color image (see Fig. 6.2(A,B)) is used to prevent interactions between pixels having a large range difference. This is done by replacing all interaction strengths J_{ij} (see (2.5) in Section 2.2.1 between pixels having a depth difference larger than a pre-defined threshold τ with the very low value $\Theta = -5.0$ according to

$$J_{ij} = \begin{cases} J_{ij} & \text{if } |z_i - z_j| \leqslant \tau, \\ \Theta & \text{otherwise.} \end{cases} , \tag{6.1}$$

where z_i and z_j are range values of pixels i and j, respectively. Matrices containing color differences involved in the formation of segments under the introduced constraint are shown in Fig. 6.2(C - F). Excluded interactions, marked by dark red, prevent in most cases neighboring pixels to be assigned to the same segment. This way segmentation of 2D images is supported by 3D data and merges of similar looking objects or object parts are prevented.

6.2.2 Linking of segments

An estimated optical flow vector field for two adjacent frames t and $t + 1$ from a test video stream is shown in Fig. 6.3(A - C). Having segments with correspondent average range values for a time step t (see Fig. 6.3(D)) and estimated optical flow vector field, labels of segments S_t are transferred to frame $t + 1$ excluding transfers between pixels having a range difference larger than a pre-defined threshold τ (see

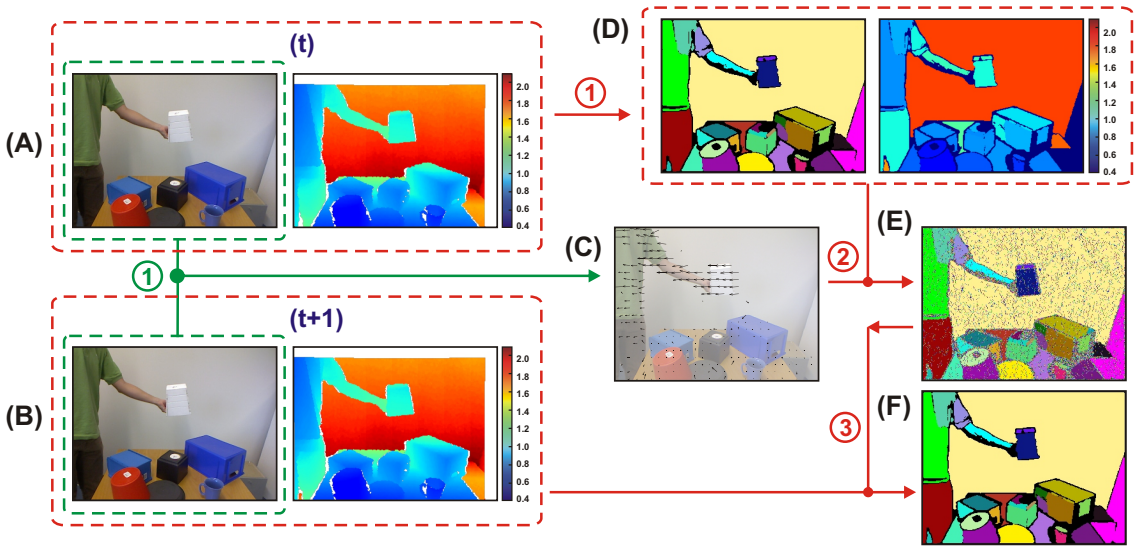

Figure 6.3: Segmentation of two adjacent frames in a sequence. Numbers at arrows show the sequence of computations. (A,B) Kinect data acquired at time steps t and $t+1$, respectively. (C) Estimated optical flow vector field (sub-sampled 11 times and scaled 10 times) (step 1). (D) Extracted segments S_t with correspondent average range values $\overline{z}$ (step 1). (E) Initialization of frame $t+1$ after the label transfer from frame t (step 2). (F) Extracted segments S_{t+1} (step 3).

Fig. 6.3(E)). We obtain

$$S_{t+1}(x_{t+1}, y_{t+1}) = \begin{cases} S_t(x_t, y_t) & \text{if } \lambda \leqslant \tau, \\ 0 & \text{otherwise.} \end{cases}, \tag{6.2}$$

$$\begin{aligned} \lambda &= |z_{t+1}(x_{t+1}, y_{t+1}) - z_t(x_t, y_t)|, \\ x_{t+1} &= x_t + v_x(x_t, y_t), \\ y_{t+1} &= y_t + v_y(x_t, y_t), \end{aligned} \tag{6.3}$$

where z is a range data obtained from the Kinect and $\mathbf{v} = (v_x, v_y)^T$ is the optical flow vector. Label transfers between segments having large range differences are excluded as well, which yields:

$$S_{t+1}(x_{t+1}, y_{t+1}) = 0 \text{ if } \xi > \tau, \tag{6.4}$$

$$\xi = |z_{t+1}(x_{t+1}, y_{t+1}) - \overline{z}_t(x_t, y_t)|, \tag{6.5}$$

and $\overline{z}$ is a matrix containing average range values for each segment (see Fig. 6.3(D)). Spin variables of pixels without correspondences are initialized by labels which are not occupied by any of the found segments (see Fig. 6.3(E)). Once frame $t+1$ is

initialized, it needs to be adjusted to the current image and depth data is required by the extended image segmentation core (see Section 6.2.1). This adjustment is needed in order to fix erroneous bonds which can take place during the transfer of spin states from frame t. The relaxation process performed by the extended image segmentation core runs until convergence and only after that the final segments can be extracted (see Fig. 6.3(F) where corresponding segments between frames t and $t+1$ are labeled with identical colors). Only segments larger than a pre-defined minimum size are extracted, thereby small segments at borders of the blue cup and at edges of the big blue box formed due to reflections and changes in contrast are excluded (see Fig. 6.3(D,F)). The use of range data allows us to distinguish between objects having very similar color values like between the white moving object and the wall and between the blue cup and the big blue box (see Fig. 6.2(A,B)).

6.3 Experimental results

In this section we present results of our method obtained for several depth/color videos acquired with the Kinect showing human manipulations of objects. The method is compared here with another state-of-the-art video segmentation technique. Our approach is evaluated in terms of the quality of the segmentation, coherence of the video segmentation, and computational speed. Again, both the quantitative and qualitative evaluations, introduced in Section 3.3, are needed to judge and compare video segmentation results.

Qualitative evaluation

Video segmentation results obtained in the CIE $(L^*a^*b^*)$ color space for the test sequence "Moving an object" without and with support of the depth data are shown in Fig. 6.4. The first and second rows show the original color frames and estimated optical flow for a few selected frames. The third row shows results obtained without usage of the range data, i.e., produced by the framework proposed in Section 3.2. We can see that video segmentation fails for fast moving objects. As was already mentioned in the previous chapters, the optical flow method has a limit of 2 pixels per scale, so using 4 scales, the limit is $2^4 = 16$ pixels (see Section 3.2.1). For this reason the white wooden object cannot be tracked along the whole sequence and some of its parts are initialized improperly in frame 530 by the label taken from the background. It occurs due to the lack of pixel correspondences between adjacent frames. Such erroneous initializations cannot be resolved by the segmentation core only. Note that both the moving object and the wall have in some frames very similar color values which make the tracking extremely difficult.

Incorporation of range data (shown in the fourth row) into the segmentation core and using it on the label transferring stage (see Sections 6.2.1 and 6.2.2) helps to

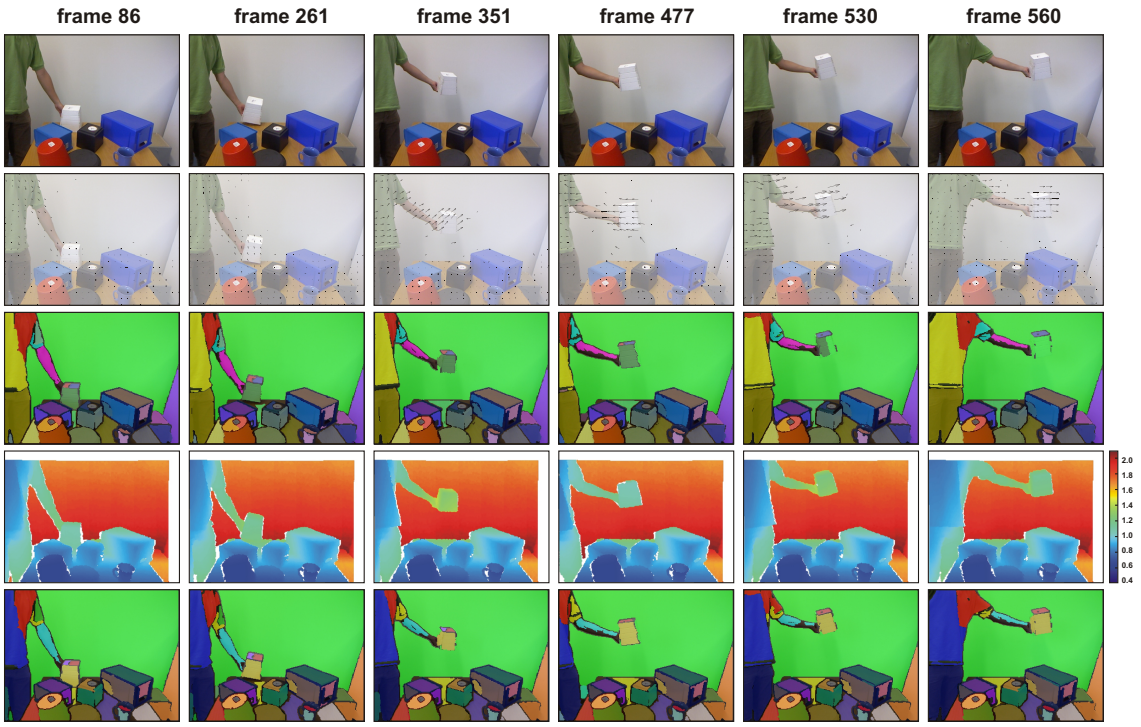

Figure 6.4: Segmentation of frame sequence "Moving an object". Original frames and estimated optical flow for selected time points are shown in the first and the second rows, respectively. Segmentation results without usage of range data are shown in the third row. The forth row shows depth data obtained from the Kinect. Segmentation results obtained using a fusion of image and range data are depicted in the last row.

resolve such problems. Segmentation results of the same frame sequence derived with the range data support are presented in the last row of the figure. Fast moving pixels cannot be initialized by labels of pixels having range differences larger than the threshold τ (see (6.2)). In the current experiment we used $\tau = 30$ cm. Furthermore, similar pixels having large range differences do not tend to interact with each other (see (6.1)). Thereby the segmentation core can recover even poorly-initialized segments which makes the tracking of the fast moving white object consistent along the whole sequence.

Next, the segmentation results for a 2 min frame sequence of the sample action "Building a pyramid" are presented in Fig. 6.5. The first and second rows show original color frames with depth data from the Kinect. The third row shows segmentation results obtained by the proposed depth-supported video segmentation method using $n_2 = 30$ relaxation iterations, $\alpha_2 = 2.5$, the starting temperature $T_{n+1} = 1.0$, and the simulated annealing factor $\gamma = 0.999$. As we can see our approach provides a temporally coherent video segmentation, in which all segments carry their initially

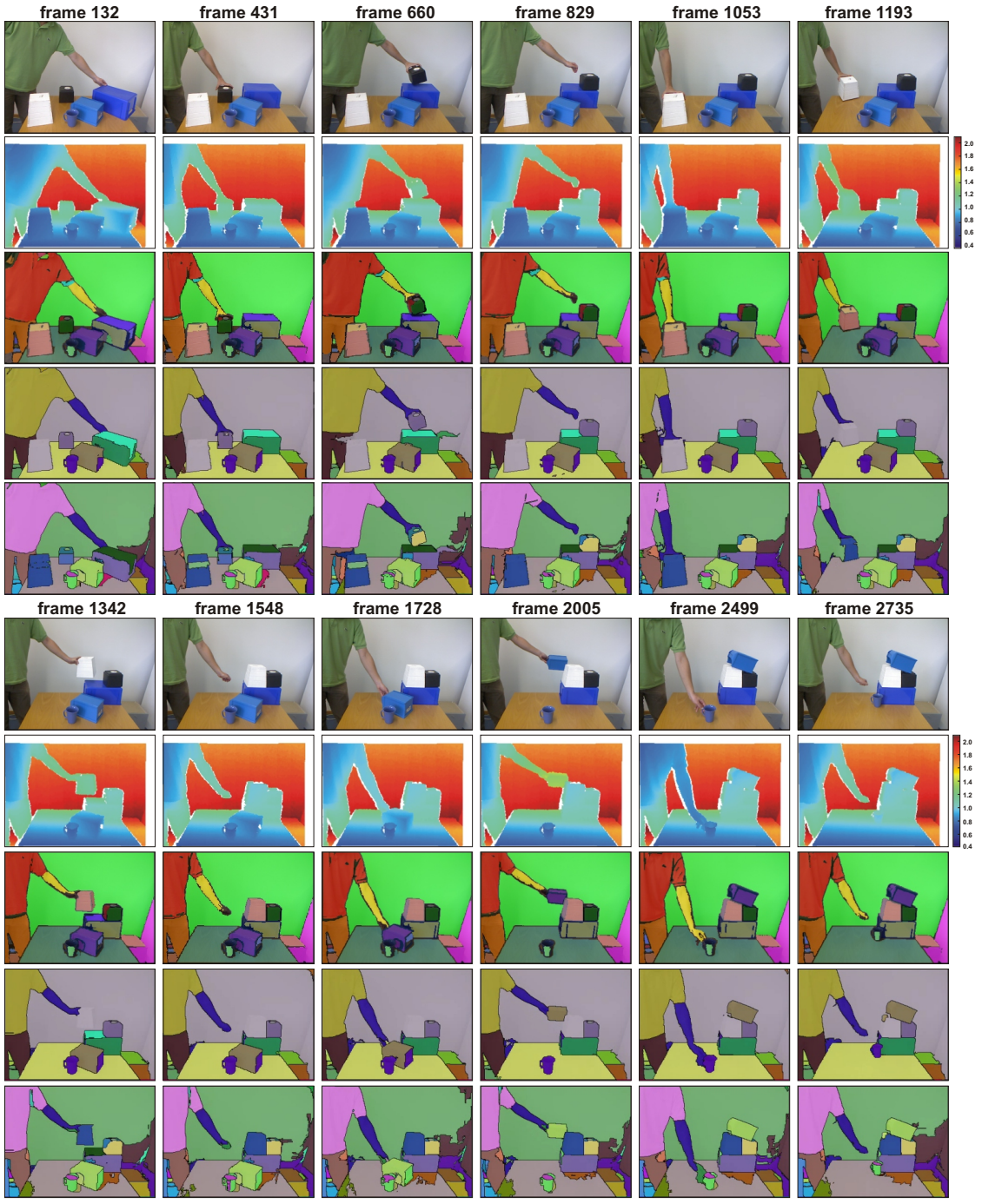

Figure 6.5: Results for frame sequence "Building a pyramid". Original frames and range data from the Kinect for selected time points are shown in the first two rows. The third row shows the segmentation results of our method ($n_2 = 30$ iterations, $\alpha_2 = 2.5$). Graph-based video segmentation results obtained at 90% and 70% of the highest hierarchy level are presented in the last two rows.

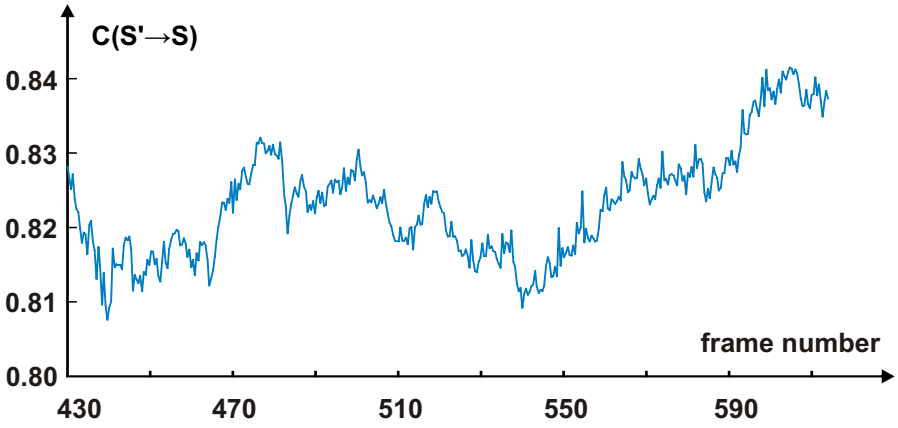

Figure 6.6: Segmentation covering for frames 430 – 630 out of the "Building a pyramid" sequence.

assigned labels along the whole video stream. The proposed video segmentation technique is compared here again to the hierarchical graph-based video segmentation, proposed by Grundmann et al. (2010), which is known as one of the most efficient spatio-temporal segmentation techniques to date. Results derived by the hierarchical graph-based video segmentation at 90% and 70% of the highest hierarchy level [2] are shown in the fourth and the fifth row, respectively. Note that both methods cannot be compared entirely, since the publicly available implementation of the graph-based approach uses future data for segmentation (off-line processing) and does not incorporate the depth data. Therefore, here we only show that the proposed approach gives output comparable to results of the conventional video segmentation methods. Depending on the hierarchy level of the graph-based method, a coarser or finer segmentation is obtained. At coarse levels, merging problems leading to under-segmentation are observed, while at finer levels, more segments are formed, leading, however, to some temporal coherence problems.

Quantitative evaluation

Fig. 6.6 shows the performance of the system for frame sequence "Building a pyramid" as the segmentation covering against the current frame number for frames $430 - 630$. As we can see, the color/depth sequence is segmented with a quite high accuracy having the average segmentation covering value 0.825.

[2] the online version of the hierarchical graph-based video segmentation for 90% and 70% of the highest hierarchy level is available under http://neumann.cc.gt.atl.ga.us/segmentation/

resolution (px)	msec / frame	frame rate (Hz)
128×160	9 – 17	111 – 59
256×320	21.5 – 39.5	47 – 25
512×640	72.5 – 145.5	14 – 7

Table 6.1: Processing times and frame rates obtained for various image resolutions with 20 – 60 relaxation iterations.

Time performance

The algorithm runs on the Nvidia GeForce GTX 295 card (with 896 MB device memory). The total processing times, frame rates for various image resolutions are summarized in Table 6.1. The proposed method runs in real-time for medium image resolutions and can process video sequences of arbitrary length, while the graph-based video segmentation needs about 20 min to process a 40 sec video and only sequences that are not longer than 40 sec (with 25 fps) can be processed in the hierarchical mode (Grundmann et al., 2010).

6.4 Discussion

We extended the image segmentation core based on the superparamagnetic clustering of data (see see Section 2.2.1) by the use of depth information in terms of the constrained parallel Metropolis updates and label transfers between adjacent frames. The Kinect device was used as a hardware setup for simultaneous real-time acquisition of color images and correspondent depth information.

Usage of depth data makes it possible to track relatively fast moving objects by preventing interactions between pixels having significant range differences. It could be shown that the incorporation of the depth data into the segmentation process makes the segmentation core more robust and reduces under-segmentation. Our method can be at match with the graph-based technique (Grundmann et al., 2010) in terms of segmentation quality for the types of movies considered. In terms of computational speed, we passed the graph-based method, which works at lower frame rates than ours. However, for complex actions and scenes, the coherence of the segmentation may be impaired due to the following problems:

1. Objects are getting partly or completely occluded during the action. It can lead to assignment of new labels when these objects reappear again which breaks the temporal coherence.

2. Objects are getting joint/disjoint. If two large parts of the same object repre-

sented by different segments are merged, we face the already mentioned domain fragmentation problem (see Section 2.2.6). In the presented algorithm the domain fragmentation problem can be resolved only by a very long annealing schedule (see Section 2.2.3) which cannot be achieved in real-time. If one object is divided into several pieces, all of them will keep an equal label even being disjoint and independent which is inconsistent in terms of some applications.

3. Objects move extremely fast, causing optical flow to fail (see Section 3.4).

4. The usage of average range values for segments during the label transfer (see Section 6.2.2) is not very accurate and can cause domain fragmentation and / or temporal coherence problems in the case of objects or object parts whose surfaces are not parallel to the image plane.

In the future, we aim to improve performance of the proposed method under these circumstances.

7 Conclusion and Outlook

"A good ending is vital to a picture,
the single most important element,
because it is what the audience
takes with them out of the theater"
– Walt Disney

Each previous chapter contained its own extensive "Discussion" section where we discussed our results and compared our methods to other approaches. Thus, in this chapter we will only briefly summarize presented work by highlighting all main findings, provide an outlook for future investigations, and conclude this thesis.

In this thesis we were investigating one of the most fundamental problems in the computer vision - establishing correspondences between images acquired from different view points or adjacent frames of a video stream. We developed a framework performing the automatic cognition of the visual scene in such a way that it transforms input visual information into symbol-like representation where all objects or object parts are detected, identified, and relations between them are determined. All components of the framework are on-line, automatic, do not use prior knowledge about the input data, and can run for some resolutions in real-time. Therefore, the proposed framework is a cognitive visual system which can be used in on-line robotic systems to close the replicated perception-action loop between sensors and robots. The framework combines both local and region correspondences in order to improve and accelerate the matching procedure as compared to both approaches applied individually.

The framework is built around the novel real-time image segmentation technique developed in the first part of the thesis (see Chapter 2). This technique solves the segmentation problem by the method of the superparamagnetic clustering of data which performs the fusion of local and region correspondences in a very efficient way. Spin states in the Potts model, designating partitioning of an image, can be easily transferred between various views of the scene or adjacent frames of a video stream taking local matchings into account. Region matchings are found then by the update of the Potts model with the Metropolis algorithm. The Metropolis algorithm

with the simulated annealing was chosen for spin state updates due to its local nature and ability for acceleration on the special hardware. As a special hardware for acceleration we used a GPU architecture with the parallel programming model of CUDA. The parallel real-time Metropolis algorithm running on the GPU with the short-cut for acceleration of the annealing procedure is the main result of Chapter 2. Extensive experimental results and evaluations shown in Section 2.3.2 demonstrate the comparability of segmentation results produced by the proposed algorithm with the conventional image segmentation techniques, such as *mean shift* and *graph-based*. The graph-based technique is slightly more precise, but almost two times slower for middle-size and large images as compared to our method. The major drawback of the presented algorithm is that it does not produce consistent results on very textured images, whereas both the mean shift and graph-based approaches perform better there. However, this problem could be solved by pre-filtering of input images applying special texture filters which smooth texture preserving edges between diverse regions.

In the second and third parts of the thesis the developed image segmentation technique was used for the real-time segmentation of monocular and stereo video streams (see Chapters 3 and 4). As opposed to the mean shift and graph-based segmentation algorithms, our framework based on the parallel Metropolis updates on the GPU does not require a very time consuming region matching procedure for finding correspondences between frames from a video stream or images acquired from different view points. The major limitation of the method is its inability to maintain the spatio-temporal coherence in the case of full occlusions. This problem cannot be resolved on the pixel domain in the context of the Metropolis algorithm, and high-level tracking mechanisms operating on the level of segments are required for that (Nummiaro et al., 2002; Wang et al., 1994). Furthermore, these techniques should also help to resolve faster the domain fragmentation problem which arises in the case of merging of previously disjoint objects or their parts. The current framework can resolve it only by a very long annealing schedule.

In the forth part established stereo-segment correspondences were employed for recovery of depth information in weakly-textured images (see Chapter 5). It was shown that performance of all traditional passive stereo techniques is extremely poor in weakly-textured environments with respect to the estimation accuracy and density due to the lack of texture, as only a few correspondences between two views can be found. However, it was shown that found stereo segments produce an additional and quite accurate information limiting the disparity search in poorly-textured regions. Using the linear least squares for optimization of surface model functions the method runs in real-time for middle-size images and close to real-time for larger images. Therefore, 3D information even in very untextured environments can be obtained and considered in real-time robotic applications. The proposed stereo approach fails only in extremely textureless regions where no local correspondences are available. It makes the usage of passive stereo techniques in such environments meaningless and

active methods for acquisition of depth data are required there.

Finally, the fifth and the last part of this thesis (see Chapter 6) demonstrated an improvement of the video segmentation by the use of depth information provided by an active sensor, here the Kinect device. Video segmentation supported by the depth data allows the tracking of relatively fast moving objects and increases the robustness of the framework. The usage of depth information leads to more precise label transfers resulting in less Metropolis iterations needed for the relaxation. Hence, the segmentation of monocular video streams using the Kinect is faster than the original framework based on the processing of color information only. But this extension does not resolve occlusions and the domain fragmentation problem.

The main achievement of the thesis is an efficient compression of the input visual data into symbol-like descriptors performed by the cognitive computer vision system serving as a visual front-end for robotic applications. We have shown that even very intensive pre-processing operations such as segmentation of the visual data, maintaining of the spatio-temporal coherence of found descriptors, and extraction of the 3D structure for weakly-textured scenes can be done fast enough in order to be incorporated in the perception-action loop replicated by robots. While our input scenarios have still some limitations, we think that the proposed framework may help in obtainment of the reduced representation of the visual input. Aksoy et al. (2011) have shown in their study that such a representation allows the encoding of various types of actions by so-called *semantic event chains*.

The presented cognitive vision system can be extended by a more sophisticated tracking considering full occlusions, very fast movements, or reappearance of previously observed objects maintaining the basic segmentation mechanism. The depth information obtained by the modern active sensors needs to be used in future more extensively which will allow us to incorporate the geometry of objects directly in the image segmentation core and perform three-dimensional video segmentation (Rusu and Cousins, 2011).

A Appendix

A.1 GPU occupancy data

GPU occupancy data and physical limits for the CUDA kernel of the parallel Metropolis algorithm (see Section 2.2.5) are summarized in Table A.1 for the GeForce GTX 295 and GTX 580 architectures of compute capabilities 1.3 and 2.0, respectively. To compute the multiprocessor occupancy of a GPU by our CUDA kernel we used the CUDA occupancy calculator [1]. On devices of compute capability 1.3 a number of active thread blocks, i.e., thread blocks running simultaneously on one streaming multiprocessor (SM), is limited by both the number of registers per SM and the amount of shared memory per SM. Since the total number of 32-bit registers per SM is $16,384$ and the number of registers allocated for one block of the kernel is $R_{block} = 6,565$, one SM has enough registers only for two thread blocks running at the same time. In terms of shared memory one thread block allocates $S_{block} = 5,632$ bytes, while only 16 KB are available per SM. Therefore, one SM has enough shared memory only for two thread blocks running at the same time. All these limitations lead to only 50% occupancy of each SM. More thread blocks can run at the same time on devices of compute capability 2.0 due to their hardware improvements (more shared memory and more 32-bit registers per SM). Thus, the 67% occupancy of each SM can be achieved on the GTX 580 card. This time the number of active blocks is limited by a number of registers per SM. The total number of 32-bit registers is $32,768$, while one block needs $R_{block} = 6,565$. Therefore, one SM has enough registers only for four thread blocks running at the same time.

The warp occupancy of SMs as a function of threads per block, registers per thread, and shared memory per block for both graphics cards is shown in Fig. A.1. The resource usage of the proposed kernel is indicated by red rectangles on all graphs. The other data points represent the range of possible block sizes, register numbers, and shared memory allocation. Note that higher occupancy does not necessarily lead to higher performance especially for kernels that are not bandwidth bound. If the kernel is bottlenecked by computation and not by global memory accesses, then

[1] available under http://developer.nvidia.com/cuda

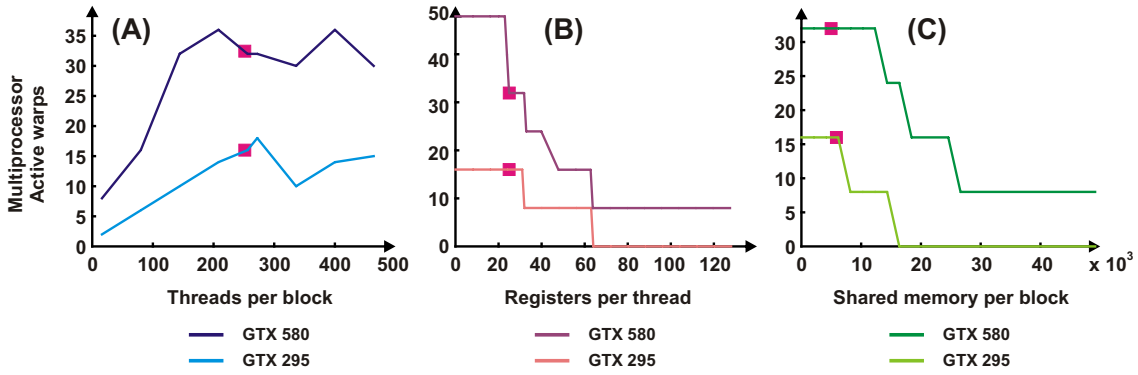

Figure A.1: Multiprocessor warp occupancy as a function of threads per block (A), registers per thread (B), and shared memory per block (C) for the proposed CUDA kernel. The current resource usage is indicated by red rectangles on the graphs for the GeForce GTX 295 and GTX 580 architectures.

increasing occupancy may have no effect. On the contrary, for bandwidth-bound applications increasing occupancy can help to hide the latency of memory accesses, and consequently improve performance.

	GTX 295	GTX 580
Compute capability	1.3	2.0
Warps per Multiprocessor	32	48
Total number of warps in a block (W_{block})	8	8
Total number of registers allocated for a block (R_{block})	6,656	6,656
Shared memory (in bytes) allocated for a block (S_{block})	5,632	5,248
Active threads per Multiprocessor	512	1,024
Active warps per Multiprocessor	16	32
Active thread blocks per Multiprocessor	2	4
Occupancy of each Multiprocessor	50%	67%

Table A.1: Physical limits of the proposed CUDA kernel on the GeForce GTX 295 and GTX 580 architectures.

A.2 General linear least squares

To present the solution by use of the normal equations for the general linear least squares (see Section 5.3.6 of), we need to introduce some notation. Let **A** be a matrix

of $N \times M$ components which are constructed from the M basis functions evaluated at the N points (x_i, y_i), and from the N measurement errors ϑ_i as

$$A_{ij} = \frac{\varphi_j(x_i, y_i)}{\vartheta_i}. \tag{A.1}$$

The matrix $\mathbf{A}$ is called the *design matrix* of the fitting problem. Not that in general $\mathbf{A}$ has more rows than columns, i.e., $N \geqslant M$, since there must be more data points than model parameters to be solved for (we can fit a straight line to two points, but not a quintic). The design matrix for the least squares fit of a linear combination of M basis functions to N data points looks as follows:

$$\mathbf{A}_{M,N} = \begin{pmatrix} \frac{\varphi_1(x_1,y_1)}{\vartheta_1} & \frac{\varphi_2(x_1,y_1)}{\vartheta_1} & \cdots & \frac{\varphi_M(x_1,y_1)}{\vartheta_1} \\ \frac{\varphi_1(x_2,y_2)}{\vartheta_2} & \frac{\varphi_2(x_2,y_2)}{\vartheta_2} & \cdots & \frac{\varphi_M(x_2,y_2)}{\vartheta_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\varphi_1(x_N,y_N)}{\vartheta_N} & \frac{\varphi_2(x_N,y_N)}{\vartheta_N} & \cdots & \frac{\varphi_M(x_N,y_N)}{\vartheta_N} \end{pmatrix}.$$

Also we define a vector $\mathbf{b}$ of length N by

$$b_i = \frac{z_i}{\vartheta_i}, \tag{A.2}$$

and denote the vector whose components are the M parameters to be fitted, $a_1, a_2, \cdots, a_M$, by $\mathbf{a}$.

The minimum of (5.11) in Section 5.3.6 occurs where the derivative of χ^2 vanishes with respect to all M parameters a_k. This condition yields the M equations called the *normal equations* of the least-squares problem (Lawson and Hanson, 1987; Press et al., 1992)

$$0 = \sum_{i=1}^{N} \frac{1}{\vartheta_i^2} \left[z_i(x_i, y_i) - \sum_{j=1}^{M} a_j \varphi_j(x_i, y_i) \right] \varphi_k(x_i, y_i), \quad k = 1, \cdots, M. \tag{A.3}$$

Interchanging the order of summations, (A.3) can be rewritten as the matrix equation

$$\sum_{j=1}^{M} \alpha_{kj} a_j = \beta_k, \tag{A.4}$$

where

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{\varphi_j(x_i, y_i)\varphi_k(x_i, y_i)}{\vartheta_i^2}, \quad \text{or equivalently} \quad [\alpha] = \mathbf{A}^T \cdot \mathbf{A}, \tag{A.5}$$

where $[\alpha]$ is a $M \times M$ matrix, and

$$\beta_k = \sum_{i=1}^{N} \frac{z_i(x_i, y_i)\varphi_k(x_i, y_i)}{\vartheta_i^2}, \quad \text{or equivalently} \quad [\beta] = \mathbf{A}^T \cdot \mathbf{b}, \tag{A.6}$$

where $[\beta]$ is a vector of length M.

The equations (A.3) and (A.4) are called the *normal equations of the least squares problem*. In matrix form, the normal equations can be written as either

$$[\alpha] \cdot \mathbf{a} = [\beta], \quad \text{or as} \quad (\mathbf{A}^T \cdot \mathbf{A}) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{b}. \tag{A.7}$$

They can be solved for the vector of parameters $\mathbf{a}$ by Choleksy decomposition, or Gauss-Jordan elimination (Press et al., 1992). In the current study we use the latter which gives us not only the solution vector $\mathbf{a}$ but also the covariance matrix.

A.3 Nelder-Mead simplex algorithm

A simplex is a multidimensional generalization of a triangle (2D) or a tetrahedron (3D). A simplex is chosen as a starting point and with each iteration it moves through the search space in a pre-defined manner. After each iteration its worst vertex is replaced with a vertex which is better than any of its other vertices. This new vertex is found using a set of pre-defined steps (Nelder and Mead, 1965). Mathematically the steps are defined as follows:

1. All vertices are ordered according to the values at each vertex:

$$z(x_1, y_1) \leqslant z(x_2, y_2) \leqslant \ldots \leqslant z(x_{N+1}, y_{N+1}). \tag{A.8}$$

2. The center of gravity (x_0, y_0) of the simplex is calculated without the using the worst vertex (x_{N+1}, y_{N+1}):

$$x_0 = 0.5 \cdot \sum_{i=0}^{i=N} x_i, \quad y_0 = 0.5 \cdot \sum_{i=0}^{i=N} y_i. \tag{A.9}$$

3. Calculate the reflected vertex:

$$x_r = x_0 + \alpha \cdot (x_0 - x_{N+1}), \tag{A.10}$$

$$y_r = y_0 + \alpha \cdot (y_0 - y_{N+1}). \tag{A.11}$$

where α represents the reflection coefficient, with a default and minimum value of 1. The next step of the iteration is determined by evaluating the value of the reflected vertex compared to the other vertices:

(a) if $z(x_1, y_1) \leqslant z(x_r, y_r) < z(x_N, y_N)$ then replace (x_{N+1}, y_{N+1}) with (x_r, y_r) and go to the next iteration

(b) if $z(x_r, y_r) < z(x_1, y_1)$ then calculate the expanded vertex

(c) else calculate the contracted vertex.

4. Determine the expanded vertex:

$$x_e = x_0 + \gamma \cdot (x_0 - x_{N+1}), \tag{A.12}$$

$$y_e = y_0 + \gamma \cdot (y_0 - y_{N+1}). \tag{A.13}$$

with γ denoting the expansion coefficient, with a default value of 2 (always larger than α). Then the following case is evaluated and afterwards the next iteration will start:

$$x_{N+1} = \begin{cases} x_e & \text{if } z(x_e, y_e) < z(x_r, y_r), \\ x_r & \text{otherwise.} \end{cases}, \tag{A.14}$$

$$y_{N+1} = \begin{cases} y_e & \text{if } z(x_e, y_e) < z(x_r, y_r), \\ y_r & \text{otherwise.} \end{cases}. \tag{A.15}$$

5. Determine the contracted vertex:

$$x_c = x_{N+1} + \rho \cdot (x_0 - x_{N+1}), \tag{A.16}$$

$$y_c = y_{N+1} + \rho \cdot (y_0 - y_{N+1}), \tag{A.17}$$

with ρ denoting the contraction coefficient which lies between 0 and 1 with a default value of 0.5. If the contracted vertex is better than the worst vertex $(z(x_c, y_c) \leqslant z(x_{N+1}, y_{N+1}))$, then the worst vertex is replaced by the contracted vertex.

6. Replace all vertices by the reduced vertices:

$$x_i = x_1 + \sigma \cdot (x_i - x_1), \quad \text{where} \quad i \in \{2, \ldots, N+1\}, \tag{A.18}$$

$$y_i = y_1 + \sigma \cdot (y_i - y_1), \quad \text{where} \quad i \in \{2, \ldots, N+1\}, \tag{A.19}$$

with σ representing the reduction coefficient which lies between 0 and 1 with a default value of 0.5.

When all these rules are followed, the method is guaranteed to find a minimum. As with other optimization techniques, the problem of Nelder-Mead method is that it usually finds a local minimum instead of the global minimum. However, this can be prevented by choosing the correct size for the starting simplex so that local minima are skipped. Another possibility is to choose multiple starting simplices. The best vertex can be chosen as a true minimum of the objective function after each simplex converges to a certain point.

A.4 Kinect calibration

In a normal stereo setup, images derived from the calibrated cameras are rectified in order to obtain correspondent horizontal lines. In such a system, the relation between disparity and depth is given by

$$z = b \cdot f/d, \tag{A.20}$$

where z is the depth value (in meters), b is the baseline between two cameras (in meters), f is the focal length of the cameras (in pixels) and d is the disparity value (in pixels). Thus, in the case of zero disparity values the rays from both cameras are parallel and depth is infinite. However, the Kinect device returns raw disparity data which is not normalized in this way. So zero disparity values do not correspond to infinite distances. The relation of raw Kinect disparity to a normalized disparity is given by

$$d = 1/8 \cdot (\text{doff} - \text{kd}), \tag{A.21}$$

where d is the normalized disparity (see (A.20)), kd is the Kinect disparity and $doff$ is the offset value particular to a given Kinect device. Values for kd and $doff$ are found at the calibration stage. Consequently, the relation between disparity and depth for the Kinect is given by

$$z = \frac{b \cdot f}{1/8 \cdot (\text{doff} - \text{kd})}. \tag{A.22}$$

In order to relate color and depth images, pixels of the color image need to be matched to pixels of the depth image. Therefore, a calibration between IR and RGB

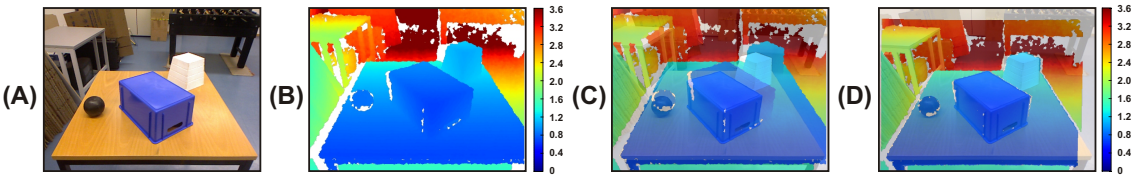

Figure A.2: Calibration of the Kinect with OpenNI toolbox. (A) Original frame acquired by RGB camera. (B) Depth data derived from IR image (in meters). (C) Mapping of depth and color pixels without calibration. (D) Mapping of depth and color pixels after calibration.

cameras needs to be performed [2,3]. First, RGB and IR cameras are calibrated separately. Once intrinsic matrices $\mathbf{K_{rgb}}$ and $\mathbf{K_{ir}}$ and distortion parameters of both cameras are derived, the geometrical relationship between two cameras needs to be obtained. This relation is expressed by a matrix of parameters consisting of a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. $\mathbf{R}$ and $\mathbf{t}$ denote the coordinate system transformations from 3D IR camera coordinates to 3D RGB camera coordinates. Intrinsic and extrinsic parameters of both cameras can be obtained by any of existing calibration toolbox (Bradski, 2000). A pixel p_{ir} from the IR image can be projected to a 3D point in the IR's camera coordinate system taking the depth value of p_{ir} as a z coordinate:

$$\begin{aligned} P_{ir} &= \mathbf{K_{ir}^{-1}} \cdot p_{ir}, \\ P_{ir}^{z} &= z(p_{ir}). \end{aligned} \tag{A.23}$$

P_{ir} is transformed to the RGB's camera coordinate system applying relative transformation $(\mathbf{R}, \mathbf{t})$:

$$P_{rgb} = \mathbf{R} \cdot P_{ir} + \mathbf{t}. \tag{A.24}$$

The derived 3D point is projected to the RGB camera image by

$$p_{rgb} = \mathbf{K_{rgb}} \cdot P_{rgb}, \tag{A.25}$$

obtaining the depth value corresponding to the location p_{rgb} in the RGB image as

$$z(p_{rgb}) = P_{rgb}^{z}. \tag{A.26}$$

[2] see http://nicolas.burrus.name

[3] see http://www.xbox.com/en-US/kinect

In the current work the OpenNI toolbox was used for the Kinect calibration and mapping of color pixels with range values [4].

[4]available under http://www.openni.org

Bibliography

Abramov, A., Aksoy, E. E., Dörr, J., Pauwels, K., Wörgötter, F., and Dellen, B. (2010a). 3D semantic representation of actions from efficient stereo-image-sequence segmentation on GPUs. In *Fifth International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT 2010)*, Paris, France.

Abramov, A., Kulvicius, T., Wörgötter, F., and Dellen, B. (2010b). Real-time image segmentation on a GPU. *Facing the Multicore-Challenge, Lecture Notes in Computer Science*, 6310:131–142.

Abramov, A., Papon, J., Pauwels, K., Wörgötter, F., and Dellen, B. (2012a). Depth-supported real-time video segmentation with the Kinect. In *IEEE workshop on the Applications of Computer Vision (WACV)*, pages 457–464.

Abramov, A., Pauwels, K., Kornewald, W., Wörgötter, F., and Dellen, B. (2012b). Real-time dense disparity from stereo-segment silhouettes for weakly-textured images. *International Journal of Computer Vision (submitted)*.

Abramov, A., Pauwels, K., Papon, J., Wörgötter, F., and Dellen, B. (2012c). Real-time segmentation of stereo videos on a portable system with a mobile GPU. *IEEE Transactions on Circuits and Systems for Video Technology (in press)*.

Aksoy, E. E., Abramov, A., Dörr, J., Ning, K., Dellen, B., and Wörgötter, F. (2011). Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research (IJRR), Special Issue on 'Semantic Perception for Robots in Indoor Environments'*, (30):1229–1249.

Anderson, B. and Nakayama, K. (1994). Toward a general theory of stereopsis. *Physical Review*, 101(3):414–445.

Arbelaez, P., Maire, M., Fowlkes, C. C., and Malik, J. (2009). From contours to regions: An empirical evaluation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2294–2301.

Azencott, R. (1992). *Simulated Annealing: Parallelization Techniques*. Wiley-Interscience, New York.

Bab-Hadiashar, A. and Gheissari, N. (2006). Range image segmentation using surface selection criterion. *IEEE Transactions on Image Processing*, 15(7):2006–2018.

Barkema, G. T. (1992). Ph.D. thesis. *Utrecht University*.

Barkema, G. T. and MacFarland, T. (1994). Parallel simulation of the ising model. *Physical Review E*, 50(2):1623–1628.

Barnard, S. T. (1989). Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32.

Beare, R. (2006). A locally constrained watershed transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1063–1074.

Belhumeur, P. N. (1996). A bayesian approach to binocular steropsis. *International Journal of Computer Vision*, 19(3):237–260.

Bergen, J. R., Anandan, P., Hanna, K. J., and Hingorani, R. (1992). Hierarchical model-based motion estimation. In *European Conference on Computer Vision (ECCV)*, pages 237–252.

Besag, J. (1986). On the statistical analysis of dirty pictures. *J. Royal Statistical Soc., Series B*, 48(3):259–302.

Besl, P. J. and Jain, R. C. (1988). Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:167–192.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY.

Blake, A. and Isard, M. (1998). *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer.

Blatt, M., Wiseman, S., and Domany, E. (1996). Superparamagnetic clustering of data. *Physical Review Letters*, 76(18):3251–3254.

Bobick, A. F. and Intille, S. S. (1999). Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181–200.

Borra, S. and Sarkar, S. (1997). A framework for performance characterization of intermediate-level grouping modules. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:1306–1312.

Boykov, Y. and Funka-Lea, G. (2006). Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision*, 70(2):109–131.

Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *International Conference on Computer Vision (ICCV)*.

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

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Gool, L. J. V. (2009). Robust tracking-by-detection using a detector confidence particle filter. In *International Conference on Computer Vision (ICCV)*, pages 1515–1522.

Brendel, W. and Todorovic, S. (2009). Video object segmentation by tracking regions. In *International Conference on Computer Vision (ICCV)*, pages 833–840.

Brice, C. R. and Fennema, C. L. (1970). Scene analysis using regions. *Artificial Intelligence*, 1:205–226.

Brodtkorb, A. R., Dyken, C., Hagen, T. R., Hjelmervik, J. M., and Storaasli, O. O. (2010). State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33.

Brown, M. Z., Burschka, D., and Hager, G. D. (2003). Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:993–1008.

Brox, T. and Malik, J. (2011). Large displacement optical flow: descriptor matching in variational motion estimation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 500–513.

Brunton, A., Shu, C., and Roth, G. (2006). Belief propagation on the GPU for stereo vision. In *The 3rd Canadian Conference on Computer and Robot Vision*, Washington, DC, USA.

Burt, P. J., Edward, and Adelson, E. H. (1983). The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31:532–540.

Chen, S., Li, Y. F., and Zhang, J. (2008). Vision processing for realtime 3-d data acquisition based on coded structured light. *IEEE Transactions on Image Processing*, 17(2):167–176.

Cheng, H. D., Jiang, X. H., Sun, Y., and Wang, J. L. (2001). Color image segmentation: Advances and prospects. *Pattern Recognition*, 34:2259–2281.

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799.

CIE-Publication-116-1995 (1995). Industrial colour-difference evaluation. Vienna: CIE Central Bureau.

Comaniciu, D., Meer, P., and Member, S. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619.

Compagner, A. and Hoogland, A. (1987). Maximum length sequences, cellular automata, and random numbers. *Journal of Computational Physics*, 71:391–428.

Cour, T., Benezit, F., and Shi, J. (2005). Spectral segmentation with multiscale graph decomposition. In *Computer Vision and Pattern Recognition (CVPR)*, Washington, DC, USA.

Dellen, B., Aksoy, E. E., and Wörgötter, F. (2009). Segment tracking via a spatiotemporal linking process including feedback stabilization in an n-d lattice model. *Sensors*, 9(11):9355–9379.

Dellen, B., Alenyà, G., Foix, S., and Torras, C. (2011). Segmenting color images into surface patches by exploiting sparse depth data. In *IEEE workshop on the Applications of Computer Vision (WACV)*, Kailua-Kona, Hawaii.

Dellen, B. and Wörgötter, F. (2009). Disparity from stereo-segment silhouettes of weakly-textured images. In *British Machine Vision Conference (BMVC)*, London, UK.

Díaz, J., Ros, E., Carrillo, R. R., and Prieto, A. (2007). Real-time system for high-image resolution disparity estimation. *IEEE Transactions on Image Processing*, 16(1):280–285.

Eckes, C. and Vorbrüggen, J. C. (1996). Combining data-driven and model-based cues for segmentation of video sequences. In *World Congress on Neural Networks*, pages 868–875.

Egnal, G. and Wildes, R. P. (2002). Detecting binocular half-occlusions: Empirical comparisons of five approaches. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1127–1133.

Estrada, F., Jepson, A., and Chennubhotla, C. (2004). Spectral embedding and min-cut for image segmentation. In *British Machine Vision Conference (BMVC)*.

Estrada, F. J. and Jepson, A. D. (2009). Benchmarking image segmentation algorithms. *International Journal of Computer Vision*, 85(2):167–181.

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

Felzenszwalb, P. F. and Huttenlocher, D. P. (2006). Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54.

Fleet, D. J. and Jepson, A. D. (1990). Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1):77–104.

Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference.

Gautama, T. and Van Hulle, M. (2002). A phase-based approach to the estimation of the optical flow field using spatial filtering. *IEEE Transactions on Neural Networks*, 13(5):1127–1136.

Geiger, D., Ladendorf, B., and Yuille, A. L. (1995). Occlusions and binocular stereo. *International Journal of Computer Vision*, 14(3):211–226.

Geman, D. and Geman, S. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):721–741.

Gonzalez, R., Woods, R., and Eddins, S. (2003). Digital image processing using matlab, chapter 11.

Grundmann, M., Kwatra, V., Han, M., and Essa, I. A. (2010). Efficient hierarchical graph-based video segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2141–2148.

He, L., Chao, Y., Suzuki, K., and Wu, K. (2009). Fast connected-component labeling. *Pattern Recognition*, 42:1977–1987.

Hedau, V., Arora, H., and Ahuja, N. (2008). Matching images under unstable segmentations. In *Computer Vision and Pattern Recognition (CVPR)*.

Heo, Y. S., Lee, K. M., and Lee, S. U. (2011). Robust stereo matching using adaptive normalized cross-correlation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):807–822.

Hirschmüller, H. (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341.

Hoffmann, K. H., Würtz, D., de Groot, C., and Hanf, M. (1991). Concepts in optimizing simulated annealing schedules: An adaptive approach for parallel and vector machines. In *Parallel and Distributed Optimization*. Springer Verlag, Heidelberg.

Hong, B.-W., Soatto, S., Ni, K., and Chan, T. F. (2008). The scale of a texture and its application to segmentation. In *Computer Vision and Pattern Recognition (CVPR)*.

Huang, Y., Liu, Q., and Metaxas, D. N. (2009). Video object segmentation by hypergraph cut. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1738–1745.

Ising, E. (1925). Beitrag zur Theorie des Ferromagnetismus. *Z. Phys.*, 31:253–258.

Johnson, D. S. and McGeogh, L. A. (1997). The traveling salesman problem: A case study. In *Local Search in Combinatorial Optimization*, pages 215–310.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Kjellström, H., Romero, J., and Kragic, D. (2011). Visual object-action recognition : Inferring object affordances from human demonstration. *Computer Vision and Image Understanding*, 115(1):81–90.

Klingbeil, E., Rao, D., Carpenter, B., Ganapathi, V., Ng, A. Y., and Khatib, O. (2011). Grasping with application to an autonomous checkout robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2837–2844, Shanghai, China.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583.

Kolmogorov, V. and Zabih, R. (2004). What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:65–81.

König, P. and Krüger, N. (2006). Perspectives: Symbols as self-emergent entities in an optimization process of feature extraction and predictions. *Biological Cybernetics*, 94(4):325–334.

Koschan, A. and Abidi, M. A. (2008). *Digital color image processing*. John Wiley and Sons.

Kyriakoulis, N. and Gasteratos, A. (2010). Light-Invariant 3D object's pose estimation using color distance transform. In *IEEE International Conference on Imaging Systems and Techniques (IST 2010)*, pages 105–110, Thessaloniki, Greece.

Ladický, Ľ., Sturgess, P., Russell, C., Sengupta, S., Bastanlar, Y., Clocksin, W., and Torr, P. (2010). Joint optimisation for object class segmentation and dense stereo reconstruction. In *British Machine Vision Conference (BMVC)*.

Lawson, C. L. and Hanson, R. J. (1987). *Solving Least Squares Problems (Classics in Applied Mathematics)*. Society for Industrial Mathematics.

Lee, H.-J. and Lei, W.-L. (1990). Region matching and depth finding for 3d objects in stereo aerial photographs. *Pattern Recognition*, 23(1-2):81–94.

Lempitsky, V. S. and Boykov, Y. (2007). Global optimization for shape fitting. In *Computer Vision and Pattern Recognition (CVPR)*.

Liang, C.-K., Cheng, C.-C., Lai, Y.-C., Chen, L.-G., and Chen, H. H. (2011). Hardware-efficient belief propagation. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(5):525–537.

Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28:39–55.

Liu, C., Freeman, W. T., Adelson, E. H., and Weiss, Y. (2008a). Human-assisted motion annotation. In *Computer Vision and Pattern Recognition (CVPR)*.

Liu, S., Dong, G., Yan, C. H., and Ong, S. H. (2008b). Video segmentation: Propagation, validation and aggregation of a preceding graph. In *Computer Vision and Pattern Recognition (CVPR)*.

Lu, J., Rogmans, S., Lafruit, G., and Catthoor, F. (2009). Stream-centric stereo matching and view synthesis: A high-speed approach on GPUs. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(11):1598–1611.

MacLean, W. J., Sabihuddin, S., and Islam, J. (2010). Leveraging cost matrix structure for hardware implementation of stereo disparity computation using dynamic programming. *Computer Vision and Image Understanding*, 114(11):1126–1138.

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 *8th Int'l Conf. Computer Vision*, volume 2, pages 416–423.

Martin, D. R., Fowlkes, C., and Malik, J. (2002). Learning to detect natural image boundaries using brightness and texture. In *Neural Information Processing Systems (NIPS)*, pages 1255–1262.

Meribout, M. and Nakanishi, M. (2005). A new real time object segmentation and tracking algorithm and its parallel architecture. *Journal of VLSI Signal Processing*, 39(3):249–266.

Mester, R., Conrad, C., and Guevara, A. (2011). Multichannel segmentation using contour relaxation: Fast super-pixels and temporal propagation. In *SCIA*, pages 250–261.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *J. of Chem. Phys.*, 21(11):1087–1091.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).

Mortensen, E. N. and Barrett, W. A. (1999). Toboggan-based intelligent scissors with a four-parameter edge model. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2452–2458.

Mosegaard, K. and Vestergaard, P. D. (1991). A simulated annealing approach to seismic model optimization with sparse prior information. *Geophysical Prospecting*, 39:599–611.

Mutto, C. D., Zanuttigh, P., Cortelazzo, G. M., and Mattoccia, S. (2011). Scene segmentation assisted by stereo vision. In *3DIMPVT*, pages 57–64.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

Nummiaro, K., Koller-Meier, E., and Van Gool, L. (2002). An adaptive color-based particle filter. *Image and Vision Computing*, 21(1):99–110.

NVIDIA-Corporation (2009). Nvidia's next generation CUDA compute architecture: Fermi.

NVIDIA-Corporation (2011). Nvidia cuda c programming guide 4.0. 7:187.

Ohlander, R., Price, K., and Reddy, D. R. (1978). Picture segmentation using a recursive region splitting method.

Opara, R. and Wörgötter, F. (1998). A fast and robust cluster update algorithm for image segmentation in split-lattice models without annealing - visual latencies revisited. *Neural Computation*, 10(6):1547–1566.

Pal, N. R. and Pal, S. K. (1993). A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294.

Papon, J., Abramov, A. Aksoy, E. E., and Wörgötter (2012). A modular system architecture for online parallel visual pipelines. In *IEEE workshop on the Applications of Computer Vision (WACV)*, pages 361–368.

Paris, S. (2008). Edge-preserving smoothing and mean-shift segmentation of video streams. In *European Conference on Computer Vision (ECCV)*, pages 460–473.

Paris, S. and Durand, F. (2007). A topological approach to hierarchical segmentation using mean shift. In *Computer Vision and Pattern Recognition (CVPR)*.

Pauwels, K. and Hulle, M. V. (2009). Optic flow from unstable sequences through local velocity constancy maximization. *Image Vision Computing*, 27(5):579–587.

Pauwels, K., Krüger, N., Lappe, M., Wörgötter, F., and Van Hulle, M. (2010). A cortical architecture on parallel hardware for motion processing in real time. *Journal of Vision*, 10(10).

Pauwels, K., Tomasi, M., Alonso, J. D., Ros, E., and Hulle, M. M. V. (2011). A comparison of fpga and GPU for real-time phase-based optical flow, stereo, and local image features. *IEEE Transactions on Computers*, 99.

Potts, R. B. (1952). Some generalized order-disorder transformations. *Proc. Cambridge Philos. Soc.*, 48:106–109.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA.

Rao, D., Le, Q. V., Phoka, T., Quigley, M., Sudsang, A., and Ng, A. Y. (2010). Grasping novel objects with depth segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Reina, A. V., Avidan, S., Pfister, H., and Miller, E. L. (2010). Multiple hypothesis video segmentation from superpixel flows. In *European Conference on Computer Vision (ECCV)*, pages 268–281.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Sabatini, S. P., Gastaldi, G., Solari, F., Pauwels, K., Hulle, M. M. V., Díaz, J., Ros, E., Pugeault, N., and Krüger, N. (2010). A compact harmonic code for early vision based on anisotropic frequency channels. *Computer Vision and Image Understanding*, 114(6):681–699.

Salamon, P., Nulton, J. D., Robinson, J., Pedersen, J. M., Ruppeiner, G., and Liao, L. (1988). Simulated annealing with constant thermodynamic speed. 49:423–428.

Salamon, P., Sibani, P., and Frost, R. (2010). *Facts, Conjectures, and Improvements for Simulated Annealing*. Society for Industrial and Applied Mathematics.

Salembier, P. and Marqués, F. (1999). Region-based representations of image and video: segmentation tools for multimedia services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1147–1169.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42.

Scharstein, D. and Szeliski, R. (2008). Middlebury stereo vision research page. *http://vision.middlebury.edu/stereo/eval/*.

Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition (CVPR)*, pages 519–528.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Sizintsev, M., Kuthirummal, S., Samarasekera, S., Kumar, R., Sawhney, H., and Chaudhry, A. (2010). GPU accelerated realtime stereo for augmented reality. In *3DPVT10*.

Snavely, N., Seitz, S. M., and Szeliski, R. (2008). Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210.

Sun, J., yeung Shum, H., and ning Zheng, N. (2003). Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:787–800.

Swendsen, R. H. and Wang, S. (1987). Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters*, 76(18):86–88.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.

Tappen, M. F. and Freeman, W. T. (2003). Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. In *International Conference on Computer Vision (ICCV)*, pages 900–907.

Tombari, F., Mattoccia, S., and di Stefano, L. (2007). Segmentation-based adaptive support for accurate stereo correspondence. In *PSIVT'07*, pages 427–438.

Trapp, R., Drüe, S., and Hartmann, G. (1998). Stereo matching with implicit detection of occlusions. In *European Conference on Computer Vision (ECCV)*, pages 17–33.

Tseng, D. and Chang, C. (1992). Color segmentation using perceptual attributes. In *International Conference on Pattern Recognition*, volume 3, pages 228–231.

Tuceryan, M. and Jain, A. K. (1998). *Texture Analysis in The Handbook of Pattern Recognition and Computer Vision (2nd Edition)*. World Scientific Publishing Co.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and its Applications*, 45:41–55.

Unger, M., Mauthner, T., Pock, T., and Bischof, H. (2009). Tracking as segmentation of spatial-temporal volumes by anisotropic weighted tv. In *EMMCVPR*, pages 193–206.

Veksler, O. (2003). Fast variable window for stereo correspondence using integral images. In *Computer Vision and Pattern Recognition (CVPR)*, pages 556–561.

Vicente, S., Kolmogorov, V., and Rother, C. (2008). Graph cut based image segmentation with connectivity priors. In *Computer Vision and Pattern Recognition (CVPR)*.

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*, 13:583–598.

von Ferber, C. and Wörgötter, F. (2000). Cluster update algorithm and recognition. *Physical Review E*, 62(2):1461–1464.

Wang, C. and Abe, K. (1995). Region correspondence by inexact attributed planar graph matching. In *International Conference on Computer Vision (ICCV)*, pages 440–446.

Wang, C., de La Gorce, M., and Paragios, N. (2009). Segmentation, ordering and multi-object tracking using graphical models. In *International Conference on Computer Vision (ICCV)*, pages 747–754.

Wang, J., Wang, J. Y. A., Edward, and Adelson, H. (1994). Representing moving images with layers. *IEEE Transactions on Image Processing*, 3:625–638.

Wedel, A., Pock, T., Zach, C., Cremers, D., and Bischof, H. (2008). An improved algorithm for TV-L1 optical flow. In *Dagstuhl Motion Workshop*, LNCS. Springer.

White, S. R. (1984). Concepts of scale in simulated annealing. In *IEEE International Conference on Computer Design*, pages 646–651.

Wildes, R. P. (1991). Direct recovery of three-dimensional scene geometry from binocular stereo disparity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):761–774.

Wolff, U. (1989). Collective Monte Carlo updating for spin systems. *Physical Review Letters*, 62(4):361–364.

Wyszecki, G. A. and Stiles, W. S. (2000). *Color Science: Concepts and Methods, Quantitative Data and Formulae (Wiley Series in Pure and Applied Optics)*. John Wiley and Sons.

Yang, Q., Engels, C., and Akbarzadeh, A. (2008). Near real-time stereo for weakly-textured scenes. In *British Machine Vision Conference (BMVC)*, pages 80–87.

Yang, Q., Wang, L., and Ahuja, N. (2010). A constant-space belief propagation algorithm for stereo matching. In *Computer Vision and Pattern Recognition (CVPR)*.

Yang, Q., Wang, L., Yang, R., Wang, S., Liao, M., and Nistér, D. (2006). Real-time global stereo matching using hierarchical belief propagation. In *British Machine Vision Conference (BMVC)*, pages 989–998.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2000). Generalized belief propagation. In *Advances in Neural Information Processing Systems*, pages 689–695.

Yoon, K., Member, S., and Kweon, I. S. (2006). Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:650–656.

Zhu, J., Wang, L., Yang, R., Davis, J. E., and Pan, Z. (2011). Reliability fusion of time-of-flight depth and stereo geometry for high quality depth maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1400–1414.

Zitnick, C. L., Kanade, T., and Government, S. (1999). A cooperative algorithm for stereo matching and occlusion detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:675–684.

B

Curriculum Vitae

ALEXEY ABRAMOV

Research Assistant at the Bernstein Center for Computational Neuroscience
Georg-August-Universität Göttingen
III Physikalisches Institut - Biophysik
Friedrich-Hund Platz 1
37077 Göttingen

Date and place of birth:	17 May 1985 Moscow, Russian Federation
Citizenship:	Russian
E-mail:	abramov@physik3.gwdg.de
Tel.:	+49(0) 551 3910 764

EDUCATION

2008 Apr – 2012 Jul	PhD Student at the Department of Computer Science Georg-August-Universität Göttingen Germany
2002 Sep – 2008 Feb	M.Sc. and B.Sc. in Computer Science Moscow Ingineering Physics Institute (State University) Faculty of Cybernetics Moscow, Russia

PROFESSIONAL EXPERIENCE

2008 Apr – present	Research assistant Georg-August-Universität Göttingen Germany
2005 Aug – 2007 Oct	Research assistant All Russian Scientific Research Institute for Control Automatization in Unprofitable Field Moscow, Russia
2004 Aug – 2005 Mar	Software developer Sputnik Labs Moscow, Russia

RESEARCH INTERESTS

Computer vision and image processing
Parallel computing and architectures
Image segmentation
Video segmentation and object tracking
Stereo vision and depth perception
Real-time visual systems

LIST OF PUBLICATIONS

Journal Papers

- Abramov, A., Pauwels, K., Kornewald, W., Wörgötter, F. and Dellen, B. Real-time Dense Disparity from Stereo-segment Silhouettes for Weakly-textured Images. *International Journal of Computer Vision* (submitted), 2012.

- Abramov, A., Pauwels, K., Papon, J., Wörgötter, F. and Dellen, B. Real-time Segmentation of Stereo Videos on a Portable System with a Mobile GPU. *IEEE Transactions on Circuits and Systems for Video Technology* (in press), 2012.

- Aksoy, E.E., Abramov A., Drr, J., Ning, K., Dellen, B. and Wörgötter, F. Learning the semantics of objectaction relations by observation. *International Journal of Robotics Research (IJRR), Special Issue on Semantic Perception for Robots in Indoor Environments*, 30: 1229-1249, 2011.

Conference Papers

- Abramov A., Papon, J., Pauwels, K., Wörgötter, F., Dellen, B. Depth-supported real-time video segmentation with the Kinect. *IEEE workshop on the Applications of Computer Vision (WACV 2012)*, Breckenridge, Colorado, USA, January 9-11, 2012.

- Papon J., Abramov, A., Aksoy, E.E., Wörgötter, F. A Modular System Architecture for Online Parallel Visual Pipelines. *IEEE workshop on the Applications of Computer Vision (WACV 2012)*, Breckenridge, Colorado, USA, January 9-11, 2012.

- Abramov A., Aksoy, E.E., Dörr, J., Pauwels, K., Wörgötter, F. and Dellen, B. 3D Semantic Representation of Actions from efficient stereo-image-sequence segmentation on GPUs. *Fifth International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT 2010)*, Paris, France, May 17-20, 2010.

- Aksoy, E.E., Abramov, A., Wörgötter, F. and Dellen, B. Categorizing Object-Action Relations from Semantic Scene Graphs. *IEEE International Conference on Robotics and Automation (ICRA 2010)*, Alaska, USA, May 3-8, 2010.

- Abramov A., Kulvicius, T., Wörgötter, F. and Dellen, B. Real-time image segmentation on a GPU. *Facing the Multicore-Challenge, Conference for young scientists*, Heidelberg, Germany, March 17-19, 2010.

Patent pending

- Abramov, A., Aksoy E.E., Dellen, B., Wörgötter, F. Method and Device for Estimating Development Parameters of Plants, Public law foundation of the Georg-August Universität Göttingen filed with European Patent Office, 2011.

Abstract

- Abramov, A., Papon, J., Wörgötter, F. Oculus Real-time Modular Cognitive Vision System. *Nvidia GPU technology conference (GTC)*, San Jose, California, USA, May 14-18, 2012.