Motion-based Segmentation and Classification of Video Objects

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.-Wirtsch.-Inf. Gerald Kühne
aus Kassel

Mannheim, 2002

Dekan:	Professor Dr. Herbert Popp, Universität Mannheim
Referent:	Professor Dr. Wolfgang Effelsberg, Universität Mannheim
Korreferent:	Privatdozent Dr. Jürgen Hesser, Universität Mannheim

Tag der mündlichen Prüfung: 09. Oktober 2002

Abstract

In this thesis novel algorithms for the segmentation and classification of video objects are developed. The segmentation procedure is based on motion and is able to extract moving objects acquired by either a static or a moving camera. The classification of those objects is performed by matching their outlines gathered from a number of consecutive frames of the video with preprocessed views of prototypical objects stored in a database.

This thesis contributes to four areas of image processing and computer vision: motion analysis, implicit active contour models, motion-based segmentation, and object classification. In detail, in the field of motion analysis, the tensor-based motion estimation approach is extended by a non-maximum suppression scheme, which improves the identification of relevant image structures significantly. In order to analyze videos that contain large image displacements, a feature-based motion estimation method is developed. In addition, to include camera operations into the segmentation process, a robust camera motion estimator based on least trimmed squares regression is presented.

In the area of implicit active contour models, a model that unifies geometric and geodesic active contours is developed. For this model an efficient numerical implementation based on a new narrow-band method and a semi-implicit discretization is provided. Compared to standard algorithms these optimizations reduce the computational complexity significantly. Integrating the results of the motion analysis into the fast active contour implementation, novel algorithms for motion-based segmentation are developed. Since both static and mobile cameras are taken into account, motion-based segmentation is possible even when camera motion is present.

In the field of object classification, a shape-based classification approach is extended and adapted to image sequence processing. Finally, a system for video object classification is derived by combining the proposed motion-based segmentation algorithms with the shape-based classification approach.

For each contribution experimental results are reported that demonstrate the effectiveness of the proposed algorithms.

Acknowledgments

This thesis was written while I was working as a research assistant at the "Lehrstuhl für Praktische Informatik IV" at the University of Mannheim, Germany. Many people supported and influenced my work, and I would like to take this opportunity to express my gratitude to them.

First of all, I would like to thank Prof. Dr. Wolfgang Effelsberg, who gave me the opportunity to pursue my ideas in the Movie Content Analysis (MoCA) group. His support, feedback and advice have been an indispensable help. Dr. Jürgen Hesser deserves many thanks for acting as a referee for this thesis. He introduced me to image processing during my time as a graduate student in his research group.

In particular, I wish to thank Prof. Dr. Joachim Weickert for his excellent lectures and many helpful discussions on partial differential equations in image processing. With pleasure I remember our joint research on implicit active contour models.

I am very grateful to my colleagues, who made our department a great place to work at. Their cooperation helped me through all problems I encountered during my work. Silvia Pfeiffer, Rainer Lienhart, and Stephan Fischer, who together pioneered the MoCA project introduced me to the subject and provided many useful tips. With Christoph Kuhmünch I worked on multimedia communication and we also shared many hours preparing exercises and exams for our students. I also wish to thank Dirk Farin, Thomas Haenselmann, Stephan Kopf, Claudia Schremmer, Oliver Schuster, and Markus Beier for excellent team work and many joint projects. The frequent discussions with Holger Füßler, Volker Hilt, Martin Mauve, Jürgen Vogel, Rüdiger Weis, and Jörg Widmer opened up many different perspectives. From Walter Müller I learned a lot about Unix/Linux systems. Betty Haire Weyerer was a great help in overcoming many hurdles with respect to the English language.

Last but not least, I thank my family: Helga, for her support and patience, and Nils and Lina, who reminded me that human vision and thoughts are much more important than computer vision.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *Making a computer see was something that leading experts in the field of Artificial Intelligence thought to be at the level of difficulty of a summer student's project back in the sixties.*
>
> Olivier Faugeras in 2001 [46]

Despite very optimistic predictions[1] in the early days of Artificial Intelligence research, a computer vision system that interprets image sequences acquired from arbitrary real-world scenes remains out of reach to this day.

Nevertheless, there has been great progress in the field since then, and a number of applications have emerged within different areas. Computer vision techniques are used in the industrial manufacturing process, for instance, to navigate industrial robots or inspect products during quality control. Security and surveillance applications are based on machine vision algorithms that analyze biometric properties like faces and fingerprints or monitor sensitive areas. Gesture recognition and the tracking of facial expressions help to define new paradigms in human computer interaction. Furthermore, computer vision-based road and traffic analysis might someday lead to automatic guidance of road vehicles. Medical image analysis applied to the spatial image sequences acquired by computer tomography or magnetic resonance imaging can indicate areas of interest and provides additional information about individual patients. Systems for document analysis

[1] Compare, for instance, Herbert A. Simon who predicted in 1965 that *machines will be capable, within twenty years, of doing any work that a man can do*, or recall a statement from the Dartmouth summer conference in 1956 (Dartmouth Summer Research Project on Artificial Intelligence): "Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.".

and image/video indexing support user retrieval of relevant information from large media archives. Computer vision techniques are employed in the entertainment industry in the production process of movies and in media distribution.

Of particular interest with regard to several applications in the above-mentioned fields are capabilities for *object segmentation* and *object recognition*. Algorithms of the former category support the segregation of the observed world into semantic entities, providing a transition from signal processing towards an object-oriented view. Object recognition approaches support the classification of objects into categories and enable conceptual representations of still images or video sequences.

The goal of this thesis is to develop a classification system for video objects. By video object we mean the collection of all two-dimensional appearances (projections) of a real-world object within an image sequence. A single appearance is called an *object view*. The classification results obtained with our system can be used to index and/or categorize videos and thus support object-based video retrieval.

In order to keep the subject manageable, the algorithms developed throughout the thesis are embedded into a set of constraints. First of all, since visual motion as perceived from changing color (or gray-value) patterns in successive frames is a specific property of video sequences (in contrast to still images), our segmentation algorithms rely on *motion cues*. Focusing on motion allows the segmentation of objects regardless of color or texture and thus captures a wide range of possible object variations. Second, our approach to video object classification relies solely on an object's shape and assumes that a collection of segmented object views is available.

To categorize the components of our approach systematically, let us follow a systems approach as proposed by Nagel [94]. Figure 1.1 depicts an image sequence evaluation system structured in layers. Real-world scenes are captured at the lowest layer, the sensor-actuator level (SAL), for instance, by a video camera. The corresponding signal stream is analyzed at the image signal level (ISL), which performs *local* signal processing. The results from the ISL are fed forward to the picture domain level (PDL) and aggregated spatio-temporally by *non-local* approaches. The PDL provides so-called picture domain cues to the scene domain level (SDL), where scene knowledge, e. g., an illumination model, is exploited. Results from the SDL will then be converted to conceptual representations within the conceptual primitives level (CPL). The behavioral representation level (BRL) aggregates SDL information into an even more abstract representation and might lead to a text description of the observed scene (natural language level (NLL)). Note that there are also feedback loops between the different layers.

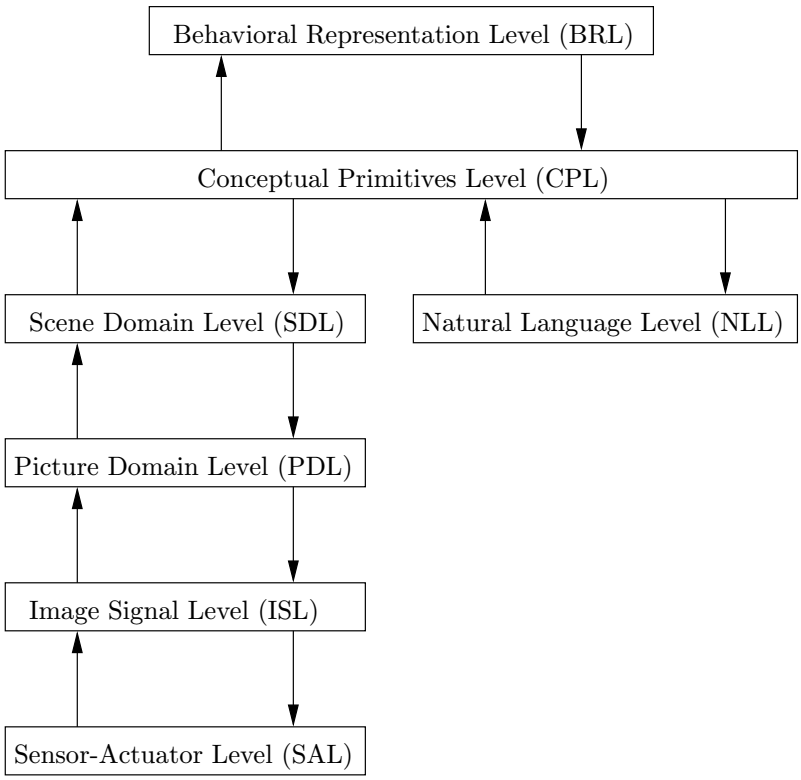

Figure 1.1: Layered structure of an Image Sequence Evaluation system (from [94]).

Thus, in accordance with the above model, the approach developed throughout this thesis comprises four levels, namely ISL, PDL, SDL, and CPL. However, we concentrate on the bottom-up process and do not consider feedback loops included in the model. First, local operators determine image features and their motion characteristics from the acquired image sequence. Then, for each frame of the sequence these features are grouped in order to approximate semantically meaningful objects, or so-called object views. To allow for both static and moving cameras, a global motion model is assumed and its parameters are calculated from the motion estimation results. Finally, object views from consecutive frames are used for video object classification.

1.1 Contributions

This thesis contributes to four areas of computer vision: motion analysis, implicit active contour models, motion-based segmentation, and object classification.

Motion Analysis

Two techniques for motion estimation have been developed. In both cases we introduce optimizations to earlier algorithms which facilitate subsequent object segmentation tasks.

The first technique relies on the three-dimensional structure tensor. In this approach, motion is calculated from orientation estimates in spatio-temporal neighborhoods. Since spatial coordinates and the temporal axis are treated equally, tensor-based motion estimation turns out to be robust to noise. Our contribution is the development of a non-maximum suppression scheme for the 3D structure tensor. This scheme enables an accurate localization of the boundaries of moving objects. Consequently, motion-based segmentation algorithms that employ the structure tensor with non-maximum suppression benefit from the additional precision.

Second, since the tensor-based approach is limited to small image displacements, we develop a feature-based motion estimator. In this approach, image features that can be reliably tracked are extracted and the corresponding motion field is determined. Here, we develop specific reliability measures to improve the feature density.

In addition, in order to cope with moving-camera scenarios, we develop a novel algorithm for robust camera motion estimation based on a global motion model. This approach relies on the motion estimation techniques mentioned above. The model parameters are calculated from the motion information by least trimmed squares regression (LTS), a robust regression technique which has not been considered for this task before.

Implicit Active Contour Models

To group image features into regions, we employ implicit active contour models. Their main advantages are topological adaptivity and numerical stability. However, commonly used implementations of those models are computationally expensive. Thus, we develop efficient implementations of the geometric active contour model and the geodesic active contour model. In particular, we provide a novel narrow-band algorithm that limits the calculations to a small area around the evolving contour. Additionally, a unified semi-implicit updating scheme is presented for both models. These implementations reduce the computational complexity significantly compared to standard algorithms.

Motion-based Segmentation

For motion-based segmentation, novel algorithms are developed by integrating the results from motion estimation and camera motion estimation into the fast implicit active contour

algorithms. Since both static and moving cameras are taken into account, motion-based segmentation is possible even when camera motion is present.

Object Classification

Based on the segmentation results object classification is performed. For this purpose, a shape-based classification approach is extended and adapted to image sequence processing. The object classification is achieved by matching a collection of object views extracted from consecutive frames with preprocessed views of prototypical objects stored in a database. By adapting the database appropriately, different problem domains can be addressed.

1.2 Structure of the Dissertation

This dissertation is structured as follows. Chapter 2 presents our approaches in the field of motion analysis. Following a discussion of the structure tensor for motion estimation, the non-maximum suppression extension is developed. The limitations of this approach are outlined, and as a remedy, feature-based motion estimation is introduced. The chapter concludes by presenting the robust camera motion estimator.

Chapter 3 introduces active contour models. First, explicit active contour models, where the evolving contour is modeled by a parametric curve, are summarized and their limitations are discussed. Second, implicit active contour models that solve several problems of their explicitly modeled counterparts are presented. Since standard algorithms for implicit active contours suffers from computational complexity, efficient numerical implementations are developed for this type of model. Then, by integrating the results of Chapter 2 into the fast contour evolution methods, motion-based segmentation algorithms are developed.

Chapter 4 presents our system for video object classification. First, an efficient shape-based representation for video objects is discussed. Afterwards, matching algorithms for single object views and for entire video objects are provided. Combining the motion-based segmentation algorithms with the classification approach yields our system for video object classification.

Chapter 5 concludes this thesis by providing a summary and discussing future work related to object segmentation and classification.

Chapter 2

Motion Analysis in Video Sequences

2.1 Introduction

An image sequence from a video is a collection of single frames that have been recorded at consecutive points in time. Thus, in addition to the spatial information already contained in still images, techniques for *image sequence evaluation* [94] also can exploit temporal information. *Motion analysis* which relies on changes in the image intensities over time, is fundamental in this context.

Motion analysis might indicate *scene motion* resulting from a moving camera. In the context of *object segmentation*, motion cues provide regions of interest and enable the pursuit of those regions over time (*object tracking*). Furthermore, motion information facilitates the inference of three-dimensional scene structures *(structure-from-motion)*. Various applications ranging from video compression to video content analysis rely on techniques of motion analysis [86, 92, 94].

Image sequence evaluation, and in particular motion analysis, has been under study for over three decades now. First publications on image sequence analysis date back to the '60s, and an early review on techniques for analyzing image sequences was provided by Nagel in 1978 [91]. Reviews dedicated particularly to motion analysis can be found in [10, 50, 86].

A widely acknowledged categorization discriminates the different motion analysis techniques into *gradient-based*, *correspondence-based*, and *frequency-based* approaches. Since our work concentrates on the former two categories, a brief overview of them is given in the following. With regard to frequency-based methods we refer to [10, 86].

Gradient-based methods estimate motion by calculating spatial and temporal derivatives of image intensities. Let us represent an image sequence as a function $f(x, y, t) \in \mathbb{R}$,

where x, y denote the spatial coordinates and t is time. Under the assumption that local intensity structures remain constant under motion during short periods, the following approximation is valid:

$$f(x, y, t) \approx f(x + u(x, y), y + v(x, y), t + \tau). \tag{2.1}$$

Here, $(u(x, y), v(x, y))$ denotes the two-dimensional velocity at the position (x, y) and τ is the time step. Application of Taylor series expansion yields the so-called *optical flow constraint equation*

$$\partial_x f u + \partial_y f v + \partial_t f = 0, \tag{2.2}$$

where $\partial_k, k \in \{x, y, t\}$, denote the spatial and temporal partial derivatives.[1]

Obviously, it is not possible to determine both velocity components from a single linear equation. Instead, additional constraints have to be imposed on the problem [49, 72, 93, 141]. Gradient-based methods demand small displacements because large image motions might lead to aliasing effects and thus render accurate numerical differentiation impractical.

Correspondence-based motion analysis techniques can handle displacements of any size. Motion is calculated in these approaches by identifying corresponding image structures in consecutive frames. Appropriate image structures include unprocessed image regions, image blocks of a predefined size, corners, and edges [17]. Those structures can be matched in consecutive frames in different ways, e. g., by relying on a two-dimensional search within a window, graph theoretic methods, or relaxation labeling [86].

The following discussion on motion analysis is guided by the segmentation problem that is one of the main topics of this thesis. Thus, the motion analysis techniques presented below are particularly optimized to facilitate subsequent motion-based segmentation algorithms. First, we present two motion analysis techniques, namely, *tensor-based motion estimation*, which belongs to the class of gradient-based approaches, and *feature-based motion estimation*, which relies on the correspondence of image features. By applying these techniques, moving image areas can be reliably detected. Thus, assuming a static camera, subsequent object segmentation steps discussed in Chapter 3 are enabled to extract moving objects from those image areas. However, in the event of a moving camera, the simple provision of moving image areas to the segmentation algorithm will not be sufficient. Instead, in order to separate objects from the background, information about

[1] We employ the usual short notations for partial derivatives, e. g., $\frac{\partial f}{\partial x}$ is abbreviated by $\partial_x f$.

the camera motion must also be available. We develop therefore a robust camera motion estimator that determines the camera parameters from a given motion vector field.

In the subsequent sections the following novelties are presented:

(1) A gradient-based motion estimation technique, the three-dimensional structure tensor, is equipped with a non-maximum suppression scheme. The use of this scheme enables a more accurate localization of the boundaries of moving objects. Consequently, subsequent motion-based segmentation approaches benefit from this improvement.

(2) On the basis of specific reliability measures, a feature-based motion estimation technique is developed. The reliability measures increase the feature density significantly compared to other, widely used schemes. This also improves subsequent segmentation stages.

(3) A robust camera motion estimator is developed. The camera parameters are determined from the motion computations by least trimmed squares regression, a robust regression technique that has not been considered for this task before.

The remainder of this chapter is structured as follows: First of all, Section 2.2 discusses related work. Section 2.3 introduces tensor-based motion analysis of spatio-temporal images (two-dimensional) and spatio-temporal volumes (three-dimensional) and provides our extensions, which enhances the localization of relevant image features significantly. In Section 2.4, the feature-based motion analysis technique and the corresponding reliability measures are presented. Section 2.5 discusses our approach to robust estimation of camera parameters based on least trimmed squares regression.

2.2 Related Work

Let us first discuss related work concerning tensor-based motion estimation. The idea of a structure tensor for estimating local orientations dates back to [41, 63].

Bigün, Granlund, and Wiklund [14, 15] and *Jähne* [53] extended the basic principles of the structure tensor to include spatio-temporal volumes and provided approaches to motion estimation in image sequences. In addition, they developed reliability measures that indicate at each image position whether or not motion estimation is possible.

Haußecker and Jähne [47] introduced an extension to the structure tensor approach for estimating dense motion vector fields. They employed a backprojection technique in order to calculate motion estimates in regions suffering from the aperture problem.

However, none of these approaches employ the underlying spatio-temporal structure to adapt the structure tensor to local image structures. Such an adaptation might improve accuracy and robustness, and would thus facilitate subsequent segmentation steps.

Nagel and Gehrke [94,95] proposed a spatio-temporally adaptive *gray-value local structure tensor* (GLST). In their approach, an initial structure tensor is computed and then used to adapt subsequent computations to the underlying spatio-temporal structure.

Spies and Scharr [121] developed an anisotropic diffusion process based on the 3D structure tensor in order to improve the robustness of optical flow computation. They employed coherence enhancing diffusion [137, 138], a technique that smooths the image/volume along gray-value flow lines. Thus, the calculations are adapted to local gray-value structures.

Our work differs from the latter two approaches in that we do not employ an iterative strategy. Instead, the standard structure tensor is calculated within the spatio-temporal domain. Then, using non-maximum suppression, only those image positions are retained where our reliability measure indicates local maxima.

Next, we discuss related work concerning feature-based motion estimation. Here, techniques for feature (or so-called interest point) detection and a correspondence process for mapping those features from one image to another are developed [86].

A number of interest-point detectors have been proposed that can be categorized as contour-based, intensity-based, or parametric model-based methods [111]. A popular feature detection method based on image intensities was developed by *Harris and Stephens* [45]. In their approach, the local neighborhood around an image position is captured by a matrix that averages derivatives of the signal. Interestingly, this is similar to a two-dimensional variant of the structure tensor as mentioned above.

The correspondence process is based on template matching. For this purpose, the cross-correlation of templates from different images can be calculated or distance functions based on similarity can be employed [17, 86].

While we use an approach similar to that proposed in [45], our non-maximum suppression scheme differs in that it retains considerably more interest points. In our context, this is a prerequisite for a successful segmentation.

With regard to camera motion estimation, our work is closely related to that of [46, 130]. In their work, feature-based correspondences are established between consecutive frames. Under the assumption of a global motion model, these correspondences are exploited to calculate the model parameters. Our work differs in that we rely on correspondences resulting from tensor-based motion estimation or feature-based motion

estimation. In addition, we employ least trimmed squares regression for robust determination of the model parameters.

2.3 Tensor-based Motion Estimation

Within consecutive frames stacked on top of each other, a video sequence can be represented as a three-dimensional volume with two spatial coordinates (x, y) and one temporal coordinate (t). Then, under the assumption of a non-varying illumination, stationary parts of a scene will result in lines of equal gray values that run parallel to the time axis t. Moving parts, however, will produce iso-gray-value lines of different orientations.

Figure 2.1 illustrates this observation by means of the *hall-and-monitor* sequence. In this sequence, recorded with a static camera, two persons are walking down a hall. As we can see from Figure 2.1b, the static parts in the scene like the floor and the walls yield iso-gray value lines running parallel to the temporal axis. The walking persons, however, cause gray-value flows in other directions (cf. Figure 2.1c). From this perspective, motion in image sequences can be estimated by analyzing *orientations* of local gray-value structures [15].

Figure 2.1: Spatio-temporal volume created from the *hall-and-monitor* sequence. (a) LEFT: Entire volume, (b) MIDDLE: Cut taken at vertical position y_0, (c) RIGHT: Cut taken at vertical position y_1.

A specific property of the tensor-based approach is the inclusion of multiple frames into the motion estimation process. Calculations are performed in a spatio-temporal volume; thus, all dimensions are treated equally. In the following two sections, we summarize the main properties of tensor-based motion estimation [15, 53, 54]. To illustrate the basic concepts, we first consider only spatio-temporal images (one spatial coordinate (x) and one temporal axis). For instance, those images might result from a one-dimensional pattern

moving over time. Then, we discuss tensor-based motion estimation in spatio-temporal volumes (two spatial coordinates (x, y) and one temporal axis t) created from consecutive frames of an image sequence.

2.3.1 Tensor-based Motion Estimation in Spatio-temporal Images

To simplify matters we start our analysis in the two-dimensional domain, i. e., our coordinate system is restricted to one spatial (x) and one temporal dimension (t). Thus, in the following a spatio-temporal image is denoted by $f(x, t)$.

Consider a one-dimensional pattern that moves constantly to the right against a static background, as depicted in Figure 2.2. Hence, with respect to the temporal axis, the spatio-temporal image $f(x, t)$ consists of sloped iso-gray-value lines in areas of motion and straight iso-gray-value lines in static regions. For certain image areas (areas A_1, A_2 in Figure 2.2a) or so-called *local neighborhoods*, a single vector n can be found, along which the gray values remain constant. Furthermore, the gray values change only in these areas in the direction perpendicular to n. A local neighborhood characterized in this way is called a *linear symmetry* [14] or *simple neighborhood* [42]. We denote the axis represented by the unit vector n (and of course also by $-n$) as *local orientation.* To put it differently, local orientation is given by the direction of constant gray values.[2]

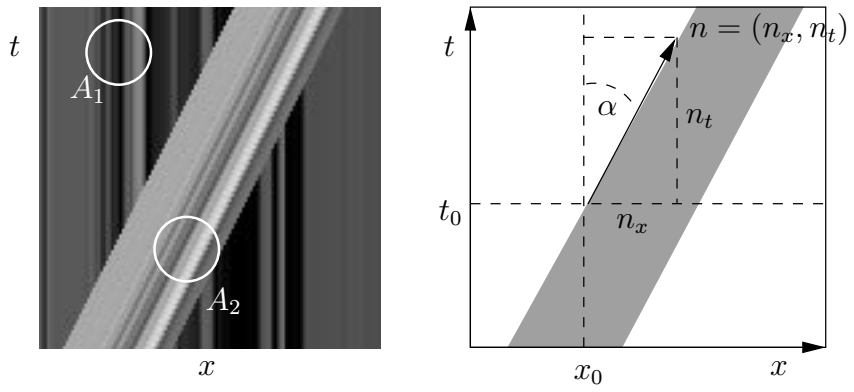

Figure 2.2: Two-dimensional spatio-temporal images. (a) LEFT: One-dimensional pattern against a textured background moving constantly to the right, (b) RIGHT: Schematic diagram of the moving one-dimensional pattern.

Determining motion is closely related to the orientation of local gray-value structures.

[2]Note that local orientation is often represented by the vector $n_\perp$ (and $-n_\perp$) [54]. However, in the setting of motion estimation we feel that our definition is more intuitive.

When estimating motion we are interested in following a certain structure contained in an image over time. Thus, under the assumption of constant illumination, the direction to follow is the one with the least gray-value deviation. In a simple neighborhood this direction is determined by the local orientation. Figure 2.2b illustrates this relationship by means of a gray one-dimensional pattern moving constantly to the right. The angle α indicates how fast the one-dimensional pattern is moving. From the vector $n = (n_x, n_t)^T$, the corresponding one-dimensional velocity v can be determined directly as $v = n_x/n_t$.

How can we determine local orientation and thus estimate motion? A straightforward approach is to model local orientation by using the spatio-temporal gradient $\nabla f = (\partial_x f, \partial_t f)^T$. Since the local orientation within simple neighborhoods is perpendicular to the gradient, we can determine the unit vector $n = (n_x, n_y)^T$ from the equations

$$n^T \cdot \nabla f(p) = 0 \quad \text{and} \quad \left(n_x^2 + n_y^2\right)^{1/2} = 1 \tag{2.3}$$

where $n^T \cdot \nabla f(p)$ denotes the scalar product. Consequently, we can formulate a gradient-based approach to motion estimation in simple neighborhoods as follows:

(1) Calculate the spatio-temporal gradient $\nabla f = (\partial_x f, \partial_t f)^T$ at each image position $p = (x, t)$.

(2) If $\nabla f \neq 0$, determine the local orientation $n(p) = (n_x, n_t)^T$ from the gradient according to Equation 2.3.

(3) If $n_t \neq 0$, then calculate the one-dimensional velocity $v(p) = n_x/n_t$.

Figure 2.3 illustrates motion estimation results obtained for a synthetic sequence by means of the gradient-based method. Figure 2.3a depicts a constantly moving pattern, while Figure 2.3b shows the motion vectors calculated by means of the gradient-based approach. Of course, only at the boundaries of the moving object can motion vectors be calculated. In real-world video sequences however small gray-value changes always occur due to noise and render our ideal scenario inappropriate. Thus, a simple neighborhood can no longer be assumed. As depicted in Figure 2.3d, a small amount of additive Gaussian noise disturbs the motion estimates obtained from gradient directions severely.

Therefore, it is useful to relax our definition of local orientation. Instead of requiring constant gray values along lines of local orientation, we now state local orientation as the axis defined by the direction of the *least* gray-value change. Furthermore, we can add reliability by determining local orientation not just from a single image position but from an entire local neighborhood of a certain size.

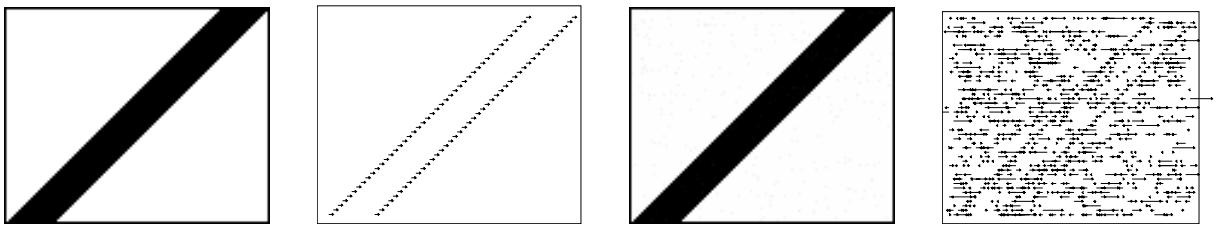

Figure 2.3: Gradient-based motion estimation. FROM LEFT TO RIGHT: (a) One-dimensional pattern moving constantly to the right, (b) motion vectors obtained from motion estimation on (a), (c) one-dimensional moving pattern with additive Gaussian noise, (d) motion estimation on (c).

Then, under ideal conditions, i. e., a linear symmetric neighborhood without noise, the local orientation is given as the unit vector n being perpendicular to all gray-value gradients in a local neighborhood Ω. However, taking local variations into account, we minimize at each image position $p = (x, t)$

$$\int\limits_{p' \in \Omega(p)} w(p - p')(\nabla f(p')^T \cdot n)^2 dx'\, dt', \tag{2.4}$$

where w denotes a Gaussian weighting function. Since the squared scalar product $\left(\nabla f^T \cdot n\right)^2$ is large for gradients parallel or anti-parallel to the vector n, unfavorable directions are penalized.

As derived in Appendix A, minimizing Equation 2.4 is equivalent to determining the eigenvector of the minimum eigenvalue of the *structure tensor*

$$J = \begin{bmatrix} j_{xx} & j_{xt} \\ j_{xt} & j_{tt} \end{bmatrix}, \tag{2.5}$$

where the matrix elements $j_{kl}, k, l \in \{x, t\}$ are calculated from

$$j_{kl}(p) = \int\limits_{p' \in \Omega(p)} w(p - p') \left(\partial_k f(p') \partial_l f(p')\right) dx'\, dt'. \tag{2.6}$$

Consequently, we can determine local orientation by solving the eigenvalue problem. Since the matrix J is symmetric, it possesses orthonormal eigenvectors $v_1 = (v_{1x}, v_{1t})^T$, $v_2 = (v_{2x}, v_{2t})^T$ and real eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$ given as

$$\lambda_{1,2} = \frac{1}{2}\left(j_{xx} + j_{tt} \pm \sqrt{(j_{xx} - j_{tt})^2 + 4j_{xt}^2}\right). \tag{2.7}$$

By analyzing the two eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$ of the structure tensor, we can classify the motion of the local neighborhood. In general, an eigenvalue λ_i greater than zero indicates that the gray values change in the direction of the corresponding eigenvector v_i. We can discriminate three combinations of eigenvalue magnitudes. Figure 2.4 illustrates the different cases.

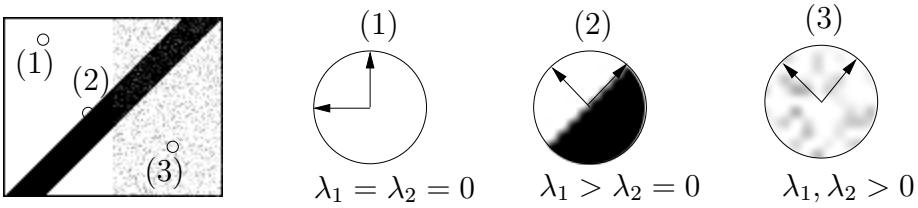

Figure 2.4: Moving one-dimensional pattern. The right half of the spatio-temporal image is distorted by Gaussian noise.

(1) $\lambda_1 = \lambda_2 = 0$
Both eigenvalues equal to zero indicates an area of constant gray values. It is not possible to determine a unique local orientation. Therefore, motion cannot be estimated.

(2) $\lambda_1 > 0, \lambda_2 = 0$
If $\lambda_1 > 0$ and $\lambda_2 = 0$, gray values change only in one direction. Consequently, the local orientation is given by the eigenvector v_2, and we can calculate velocity $v = v_{2x}/v_{2t}$.

(3) $\lambda_1 > 0, \lambda_2 > 0$
If both eigenvalues are greater than zero, gray values change in both directions. This is the case, for instance, in noisy image areas. Hence, local orientation cannot be identified uniquely.

In real-world video sequences, it is impractical to compare the eigenvalues to zero since due to noise in the sequence small gray-value changes always occur. Thus, we introduce a normalized coherence measure c that quantifies the certainty of the calculations. The coherence measure c indicates whether or not a reliable calculation of motion is possible.

It is defined as

$$c = \begin{cases} 0 & \lambda_1 = \lambda_2, \\ \exp\left(\frac{-C}{|\lambda_1 - \lambda_2|}\right) & \text{else} \end{cases} \tag{2.8}$$

where $C > 0$ denotes a contrast parameter. Figure 2.5 depicts the coherence measure for different contrast parameters. Areas with $|\lambda_1 - \lambda_2| \ll C$ are regarded as almost constant local neighborhoods [138]. A value of the coherence measure c near 1.0 indicates that $\lambda_1 \gg \lambda_2$, therefore motion can be estimated reliably. The coherence measure approaches zero if the eigenvalues are equal or do not differ significantly.

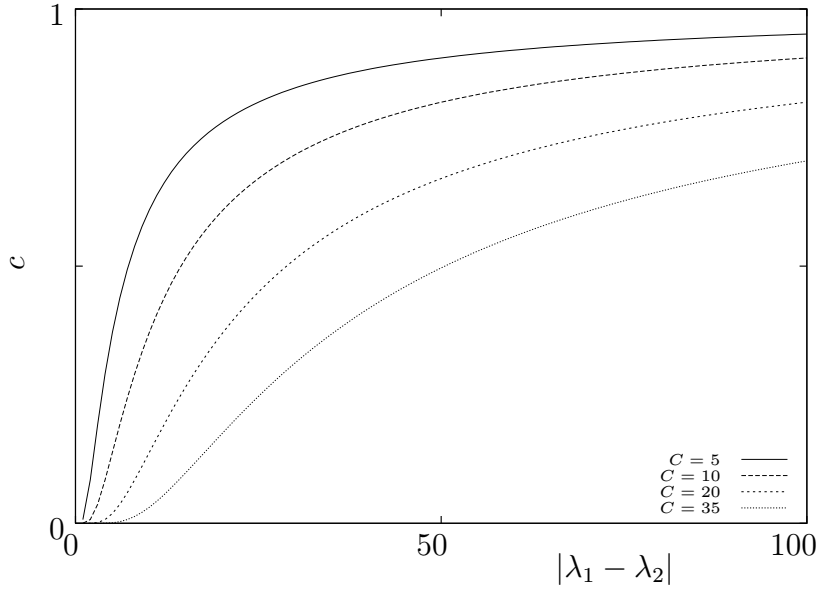

Figure 2.5: Coherence measure c for different contrast parameters.

Thus, by using the structure tensor in combination with Equation 2.8, we can refine our simple motion estimation approach as follows:

(1) Calculate the structure tensor J at each position (x, t).

(2) Calculate the coherence c from the eigenvalues of J (Equation 2.8).

(3) For $(c > 1 - \epsilon)$ calculate the velocity from the eigenvector corresponding to the lesser eigenvalue as $v = v_{2x}/v_{2t}$.

Figure 2.6 shows the result of this technique when applied to the noisy spatial-temporal image where the simple gradient-based approach failed. As expected, motion estimation is not possible in the object's interior or in the background. However, at the border of the object the coherence measure indicates that motion can be calculated reliably.

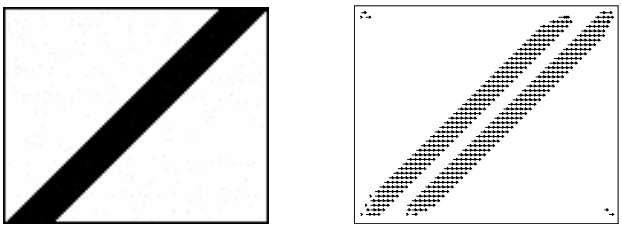

Figure 2.6: Tensor-based motion estimation in two dimensions. (a) LEFT: One-dimensional moving pattern, (b) RIGHT: Motion estimation on (a) with $\sigma = 1, C = 2$.

We should note that motion estimates are not only obtained at the object's boundary but also nearby. The reason is that the approach works on local neighborhoods. Consequently, the boundary of the moving object cannot be determined precisely from motion information. As a remedy, we will present a novel scheme to improve localization in Section 2.3.3.

2.3.2 Tensor-based Motion Estimation in Spatio-temporal Volumes

The estimation of local orientation in three-dimensional spatio-temporal volumes resembles the procedure described in the preceding section. Again, the eigenvector corresponding to the lowest eigenvalue of the structure tensor provides the local orientation. However, we have to adapt the problem definition to three dimensions. For the sake of clarity, we employ the same notation as in the two-dimensional case and simply add the second spatial dimension denoted by y. Then, at each image position $p = (x, y, t)$ the functional

$$\int\limits_{p' \in \Omega(p)} w(p - p')(\nabla_3 f(p')^T \cdot n)^2 dx'\, dy'\, dt' \tag{2.9}$$

is minimized, where $\nabla_3 := (\partial_x f, \partial_y f, \partial_t f)^T$ denotes the spatio-temporal gradient. From this minimization, the vector $n = (n_x, n_y, n_t)$ is determined that represents the local orientation. Ideally, n is perpendicular to all gradients in the local neighborhood Ω.

Similar to Appendix A, we derive the 3×3 structure tensor

$$J = \begin{bmatrix} j_{xx} & j_{xy} & j_{xt} \\ j_{xy} & j_{yy} & j_{yt} \\ j_{xt} & j_{yt} & j_{tt} \end{bmatrix}. \tag{2.10}$$

The tensor components $j_{kl}, k, l \in \{x, y, t\}$ are given as

$$j_{kl}(p') = \int_{p' \in \Omega(p)} w(p - p') \partial_k f(p') \partial_l f(p') dx' \, dy' \, dt'. \tag{2.11}$$

Solving the eigenvalue problem for the 3×3 structure tensor is not as simple as it was in the two-dimensional case. However, it is a standard problem in numerical mathematics, and efficient techniques can be found in [39, 103]. With respect to the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$, we can distinguish four cases (cf. Figure 2.7).

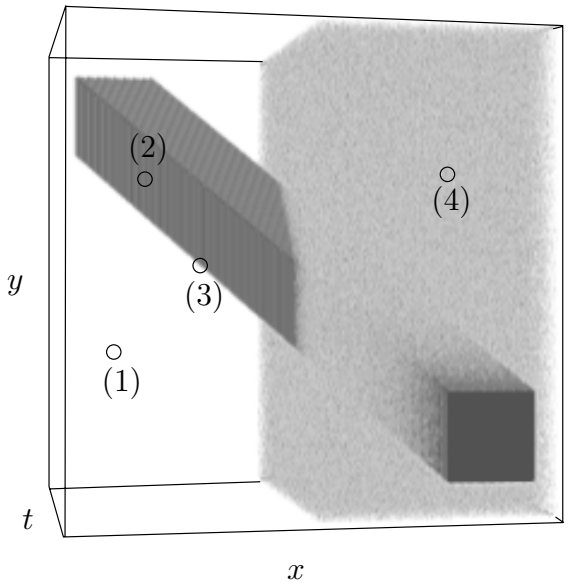

Figure 2.7: Moving two-dimensional pattern. The right half of the volume is distorted by Gaussian noise.

(1) $\lambda_1 = \lambda_2 = \lambda_3 = 0$
If all three eigenvalues are equal to zero, the local neighborhood consists of a constant gray value (see Figure 2.7, case (1)). Thus, local orientation cannot be determined, nor is it possible to estimate motion.

(2) $\lambda_1 > 0, \lambda_2 = \lambda_3 = 0$
In this case, the gray values change only in one direction. This occurs if a linearly symmetrical spatial pattern moves. Consider, for instance, the edge of a moving black square against a white background (see Figure 2.7, case (2)). Consequently, due to the aperture problem (see, e. g., [10]), we can calculate only *normal* velocity,

i. e., the velocity vector perpendicular to the edge [15, 54]:

$$v = -\frac{v_{1t}}{v_{1x}^2 + v_{1y}^2} \begin{bmatrix} v_{1x} \\ v_{1y} \end{bmatrix}. \tag{2.12}$$

In addition, this eigenvalue combination may result from a motion discontinuity, i. e., if the gray values are constant with respect to the spatial coordinates but change in the temporal dimension. However, we can detect this case due to the fact that the spatial components of eigenvector v_1 are zero.

(3) $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 = 0$
Here, we are able to identify the local orientation uniquely. This configuration appears in the case of a moving textured region. Consider, for example, the corner of a moving black square against a white background (see Figure 2.7, case (3)). Since gray values remain constant along the direction of the eigenvector v_3, we can calculate the *full* velocity, i. e., both components of the motion vector v, as

$$v = \frac{1}{v_{3t}} \begin{bmatrix} v_{3x} \\ v_{3y} \end{bmatrix}. \tag{2.13}$$

Note that the same eigenvalue magnitudes may result from a motion discontinuity, i. e., the gray values change in one spatial dimension and in the temporal dimension. In this case, the eigenvalue v_3 is parallel to the spatial plane, thus, the temporal component v_{3t} is zero.

(4) $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0$
All three eigenvalues greater than zero indicate a noisy spatio-temporal area (see Figure 2.7, case (4)) or a motion discontinuity. In any case, there is insufficient evidence to estimate motion.

Again, coherence measures are required that express the reliability of the calculations. In contrast to the two-dimensional case, two different coherence measures are needed here: one to indicate whether or not motion estimation is possible, and the other to distinguish between full and normal motion estimation. The coherence measure c_t indicates whether or not motion calculation is possible and is defined as

$$c_t = \begin{cases} 0 & \lambda_1 = \lambda_3, \\ \exp\left(\frac{-C}{|\lambda_1 - \lambda_3|}\right) & \text{else} \end{cases}. \tag{2.14}$$

A value of c_t near 1.0 indicates that $\lambda_1 \gg \lambda_3$, therefore a reliable calculation of motion can be performed. To discriminate between full and normal motion estimation the coherence measure c_s is defined as

$$c_s = \begin{cases} 0 & \lambda_2 = \lambda_3, \\ \exp\left(\frac{-C}{|\lambda_2 - \lambda_3|}\right) & \text{else} \end{cases} . \tag{2.15}$$

Here, values near 1.0 allow the calculation of both motion components. Otherwise, only normal velocities can be specified. Based on these coherence measures, a tensor-based motion estimation approach in spatio-temporal volumes can be specified as follows:

(1) Calculate the structure tensor J at each position $p = (x, y, t)$.

(2) Calculate the coherence measures c_t and c_s from the eigenvalues of J (Equations 2.14 and 2.15).

(3) Calculate for $(c_t > 1-\epsilon) \wedge (c_s > 1-\epsilon)$ the velocity from the eigenvector corresponding to the smaller eigenvalue. Determine for $(c_t > 1-\epsilon) \wedge (c_s \leq 1-\epsilon)$ normal motion. In all other cases, motion cannot be estimated.

We conclude this section by applying tensor-based motion estimation to a synthetic image sequence that contains a square moving constantly from bottom right to top left. Figure 2.8a displays one frame of this sequence. In order to clarify the square's motion, several frames from different points in time have been superimposed. Figure 2.8b shows the motion vectors obtained from the tensor-based motion approach.

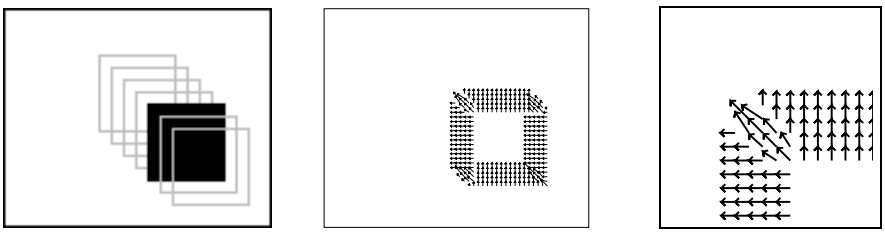

Figure 2.8: Tensor-based motion estimation in three dimensions. (a) LEFT: Constantly moving square at different points in time, (b) MIDDLE: Motion estimation on (a) with $\sigma = 1, C = 2$, (c) RIGHT: Enlarged top left part of (b).

We observe that motion cannot be estimated inside the object or within the background. This is due to the fact that the local neighborhoods in these areas are uniform. As stated earlier, in this case all three eigenvalues are zero, thus, the coherence measure

c_t is below $(1 - \epsilon)$. As expected, full motion can only be calculated near the corners of the moving square (cf. Figure 2.8c). Here, gray values remain constant only in one direction. Thus, $\lambda_1 > 0, \lambda_2 > 0$, and both coherence measures indicate sufficient information. Finally, near the edges of the square, normal motion, i. e., the motion of the object in the direction of the spatial gradient, can be estimated. The gray values in these areas remain constant in two directions. Therefore, the coherence measure c_t is close to one and c_s is below $(1 - \epsilon)$.

Similar to the two-dimensional case, we observe here that motion boundaries do not coincide with the real boundaries of the object. Of course, this is also due to the computations performed on local neighborhoods. Within the following section, this problem will be addressed by a non-maximum suppression scheme.

2.3.3 Non-maximum Suppression for Tensor-based Motion Estimation

In the preceding sections, we introduced the structure tensor as an adequate means of reliable motion estimation. By calculating the tensor components within a given spatio-temporal neighborhood, the approach becomes robust to noise. However, we have observed a shortcoming of the approach that will be discussed subsequently.

Tensor-based motion estimation entails a trade-off between robustness and localization. Correct estimation of local orientation under the influence of noise is achieved by averaging the calculations within a local neighborhood Ω. Increasing the size of the neighborhood enlarges the area that is averaged, and thus adds reliability to the estimates. As previously said, this is usually achieved by means of a Gaussian weighting function.

However, the smoothing process extends the support of local gray value structures. Consequently, both at and near the original boundary of a moving object unique local orientations are identified and contribute to motion estimation. Figure 2.9 exemplifies this relationship for the two-dimensional case. Figure 2.9a displays local orientations for a one-dimensional moving pattern. For illustrative purposes, the moving object's boundary has been superimposed on the orientation vectors. We observe that motion estimation is possible in a certain neighborhood around the boundary; thus, motion boundaries do not coincide with the true object boundaries.

Let us consider the coherence measure c for the given example. Recall that $c(p)$ indicates whether or not reliable estimation of motion is possible at the image position p. Figure 2.9b displays the corresponding surface of coherence values calculated for the entire image domain. We observe that local maxima exist at the exact position of the

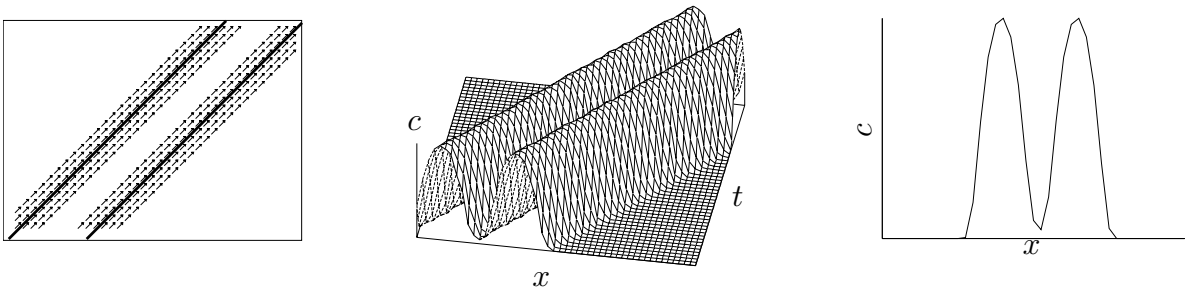

Figure 2.9: Coherence value surface. (a) LEFT: Spatial boundaries of a one-dimensional moving object superimposed on tensor-based orientation estimations, (b) MIDDLE: Surface of coherence values, (c) RIGHT: Cross section of the coherence surface in the direction of the greatest gray-value change.

moving object's boundary. In particular, starting from a position on the boundary, the coherence values decrease along the axis defined by the vector v_1. Recall that v_1 points in the direction of the maximal gray value change. Figure 2.9c depicts a cross-section of the coherence surface along the axis defined by the vector v_1. The local maxima can be clearly identified.

The local maxima of the coherence measure result from the fact that the greatest gray-value change occurs at the boundary. Thus, the eigenvalue λ_1, which measures the amount of gray-value changes in the direction of v_1, reaches its maximum at this position. In addition, since the gray values do not change in the direction perpendicular to v_1, the eigenvalue λ_2 is zero. Therefore, the coherence measure c reaches a local maximum precisely at the boundary of the moving object. Note, however, that this observation only holds true under the assumption of a simple neighborhood and a Gaussian window function.

Thus, by developing a scheme to suppress coherence values that are not local maxima (*non-maximum suppression*), we can determine the boundaries of moving objects more precisely. A suppression scheme proposed by Harris and Stephens [45] in the context of corner and edge detection in still images nominates a corner region pixel as a corner pixel if it contains an 8-way local maximum. I. e., a value is not suppressed if it is a local maximum with respect to a given neighborhood. An edge region pixel is retained as an edge pixel if the corresponding response is a local minimum in either the x or y directions.

A different non-maximum suppression proposed by Canny [20] for edge detection in still images marks edges at local maxima in gradient magnitude of a Gaussian-smoothed

image. This is achieved by determining the edge direction from the gradient and suppressing non-maximum gradient magnitudes in that direction.

In replacing the gradient by eigenvector v_1 of the structure tensor and by suppressing non-maxima of the coherence measure c in that direction, Canny's technique can be readily adapted to our setting. In the following, we outline non-maximum suppression in two-dimensional spatio-temporal images. Subsequently, we extend the approach to encompass three dimensions.

2.3.3.1 Non-maximum Suppression in Spatio-temporal Images

In the following, a two-dimensional spatio-temporal image $f(x,t)$ is represented as a uniform mesh of spacing $h = 1$ that divides the image domain into grid nodes (i,j). The decision as to whether or not a pixel p is a local maximum is made within a 3×3 neighborhood (cf. Figure 2.10). Without loss of generality, we translate the position of p to the origin of the coordinate system. Thus, the eight neighboring pixels are located within $[-1,1] \times [-1,1]$.

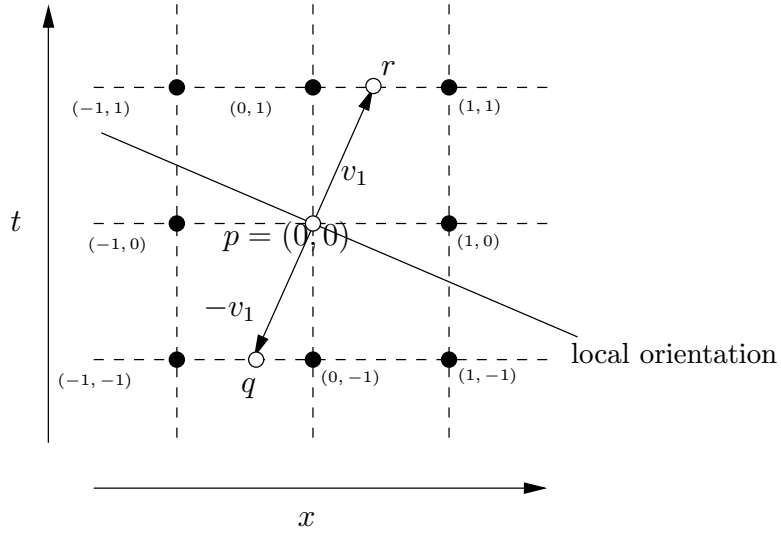

Figure 2.10: Non-maximum suppression in two-dimensional spatio-temporal images (cf. [19]).

As depicted in Figure 2.10, to perform non-maximum suppression parallel to the direction of the eigenvector $v_1 = (v_{1x}, v_{1y})^T$ the values of three image positions are required: the center point p of the 3×3 neighborhood itself and the neighboring points $r = (r_x, r_t)$ and $q = (q_x, q_t)$ displaced from p by v_1 and $-v_1$, respectively. To determine the image positions r and q, one has to calculate the position at which the axis defined by v_1 inter-

cepts the two surrounding parallel grid lines. From the vector components (v_{1x}, v_{1t}), one can easily determine which grid lines have to be considered: $|v_{1t}| > |v_{1x}|$ indicates that the axis given as v_1 intersects the grid lines at $t = \pm 1$, whereas $|v_{1x}| > |v_{1t}|$ leads to the grid lines given as $x = \pm 1$.

More formally, let us represent the grid lines as two straight line equations

$$\begin{aligned} g_1(s) &= es + o \\ g_1'(s) &= es - o, \qquad s \in I\!R, \end{aligned} \tag{2.16}$$

where the unit vectors $e = (e_x, e_t)^T$ and $o = (o_x, o_t)^T$ chosen from $\{(1,0)^T, (0,1)^T\}$ with $e \neq o$ denote direction and offset. For instance, in our example the grid lines at $t = -1$ and $t = 1$ are obtained from g_1 and g_1' by setting $e = (1,0)^T$ and $o = (0,1)^T$. In addition, we denote the axis given as the eigenvector $v_1 = (v_{1x}, v_{1t})^T$ as

$$g_2(t) = v_1 t, \qquad t \in I\!R. \tag{2.17}$$

Then, the point r is determined from the intersection of g_1 and g_2. Accordingly, q is given as the interception point of g_1' and g_2. Thus, denoting the determinant of a matrix as $|\cdot|$, r and q can be calculated as follows:

$$r = \frac{\begin{vmatrix} o_x & e_x \\ o_t & e_t \end{vmatrix}}{\begin{vmatrix} v_{1x} & e_x \\ v_{1t} & e_t \end{vmatrix}} \cdot v_1, \quad q = \frac{\begin{vmatrix} o_x & e_x \\ o_t & e_t \end{vmatrix}}{\begin{vmatrix} v_{1x} & e_x \\ v_{1t} & e_t \end{vmatrix}} \cdot (-v_1). \tag{2.18}$$

For example, in Figure 2.10 the interception points are determined from $e = (1,0)^T$ and $o = (0,1)^T$ as $r = (v_{1x}/v_{1t}, 1)$ and $q = -r$.

Since only at the grid points are coherence values available, it is necessary to interpolate whenever the direction of v_1 is different from horizontal and vertical. By using linear interpolation the coherence values at points r and q are approximated as

$$\begin{aligned} c(r) &\approx (1 - \min(|r_x|, |r_t|)) \cdot c(\lfloor r \rfloor) + \min(|r_x|, |r_t|) \cdot c(\lceil r \rceil) \\ c(q) &\approx (1 - \min(|q_x|, |q_t|)) \cdot c(\lfloor q \rfloor) + \min(|q_x|, |q_t|) \cdot c(\lceil q \rceil), \end{aligned} \tag{2.19}$$

where $\lfloor r \rfloor$ denotes that the components of the vector r are truncated to the nearest integer not larger in absolute value. Similarly, $\lceil r \rceil$ denotes that the components of the vector r

are rounded to the nearest integer not smaller in absolute value. In our example, the coherence values $c(r)$ and $c(q)$ are computed from:

$$\begin{aligned} c(r) &\approx \left(1 - \frac{v_{1x}}{v_{1t}}\right) c(0,1) + \frac{v_{1x}}{v_{1t}} c(1,1) \\ c(q) &\approx \left(1 - \frac{v_{1x}}{v_{1t}}\right) c(0,-1) + \frac{v_{1x}}{v_{1t}} c(-1,-1). \end{aligned} \tag{2.20}$$

Finally, the value at point p is retained only if it is a local maximum ($c(p) > c(r)$ and $c(p) > c(q)$), otherwise it is set to zero.[3]

Figure 2.11 displays the result obtained by tensor-based motion estimation combined with this suppression scheme. In contrast to the tensor-based approach without non-maximum suppression, the boundary of the moving object can be determined precisely.

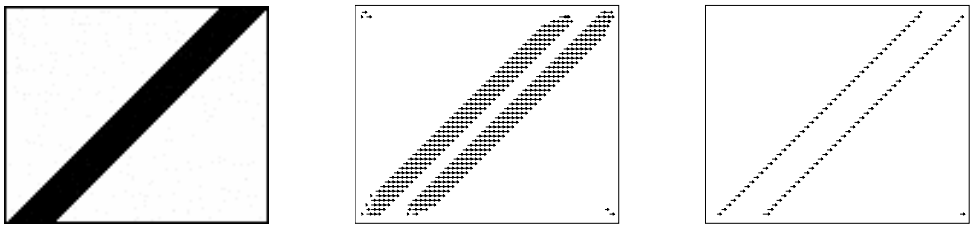

Figure 2.11: Tensor-based motion estimation in two dimensions with non-maximum suppression. (a) LEFT: One-dimensional moving pattern, (b) MIDDLE: Motion estimation on (a) with $\sigma = 1, C = 2$, (c) RIGHT: Motion estimation on (a) with $\sigma = 1, C = 2$ and non-maximum suppression.

2.3.3.2 Non-maximum Suppression in Spatio-temporal Volumes

Let us now turn to non-maximum suppression in spatio-temporal volumes. For this purpose, the volume is represented as a 3D discrete grid. For the sake of clarity, we use the same notation as in the two-dimensional case and simply add a third vector component. Again, the suppression axis is given as the eigenvector v_1 of the structure tensor. As depicted in Figure 2.12, the suppression is carried out within a $3 \times 3 \times 3$ neighborhood.

In analogy to the two-dimensional case, the coherence values are required at three volume positions $p = (0,0,0)$, $r = (r_x, r_y, r_t)$ and $q = (q_x, q_y, q_t)$. Note that to simplify the notation, p is translated to the origin of the coordinate system. In order to determine p

[3]Note that in an implementation with respect to discrete images, one should employ ($c(p) \geq c(r)$ and $c(p) > c(q)$) or ($c(p) > c(r)$ and $c(p) \geq c(q)$).

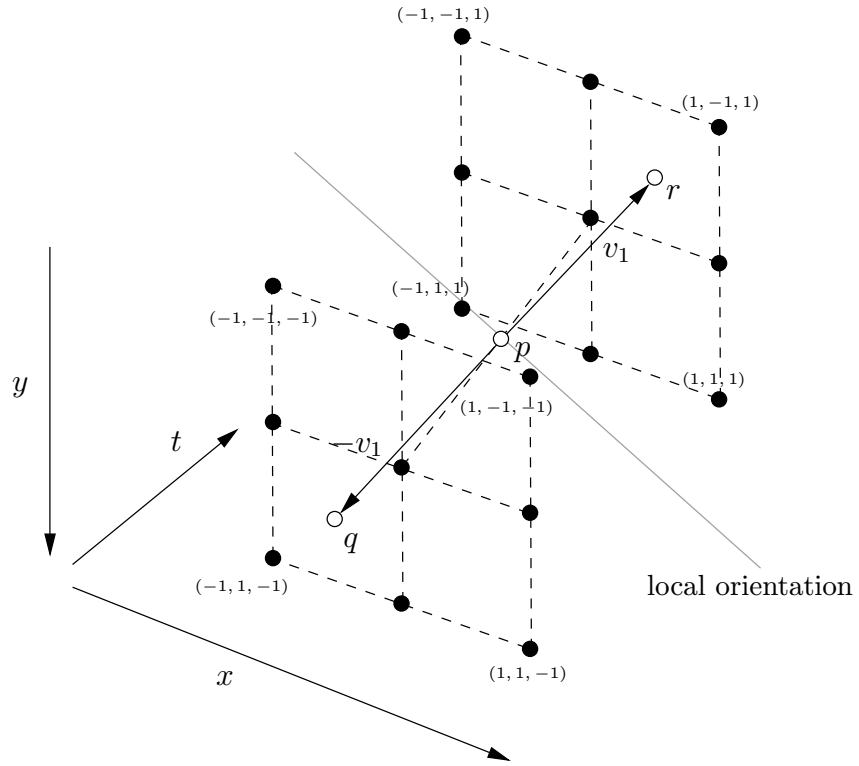

Figure 2.12: Non-maximum suppression in three-dimensional spatio-temporal volumes.

and q, one has to calculate the points at which the straight line given by v_1 intercepts with two grid planes. The appropriate grid planes can be chosen according to the magnitudes of the vector components v_{1x}, v_{1y}, v_{1t}.

Let $g_2(t) = v_1 t$, $t \in \mathbb{R}$, denote the axis given as $v_1 = (v_{1x}, v_{1y}, v_{1t})^T$ which represents the direction of the maximum gray-value change. Additionally, two parallel grid planes are represented by the plane equations

$$\begin{aligned} g_1(p,s) &= e_1 p + e_2 s + o \\ g_1'(p,s) &= e_1 p + e_2 s - o, \qquad p, s \in \mathbb{R}, \end{aligned} \tag{2.21}$$

where the unit vectors $e_1 = (e_{1x}, e_{1y}, e_{1t})^T$, $e_2 = (e_{2x}, e_{2y}, e_{2t})^T$ and $o = (o_x, o_y, o_t)^T$ are chosen from $\{(1,0,0)^T, (0,1,0)^T, (0,0,1)^T\}$ with $e_1 \neq e_2$, $o \neq e_1$, and $o \neq e_2$.

Considering the example displayed in Figure 2.12, the grid planes at $t = -1$ and $t = 1$ are obtained by setting $e_1 = (1,0,0)^T$, $e_2 = (0,1,0)^T$ and $o = (0,0,1)^T$. In general, r and q are obtained from the points at which g_2 intercepts with the two grid planes and can

be calculated as follows:

$$r = \frac{\begin{vmatrix} o_x & e_{1x} & e_{2x} \\ o_y & e_{1y} & e_{2y} \\ o_t & e_{1t} & e_{2t} \end{vmatrix}}{\begin{vmatrix} v_{1x} & e_{1x} & e_{2x} \\ v_{1y} & e_{1y} & e_{2y} \\ v_{1t} & e_{1t} & e_{2t} \end{vmatrix}} \cdot v_1, \quad q = \frac{\begin{vmatrix} o_x & e_{1x} & e_{2x} \\ o_y & e_{1y} & e_{2y} \\ o_t & e_{1t} & e_{2t} \end{vmatrix}}{\begin{vmatrix} v_{1x} & e_{1x} & e_{2x} \\ v_{1y} & e_{1y} & e_{2y} \\ v_{1t} & e_{1t} & e_{2t} \end{vmatrix}} \cdot (-v_1). \tag{2.22}$$

Returning to our example, r and q are determined from $e_1 = (1,0,0)^T$, $e_2 = (0,1,0)^T$, and $o = (0,0,1)^T$, as $r = (v_{1x}/v_{1t}, v_{1y}/v_{1t}, 1)$ and $q = -r$.

In order to calculate the coherence values for volume positions not located on the discrete grid, interpolation is required. Let us again consider the example displayed in Figure 2.12. Here, the coherence value at position r may be computed by bilinear interpolation from the four surrounding values at the positions

$$\begin{aligned} x_0 &= (\lfloor rx \rfloor, \lfloor ry \rfloor, 1) = (0,0,1) \\ x_1 &= (\lceil rx \rceil, \lfloor ry \rfloor, 1) = (1,0,1) \\ x_2 &= (\lfloor rx \rfloor, \lceil ry \rceil, 1) = (0,1,1) \\ x_3 &= (\lceil rx \rceil, \lceil ry \rceil, 1) = (1,1,1) \end{aligned} \tag{2.23}$$

as follows:

$$\begin{aligned} c(r) \approx\ & (1-|ry|)(1-|rx|)c(x_0) + (1-|ry|)|rx|c(x_1) \\ & + |ry|(1-|rx|)c(x_2) + |ry||rx|c(x_3). \end{aligned} \tag{2.24}$$

The coherence value for q can be determined accordingly. To approximate coherence values located on different grid planes, one has only to adapt Equation 2.24 appropriately.

Figure 2.13 illustrates the efficiency of the non-maximum suppression scheme for spatio-temporal volumes. As depicted, the motion boundaries resulting from the enhanced tensor-based motion estimation approximate the real boundaries of the square fairly accurately.

2.3.4 Experimental Results

To illustrate the main ideas and properties of tensor-based motion estimation, we have concentrated so far on simple synthetic objects. In this section, the different approaches

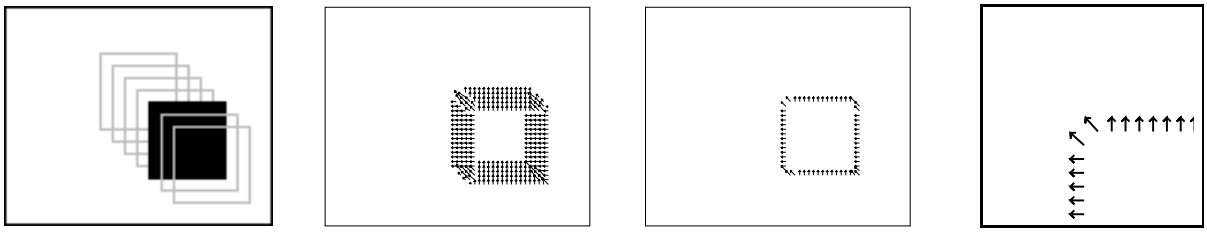

Figure 2.13: Tensor-based motion estimation in three dimensions with non-maximum suppression. FROM LEFT TO RIGHT: (a) Constantly moving square at different points in time, (b) motion estimation on (a) with $\sigma = 1, C = 2$, (c) motion estimation on (a) with $\sigma = 1, C = 2$ with non-maximum suppression, (d) enlarged top left part of (c).

are investigated in more detail. First of all, however, we briefly discuss some implementation issues. Recall that a tensor component j_{kl}, $k, l \in \{x, y, t\}$, is calculated on a continuous domain as follows

$$j_{kl}(p') = \int_{p' \in \Omega(p)} w(p - p') \partial_k f(p') \partial_l f(p') dx' \, dy' \, dt'. \tag{2.25}$$

In the context of discrete digital images, the integration and the partial derivatives have to be approximated appropriately. The integration is discretized by a convolution in which the weights are calculated from the Gaussian weighting function w. The partial derivatives are approximated by discrete operators. Here, we employ first-order discretizations optimized with respect to rotation invariance as proposed in [109, 121].

We proceed by presenting some quantitative results calculated with the proposed algorithms. For this purpose, our implementations were applied to a common set of synthetic image sequences, namely, the *translating tree* sequence and the *diverging tree* sequence [9]. Note that the exact motion fields for those sequences are known and thus can be compared to the results of a motion estimation. Both sequences simulate camera motion with respect to a textured surface (cf. Figure 2.14). The camera in the *translating tree* sequence moves to the left at speeds between 1.73 and 2.26 pixels/frame. Consequently, all motion vectors are parallel to the horizontal image axis. The camera in the *diverging tree* sequence zooms into the scenery, where the focus of expansion is located at the image center. Here, the velocities vary from 1.29 pixels/frame on the left to 1.86 pixels/frame on the right.

For the sake of comparison, we employ the widely used angular error measure [9]. Here, velocity is viewed as orientation in space-time. Let $v = (v_x, v_y)$ and $w = (w_x, w_y)$ denote the correct motion vector and the estimated motion vector. Those vectors are represented

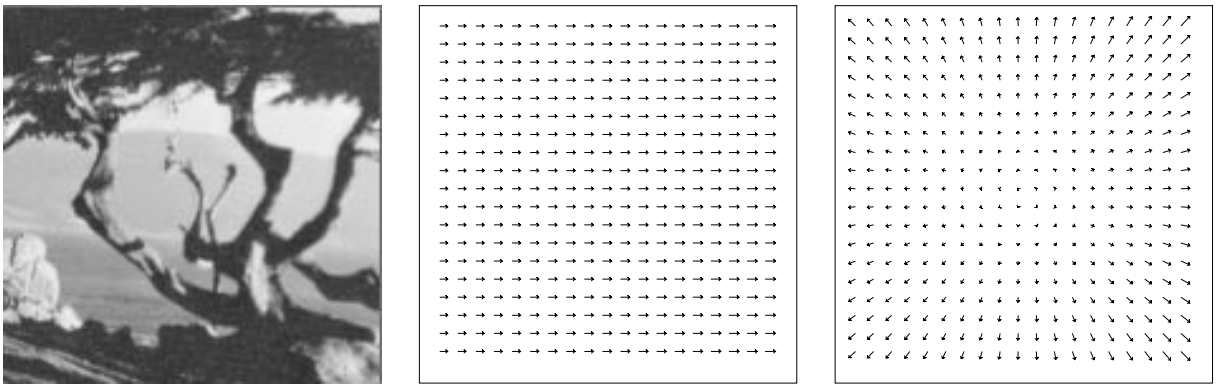

Figure 2.14: Synthetic tree sequences. FROM LEFT TO RIGHT: (a) Surface texture, (b) motion field of frame 20 of the *translating tree* sequence, (c) motion field of frame 20 of the *diverging tree* sequence (cf. [9]).

as space-time vectors by $\hat{v} = \frac{1}{\sqrt{v_x^2+v_y^2+1}}(v_x, v_y, 1)^T$ and $\hat{w} = \frac{1}{\sqrt{w_x^2+w_y^2+1}}(w_x, w_y, 1)^T$. Then, the angular error between the correct motion vector and the estimate is given as

$$\psi = \arccos(\hat{v}^T \cdot \hat{w}). \tag{2.26}$$

In order to evaluate our algorithms, we applied tensor-based motion estimation without non-maximum suppression (*3dst*) and with non-maximum suppression (*3dst-nonms*) to the tree sequences. Table 2.1 summarizes average angular error, standard deviation and density of the estimated flow fields. In addition, for the purpose of comparison, several results obtained in [9] with well-known optical flow techniques are displayed. In both cases, we obtain excellent results, the motion fields estimated by the tensor-based approaches match the correct motion fields very closely.

For visual inspection, Figures 2.15 and 2.16 display the flow fields as calculated by the different algorithms. As expected, the algorithms do not provide dense motion vector fields but eliminate some areas due to lacking texture. In particular, the non-maximum suppression reduces the density of the flow field but also slightly reduces the average angular error for the *diverging tree* sequence.

Next, we examine the results of *3dst* and *3dst-nonms* on well-known real-world sequences, namely, the *hall-and-monitor* (Figure 2.18) and the *Hamburg taxi* (Figure 2.19) sequences, both acquired by means of a static camera. The former displays a street scene and contains four moving objects: a taxi turning the corner; a car on the left, driving to the right; a van on the right, driving to the left; and a pedestrian in the upper left. In

sequence	technique	average error	standard deviation	density
translating tree (frame 20)	3dst	0.52°	0.45°	63.18%
diverging tree (frame 20)	3dst	1.38°	1.38°	53.25%
translating tree (frame 20)	3dst-nonms	0.52°	0.45°	13.33%
diverging tree (frame 20)	3dst-nonms	1.22°	1.25°	11.32%
translating tree (frame 20)	Lucas-Kanade ($\lambda_2 > 5.0$)	0.56°	0.58°	13.10%
diverging tree (frame 20)	Lucas-Kanade ($\lambda_2 > 5.0$)	1.65°	1.48°	24.30%
translating tree (frame 20)	Horn-Schunck (modified)	1.89°	2.40°	53.20%
diverging tree (frame 20)	Horn-Schunck (modified)	2.50°	3.89°	32.90%

Table 2.1: Tensor-based motion estimation: Full velocity results for translating and diverging tree sequences ($\rho = 2, C = 1$). The results for the Lucas-Kanade and the Horn-Schunck techniques are taken from [9] for comparison.

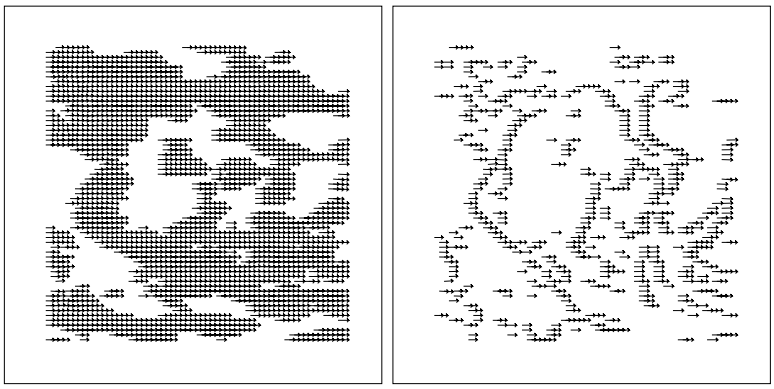

Figure 2.15: Tensor-based motion estimation on the *translating tree* sequence. (a) LEFT: Without non-maximum suppression, (b) RIGHT: With non-maximum suppression.

the latter sequence, two persons are walking down a hall.

In contrast to the synthetic sequences above, the correct flow fields are not available in either of these cases, and we have to rely on visual impressions. Since both sequences were recorded by a static camera, we concentrate on the moving objects. To illustrate the performance of the algorithms so-called *motion images* are calculated. In these images, only those positions are marked where tensor-based motion estimation indicates motion. Here, both full and normal velocities are included. Figures 2.18 and 2.19 display the original frames of both sequences along with the motion images calculated by *3dst* and *3dst-nonms*. We observe that both algorithms detect motion areas reliably. Since the calculations are performed within a spatio-temporal neighborhood, noise contained in

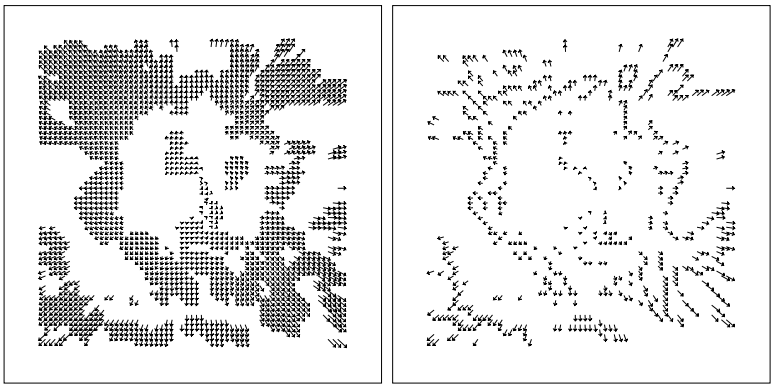

Figure 2.16: Tensor-based motion estimation on the *diverging tree* sequence. (a) LEFT: Without non-maximum suppression, (b) RIGHT: With non-maximum suppression.

the sequence is efficiently suppressed. In addition, the non-maximum suppression scheme enables a more accurate detection of the moving object's boundary. Figure 2.17 underlines this observation by depicting results from tensor-based motion estimation superimposed on an original frame from the taxi sequence.

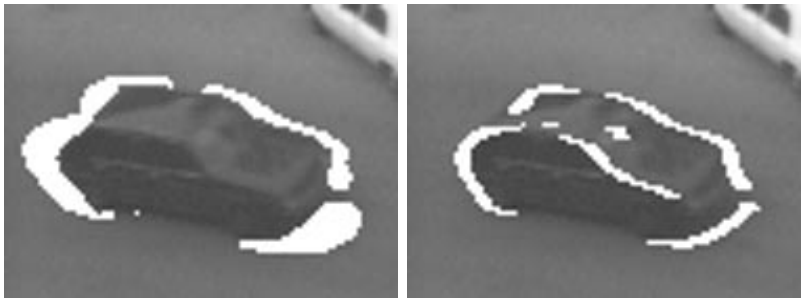

Figure 2.17: Motion image of a part of a frame from the *Hamburg taxi* sequence. (a) LEFT: Tensor-based motion estimation, (b) RIGHT: Tensor-based motion estimation with non-maximum suppression.

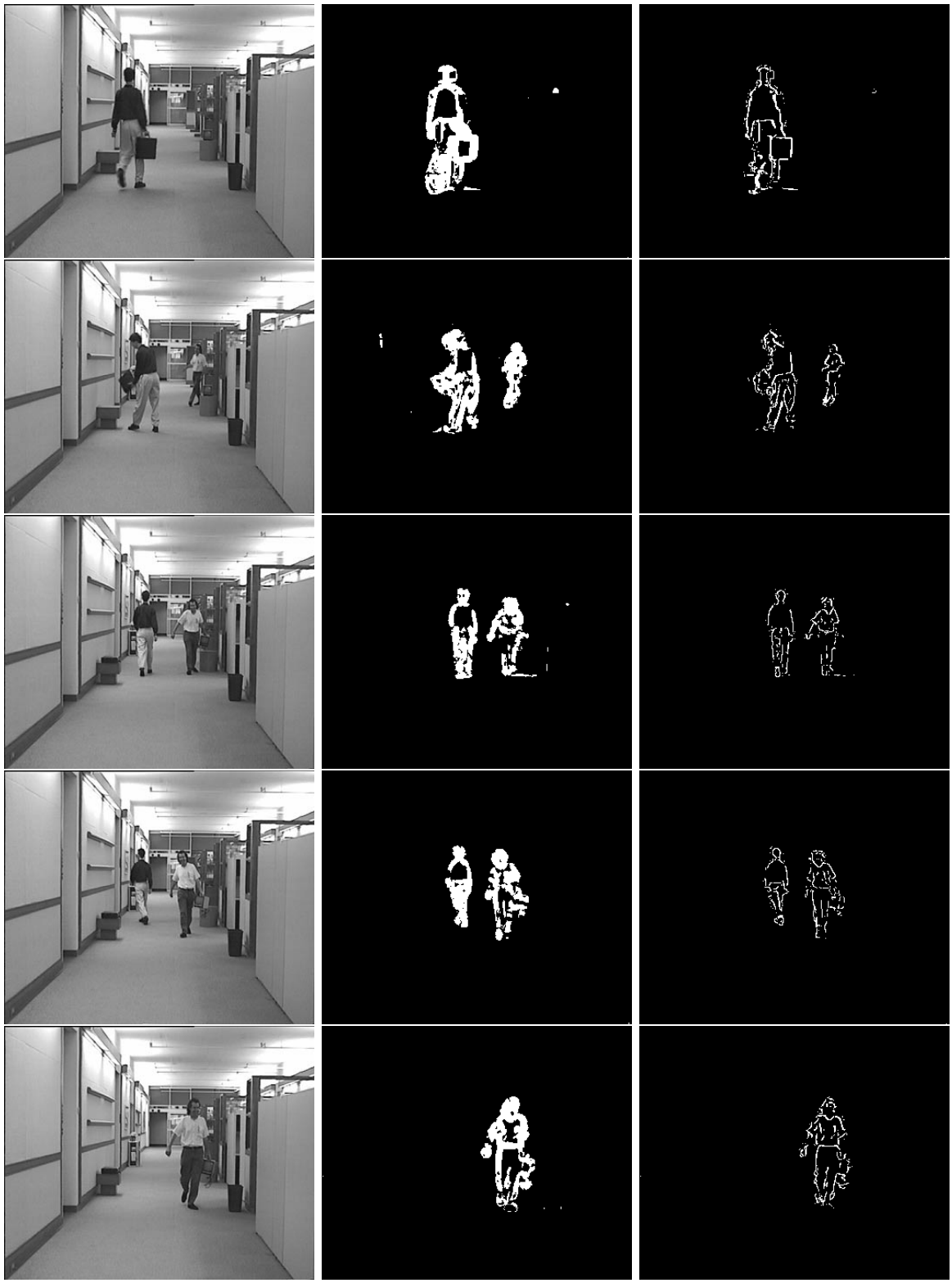

Figure 2.18: Motion images of frames from the *hall-and-monitor* sequence. (a) LEFT ROW: Original frames, (b) MIDDLE ROW: Tensor-based motion estimation, (c) RIGHT ROW: Tensor-based motion estimation with non-maximum suppression ($\rho = 1.0, C = 15$).

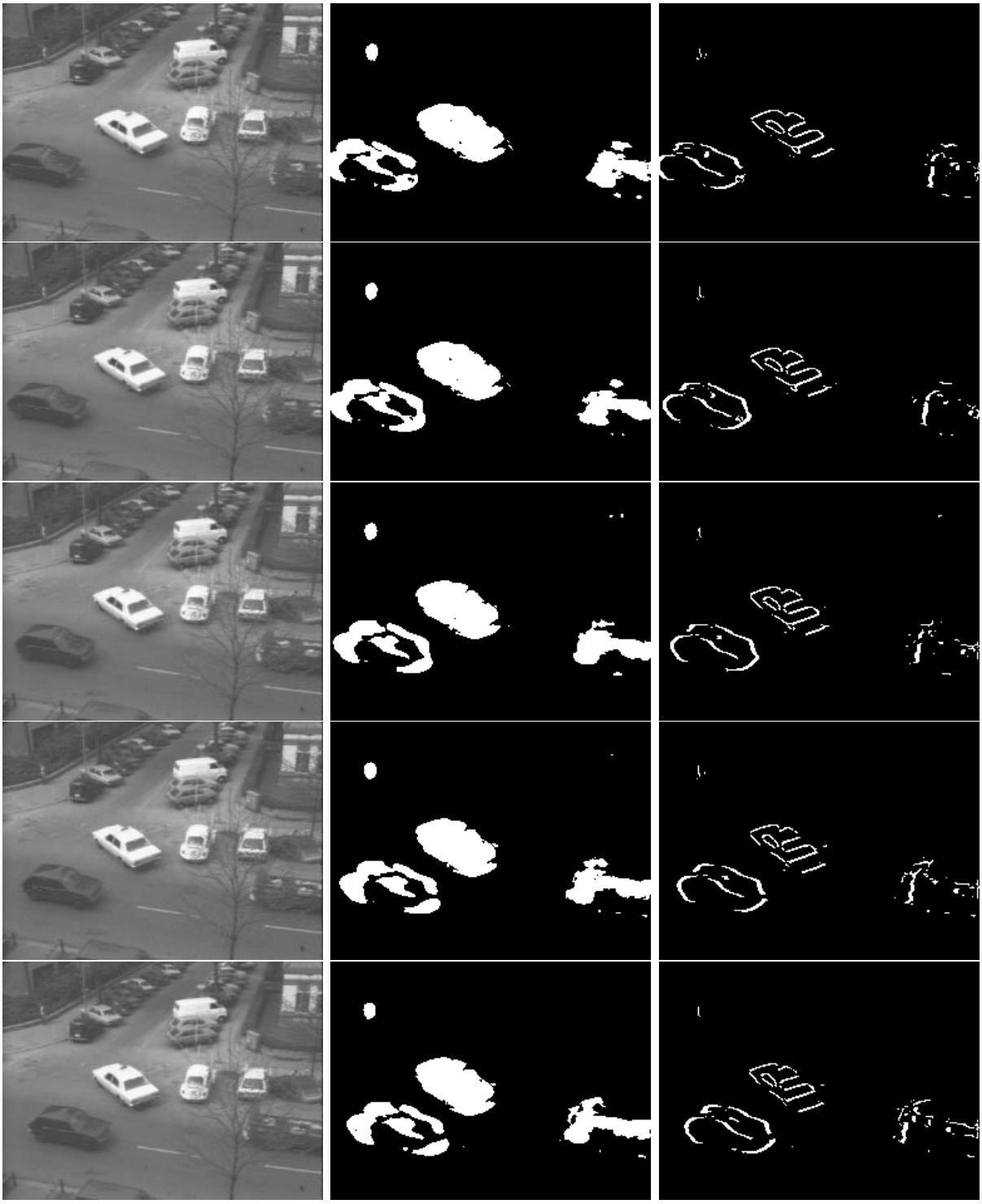

Figure 2.19: Motion images of frames from the *Hamburg taxi* sequence. (a) LEFT ROW: Original frames, (b) MIDDLE ROW: Tensor-based motion estimation, (c) RIGHT ROW: Tensor-based motion estimation with non-maximum suppression ($\rho = 1.5$, $C = 5$).

2.4 Feature-based Motion Estimation

Tensor-based motion estimation, as discussed in the preceding sections, relies on gradient information and thus belongs to the class of gradient-based methods. However, gradient-based motion estimation methods are applicable only under the assumption of small image displacements [9,86]. It is therefore necessary to develop an additional technique that can cope with large velocities. Before proceeding, let us first illustrate the limitations of gradient-based methods and discuss two approaches to diminish those problems.

We applied tensor-based motion estimation to a synthetic image sequence similar to those shown in Figure 2.3. In this sequence, a one-dimensional pattern moves to the right at constant speed v. In order to determine the accuracy with respect to different image displacements, several sequences of this type were created differing in the magnitude of v. In detail, tensor-based motion estimation was applied to sequences resulting form $v = 2$, $v = 4$, $v = 8$, and $v = 16$ (pixels/frame).

For the purpose of evaluation, we tracked one image feature through each sequence and estimated its velocity. Then, the velocity estimate was compared to the real velocity v by employing the angular error measure as defined in Equation 2.26. The calculations were performed using different sizes of the spatio-temporal neighborhood Ω. Recall that the tensor components are determined from this local neighborhood. Figure 2.20a displays the angular error of tensor-based motion estimation with respect to a given velocity v and the size of the local neighborhood. Note that in the figure, the radius of the neighborhood denoted as ρ is given in pixels.

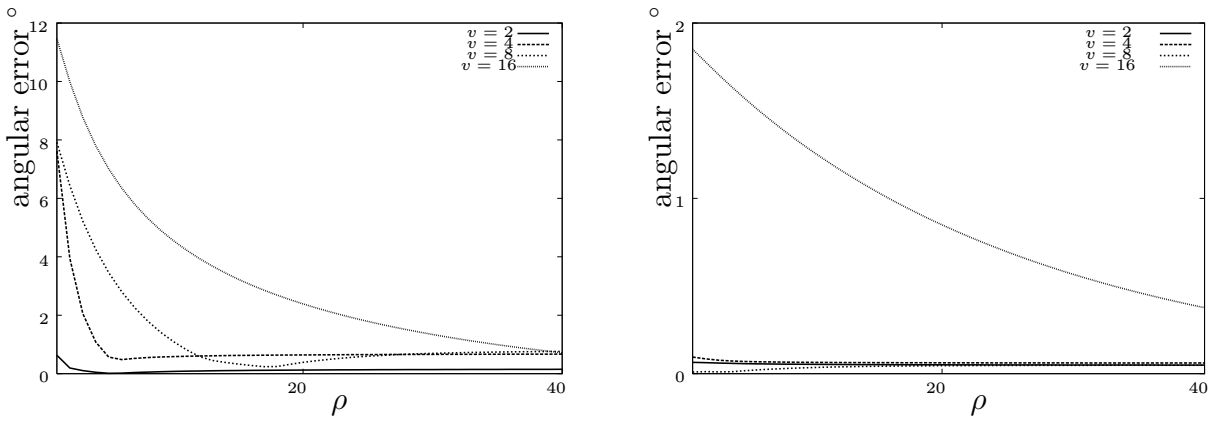

Figure 2.20: Tensor-based motion estimation on sequences containing large displacements. (a) LEFT: Angular errors for standard technique calculated for local neighborhoods of different size (ρ denotes the radius in pixels), (b) RIGHT: Angular errors for standard technique combined with coherence-enhancing diffusion.

We observe that for image displacements of more than two pixels per frame, calculations over small neighborhoods yield significant differences between motion estimates and real velocities. Only for very large mask sizes is the accuracy sufficient. Unfortunately, calculating the tensor components on such large neighborhoods is impractical since it is likely that the basic assumption of linear symmetry is not fulfilled in these cases. Therefore, one can employ the standard tensor-based technique only for small displacements.

The limitations discussed above are due to aliasing effects. If a sequence containing a fast-moving object is recorded at a low frame rate, aliasing or "stair-case effects" will occur. Figure 2.21 illustrates this relationship for a fast-moving one-dimensional pattern. Iso-gray-value lines indicating the local orientation can be determined only by considering a rather large local neighborhood.

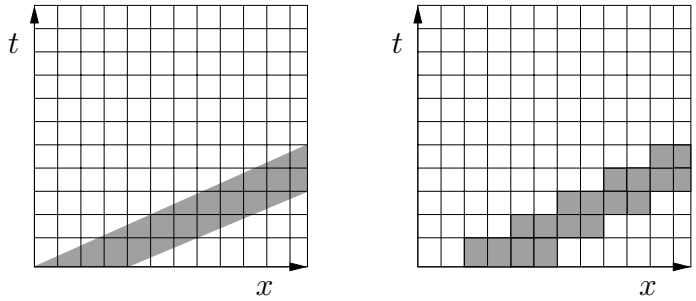

Figure 2.21: Discretization of an image sequence. (a) *Left:* Continuously moving one-dimensional pattern with pixel grid superimposed, (b) RIGHT: Discretization of the image sequence.

A standard solution to this problem is the use of multiresolution methods [7, 18, 33, 36]. A multiresolution method for tensor-based motion estimation has been proposed in [67]. Here, the basic idea is to calculate an image pyramid containing images of different resolutions or so-called levels. These levels are created by spatial subsampling, e. g., from an original image of size 256×256, three levels of sizes 256×256, 128×128, and 64×64 might be constructed. It is then possible to determine large velocities by estimating motion first at the coarsest level of the pyramid and propagating these estimates to levels of finer resolutions. However, motion estimation is possible only for image features than can be reliably identified at all levels of the pyramid. In addition, it is likely that at coarser levels image features will merge and thus might steer motion estimates in the wrong direction.

A different approach comprises the reduction of aliasing effects. To this end, we combined tensor-based motion estimation with a specific anisotropic diffusion technique, namely coherence-enhancing diffusion [137,138]. In this technique, the image is smoothed

along local orientations or flow lines. Figure 2.22 displays the results of such a diffusion process. As can be observed from the figure, the stair-case effects are significantly reduced. To evaluate the impact of the diffusion technique on the accuracy of motion estimation, we repeated the experiment with the one-dimensional patterns moving at different speeds. Figure 2.20b shows the angular error for the combination of coherence-enhancing diffusion and tensor-based motion estimation. In all cases, 200 diffusion iterations were applied to the test sequences prior to motion estimation. We observe that the accuracy of the motion estimates is sufficient even for small local neighborhoods up to displacements of 8 pixels/frame. For larger displacements, however, larger image areas have to be evaluated.

Figure 2.22: Reducing aliasing effects by coherence-enhancing diffusion. (a) LEFT: Original spatio-temporal image, (b) MIDDLE: Coherence-enhancing diffusion (20 iterations, $\tau = 0.2$), (c) RIGHT: Coherence-enhancing diffusion (100 iterations, $\tau = 0.2$).

While it is possible to extend the scope of tensor-based motion estimation by multiresolution or diffusion-based approaches as discussed above, limitations still remain. Often, it is more appropriate to employ correspondence-based or feature-based techniques in the event of large velocities. Here, the basic idea is to identify certain image features in a frame of the sequence and to determine corresponding features in consecutive frames. Since this approach does not depend on continuous iso-gray-value lines but on corresponding structures large velocities do not pose problems.

Our technique, which will be discussed in the following, relies on work conducted in [14, 17, 41, 45, 110, 111, 130]. Similar to their results, appropriate feature points (also called *interest points*) are detected in a frame of the sequence and correspondences in consecutive frames are determined by a matching procedure. In addition, we develop coherence measures that allow us to distinguish between features where full motion can be calculated and those where only normal motion can be determined.

2.4.1 Identifying and Tracking of Image Features

Let us recall some results from tensor-based motion estimation that will prove useful further on. As discussed in Section 2.3.2, motion estimation is possible only for specific patterns of spatio-temporal intensity. In detail, by considering the eigenvalues λ_1, λ_2, λ_3 of the three-dimensional structure tensor, these patterns can be distinguished. For instance, $\lambda_1 > 0$ and $\lambda_2 > 0$ indicates a textured image patch or a corner. Since this patch is likely to be found in consecutive frames, image positions where $\lambda_1 > 0$ and $\lambda_2 > 0$ are good candidates for feature points.

Furthermore, $\lambda_1 > 0$ and $\lambda_2 = 0$ indicates a linearly symmetrical pattern, e. g., an edge. Again, it is likely that we can determine the position of this structure in consecutive frames. However, the aperture problem restricts the calculations to normal motion. Nevertheless, keeping this limitation in mind, image positions where $\lambda_1 > 0$ and $\lambda_2 = 0$ are also candidates for feature points.

To apply the structure tensor to single frames rather than video sequences, we adapt its two-dimensional variant which was discussed in Section 2.3.1. Replacing the temporal dimension by the second spatial dimension (y) yields

$$J = \begin{bmatrix} j_{xx} & j_{xy} \\ j_{xy} & j_{yy} \end{bmatrix}, \tag{2.27}$$

where the matrix elements $j_{kl}, k, l \in \{x, y\}$ are calculated from

$$j_{kl}(p) = \int\limits_{p' \in \Omega(p)} w(p - p') \left(\partial_k f(p') \partial_l f(p')\right) dx'\, dy'. \tag{2.28}$$

Consequently, the meaning of the eigenvalues differs from the definition presented in Section 2.3.1 and is given as follows [54]:

(1) $\lambda_1 = \lambda_2 = 0$
Both eigenvalues equal to zero indicates an area of constant gray values. Reliable tracking of this area is not possible.

(2) $\lambda_1 > 0, \lambda_2 = 0$
If $\lambda_1 > 0$ and $\lambda_2 = 0$, gray values change only in one direction. One might be able to establish correspondences between consecutive frames. However, the aperture problem allows only calculation of normal motion.

(3) $\lambda_1 > 0, \lambda_2 > 0$

Finally, if both eigenvalues are greater than zero, gray values change in both directions. Reliable tracking is possible.

To obtain coherence measures that indicate the reliability of the calculations, we adapt the measures c_t and c_s as presented in Section 2.3.2 by setting $\lambda_3 = 0$. Then, $c_t \approx 1$ indicates a feature point which might be tracked in consecutive frames at least by calculating normal motion. Furthermore, $c_s \approx 1$ indicates a textured image patch, thus, the corresponding feature point can be tracked reliably. To further improve the extraction of feature points, we employ the non-maximum suppression technique as presented in Section 2.3.3.1. Here, non-maximum values of the coherence measure c_t are suppressed in the direction of the eigenvector belonging to the larger eigenvalue λ_1.

These considerations result in the following approach for the identification of feature points:

(1) Calculate the structure tensor J at each position $p = (x, y)$.

(2) Calculate the coherence measures c_t and c_s from the eigenvalues of J.

(3) Perform non-maximum suppression on c_t in the direction of the eigenvector belonging to λ_1.

(4) Mark image positions where $(c_t > 1 - \epsilon)$ and $(c_s > 1 - \epsilon)$ as feature points.

In addition to the identification of feature points, an appropriate tracking procedure is required. In our approach, we apply standard block-based motion estimation methods (see, e. g., [12]). Each feature point is considered to be the central point of a small image patch, e. g., a 9×9 block. Then, using the L_1 distance the best match for each block centered on a feature point is determined within a given search range. The displacements calculated this way determine the motion vector field.

2.4.2 Experimental Results

As in Section 2.3.4, we applied our algorithms for feature-based motion estimation to the synthetic *translating tree* and the synthetic *diverging tree* sequences. Table 2.2 summarizes the results with respect to angular error and feature density. We applied our algorithm in two variants that differed only in their choice of the non-maximum suppression scheme. In the first run, we employed the scheme for the detection of corner pixels as proposed in [45]. In this scheme, an image position is marked as a corner pixel if it contains a local

sequence	technique	average error	standard deviation	density
translating tree (fr. 20)	suppression as in [45]	1.49°	1.64°	1.21%
diverging tree (fr. 20)	suppression as in [45]	11.61°	6.62°	1.30%
translating tree (fr. 20)	$(c_t > 1 - \epsilon) \wedge (c_s > 1 - \epsilon)$	1.43°	1.57°	9.94%
diverging tree (fr. 20)	$(c_t > 1 - \epsilon) \wedge (c_s > 1 - \epsilon)$	11.35°	6.39°	9.84%
translating tree (fr. 20)	Anandan	4.54°	3.10°	100.00%
diverging tree (fr. 20)	Anandan	7.64°	4.96°	100.00%
translating tree (fr. 20)	Singh	0.72°	0.75°	41.40%
diverging tree (fr. 20)	Singh	7.09°	6.59°	3.90%

Table 2.2: Feature-based motion estimation: Full velocity results for translating and diverging tree sequences ($\rho = 1, C = 2$). The results for the Anandan and the Singh technique are taken from [9] for comparison.

maximum of the response function within a 3×3 neighborhood. In the second run, we used our own feature selection scheme as described above.

For the purpose of comparison, we also display results for correspondence-based techniques (Anandan and Singh) taken from [9]. In general, the results for all correspondence-based techniques are inferior to those calculated by means of gradient-based approaches (cf. Section 2.3.4). This is due to the fact that the sub-pixel accuracy of the latter class of algorithms is higher. In the case of the *translating tree* sequence, the results for our algorithms are better than those gained by Anandan's method and inferior to Singh's results. Note, however, that we chose the best results for Singh's method from those reported in [9]. The results for the *diverging tree* sequence are inferior to both other methods. This might be due to the fact that our current block-matching algorithm is restricted to half-pixel accuracy. The large difference in angular errors between *translating tree* and *diverging tree* sequence which can be observed for all correspondence-based algorithms is due to the sub-pixel problem [9]. In the *translating tree* sequence, velocities are very close to integer displacements, while the *diverging tree* sequence has a wide range of velocities with noninteger displacements.

With respect to feature density, our feature selection scheme provides significantly more interest points than the method proposed in [45]. In addition, we observe that the results for our algorithms are almost identical for the different non-maximum suppression schemes. Note that the density of 100% achieved by Anandan's method results from an additional smoothing constraint.

Let us now turn to real-world sequences. For the purpose of evaluation, we have applied

the algorithm presented above to the *Stefan* sequence and the *coastguard* sequence. Since the correct flow fields are not known in either case, one has to rely on qualitative results. Figures 2.23 and 2.24 illustrate the feature detection step. At the top left of each figure, sample frames from the sequences are displayed. In those frames, image features were detected. The images at the top right depict corner pixels calculated using the structure tensor and the non-maximum suppression scheme proposed in [45]. In the bottom row on the left, features are shown which result from calculations where only the coherence measure c_t ($c_t > 1 - \epsilon$) has been evaluated. Thus, at positions marked in white in these images, it is possible to determine motion by comparing features in consecutive frames. Note that for some of those positions, only normal motion can be determined, e. g., vertical and horizontal lines. However, under the assumption of a static camera, it might be useful to detect all features that move, regardless of whether or not normal or full motion can be estimated. The bottom row on the right shows results that were obtained by applying both coherence measures. Thus, for points marked white in these frames, it should be possible to calculate full motion. These feature points are useful with respect to any camera operation.

Figure 2.23: Feature point detection on the *Stefan* sequence. (a) TOP LEFT: Original frames, (b) TOP RIGHT: Corner pixels marked (cf. [45]), (c) BOTTOM LEFT: Image positions marked where ($c_t > 1 - \epsilon$), (d) BOTTOM RIGHT: image positions marked where ($c_t > 1 - \epsilon$) and ($c_s > 1 - \epsilon$).

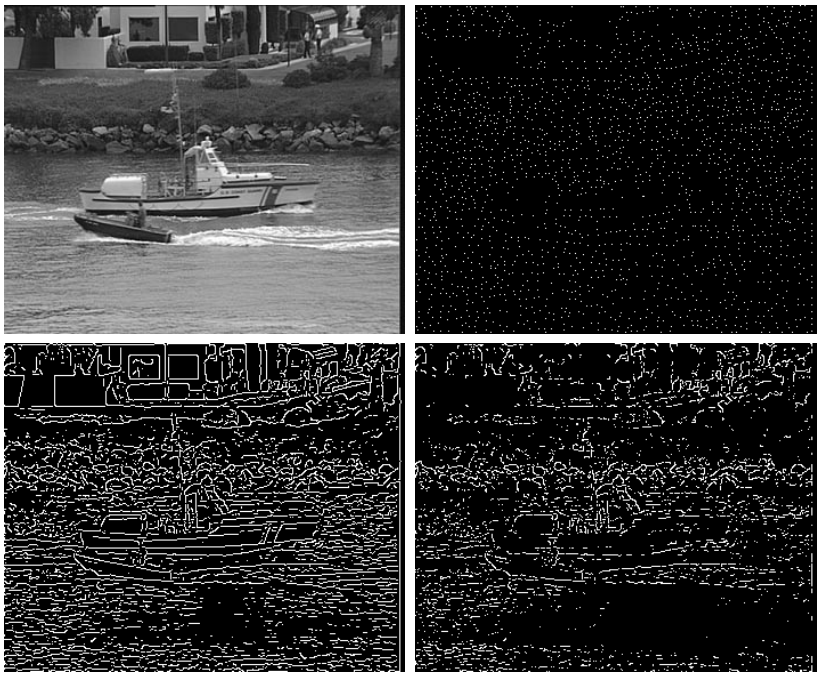

Figure 2.24: Feature point detection on the *coastguard* sequence. (a) TOP LEFT: Original frames, (b) TOP RIGHT: Corner pixels marked (cf. [45]), (c) BOTTOM LEFT: image positions marked where $(c_t > 1 - \epsilon)$, (d) BOTTOM RIGHT: image positions marked where $(c_t > 1 - \epsilon)$ and $(c_s > 1 - \epsilon)$.

Based on the feature points obtained for $c_t \approx 1$ and $c_s \approx 1$, motion vector fields were calculated for both sequences by using a block-matching technique. We chose parts of the sequences that contain considerable image displacements. For instance, the frames around frame 250 of the *Stefan* sequence contain image displacements of up to 30 pixels. For the *coastguard* sequence, a part with displacements of about four pixels was chosen. Figure 2.25 displays examples from both video sequences. In both cases, one can clearly observe how the background moves. Due to the feature extraction step, only some outliers emerge in the motion field of the background. In addition, it is possible to identify parts that belong to the moving objects.

Thus, by calculating the basic camera operation from the flow field and by removing motion vectors caused by camera motion, it should be possible to extract moving objects from the sequence. The following section will discuss an approach to calculate camera parameters from flow fields obtained by tensor-based or feature-based motion estimation.

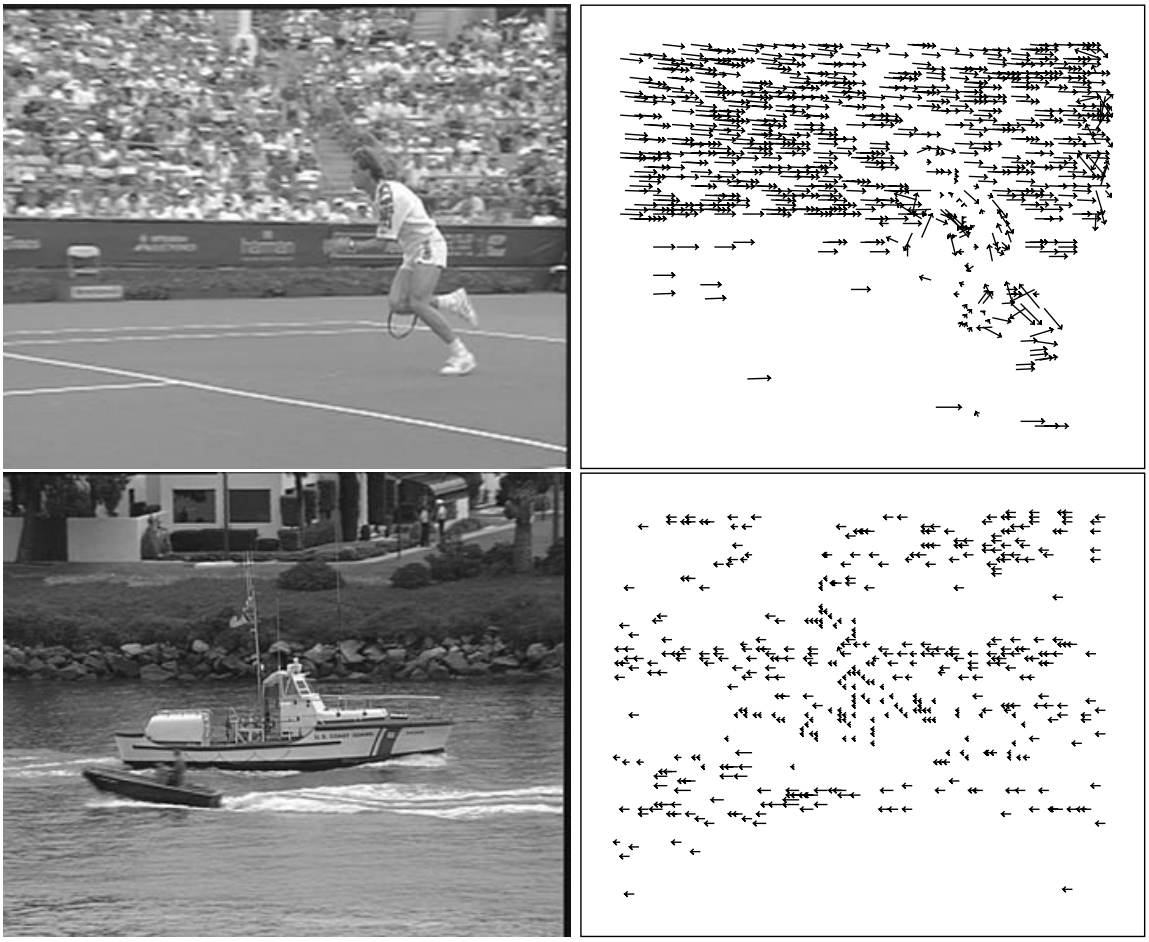

Figure 2.25: Feature-based motion estimation. (a) LEFT COLUMN: Original frames, (b) RIGHT COLUMN: Motion fields. The flow fields are subsampled for better visibility.

2.5 Estimation of Camera Motion

In the preceding sections we have discussed motion analysis techniques that indicate moving image areas reliably. As depicted in Figures 2.18 and 2.19, these image areas provide, under the assumption of a static camera, useful cues for subsequent segmentation algorithms.

However, as noted above, the situation is different when a moving camera is involved. Marking moving image positions while ignoring static areas is obviously not sufficient. Instead, one has to distinguish between camera motion and object motion in the image (cf. Figure 2.25). For this purpose, it is necessary to determine the parameters of the camera motion.

Consequently, first of all, an appropriate camera motion model is required that covers

the different camera operations. Then, a technique must be developed that estimates the parameters of the model from the image sequence. In our approach, we employ the perspective camera model, which will be summarized in the following section. Afterwards we concentrate on a robust estimation algorithm that derives the model parameters from motion information calculated by our motion analysis techniques presented above.

2.5.1 Modeling Camera Motion

Consecutive images acquired by a panning, tilting or zooming camera are related by a *planar homography* [46, 130]. Thus, under perspective projection, corresponding image positions (x', y') and (x, y) in two consecutive frames may be mapped to one another by the following non-linear equations with eight parameters $\theta := \{\theta_0, \ldots, \theta_7\}$:

$$x' = \frac{\theta_0 x + \theta_1 y + \theta_2}{\theta_6 x + \theta_7 y + 1} \quad , \quad y' = \frac{\theta_3 x + \theta_4 y + \theta_5}{\theta_6 x + \theta_7 y + 1}. \tag{2.29}$$

As a consequence, if the parameters are known, it is a straightforward process to establish correspondences between image features detected in two consecutive frames of a video sequence. Conversely, it is possible to determine the eight parameters from at least four image features if the exact positions in the two frames are known.

For this purpose, Equation 2.29 can be rewritten as

$$\begin{aligned} x' &= \theta_0 x + \theta_1 y + \theta_2 - \theta_6 x x' - \theta_7 y x' \\ y' &= \theta_3 x + \theta_4 y + \theta_5 - \theta_6 x y' - \theta_7 y y' \end{aligned} . \tag{2.30}$$

Consider four image features (e. g., corners) located in frame f_0 at positions $p_i := (x_i, y_i)$, and in the consecutive frame f_1 at positions $p'_i := (x'_i, y'_i)$. Then, each point correspondence $p_i \leftrightarrow p'_i$ generates two linear equations leading to a linear system of equations. The camera parameters can then be determined by solving the following linear system:

$$\begin{pmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0x'_0 & -y_0x'_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0y'_0 & -y_0y'_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \end{pmatrix}. \quad (2.31)$$

However, in most cases, more than four correspondences can be determined from consecutive frames of an image sequence. In addition, several of these correspondences either may be incorrect or cannot be approximated by a single camera motion model. Therefore, it is necessary to develop estimation methods that take those cases into account.

2.5.2 Robust Estimation of Camera Motion

By applying motion estimation techniques to a video sequence, it is possible to establish correspondences between image features in consecutive frames. Thus, in general, motion analysis reveals a collection of N image features where their positions p_i and p'_i in two consecutive frames are known. Since both the motion estimation algorithms developed above are accompanied by reliability measures, we can select appropriate motion estimates. In fact, it would be possible to choose the four motion-based correspondences having the highest reliabilities and then calculate the camera parameters according to Equation 2.31. However, since the correspondences might belong to different motion models (e. g., background motion and object motion), the calculations might turn out to be incorrect. Consequently, it is more appropriate to integrate into the calculations all correspondences reaching a certain level of reliability. As previously said, each point correspondence leads to two linear equations, thus, N image features results in an overdetermined linear system of $2N$ equations in eight unknowns.

The most popular technique for solving this linear system is the well-known *least sum of squares* (LS) method. Let us denote estimates of the unknown parameters θ_i by $\hat{\theta}_i$. Given those estimated parameters, one can calculate estimates $(\hat{x}'_i, \hat{y}'_i)$ of the feature positions (x'_i, y'_i) from Equation 2.30. Then, determining the difference between the actual feature positions (x'_i, y'_i) and their estimates will yield the *residuals* r_{x_i} and r_{y_i} given as:

$$r_{x_i} = x'_i - \hat{x}'_i \quad , \quad r_{y_i} = y'_i - \hat{y}'_i, \quad i = 1, \ldots, N. \tag{2.32}$$

The basic idea of the LS method is to determine the parameters by minimizing the residuals, which yields

$$\min_{\hat{\theta}} \sum_{i=1}^{N} \left(r_{x_i}^2 + r_{y_i}^2 \right). \tag{2.33}$$

A number of efficient numerical algorithms are known for this standard problem [103]. However, since the LS method is optimal for a Gaussian error distribution, it is not applicable in our context. Figure 2.26 illustrates the problems that occur when camera parameters are estimated by means of the LS method. For testing purposes, we created two synthetic flow fields: The first flow field simulates a camera panning over a constant scene (cf. Figure 2.26a). About 10% of the image pixels were assigned to the background motion, the remaining positions were ignored. The second flow field also indicates a panning camera. However, in this case, a moving object is overlaid (cf. Figure 2.26c). Here, about 8% of the image pixels were assigned to background motion and about 2% of the motion vectors resulted from object motion. As expected, LS minimization reveals the correct camera parameters in the first case of the perfect flow field without any interfering object motion (cf. Figure 2.26b). However, the mixture of background and object motion did not lead to satisfactory results (cf. Figure 2.26d). Instead, the camera parameters were computed such that they minimized the residuals for both the displacements caused by background motion and for the motion vectors resulting from the moving object. To put it differently, in the event of *outliers* — resulting in our example from object motion — LS minimization is not the method of choice. We should note that, in general, it is not possible to discover outliers by examining the residuals of the LS minimization [106].

In order to estimate camera parameters reliably when both background and object motion is present, one has to apply *robust regression methods*. Widely used robust regression methods in computer vision comprise the *random sample consensus* (RANSAC) proposed by Fischler and Bolles [34] and the *least median of squares* (LMedS) regression developed by Rousseeuw [106]. Both methods are able to calculate the parameters of a linear regression problem even though a large portion of the data consists of outliers.

However, RANSAC requires a thresholding procedure to classify data as inliers or outliers. The LMedS method does not require this threshold but its convergence rate is rather poor.

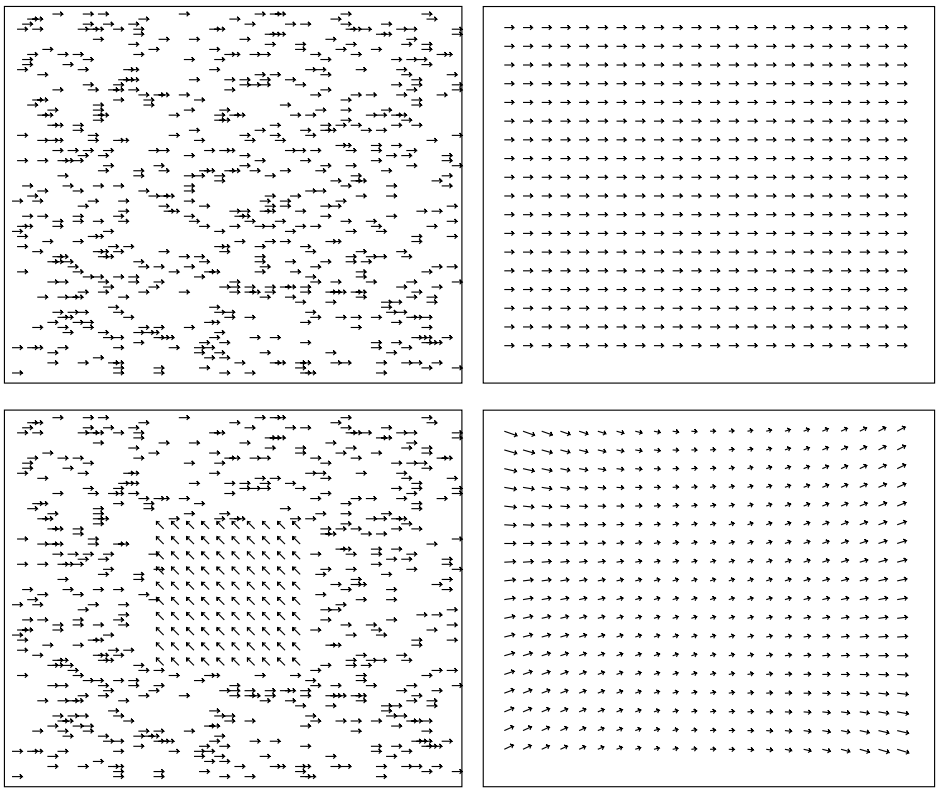

Figure 2.26: Determining camera parameters by means of least sum of squares minimization. (a) TOP LEFT: Flow field calculated from a sequence without moving objects, (b) TOP RIGHT: Estimated camera motion, (c) BOTTOM LEFT: Flow field calculated from a sequence containing one moving object, (d) BOTTOM RIGHT: Estimated camera motion. The flow fields are subsampled for better visibility.

Another robust regression estimator, the *least trimmed squares* (LTS) regression, also proposed by Rousseeuw [106], has received considerably less attention in the field of computer vision. Indeed, surveys on robust regression methods in computer vision [84, 85, 122, 129] did not take LTS into account. To the best of our knowledge, only Ye and Haralick [144] have considered LTS for a computer vision task, namely optical flow computation. We should note that they developed a technique similar to ours at around the same time [145].

Similar to LMedS regression, the LTS method does not require a selection threshold. In addition, it has a higher convergence rate than LMedS. In spite of these advantages, we concentrate in the following on LTS regression to estimate camera parameters from motion information. To simplify the notation in the following, we combine the residuals r_{x_i} and r_{y_i} into a single vector r_i of size $M := 2N$. The LTS estimator then may be written as

$$\min_{\hat{\theta}} \sum_{i=1}^{h} \left(r^2\right)_{i:M} . \tag{2.34}$$

Similar to the LS method, the sum of squared residuals is minimized. However, only a subset of h cases is taken into account. These cases are chosen from the data set according to the squared magnitude of the residuals. Let $(r^2)_{1:M} \leq \ldots \leq (r^2)_{M:M}$ denote the squared residuals given in ascending order. Then, only the h cases belonging to the least squared residuals $(r^2)_{1:M} \leq \ldots \leq (r^2)_{h:M}$ are considered. Setting h to approximately $n/2$ eliminates half of the cases from the summation, thus, the method can cope with about 50% outliers.

Since the computation of the LTS regression coefficients is not straightforward, the basic steps are summarized in the following. For a detailed explanation and a fast implementation called FAST-LTS we refer to [106, 107].

First of all, in order to calculate residuals, an initial estimate of the regression parameters is required. Let us assume that we want to estimate p parameters $\hat{\theta}^0 := \{\hat{\theta}^0_0, \ldots, \hat{\theta}^0_{p-1}\}$ ($p = 8$ in our context). Then, a subset of p cases, a so-called p-subset, denoted as S^0, is drawn randomly from the data set. Solving a linear system created by this subset yields an initial estimate of the parameters. Note that if the matrix constructed from the p-subset is of rank less than p, random cases are added until this requirement has been fulfilled.

Using $\hat{\theta}^0$, we can calculate the residuals for all cases in the data set. Let us denote these residuals by $r_{0i}, i = 1, \ldots, n$. Sorting r_{0i} by absolute value, we can create a subset H^0 that contains the h cases with the least absolute residuals. Furthermore, a first quality of fit measure can be calculated from this subset as $Q^0 := \sum_{i \in H^0} (r^2_0)_{i:M}$. Then, based on H^0, a least squares fit is calculated, yielding a new parameter estimate $\hat{\theta}^1$. Again, the residuals r_{1i} are calculated and sorted by absolute value in ascending order, yielding a subset H^1 that contains the h cases that possess the least absolute residuals with respect to the parameter set $\hat{\theta}^1$. The quality of fit for H^1 is given by $Q^1 := \sum_{i \in H^1} (r^2_1)_{i:M}$. Due to the properties of LS regression, it can be assured that $Q^1 \leq Q^0$. Thus, by iterating the above procedure until $Q^n = Q^{n-1}$, the optimal parameter set can be determined.

However, in general, Q^n constitutes only a local minimum of the LTS objective function (cf. Equation 2.34), which depends on the initial p-subset S^0. In order to find the global optimum, one must ensure that the initial subset does not contain any outliers. Let us denote the fraction of outliers in the data set as ϵ. Then, the probability that a subset of p cases will contain at least one outlier is $(1 - (1 - \epsilon)^p)$. Consequently, if m initial subsets

$S^i, i = 0, \ldots, m-1, S^i \neq S^j$ for $i \neq j$, are drawn from the data set, the probability of one "good" subsample, i. e., a p-subset without outliers, is given as

$$P(\epsilon, p, m) := 1 - (1 - (1 - \epsilon)^p)^m. \tag{2.35}$$

By requiring that this probability be close to one, a suitable value for m can be calculated for given ϵ and p. For example, in the context of camera motion estimation ($p = 8$), under the assumptions that nearly 50% outliers are present in the data set and that $P(0.5, 8, m) \geq 0.95$, 766 randomly drawn subsamples are required to obtain a good p-subsample. Then, the parameters corresponding to the global minimum of Equation 2.34 can be estimated up to a certain probability by using each subsample S^i as an initial subset for the iterations as described above. Finally, the solution is retained that provides the lowest value of the LTS objective function. We should note that the FAST-LTS algorithm [107] applies several techniques to reduce the computational complexity. First, the number of iterations is reduced by a technique called *selective iteration*. Second, special data structures, so-called *nested extensions*, are employed in the case of very large data sets.

We are now in a position to summarize our algorithm for the robust estimation of camera parameters.

(1) Determine N feature correspondences $(x'_i, y'_i) \leftrightarrow (x_i, y_i)$ by employing the motion analysis techniques described in the previous sections. Do not consider image positions where only normal motion can be calculated.

(2) Derive the overdetermined linear system of equations from the correspondences and the underlying perspective camera motion model (Equation 2.30).

(3) Estimate the camera parameters $\theta_0, \ldots, \theta_7$ by applying LTS regression to the linear system (Equation 2.34).

2.5.3 Experimental Results

First of all, we illustrate the robustness of the algorithm by repeating our experiment based on synthetic flow fields. Figure 2.27 displays the results obtained for different mixtures of object and background motion. The first case shown in the top row resulted from the configuration already used to illustrate the limitations of LS minimization (see Figure 2.26). As previously said, about 8% of the pixels were assigned to background motion and 2% belonged to the moving object. In subsequent experiments, we changed this ratio

and added further object pixels. The middle row of Figure 2.27 depicts a combination of 5.5% background pixels and 4.5% object pixels. Similar to the first case, the robust camera motion estimator was able to calculate the correct parameters as indicated by the accompanying flow field.

In the last case (cf. Figure 2.27e), containing 4% background pixels and 6% object pixels, the camera motion estimation fails. Since there are more object pixels than background pixels, the algorithm calculates the camera parameters from the moving object, while it ignores the background pixels. Consequently, in order to function properly, the camera motion estimator requires a dominant camera motion, i. e., the motion vectors resulting from background motion must outnumber those obtained from moving objects.

Next, we applied our algorithm to the synthetic tree sequences already used in Sections 2.3.4 and 2.4.2. The goal in these experiments was to reconstruct a dense motion vector field from sparse motion estimates and the estimated camera parameters. By comparing the reconstructed motion vector fields to the known motion fields of the synthetic sequences, we examined the accuracy of the calculated camera parameters.

First, we employed tensor-based motion estimation (*3dst-nonms*) to calculate sparse motion vector fields for both the *translating tree* and the *diverging tree* sequences (cf. Section 2.3.4). The camera parameters were then determined, and a dense motion vector field was constructed by evaluating the camera parameters at each image position. Finally, we compared the reconstructed motion fields to the known motion vector fields for both sequences. Table 2.3 shows the results obtained using our method. Both results indicate that the camera parameters could be extracted from the sparse motion vector field fairly accurately.

sequence	average error	standard deviation	density
translating tree (frame 20)	0.51°	0.11°	100%
diverging tree (frame 20)	1.57°	0.30°	100%

Table 2.3: Camera motion estimation: Full velocity results for translating and diverging tree sequences ($\rho = 2, C = 1$).

In the remainder of the section, we examine the performance of the algorithm on real-world sequences containing moving objects. For this purpose, we calculated camera motion estimates for four well-known video sequences, namely the *Hamburg taxi* sequence (static camera), the *hall-and-monitor* sequence (static camera), the *coastguard* sequence (panning and tilting camera), and the *Stefan* sequence (panning and zooming camera).

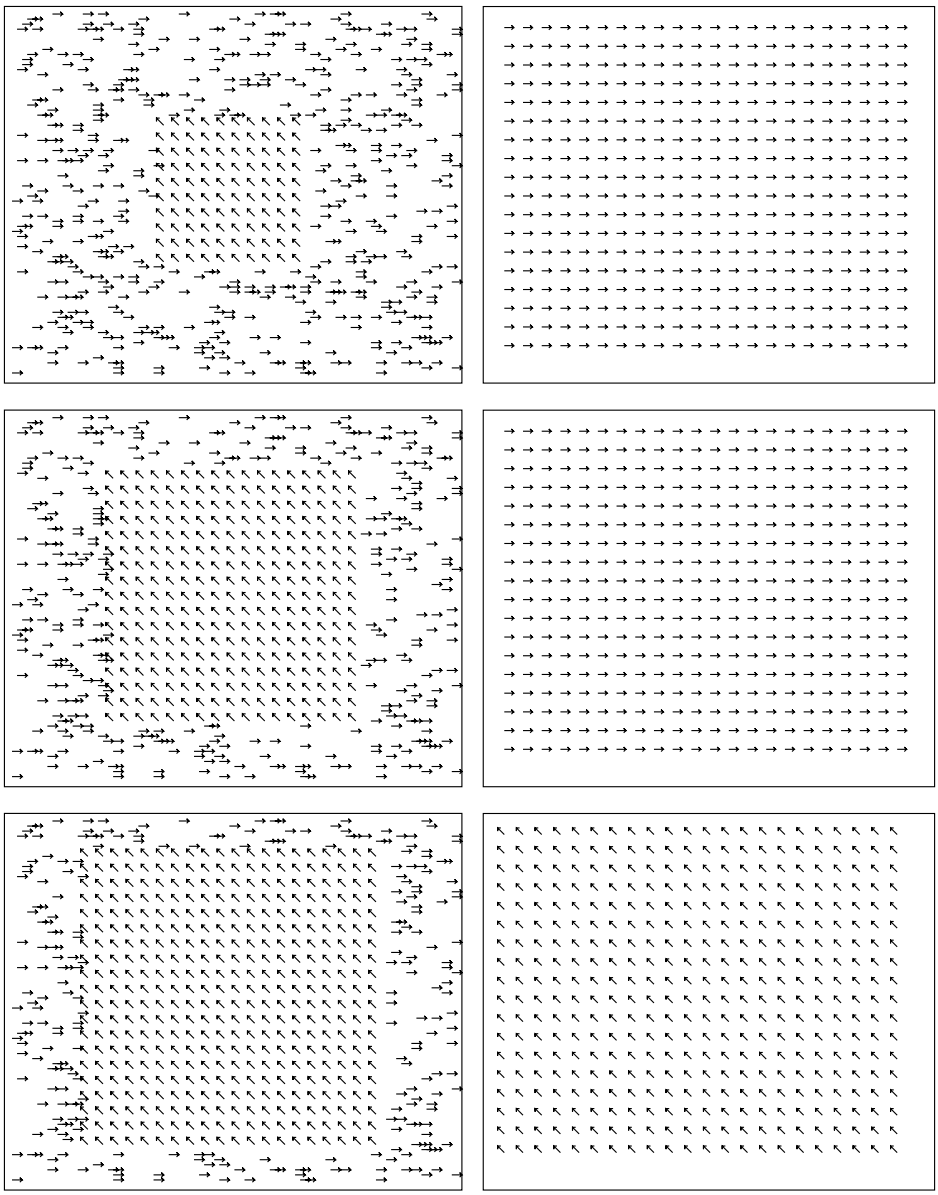

Figure 2.27: Determining camera parameters by means of least trimmed squares regression. (a) Top left: Flow field calculated from a sequence containing one moving object (8% background pixels, 2% object pixels), (b) Top right: Estimated camera motion, (c) Middle left: 5.5% background pixels, 4.5% object pixels, (d) Middle right: Estimated camera motion, (e) Bottom left: 4% background pixels, 6% object pixels, (f) Bottom right: estimated camera motion. The flow fields are subsampled for better visibility.

Correspondences between image features in consecutive frames were determined by either tensor-based motion estimation or feature-based motion estimation.

Figures 2.28–2.31 display several examples from the different sequences. The top row of each figure shows the original frame and the calculated flow field. The bottom row displays the flow field generated from the estimated camera parameters. For better visibility, all flow fields are subsampled. Note that the camera flow fields indicate how image positions in consecutive frames are mapped to one another rather than show how the camera moves.

In all cases, the robust camera motion estimator was able to determine the camera operation, despite the presence of considerable object motion in each case. In detail, for the example from the *Hamburg taxi* sequence, 8871 feature correspondences were calculated by tensor-based motion estimation. As can be seen in Figure 2.28, four moving objects, namely a taxi (middle), a car (left), a van (right), and a pedestrian (top left) are the source of image motion. However, the features in the static background outnumber the motion vectors resulting from the moving objects. Thus, the camera parameters could be calculated reliably.

The flow field of the *coastguard* example (cf. Figure 2.29) resulted from 9315 motion vectors determined by means of tensor-based motion estimation with non-maximum suppression. Here, object motion results from two boats. Again, it was possible to detect sufficient features in the background and thus enable camera motion estimation.

Since the *Stefan* sequence (cf. Figures 2.30 and 2.31) contains large displacements, we employed feature-based motion estimation. In the first example from the *Stefan* sequence, 6880 correspondences were determined. Motion vectors that did not fit into the camera motion model resulted from the moving player and motion in the audience. Nevertheless, the algorithm determined the camera parameters correctly, indicating a pan. In the second example from the *Stefan* sequence, 9618 features could be identified. From these features, some belonging to the moving player and the tennis net, the camera zoom could be calculated.

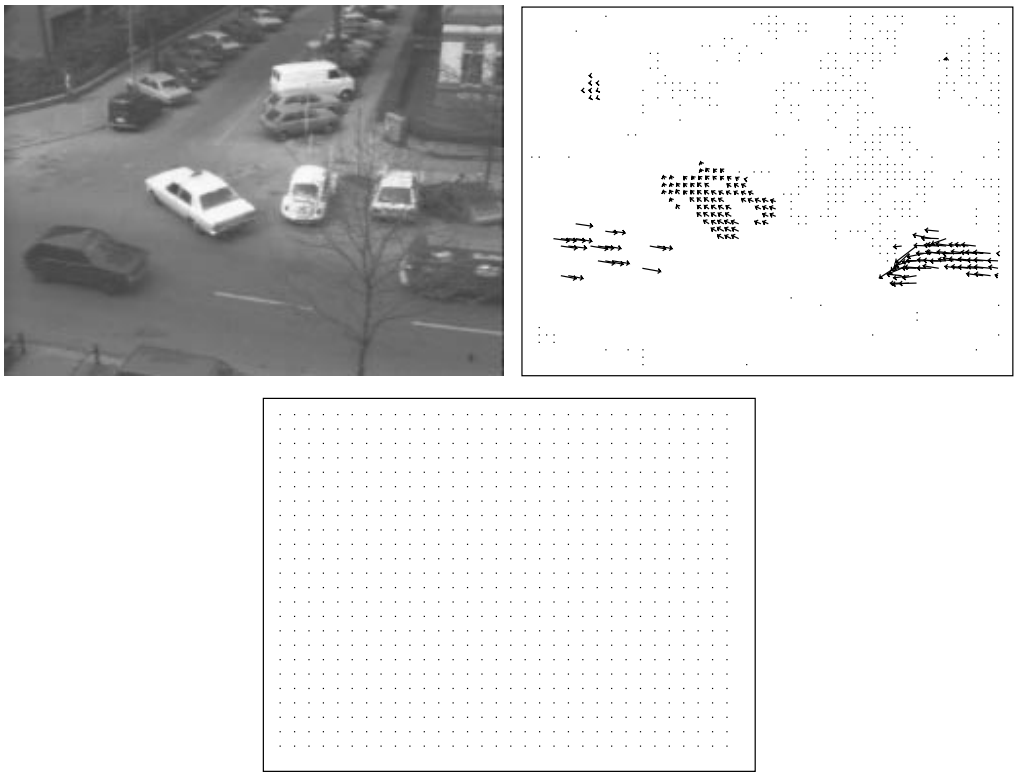

Figure 2.28: Camera motion estimation for the *Hamburg taxi* sequence.

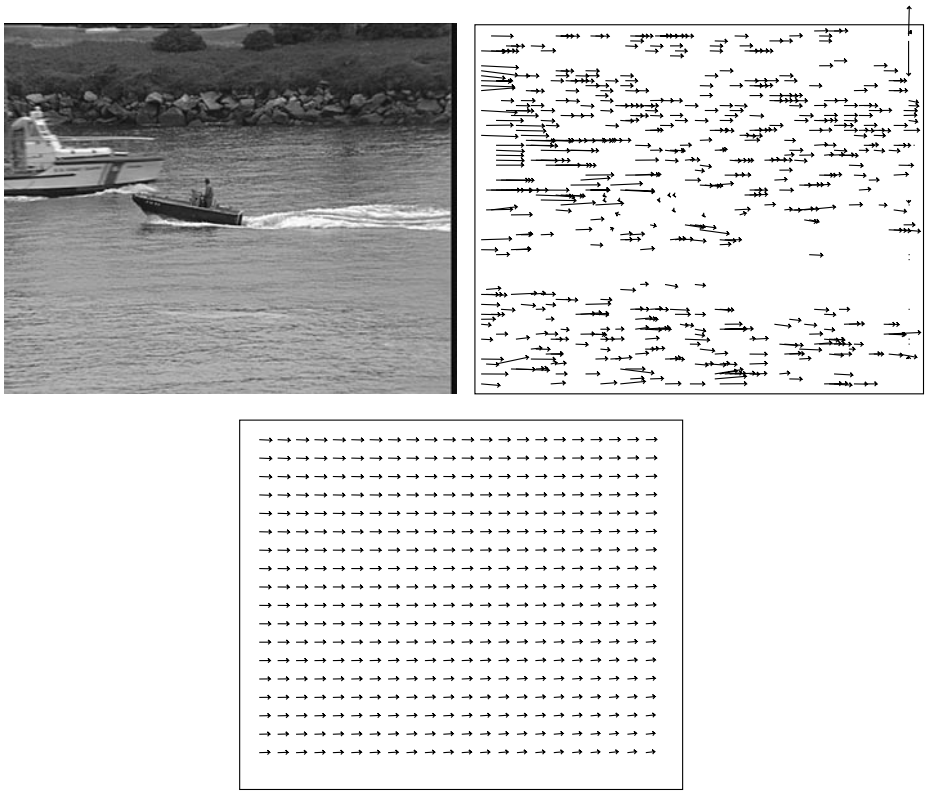

Figure 2.29: Camera motion estimation for the *coastguard* sequence.

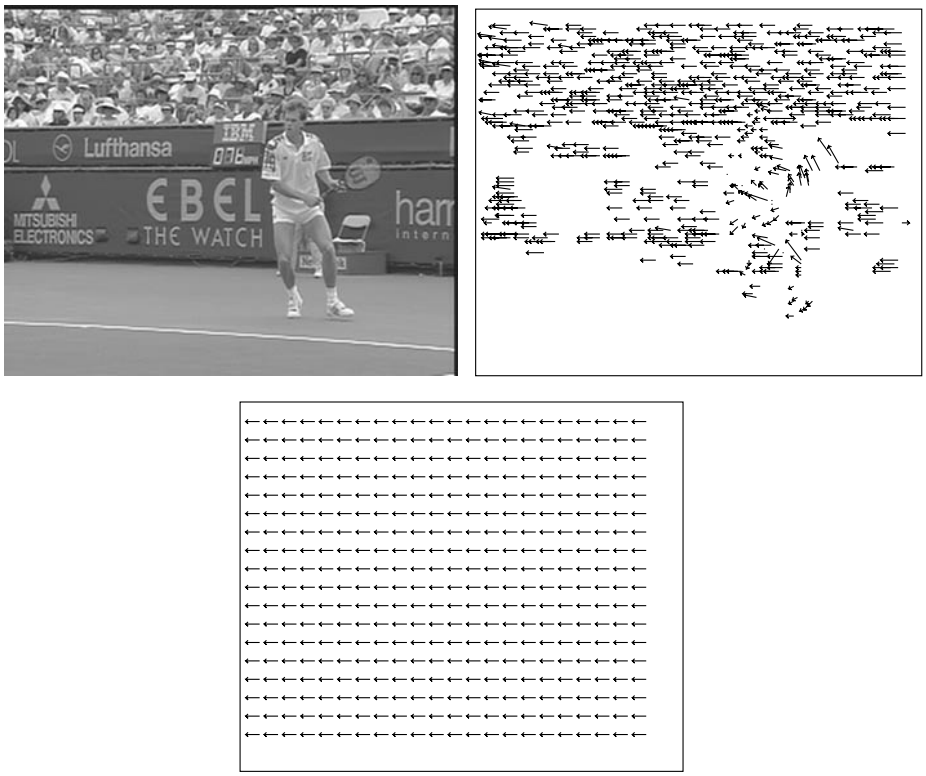

Figure 2.30: Camera motion estimation for the *Stefan* sequence (1).

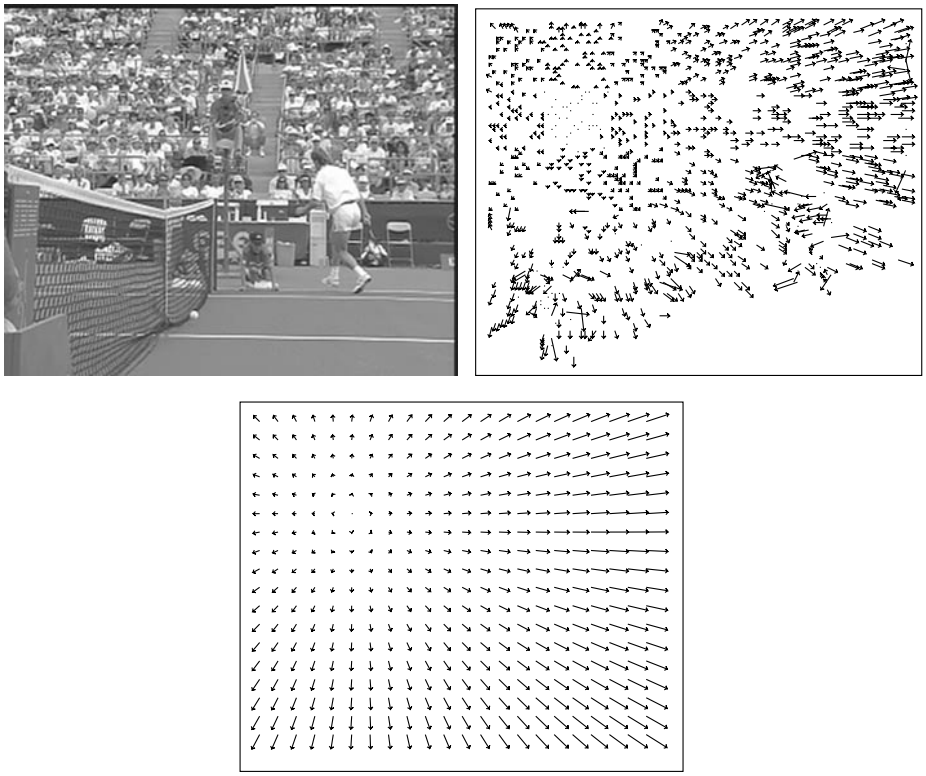

Figure 2.31: Camera motion estimation for the *Stefan* sequence (2).

Chapter 3

Motion-based Object Segmentation

3.1 Introduction

The motion-based approaches developed in the previous chapter are able to detect motion areas reliably. However, as described above, the aperture problem might cause parts of the objects to be left out. Due to areas of constant gray values within the moving objects, we do not receive dense motion vector fields. The identified regions of a moving object thus contain gaps and holes. Consequently, to extract the complete object from the video sequence, a grouping step is needed that integrates local information obtained from the motion detection algorithms. Ideally, such a grouping step should fulfill a number of properties: The final contour which separates objects from the background should reproduce the boundaries apparent in the image. In addition, the contour should be piecewise smooth and respect singularities such as angles and corners. Furthermore, missing parts should be approximated naturally, and the grouping step should be able to find several objects simultaneously. Finally, since we are working on image sequences, computational efficiency is an another important aspect.

Widely used in the problem domain of grouping local information are *active contour models*, also known as *deformable models*, *snakes* (in 2D), or *active surfaces* (in 3D). In the following we restrict our discussion to the two-dimensional case. However, most of the explanations below can be extended to three dimensions.

Thus, by applying active contour models to information gained from the motion analysis techniques as developed in the previous chapter, we can develop efficient approaches for moving object segmentation. Furthermore, by taking camera motion into account, segmentation becomes possible in the event of a moving camera.

The novelties presented in this chapter can be subdivided as follows:

(1) Efficient numerical implementations for implicit active contour models are developed. In particular, we provide a unified model for geometric and geodesic active contours. The implementation consists of a novel narrow-band approach and an efficient semi-implicit numerical discretization.

(2) Two new algorithms for moving object segmentation based on the motion detection results from the previous chapter and the new implicit active contour algorithms are developed. The first algorithm is related to object segmentation from sequences recorded with a static camera, while the second is able to cope with a moving observer.

The remainder of this chapter is organized as follows: After discussing related work in Section 3.2, Section 3.3 introduces active contour models. In Section 3.4 our fast algorithms for implicit geometric and implicit geodesic active contours are presented. Section 3.5 describes the motion-based segmentation algorithms based on the integration of motion information into the implicit active contour models and provides experimental results.

3.2 Related Work

Motion-based segmentation techniques can be categorized based on their basic assumptions as to sequence acquisition. In the simplest case it is taken for granted that a *static camera* records a *moving object*. Even more constraining is the assumption that a background reference frame of the sequence be available.

Performance of the segmentation task under the assumption of a *moving camera* is a greater challenge. Here, it is often assumed that a dominant camera motion can be determined from the sequence. Usually this implies that a large part of the image domain is taken up by background. Given these assumptions, camera motion compensation can be performed.

The most general approaches partition an image into regions of different motion characteristics, so-called *motion regions*. However, this may result in a large number of regions since different parts of a non-rigid moving object might undergo different motions. Consequently, recovery of objects from motion regions is no trivial task.

In recent years various motion segmentation algorithms have been proposed. An early survey dating from 1981 can be found in [104], while a more recent overview is provided by Mitiche and Bouthemy [86]. Since our approach is based on the integration of motion

cues into an implicit active contour model, we restrict our discussion on related work to that which also employs implicit contour evolution methods.

Caselles and Coll [22] proposed a geometric partial differential equation for segmenting and tracking moving objects in a video sequence. For this purpose, they modified their earlier geometric model for active contours [21] by adding a motion-based term. First, the object to be tracked is detected from an initial image based on spatial edges. Then, the motion vector field along the contour of the detected object is calculated and used to compute an initial estimate of the contour in the next frame. Finally, to account for possible errors, the estimated contour is dilated, and the process iterates. While the approach of Caselles and Coll combines segmentation and tracking, the segmentation part holds only under the assumption that the object can be segregated from the background based on its spatial edges, i. e., either the object must be placed on a uniform background or an initial contour close to the object must be provided by the user. Moreover, the model does not include any camera operations.

Goldenberg, Kimmel, Rivlin, and Rudzsky [37, 38] developed a fast version of the geodesic active contour model and demonstrated the capacity of their method to segment and track moving objects. In their approach, object segmentation is achieved by integrating temporal and spatial edges into the contour model. The segmentation procedure as proposed by Goldenberg et al. assumes that the moving objects have been recorded by a static camera. Note that a specific reinitalization procedure has to be carried out in each iteration of the contour evolution.

Paragios and Deriche [99] proposed a variational framework based on geodesic active contours to segment and track moving objects. Segmentation is performed by using inter-frame differences within a statistical framework. In addition, the spatial edges in the original frame contribute to the boundaries of the moving object. Like the other techniques discussed above, the approach of Paragios and Deriche is limited to the static camera scenario.

Jehan-Besson, Barlaud, and Aubert [55, 56] employed region-based active contours to segment moving objects in a video sequence captured by a moving camera. Camera motion compensation and segmentation are formulated jointly in their approach within a minimization criterion. First, the motion vector field is computed by a classical block-matching algorithm. Then, by using a 6-parameter affine motion model and by introducing a potential function for eliminating outliers, a descriptor for the region-based model is obtained. While Jehan-Besson et al. provide a powerful framework for object segmentation under dominant camera motion, they do not discuss any efficient numerical implementations.

Zhang, Gao, and Liu [146] combine similar to our approach the 3D structure tensor with an implicit active contour model. However, in their approach the structure tensor is not used for computing motion estimates explicitly. Instead, the smallest eigenvalue of the tensor is used as a robust frame difference. To handle camera movements, a global motion compensation is performed prior to the tensor calculations. Since the structure tensor in its standard implementation can be applied only to sequences containing small displacements, the approach of Zhang et al. is not applicable in the event of large object motion.

Sifakis, and Tziritas [118] propose a multi-label fast marching algorithm for segmentation. Moving objects are detected by means of inter-frame differences modeled within a statistical framework. Based on these statistics the boundary of the moving object is determined by means of two curves evolving in opposite directions. *Sifakis, Grinias, and Tziritas* [116, 117] extend this approach by also considering moving cameras. For this purpose, they estimate a three-parameter camera motion model (two translation and one zoom parameter). Finally, *Sifakis, Garcia, and Tziritas* [115] integrate also the segmentation of motion fields.

Mansouri, Sirivong, and Konrad [75] provide a level set based approach for multiple motion segmentation. In contrast to the approaches of Jehan-Besson et al., Zhang et al. summarized above and to our own they do not assume a dominant camera motion. Instead, based on the correspondence of feature points and an affine motion model a number of motion classes are determined. These motion classes serve to initialize the segmentation process.

In our approach, we develop motion-based segmentation algorithms that incorporate both fast implicit active contour models and camera motion estimation. In addition, we provide a unified model for geometric and geodesic active contours.

3.3 Active Contour Models

Active contours are based on ideas of curve evolution. Basically, a closed planar curve is placed around image parts of interest and evolves under equations of motion. Usually smoothness control (internal contour energy) and force terms derived from the image (external image energy) are included in the evolution process. The main advantage of this approach is the imposition of geometric constraints on the segmentation problem. Missing information, e. g., parts of object boundaries, are approximated by smooth segments of the evolving contour.

A fundamental aspect is the representation of the evolving curve. A straightforward approach is to describe the contour *explicitly* as a parametric curve. On the other hand, the curve can be represented *implicitly* by embedding it as a level set into a function. Both representations will be considered in more detail in the following.

We now discuss the main active contour models while concentrating upon implicit snakes. We begin with a description of explicit snakes, and then present implicit active contour models. Finally, we proceed to develop novel fast algorithms for the latter class of models.

3.3.1 Explicit Active Contour Models

An explicit snake is represented as a parametric curve $C_0(s) = (x(s), y(s)), s \in [0, 1]$, where s is the arc length parameter, and $x(s)$, $y(s)$ denote the x and y coordinates of the curve at arc length s. Let us recall the basic principle of parametric curves by considering Figure 3.1. As depicted, a curve in the plane is represented by two independent functions both depending on a parameter $s \in [0, 1]$.

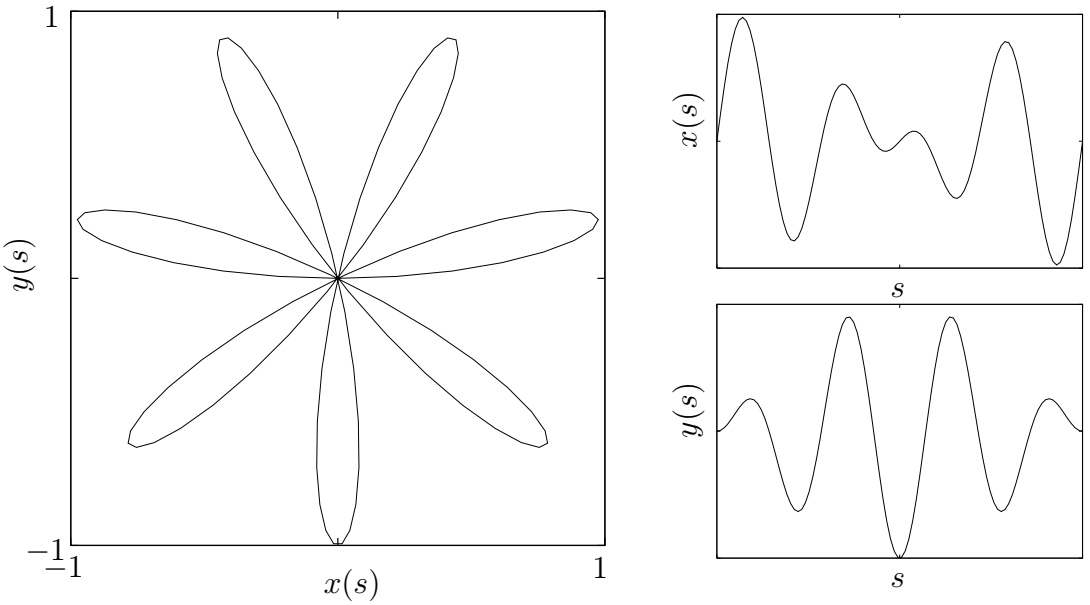

Figure 3.1: A parametric curve $C_0(s) = (\sin(7\pi s)\cos(\pi s), \sin(7\pi s)\sin(\pi s))$, $s \in [0, 1]$. (a) LEFT: The curve $C_0(s)$, (b) RIGHT: Individual components $x(s) = \sin(7\pi s)\cos(\pi s)$, $y(s) = \sin(7\pi s)\sin(\pi s)$.

In the context of explicit snakes, however, only planar curves are considered. Figure 3.2a displays an example. In our discussion, the curve is parameterized counterclockwise,

thus, its interior is on the left in the direction of increasing s. In order to facilitate the transfer of the results obtained below to digital images, we chose a different orientation of the y-axis as usual. In the context of digital images, the parametric curve must be discretized. As illustrated in Figure 3.2b, this is usually achieved by placing marker points along the curve. During curve evolution these marker points are relocated on their way through the fixed grid. The final contour can be determined from the marker particles by linear or spline interpolation. In the following sections two classical approaches for developing curve evolution equations are summarized.

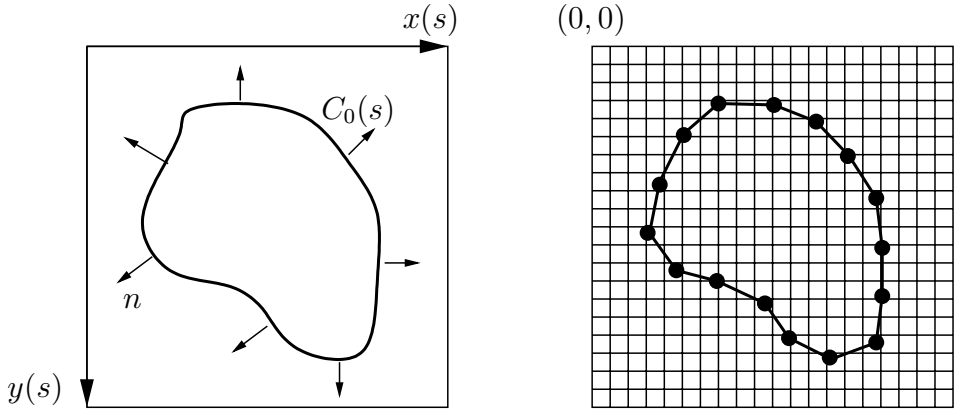

Figure 3.2: Explicit representation of an active contour. (a) LEFT: Parametric curve $C_0(s) = (x(s), y(s))$ in $\mathbb{R}^2$, with outward normals n_i (b) RIGHT: Discretization by placing marker particles and interpolating linearly.

3.3.1.1 Explicit Geometric Active Contour Model

Consider a planar curve C_0 placed at a certain distance around an object of interest. To extract the object it is necessary to move the contour towards the object's boundary. Consequently, some kind of shrinkage procedure has to be developed. In the following we discuss such a procedure based on geometric considerations. The final model obtained in this section is termed the *geometric active contour model*.

Let $C(s,t)$ denote the family of curves parameterized by $s \in [0,1]$, where $t \in [0,\infty)$ is time. Then, the following evolution equation propagates the curve along its normal vector field (see Figure 3.2a) at a fixed speed [112]:

$$\begin{aligned} \frac{\partial C(s,t)}{\partial t} &= k \cdot n(s,t) \quad \text{on} \quad [0,1] \times (0,\infty) \\ C(s,0) &= C_0(s) \quad \text{on} \quad [0,1]. \end{aligned} \tag{3.1}$$

In the equation, k is a scalar value (the force term) and $n(s,t)$ denotes the outward unit normal vector at a curve position $(x(s,t), y(s,t))$ given as

$$n = \frac{1}{(x_s^2 + y_s^2)^{1/2}} \begin{pmatrix} -y_s \\ x_s \end{pmatrix}. \tag{3.2}$$

In this context, x_s and y_s denote the usual abbreviations for the first derivative with respect to s. Figure 3.3 illustrates the evolution process under the constant force k. While for a positive force term the contour expands (top row), a negative value for k causes the contour to shrink (bottom row).

Thus, using a negative force term, the evolution equation guides an explicit snake surrounding an object of interest towards the object boundary. However, stopping the curve when reaching the boundary requires additional terms in the equation. Furthermore, it is often desired to keep the evolving contour as smooth as possible. For instance, this is useful when missing parts of the object must be replaced by curve segments.

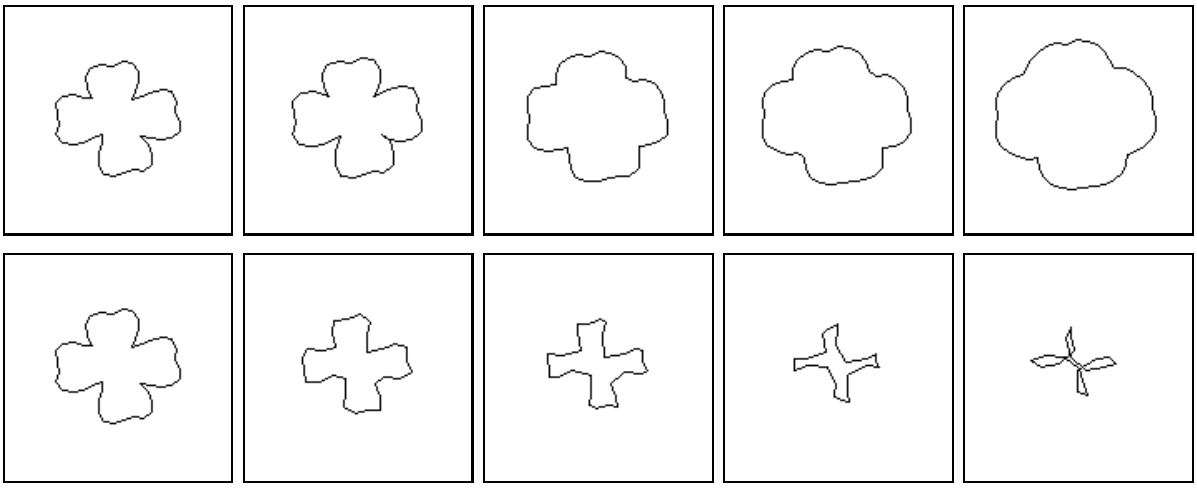

Figure 3.3: Explicit snake evolving at constant speed. (a) TOP ROW: $k > 0$, (b) BOTTOM ROW: $k < 0$.

To impose smoothness constraints on the evolution process the constant force term k can be replaced by the contour's curvature κ [112]. Recall that the curvature expresses how much the curve "bends" at each point. It is well known that the curvature of a parametric curve at a point $p = (x, y)$ is calculated as follows:

$$\kappa = \frac{y_{ss}x_s - x_{ss}y_s}{(x_s^2 + y_s^2)^{3/2}}. \tag{3.3}$$

According to the chosen parameterization and the underlying coordinate system, the curvature of a concave curve segment is positive, while a convex segment results in a negative curvature. Consequently, different segments of the curve cause motion in different directions. In our case, the contour moves outwards for $\kappa > 0$, but inwards for $\kappa < 0$. Figure 3.4 displays the curve's evolution under mean curvature motion. We observe that the evolving curve is smoothed, becoming convex after a certain period of time. In fact, each planar curve evolving under mean curvature becomes convex and shrinks to a round point [43].

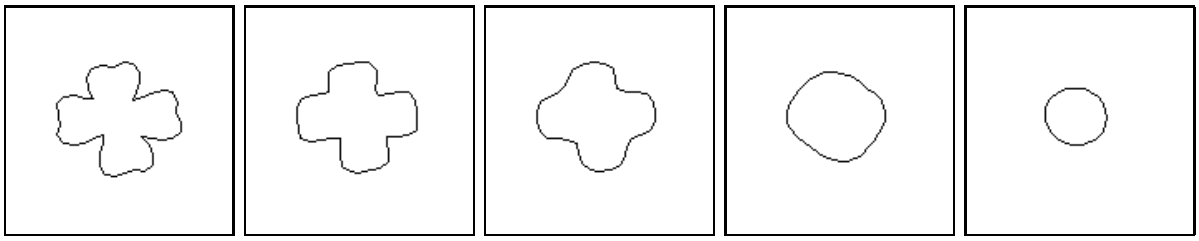

Figure 3.4: Explicit snake evolving under mean curvature motion.

In order to allow initially convex curves to become non-convex, and so as to detect non-convex objects, constant evolution and curvature-based smoothing must be combined, yielding the following equation:

$$\frac{\partial C(s,t)}{\partial t} = (k + \kappa(s,t)) \cdot n(s,t). \tag{3.4}$$

Finally, to integrate information from the object to be detected, we need an external force that will stop the evolution once the contour reaches the object's boundary. Let us define a *stopping function* $g : I\!R^2 \rightarrow (0, 1]$ based on a simple edge detector (we will discuss the integration of results gained from the motion analysis later on):

$$g(x) = \frac{1}{1 + |\nabla f_\sigma(x)|^2}. \tag{3.5}$$

Here, $\nabla f = (\partial_x f, \partial_y f)^T$ denote the spatial gradient and f_σ denotes the image containing the object under consideration smoothed with a Gaussian kernel of standard deviation σ. While close to edges (high gradient magnitudes) the stopping function approaches zero, it reaches one in flat areas of the image (low gradient magnitudes).

Thus, the final curve evolution equation is obtained from Equations 3.4 and 3.5 as

follows:

$$\frac{\partial C(s,t)}{\partial t} = g(C(s,t))(k + \kappa(s,t)) \cdot n(s,t). \tag{3.6}$$

3.3.1.2 Explicit Energy-based Active Contour Model

Energy-based snakes were introduced by Kass, Witkin and Terzopoulos in [57, 58] and have been used in a wide range of applications since then. In their approach a parametric curve is embedded into an *energy-minimizing framework*. The energy

$$E(C(s,t)) = E_{\text{int}} + E_{\text{ext}} \tag{3.7}$$

is composed of an internal and an external energy term.

The internal energy of the snake is given as

$$E_{\text{int}}(C(s,t)) = \oint_{C(s,t)} \left(\tfrac{\alpha}{2}|C_s(s,t)|^2 + \tfrac{\beta}{2}|C_{ss}(s,t)|^2\right) ds \tag{3.8}$$

where the parameters $\alpha > 0$ and $\beta > 0$ are the elasticity and the rigidity coefficient, respectively. Minimizing the internal energy results in a smooth contour.

The external energy depends on image features, for instance, on edge information, as in the following definition

$$E_{\text{ext}}\left(C(s,t)\right) = -\oint_{C(s,t)} \gamma|\nabla f_\sigma(C(s,t))|^2 ds. \tag{3.9}$$

Minimizing E_{ext} pushes the contour towards the selected image features.

In [58] the functional defined in Equation 3.7 is minimized using the calculus of variations. From the Euler-Lagrange equations the following iterative procedure is derived:

$$\frac{\partial C(s,t)}{\partial t} = \alpha C_{ss}(s,t) + \beta C_{ssss}(s,t) - \gamma\nabla(|\nabla f_\sigma(C(s,t))|^2). \tag{3.10}$$

The final contour is obtained by iterating Equation 3.10 until stability has been achieved.

Several alternate approaches to energy minimization have been proposed. For instance, in [5, 6] an algorithm based on dynamic programming is described. Williams and Shah developed a greedy optimization technique that relies on local neighborhood search [143].

In addition to the different energy minimization techniques a number of modifications to the original model were introduced. Here, we mention only the balloon force proposed

by Cohen [27, 28] since it is related to the constant force term of the above-mentioned geometric model. In Cohen's approach the following force term is added to the right-hand side of Equation 3.10:

$$k \cdot n(s,t) \tag{3.11}$$

where k denotes the amplitude of the force and $n(s,t)$ is the unit normal vector to the curve at point $C(s,t)$.[1] Depending on the sign of k, this additional term acts as an inflation or deflation force in the model. Thus, the curve evolution is not disturbed by weak external forces such as those caused by small image gradients.

3.3.1.3 Numerical Implementation

In the following we discuss the numerical implementation of the active contour models described above. While we concentrate on the geometric model, the results obtained below can be adapted to energy-based snakes as well.

In order to provide a numerical algorithm for Equation 3.6, one has to consider discretizations of space (x_s, y_s, x_{ss}, y_{ss}) and time ($\partial C/\partial t$). First, we discretize the parametric curve by a collection of equally distributed marker points (x_i, y_i) (cf. Figure 3.2b). Each marker point corresponds to some location $i \cdot h$ on the contour, where h denotes the distance between two neighboring marker points. Second, we employ discrete time steps by dividing the continuous time interval $[0, T]$ into discrete steps τ. Thus, combining space and time discretizations, the collection of marker particles (x_i^n, y_i^n) approximates the parametric curve C at time $n\tau$.

We approximate the spatial derivatives needed for the calculation of the normal and the curvature by central differences based on Taylor series as

$$\begin{aligned} \left.\frac{\partial x(s)}{\partial s}\right|_i^n &\approx \frac{x_{i+1}^n - x_{i-1}^n}{2h}, & \left.\frac{\partial y(s)}{\partial s}\right|_i^n &\approx \frac{y_{i+1}^n - y_{i-1}^n}{2h} \\ \left.\frac{\partial^2 x(s)}{\partial s^2}\right|_i^n &\approx \frac{x_{i+1}^n - 2x_i^n + x_{i-1}^n}{h^2}, & \left.\frac{\partial^2 y(s)}{\partial s^2}\right|_i^n &\approx \frac{y_{i+1}^n - 2y_i^n + y_{i-1}^n}{h^2}. \end{aligned} \tag{3.12}$$

The time derivatives are discretized by forward difference approximations

$$\left.\frac{\partial x(s)}{\partial t}\right|_i^n \approx \frac{x_i^{n+1} - x_i^n}{\tau}, \qquad \left.\frac{\partial y(s)}{\partial t}\right|_i^n \approx \frac{y_i^{n+1} - y_i^n}{\tau}. \tag{3.13}$$

[1] Note that in the original formulation, Cohen also modified the external force term. However, to keep things simple, we neglect this detail.

Using the above discretizations, the computation of an approximation to the curvature κ_i^n from Equation 3.3 is straightforward:

$$\kappa_i^n = 4\frac{(y_{i+1}^n - 2y_i^n + y_{i-1}^n)(x_{i+1}^n - x_{i-1}^n) - (x_{i+1}^n - 2x_i^n + x_{i-1}^n)(y_{i+1}^n - y_{i-1}^n)}{((x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2)^{3/2}}. \tag{3.14}$$

Using these approximations we can now formulate a simple updating scheme for evolving an active contour under Equation 3.6, i. e., for producing new values (x_i^{n+1}, y_i^{n+1}) from the marker particles given at the previous time step:

$$\begin{aligned} x_i^{n+1} &= x_i^n + \tau g_{x_i^n y_i^n}(\kappa_i^n + k)\frac{-(y_{i+1}^n - y_{i-1}^n)}{((x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2)^{1/2}} \\ y_i^{n+1} &= y_i^n + \tau g_{x_i^n y_i^n}(\kappa_i^n + k)\frac{x_{i+1}^n - x_{i-1}^n}{((x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2)^{1/2}}. \end{aligned} \tag{3.15}$$

Note that $g_{x_i^n y_i^n}$ approximates the stopping function g at position (x_i^n, y_i^n). Since the positions of the marker particles are not necessarily integers, the values of the stopping function must be determined by a suitable interpolation (e. g., bilinear interpolation).

Implementation of the approach described so far is straightforward. However, it suffers from a number of limitations [21, 112]. First of all, since the number of marker particles determines the accuracy of the boundary detection, the segmentation result, i. e., the placement of the final contour, depends on the chosen parameterization.

Second, as marker particles come closer together during curve evolution, quotients of Equations 3.14 and 3.15 approach zero over zero. Thus, the simple iterative updating scheme might become unstable. This problem might be avoided by introducing a redistribution process. Based on given minimum/maximum distance thresholds or a fixed time interval, the marker points are redistributed on the contour according to arc length. However, such a redistribution process might introduce unwanted smoothing effects into the curve evolution.

Finally, the preceding model is not able to handle topological changes, thus, it cannot segment several objects simultaneously. Figure 3.5 illustrates this problem. Here, the goal is to separate four objects from the background. For this purpose, an explicit active contour is initialized appropriately and evolves under the above-mentioned motion equations. Although during the iterations the curve approaches the objects, a correct segmentation cannot be achieved due to its topological inflexibility. Since the marker particles are tracked explicitly and stored, e. g., in an array, the neighborhood relations

between the marker particles are fixed. Therefore, it is not trivial to rearrange the marker particles and split up the snake into multiple curves. We should note, however, that several modifications have been proposed to overcome this limitation [77–81].

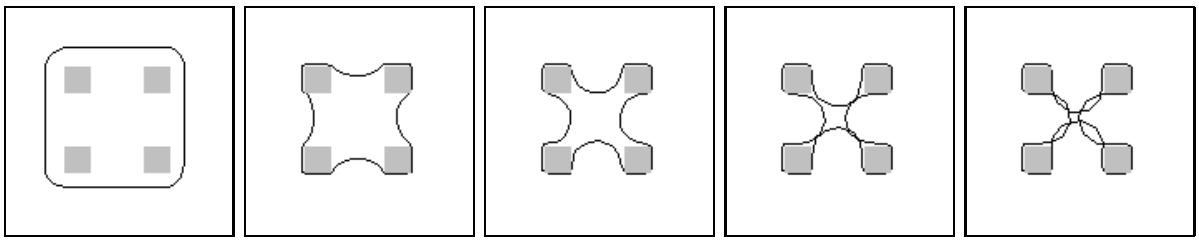

Figure 3.5: Object segmentation using an explicit geometric snake.

Implicit active contour models that will be discussed in the following section are not constrained by these limitations. Indeed, numerical stability, accuracy and topological adaptivity are naturally included in the model.

3.3.2 Implicit Active Contour Models

So far we have represented an active contour by a parametric curve. In a numerical implementation this curve might be approximated by a collection of equidistant marker particles. In the following a different, less intuitive, representation for the active contour is chosen. To remove the need for parametrization, the contour is embedded into an image. For instance, the contour is given by all image positions containing a value of zero. To evolve this *implicitly* represented contour, the entire image has to be considered, and the values must be changed appropriately. Note that the image that represents the contour is different from those which contains, for instance, some objects to segment. Of course, the latter guides the evolution process, but it is independent of the image that holds the contour.

More formally, to represent a snake implicitly, the initial curve C_0 is embedded as a *level set* into a function $u : \mathbb{R}^2 \rightarrow \mathbb{R}$ [98], i. e., the snake is represented by a collection of points having the same value. Figure 3.6a illustrates the level set idea. Here, C_0 is given as the set of points $x_i \in \mathbb{R}^2$ colored in black. To put it differently, under the assumption that a black pixel corresponds to a value of zero, the contour is represented by the set of points x_i with $u(x_i) = 0$. Usually the *signed distance function* is employed to create such

a level set representation:

$$u_0(x) = \begin{cases} d(x, C_0), & \text{if } x \text{ is inside } C_0 \\ 0, & \text{if } x \text{ is on } C_0 \\ -d(x, C_0), & \text{if } x \text{ is outside } C_0 \end{cases} \tag{3.16}$$

where $d(x, C_0)$ denotes the distance from an arbitrary position to the curve. Thus, the initial curve C_0 is embedded as a *zero level set* into the function u_0.

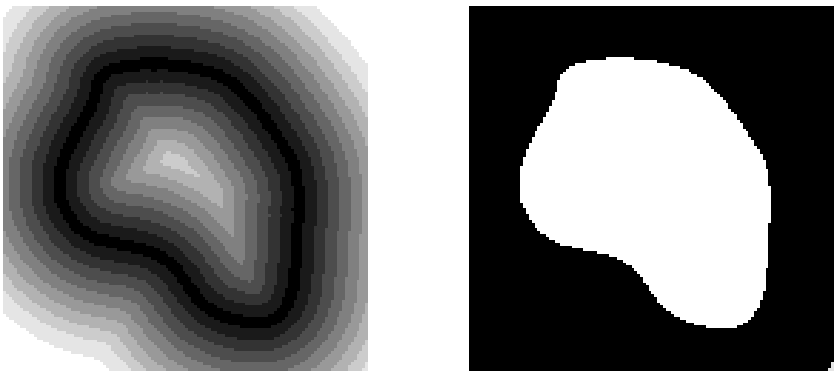

Figure 3.6: Implicit representation of an active contour. (a) LEFT: Discretization by a Euclidean distance transform, (b) RIGHT: Bi-level discretization.

We should emphasize the fundamental difference between explicit and implicit snakes. To evolve an explicit snake, the equations of motion have to be evaluated for a collection of marker particles representing the curve. Moving an implicit active contour, however, requires the evaluation of the motion equations on the entire image domain. This is due to the fact that the contour is modeled implicitly, its exact location is not given explicitly by a collection of marker particles.

In order to work with digital images, we have to consider an appropriate discretization of the level set representation. The signed distance function can be transferred to the discrete domain by applying a Euclidean distance transform and setting the signs of the resulting distance values appropriately. Figure 3.6a illustrates the discretization of the level set representation by such a transform. Note that only absolute values are shown. The contour is represented as the set of points where $u = 0$ (black pixels in the figure). The Euclidean distance for a point lying inside or outside the contour is assigned to the corresponding pixel (the greater the distance, the brighter the pixel value).

Several algorithms have been proposed for calculating the signed distance function. As mentioned above, a straightforward approach is to apply a Euclidean distance transform

and set the signs afterwards. Van den Boomgard [134] introduced an efficient algorithm for computing the exact Euclidean distance based on separable erosions. Another approach to calculate distances is provided by the fast marching method [48, 113, 132]. In this method grid points near the contour are detected, and their distances are calculated with sub-pixel accuracy. Those grid points are used for initialization, and the distances are propagated away from the contour. However, in order to obtain a signed distance function, the fast marching method has to be executed twice, first, to calculate the distances outside and second, to compute those inside the contour.

The PDE-based reinitialization technique introduced by Sussman, Smereka, and Osher [125] and modified in [102, 123, 124] does not have this drawback. In their approach, the auxiliary partial differential equation

$$\begin{aligned} \frac{\partial \phi(x,t)}{\partial t} &= \operatorname{sign}(\phi_0(x))(1 - |\nabla \phi(x,t)|) \\ \phi(x,0) &= \phi_0(x) \end{aligned} \tag{3.17}$$

is solved. Given any initial data for ϕ_0, iterating this equation until convergence results in a signed distance function whose zero level set is the same as that of the initial data. Note, however, that usually a number of iterations are necessary for convergence.

In addition to the signed distance representation, we consider a simpler discretization that we call *bi-level representation* in the following. The level set function is discretized by two values a and b, $a < b$, with one value (a) for positions outside the curve and the other (b) for positions inside the curve. The curve itself is given as the transition between the two levels. Figure 3.6b displays a bi-level representation.

In order to move the curve, the function u is evolved under a partial differential equation. Thus, instead of being tracked explicitly, the curve is implicitly captured on the fixed grid. The final contour can be easily extracted from the function u.

To carry over curve evolution results obtained above on explicitly modeled (parametric) curves to the implicit domain, we consider the zero level set of the function u and its changes over time. Recall that the zero level set represents the curve. Therefore, we differentiate $u(C,t) = 0$ with respect to time t, yielding

$$\begin{aligned} 0 = \frac{\partial u}{\partial t}(x(t), y(t), t) &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial u}{\partial t} \\ &= \nabla u^T \cdot C_t + u_t. \end{aligned} \tag{3.18}$$

Thus, we can employ Equation 3.18 to advance from explicit curve evolution $\partial C/\partial t$

to image evolution $\partial u/\partial t$.

As for the explicit contour models described above, implicit snakes were derived (1) under geometric considerations and (2) within an energy minimization framework. We consider both approaches in the following. Let us first provide two definitions that will prove useful further on. It is well known that normal n and curvature κ with respect to a level set are given by

$$n = -\frac{\nabla u}{|\nabla u|}, \quad \kappa = \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) = \frac{u_{xx}u_y^2 - 2u_x u_y u_{xy} + u_{yy}u_x^2}{(u_x^2 + u_y^2)^{3/2}}. \tag{3.19}$$

Here, with respect to the definition of the level set representation in Equation 3.16, n is the outward normal to a level set. While the curvature κ is negative for convex curve segments, it is positive for concave parts of the curve.

3.3.2.1 Implicit Geometric Active Contour Model

The implicit geometric active contour model was proposed by Caselles, Catté, Coll, and Dibois [21] and later on by Malladi, Sethian, and Vemuri [73]. It can be obtained by replacing C_t in Equation 3.18 by Equation 3.6:

$$\frac{\partial u}{\partial t} = -\nabla u^T \cdot (g(x)(\kappa + k) \cdot n)\,. \tag{3.20}$$

The original model is obtained by replacing the normal n and the curvature κ by the expressions from Equation 3.19:

$$\frac{\partial u}{\partial t} = g(x)|\nabla u| \left(\operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) + k\right). \tag{3.21}$$

Remember that k denotes a constant force that causes the contour either to shrink or to expand. And g denotes the stopping function as given in Equation 3.5.

In order to illustrate how the level set function evolves under constant force and mean curvature, we applied the above model to a test image once without the curvature term (pure erosion/dilation) and once without the force term (mean curvature motion). Figures 3.7 and 3.8 display the evolution of the signed distance representation, the bi-level representation and the zero level set (extracted from the signed distance representation) over time. Note that the evolution of a contour represented by a signed distance function slightly differs from the corresponding bi-level representation. We will discuss this difference later on.

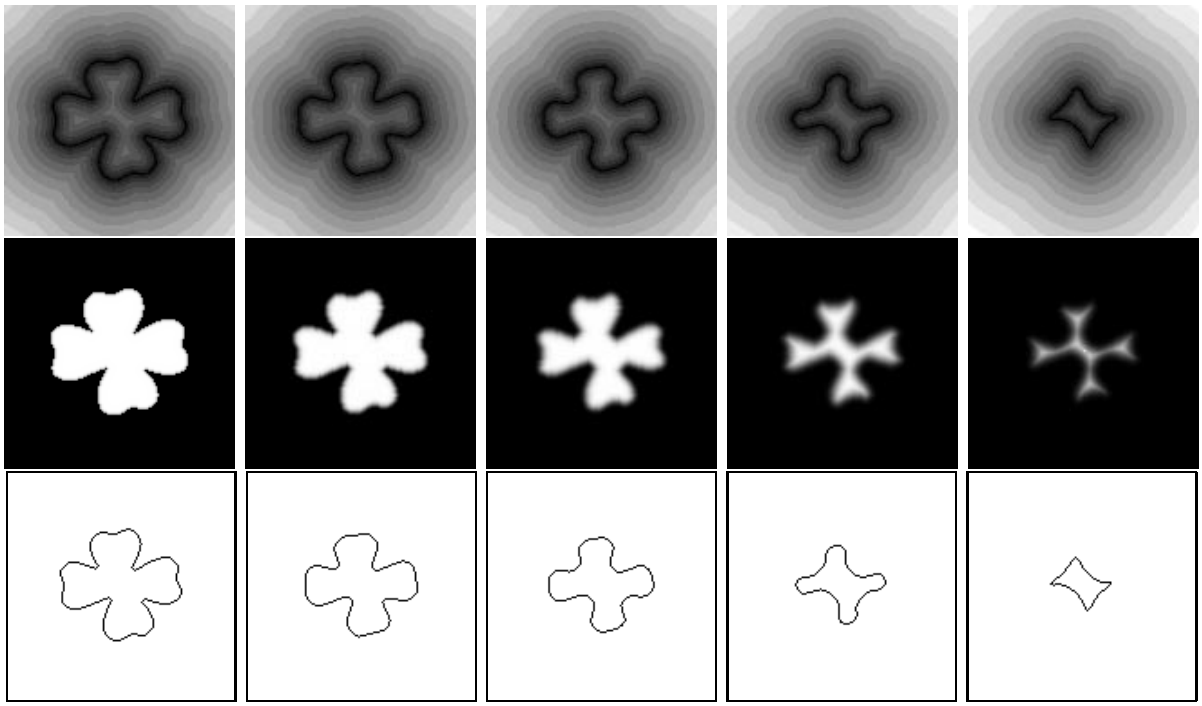

Figure 3.7: Implicit snake evolving at constant speed ($k < 0$). (a) TOP ROW: Signed distance representation, (b) MIDDLE ROW: Bi-level representation, (c) BOTTOM ROW: Extracted contour (zero level set) from signed distance representation.

Let us now consider object segmentation with the implicit geometric active contour model. Figure 3.9 displays the segmentation of an object (shown in light gray) by an implicit snake. Setting the force amplitude k to zero (see top row) yields the following observations: The shrinking process of the contour starts at the corners of its initialization. This is due to the fact that the local curvature κ is unequal to zero at these points. Thus, the temporal change $\partial_t u = g(x)|\nabla u|\kappa$ is also unequal to zero, hence the contour moves. During the evolution process the contour approximates a circle. When approaching the object's boundary, parts of the contour cease to move due to the vanishing stopping function ($g \approx 0$). Note that the contour remains always convex. Consequently, the model is not able to approximate concave parts of the boundary. This property can also be obtained directly from the equation $\partial_t u = g(x)|\nabla u|\kappa$. The temporal change at a position is (close to) zero if (1) the stopping function g vanishes, i. e., an object boundary is reached, or (2) if the local curvature κ is zero, i. e., the contour segment under consideration approximates a straight line.

In the bottom row of Figure 3.9, the force term pushes the curve towards the object boundaries. Note, however, that the model fails to approximate the object boundaries

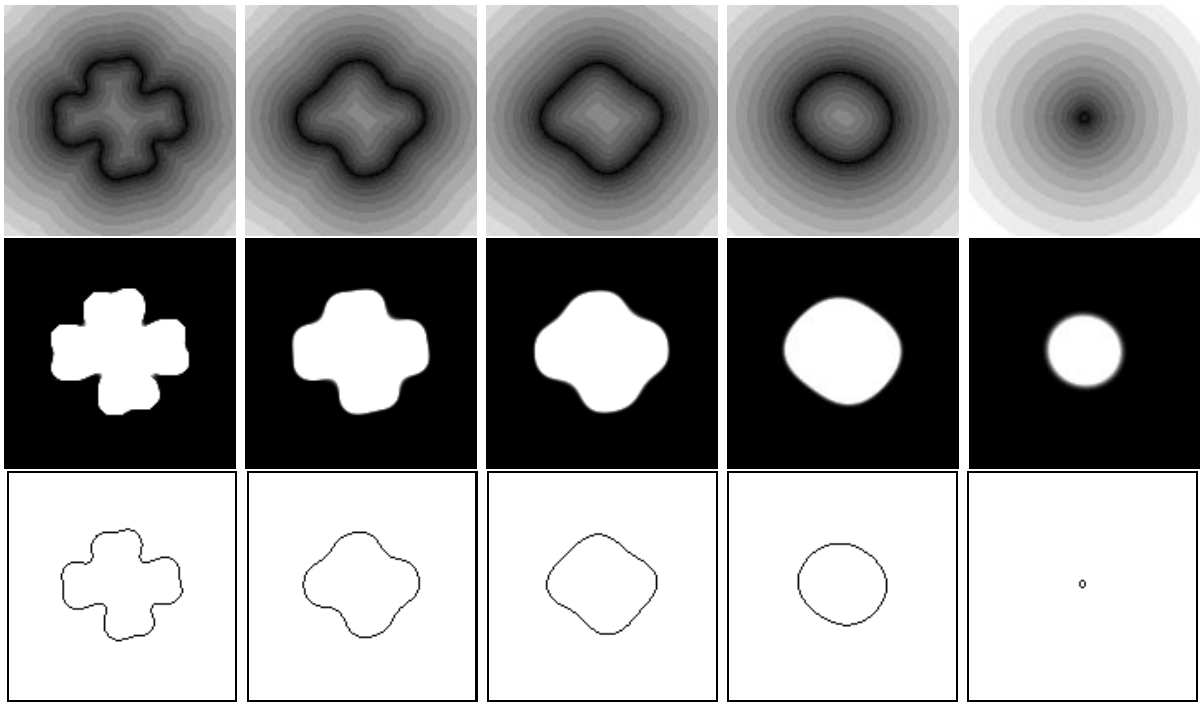

Figure 3.8: Implicit snake evolving under mean curvature motion. (a) TOP ROW: Signed distance representation, (b) MIDDLE ROW: Bi-level representation, (c) BOTTOM ROW: Extracted contour (zero level set) from signed distance representation.

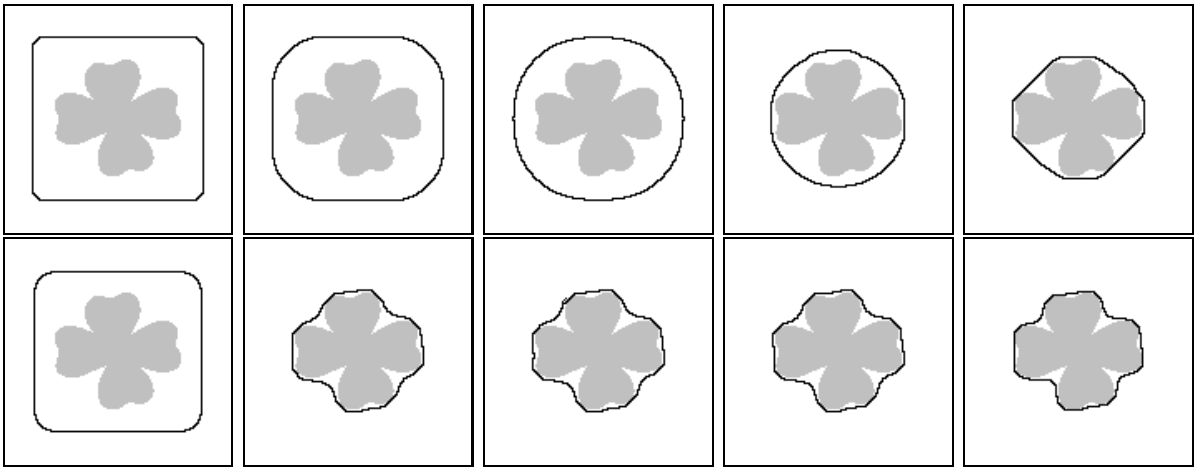

Figure 3.9: Evolution of an implicit geometric snake. (a) TOP ROW: $k = 0$, (b) BOTTOM ROW: $k = -0.25$.

near the center. At these positions large curvature values of the contour neutralize the constant force term.

Thus, we observe that the force term and its value are of crucial importance for the

geometric model. Next, we discuss the *geodesic* model that introduces an additional term, which reduces the role of the force term.

3.3.2.2 Implicit Geodesic Active Contour Model

An implicit snake model has also been developed based on energy minimization. This model is usually called the *implicit geodesic active contour model* and has been proposed simultaneously by Caselles, Kimmel and Sapiro [23] and Kichenassamy, Kumar, Olver, Tannenbaum and Yezzi [59]. In general, the shortest distance between two points on a surface is denoted as *geodesic*. By defining an image-induced distance metric that is minimal at object boundaries, a geodesic curve will be attracted to these positions and thus segregates the object from the background.

Based on the observation that minimizing the explicit snake energy defined in Equation 3.7 with $\alpha > 0$ and $\beta = 0$ decreases also the curvature [21], a modified contour energy is proposed, namely,

$$E_{\text{int}}(C(s,t)) = \oint_{C(s,t)} \left(\tfrac{\alpha}{2}|C_s(s,t)|^2 - \gamma|\nabla I(C(s,t))|^2\right) ds. \tag{3.22}$$

Under some additional assumptions (see [23]) the following curve evolution equation is derived

$$\frac{\partial C(s,t)}{\partial t} = g(x)\kappa(s,t)n(s,t) - (\nabla g(x)^T \cdot n(s,t))n(s,t). \tag{3.23}$$

Again, the balloon force kn introduced in Equation 3.11 can be easily integrated into the model, yielding

$$\frac{\partial C(s,t)}{\partial t} = g(x)(\kappa(s,t) + k)n(s,t) - (\nabla g(x)^T \cdot n(s,t))n(s,t). \tag{3.24}$$

By embedding Equation 3.24 as zero level set into u, we obtain the *implicit geodesic active contour* model, namely,

$$\frac{\partial u}{\partial t} = |\nabla u|\text{div}\left(g(x)\frac{\nabla u}{|\nabla u|}\right) + kg(x)|\nabla u|. \tag{3.25}$$

Using the fact that

$$\text{div}\left(g(x)\frac{\nabla u}{|\nabla u|}\right) = g(x)\text{div}\left(\frac{\nabla u}{|\nabla u|}\right) + \nabla g^T \cdot \frac{\nabla u}{|\nabla u|} \tag{3.26}$$

we can rewrite Equation 3.25 as

$$\frac{\partial u}{\partial t} = g(x)|\nabla u| \left(\text{div} \left(\frac{\nabla u}{|\nabla u|} \right) + k \right) + \nabla u^T \cdot \nabla g. \tag{3.27}$$

We observe that the implicit geodesic active contour model differs from its geometric counterpart only by the addition of the term $\nabla u^T \cdot \nabla g$.

Figure 3.10 displays the behavior of the implicit geodesic model. As in the geometric model, the shrinkage process starts at the corners of the initialization. During the first period the contour approximates a circle due to the mean curvature component. However, we observe that even without a force term the model is capable of approximating the concave parts of the object under consideration. This is due to the additional term $\nabla u^T \cdot \nabla g$ which increases the attraction of the contour near the object boundary.

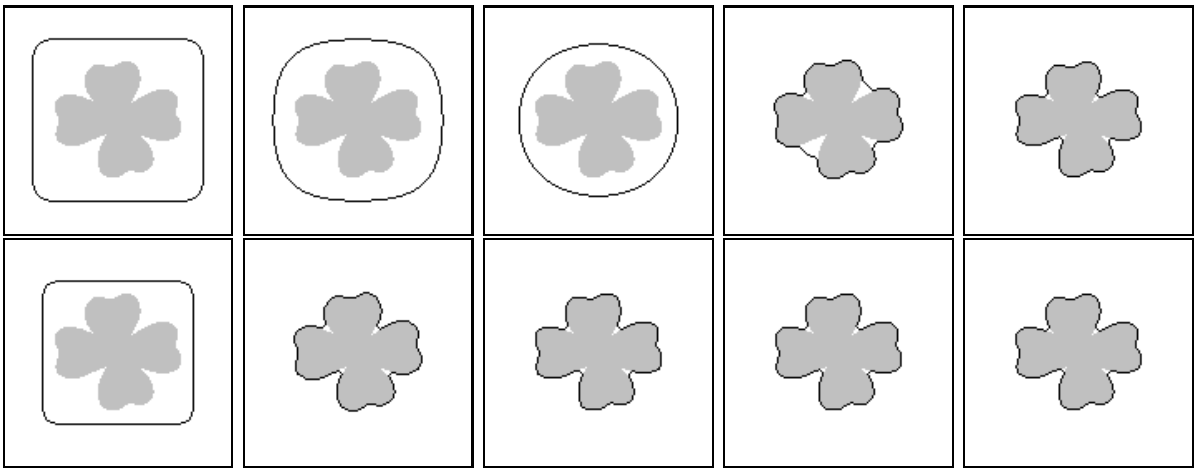

Figure 3.10: Evolution of an implicit geodesic snake. (a) TOP ROW: $k = 0$, (b) BOTTOM ROW: $k = -0.25$.

Finally, we repeat the segmentation task as depicted in Figure 3.5. This time an implicit geodesic snake is employed for object segmentation. Figure 3.11 displays the evolution of the implicit snake. In contrast to the explicit model, this model is able to split up and to detect the four objects simultaneously. The topological flexibility is due to the implicit representation of the contour. Since calculations are performed on the entire image domain rather than on explicit contour particles, splitting and merging is naturally included in the process.

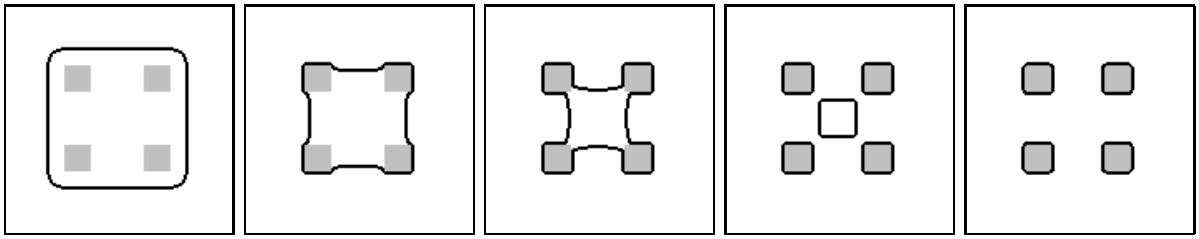

Figure 3.11: Object segmentation using an implicit geodesic snake.

3.3.2.3 Numerical Implementation

Let us rewrite Equation 3.27 to emphasize the different terms involved in the process of curve evolution:

$$\frac{\partial u}{\partial t} = \underbrace{g(x)|\nabla u|\kappa}_{\text{mean curvature motion}} + \overbrace{g(x)|\nabla u|k}^{\text{constant force}} + \underbrace{\nabla u^T \cdot \nabla g}_{\text{pure advection}} . \tag{3.28}$$

In the following we consider discretizations of these three terms separately [8,114]. To simplify matters we assume for a moment that the stopping function g is equal to one on the entire image domain ($g := 1$).

Again, we employ discrete times $t_n := n\tau$, where $n \in I\!N_0$ and τ denotes the size of the time step. Additionally, an image is divided by a uniform mesh of spacing $h = 1$ into grid nodes (i, j). Thus, using standard notation, u_{ij}^n denotes the approximation of $u(ih, jh, t_n)$.

We employ the following discretizations for space and time derivatives. The time derivatives are discretized by forward difference approximations:

$$\left.\frac{\partial u}{\partial t}\right|_{ij}^{n} \approx \frac{u_{ij}^{n+1} - u_{ij}^{n}}{\tau}. \tag{3.29}$$

For the spatial derivatives, we define the following approximations. First, similar to the approximations used for the explicit active contour models, we define central differences

as usual:

$$\left.\frac{\partial u}{\partial x}\right|_{ij}^{n} \approx \frac{u_{i+1j}^{n} - u_{i-1j}^{n}}{2h}, \quad \left.\frac{\partial u}{\partial y}\right|_{ij}^{n} \approx \frac{u_{ij+1}^{n} - u_{ij-1}^{n}}{2h}$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{ij}^{n} \approx \frac{u_{i+1j}^{n} - 2u_{ij}^{n} + u_{i-1j}^{n}}{h^2}, \quad \left.\frac{\partial^2 u}{\partial y^2}\right|_{ij}^{n} \approx \frac{u_{ij+1}^{n} - 2u_{ij}^{n} + u_{ij-1}^{n}}{h^2} \tag{3.30}$$

$$\left.\frac{\partial^2 u}{\partial x \partial y}\right|_{ij}^{n} \approx \frac{(u_{i+1j+1}^{n} - u_{i+1j-1}^{n}) - (u_{i-1j+1}^{n} - u_{i-1j-1}^{n})}{4h^2}.$$

Additionally, we define forward (D^+) and backward (D^-) approximations of the spatial derivatives:

$$\begin{aligned} D^{+x}u_{ij}^{n} &= \frac{u_{i+1j}^{n} - u_{ij}^{n}}{h}, \quad D^{-x}u_{ij}^{n} = \frac{u_{ij}^{n} - u_{i-1j}^{n}}{h} \\ D^{+y}u_{ij}^{n} &= \frac{u_{ij+1}^{n} - u_{ij}^{n}}{h}, \quad D^{-y}u_{ij}^{n} = \frac{u_{ij}^{n} - u_{ij-1}^{n}}{h}. \end{aligned} \tag{3.31}$$

First, we provide a discretization of the mean curvature term [8,114], thus considering the evolution equation $\partial_t u = |\nabla u|\kappa$. This equation is a parabolic partial differential equation, it can be approximated by using central differences in space. Therefore the curvature and the gradient are approximated by using the spatial derivatives as given in Equation 3.30.

Thus, we obtain the following discretization for the mean curvature motion equation

$$u_{ij}^{n+1} = u_{ij}^{n} + \tau\kappa_{ij}^{n}|\nabla^0 u_{ij}^{n}| \tag{3.32}$$

where ∇^0 denotes the approximation of the gradient by using central differences.

Second, we discretize the erosion/dilation equation $\partial_t u = k|\nabla u|$. This equation derived for constant speed evolution is a hyperbolic partial differential equation. Therefore, it is necessary to approximate the gradient by an upwind scheme, for instance [90, 114],

$$|\nabla u_{ij}^{n}| \approx \begin{cases} |\nabla^- u_{ij}^{n}| = & \big(\max(D^{-x}u_{ij}^{n},0)^2 + \min(D^{+x}u_{ij}^{n},0)^2 \\ & \quad + \max(D^{-y}u_{ij}^{n},0)^2 + \min(D^{+y}u_{ij}^{n},0)^2\big)^{1/2} \quad , \quad \text{if } k<0 \\ |\nabla^+ u_{ij}^{n}| = & \big(\min(D^{-x}u_{ij}^{n},0)^2 + \max(D^{+x}u_{ij}^{n},0)^2 \\ & \quad + \min(D^{-y}u_{ij}^{n},0)^2 + \max(D^{+y}u_{ij}^{n},0)^2\big)^{1/2} \quad , \quad \text{if } k>0 \end{cases} \tag{3.33}$$

Consequently, the numerical algorithm for constant speed evolution is given as

$$u_{ij}^{n+1} = u_{ij}^n + \tau \left(\min(k,0)|\nabla^- u_{ij}^n| + \max(k,0)|\nabla^+ u_{ij}^n| \right) \tag{3.34}$$

where the appropriate gradient approximation is chosen depending on the sign of the constant force k.

Finally, we consider the advection equation $\partial u/\partial t = \nabla u^T \cdot \nabla g$. Again, we employ a simple upwind scheme, using only information from the appropriate direction,

$$\begin{aligned} u_{ij}^{n+1} = u_{ij}^n + \tau \big(& \max(D^{0x} g_{ij}, 0) D_x^+ u_{ij}^n + \min(D^{0x} g_{ij}, 0) D_x^- u_{ij}^n \\ & + \max(D^{0y} g_{ij}, 0) D_y^+ u_{ij}^n + \min(D^{0y} g_{ij}, 0) D_y^- u_{ij}^n \big) . \end{aligned} \tag{3.35}$$

Putting things together, we can formulate a discretization of the implicit geometric and the implicit geodesic active contour model, respectively. To this end, we combine the terms given in Equations 3.32, 3.34, 3.35 and integrate the stopping function g. This results in

$$u_{ij}^{n+1} = u_{ij}^n + \tau \left(\, g_{ij} \kappa_{ij}^n |\nabla^0 u_{ij}^n| + \min(k g_{ij}, 0)|\nabla^- u_{ij}^n| + \max(k g_{ij}, 0)|\nabla^+ u_{ij}^n| \right) \tag{3.36}$$

for the geometric model. Similarly, the geodesic model is given by

$$\begin{aligned} u_{ij}^{n+1} = u_{ij}^n + \tau \quad \big[& \, g_{ij} \kappa_{ij}^n |\nabla^0 u_{ij}^n| + \min(k g_{ij}, 0)|\nabla^- u_{ij}^n| + \max(k g_{ij}, 0)|\nabla^+ u_{ij}^n| \\ & + \big(\max(D^{0x} g_{ij}, 0) D_x^+ u_{ij}^n + \min(D^{0x} g_{ij}, 0) D_x^- u_{ij}^n \\ & \quad + \max(D^{0y} g_{ij}, 0) D_y^+ u_{ij}^n + \min(D^{0y} g_{ij}, 0) D_y^- u_{ij}^n \big) \, \big] . \end{aligned} \tag{3.37}$$

We are now able to compare the numerical implementations of the explicit and implicit active contour models presented so far. For this purpose we highlight an experimental result that will be discussed in detail in Section 3.4.3. A simple segmentation task, the segregation of a square from a uniform background, was performed by both an explicit and an implicit geometric active contour model. With parameters set to $\tau = 0.25$, $k = -0.1$, both implementations required the same number of iterations. However, a single iteration of the implicit model is computationally more complex. Thus, the explicit snake was about 100 times faster than the implicit implementation.

The main reason for the computational inefficiency of the latter implementation is the implicit representation of the contour. Since computations are not limited to a certain

number of marker particles but are carried out on the entire image domain, the simple algorithm for implicit active contours is considerably slower than its explicit variant. Thus, in the following section we will discuss methods that improve the computational efficiency of implicit active contour models significantly.

3.4 Efficient Implementation of Implicit Active Contour Models

Reconsidering the numerical implementation for implicit snakes presented in the previous section, we observe two areas with room for improvement. First, although only the zero level set is relevant to the final segmentation result, the partial differential equation is evaluated on the entire image domain (*full-matrix method*). Therefore, it might be useful to limit the calculation area. However, special care must be taken to retain the splitting and merging capabilities of the implicit model.

Second, in all experiments conducted so far, we have employed a rather small time step $\tau \leq 0.25$. This is due to the fact that the above-mentioned numerical implementations are stable only for small time steps. Developing a numerical scheme that would remove this limitation should thus improve the computational efficiency.

We start with so-called narrow-band methods which exploit the first observation and develop a novel scheme for creating small calculation areas based on simple morphological operators. We then proceed with efficient numerical schemes and present a unified numerical approximation of the implicit geometric and the implicit geodesic active contour models. Note that our algorithms can be easily combined with well-known multiresolution methods [18].

3.4.1 Narrow-band Methods

The narrow-band method was introduced by Chopp [26]. The basic idea is to perform calculations only within a small tube around the evolving curve. Positions outside the tube do not change their values. Figure 3.12 displays an example of a narrow band. During the evolution of the curve the narrow band must also move or has to be rebuilt once the curve reaches the tube's boundary. A number of approaches to constructing and maintaining narrow-band structures have been proposed [3, 102].

The main advantage of our scheme is that it is applicable to both the signed distance and the bi-level representations. Furthermore, it relies on simple morphological operators

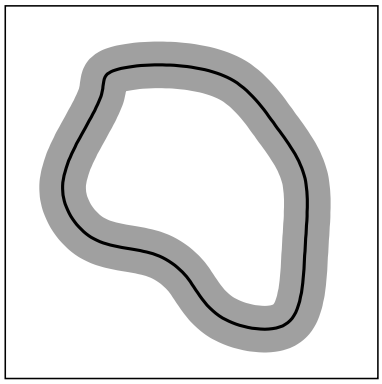

Figure 3.12: Narrow band constructed around an evolving contour.

which can be implemented very efficiently. The only condition that must be fulfilled is that pixels outside the contour have smaller values than those inside. Note that both the signed distance discretization and the bi-level representation meet this requirement by definition. In the following we show that for the purpose of narrow-band creation both representations can be reduced to binary images.

Consider an initial image u_0 constructed from an initial contour C_0 by a signed distance transform. Then, as described above, the contour is given as the zero level set and at each image position the distance to the contour is known. In addition, pixel values inside the contour are positive, while those outside are negative. It is straightforward to construct a narrow band of width $2M + 1$ ($2M$ if the zero level set is located on a sub-pixel position) from the initial image u_0 by only considering image positions x_j with $|u_0(x_j)| \leq M$. Then, to evolve the curve only positions within the tube are considered, while the values outside are frozen. After a number of iterations, the contour, i. e., the zero level set, might reach the boundary of the narrow band at some position. To capture the contour in further iterations, it is necessary to rebuild the tube. This is not as simple as in the initial case since the values outside the tube are frozen. Thus, considering image positions x_j with $|u_0(x_j)| \leq M$ does not necessarily move the boundary of the narrow band away from the curve.

Hence, we must develop a technique that constructs a new narrow band based on the contour's current location. Using the fact that values outside (inside) the contour are negative (positive), we can reduce the problem to a binary image:

$$b(x) = \begin{cases} 0, & \text{if } u(x) < 0 \\ 1, & \text{if } u(x) \geq 0 \end{cases} . \tag{3.38}$$

When the bi-level representation is used, the pixel values do not contain distance information. Therefore, the straightforward narrow-band creation as described above is not possible. However, we also can reduce the bi-level representation to a binary image:

$$b(x) = \begin{cases} 0, & \text{if } u(x) < \dfrac{a+b}{2} \\ 1, & \text{if } u(x) \geq \dfrac{a+b}{2} \end{cases} \tag{3.39}$$

where a and b ($a < b$) denote the two values of the bi-level initialization.

The construction of the new narrow band is then achieved by applying two basic morphological operators, binary dilation and binary erosion, to the image f. Let us recall the definitions of the following morphological operators:

$$\begin{aligned} \text{binary erosion:} \quad & (f(p) \ominus B) := \min\{f(q) | q \in B(p)\}, \\ \text{binary dilation:} \quad & (f(p) \oplus B) := \max\{f(q) | q \in B(p)\}, \\ \text{morphological gradient:} \quad & (f(p) \oplus B) - (f(p) \ominus B), \end{aligned} \tag{3.40}$$

where B is a so-called flat-structuring element usually centered over the image position p. Figure 3.13 displays the results of the operators applied to a binary image. While erosion shrinks the white blob on the background, dilation inflates its boundaries. Finally, the morphological gradient detects edges. Note that the radius of shrinkage, inflation, and edge detection depends on the size of the structuring element.

Figure 3.13: Narrow-band construction with morphological operators. FROM LEFT TO RIGHT: (a) Binary image from signed distance function, (b) erosion with square-shaped structuring element, (c) dilation with square-shaped structuring element, (d) morphological gradient.

We observe that to construct a narrow band of size $2M$, one simply has to apply the morphological gradient with a structuring element of radius M. The gradient value

determines whether an image pixel belongs to the narrow band or not: Image positions with a gradient value equal to zero are outside the tube while the other positions are located in the narrow band.

In the case of a binary image, the morphological operators defined above can be calculated by logical *and* operations (erosion) and logical *or* operations (dilation) in the area of the structuring element. When a square-shaped structuring element is used, both the erosion and the dilation operator are separable; thus, they can be implemented very efficiently.

To illustrate the process by an example, we consider in the following a one-dimensional image evolving under the equation $\partial_t u = -|\partial_x u|$ over time. This equation is the one-dimensional equivalent to the constant force term ($k < 0$) used in both implicit active contour models. The numerical implementation is given by

$$u_i^{n+1} = u_i^n - \tau \sqrt{\left(\max\left(\frac{u_i^n - u_{i-1}^n}{h}, 0\right)\right)^2 + \min\left(\left(\frac{u_{i+1}^n - u_i^n}{h}, 0\right)\right)^2}. \quad (3.41)$$

Consider as the initial image $u(x, 0)$ the following pixel data obtained from a signed distance function. We use a narrow band of width 5 and print values inside the narrow band in bold face.

$u(x, 0.0)$:	-5	-4	-3	**-2**	**-1**	**0**	**1**	**2**	3	4	5

Iterating u under Equation 3.41 considering only the image position within the narrow band yields the following:

$u(x, 0.5)$:	-5.0	-4.0	-3.0	**-2.5**	**-1.5**	**-0.5**	**0.5**	**1.5**	3.0	4.0	5.0
$u(x, 1.0)$:	-5.0	-4.0	-3.0	**-2.8**	**-2.0**	**-1.0**	**0.0**	**1.0**	3.0	4.0	5.0
$u(x, 1.5)$:	-5.0	-4.0	-3.0	**-2.9**	**-2.4**	**-1.5**	**-0.5**	**0.5**	3.0	4.0	5.0
$u(x, 2.0)$:	-5.0	-4.0	-3.0	**-2.9**	**-2.6**	**-1.9**	**-1.0**	**0.0**	3.0	4.0	5.0

We observe that the zero level set reaches the tube's boundary at $T = 2.0$. Thus, a reconstruction of the narrow band is necessary. According to Equation 3.38, the following

binary image is derived from u:

$b(x, 2.0)$:	0	0	0	0	0	0	0	1	1	1	1

By applying the morphological gradient with a structuring element of radius $M = 2$, the new narrow band (of size $2M$) is obtained. Note that we assume that the border values have been replicated.

$b'(x, 2.0)$:	0	0	0	0	0	1	1	1	1	0	0

$u(x, 2.0)$:	-5.0	-4.0	-3.0	-2.9	-2.6	**-1.9**	**-1.0**	**0.0**	**3.0**	4.0	5.0

We observe that the signed distance function is not maintained on u during the evolution. This is due to the fact that the values outside the narrow band are frozen.[2] To keep u as a signed distance function inside the narrow band, a technique as the one described in Section 3.3.2 might be used.

Returning to our example, we apply the numerical implementation of the one-dimensional variant of Equation 3.17, $\partial_t \phi = \text{sign}(\phi_0)(1 - |\partial_x \phi|)$, to the one-dimensional image u (see [125] for details). This yields after a few iterations

$u'(x, 2.0)$:	-5.0	-4.0	-3.0	-2.9	-2.6	**-2.0**	**-1.0**	**0.0**	**1.0**	4.0	5.0

As previously mentioned, the narrow band must be reconstructed if the contour is close to the tube's boundary. Thus, indicators are required to trigger the reconstruction process. For this purpose we place so-called *guards* at the boundaries of the narrow band [3, 114]. If the approaching contour is detected at one of these guard positions, the narrow band will be reconstructed. In the case of the signed distance representation this detection is straightforward since at each position within the narrow band the distance to the contour is known.

For the bi-level representation the situation is different. Let $a < b$ denote the two levels used to create the initial image u_0 and let $g = (g_i, g_j)$ denote a guard position. Then, a transition from a to b at such a guard position indicates that the contour has passed by the narrow band's boundary:

[2] Even if narrow-band techniques are not employed, u might deviate from the signed distance function during evolution [8, 102, 124, 125].

$$\left(u_g^{n+1} - \frac{a+b}{2}\right)\left(u_g^n - \frac{a+b}{2}\right) < 0. \tag{3.42}$$

However, since the narrow band should be reconstructed when the contour is close to the boundary but has not yet passed it, it is useful to introduce a distance offset into the previous equation. For instance, denoting the distance offset by d, the reconstruction process is triggered if

$$\left(u_g^{n+1} - \frac{a+b}{2} - d\right)\left(u_g^n - \frac{a+b}{2} - d\right) < 0, \tag{3.43}$$

or

$$\left(u_g^{n+1} - \frac{a+b}{2} + d\right)\left(u_g^n - \frac{a+b}{2} + d\right) < 0. \tag{3.44}$$

The distance offset d moves the transition point closer to a or b depending on its sign. Therefore the reconstruction process is triggered before the contour reaches the boundary of the narrow band.

3.4.2 Semi-implicit Discretization

Obviously, the performance of implicit snake algorithms depends on the time step. However, the numerical implementations presented so far restrict the time step to small values. As we will see further on, this restriction results from the chosen temporal discretization. Let us recall the discretization of the mean curvature motion equation $\partial_t u = \operatorname{div}(\nabla u/|\nabla u|)$, namely,

$$u_{ij}^{n+1} = u_{ij}^n + \tau\kappa_{ij}^n|\nabla^0 u_{ij}^n|. \tag{3.45}$$

Note that only values from the previous time level n are used to compute values on the time level $n+1$. This updating scheme is often referred to as the *explicit* updating scheme.[3]

To overcome the limitation caused by the explicit updating scheme, we can employ different temporal discretizations, for instance, an *implicit* updating scheme as

$$u_{ij}^{n+1} = u_{ij}^n + \tau\kappa_{ij}^{n+1}|\nabla^0 u_{ij}^{n+1}|. \tag{3.46}$$

Here, in contrast to the explicit scheme, values on time level $n+1$ depend on other

[3]We should note the different meanings of the word *explicit* in the terms *explicit snake* and *explicit updating scheme*. While in the first case the word *explicit* is used to define a specific contour representation, it describes a numerical approximation in the latter case.

values at the same time level. Note that the solution u_{ij}^{n+1} cannot be directly determined from this scheme. Instead, a system of equations has to be solved. However, the implicit updating scheme is well behaved for any choice of τ, thus, it is referred to as being unconditionally stable.

In the following we develop a numerical algorithm for both the geometric and the geodesic active contour models based on a *semi-implicit* updating scheme [68, 140]. It provides the same stability properties as an implicit scheme. In addition, it is much easier to implement. Note that the narrow-band technique developed above can be easily combined with this implementation.

We should note that our approach is not the first semi-implicit scheme proposed for implicit snakes. The first proposal for geodesic active contours was made by Goldenberg, Kimmel, Rivlin, and Rudzsky [37, 38]. Their implementation is based on the observation that the magnitude of the gradient of an image initialized by the signed distance function is equal to 1.0 at each position, $|\nabla u|_{ij} = 1$ for all i, j. Thus, Equation 3.25 (without the force term) reduces to the following equation

$$\frac{\partial u}{\partial t} = \operatorname{div}\left(g \nabla u\right). \tag{3.47}$$

which is closely related to the nonlinear diffusion equation [24].

An efficient and reliable scheme for this equation, the so-called additive operator splitting (AOS) scheme, was introduced by Weickert et al. [142] (see also [71]). However, as observed in the previous section, the image u deviates quickly from the signed distance function under image evolution. Therefore, the scheme by Goldenberg et al. requires a reinitialization of u in each iteration. As a remedy Weickert [139] proposed an AOS-based scheme for geodesic snakes with harmonic averaging. In the following we extend Weickert's scheme to include both the geometric and the geodesic active contour models and also cover the force term that is of crucial importance in the geometric model.

Let us consider the following equation which unifies the geometric (Equation 3.21) and the geodesic active contour models (Equation 3.25) by introducing two additional functions a and b:

$$\frac{\partial u}{\partial t} = a(x)|\nabla u| \operatorname{div}\left(\frac{b(x)}{|\nabla u|}\nabla u\right) + |\nabla u| k g(x). \tag{3.48}$$

Setting $a := g$, $b := 1$ yields the geometric model, while $a := 1$, $b := g$ results in the geodesic model. For the sake of clarity, we assume a constant force $k = 0$ in the following and discuss the complete model later on.

Interpreting the term $\frac{b(x)}{|\nabla u|}$ as "diffusivity", we can employ techniques similar to those described in [142] in the context of nonlinear diffusion filtering. As usual, an image is divided by a uniform mesh of spacing $h = 1$ into grid nodes x_i. Again, u_{ij}^n denotes the approximation of $u(ih, jh, \tau n)$. Then, the term $\operatorname{div}((b(x)/|\nabla u|)\nabla u)$ can be discretized as follows:

$$\begin{aligned}
\operatorname{div}\left(\frac{b(x)}{|\nabla u|}\nabla u\right) &\approx \partial_x\left(\left(\frac{b}{|\nabla u|}\right)_{ij}\frac{u_{i+\frac{1}{2}j}-u_{i-\frac{1}{2}j}}{h}\right)+\partial_y\left(\left(\frac{b}{|\nabla u|}\right)_{ij}\frac{u_{ij+\frac{1}{2}}-u_{ij-\frac{1}{2}}}{h}\right)\\
&\approx \left(\frac{b}{|\nabla u|}\right)_{i+\frac{1}{2}j}\frac{u_{i+1j}-u_{ij}}{h^2}-\left(\frac{b}{|\nabla u|}\right)_{i-\frac{1}{2}j}\frac{u_{ij}-u_{i-1j}}{h^2}+\\
&\quad \left(\frac{b}{|\nabla u|}\right)_{ij+\frac{1}{2}}\frac{u_{ij+1}-u_{ij}}{h^2}-\left(\frac{b}{|\nabla u|}\right)_{ij-\frac{1}{2}}\frac{u_{ij}-u_{ij-1}}{h^2}\\
&\approx \frac{\left(\frac{b}{|\nabla u|}\right)_{i+1j}+\left(\frac{b}{|\nabla u|}\right)_{ij}}{2}\frac{u_{i+1j}-u_{ij}}{h^2}-\frac{\left(\frac{b}{|\nabla u|}\right)_{ij}+\left(\frac{b}{|\nabla u|}\right)_{i-1j}}{2}\frac{u_{ij}-u_{i-1j}}{h^2}+\\
&\quad \frac{\left(\frac{b}{|\nabla u|}\right)_{ij+1}+\left(\frac{b}{|\nabla u|}\right)_{ij}}{2}\frac{u_{ij+1}-u_{ij}}{h^2}-\frac{\left(\frac{b}{|\nabla u|}\right)_{ij}+\left(\frac{b}{|\nabla u|}\right)_{ij-1}}{2}\frac{u_{ij}-u_{ij-1}}{h^2}.
\end{aligned} \tag{3.49}$$

Note that the values for $(b/|\nabla u|)_{i+1/2j}$ and $(b/|\nabla u|)_{ij+1/2}$ are obtained by linear interpolation.

To simplify the notation, a discrete image is represented in the following as a vector $f \in \mathbb{R}^N$, whose components $f_i, i \in \{1, \ldots, N\}$ contain the pixel values. Consequently, pixel i corresponds to a grid node x_i. Then, u_i^n denotes the approximation of $u(x_i, t_n)$.

Thus, using the above discretization, Equation 3.48 with $k = 0$ can be written as

$$u_i^{n+1} = u_i^n + \tau\left(a_i|\nabla u|_i^n \sum_{j\in\mathcal{N}(i)} \frac{\left(\frac{b}{|\nabla u|}\right)_i^n + \left(\frac{b}{|\nabla u|}\right)_j^n}{2}\frac{u_j^{n+1}-u_i^{n+1}}{h^2}\right), \tag{3.50}$$

where $\mathcal{N}(i)$ denotes the 4-neighborhood of the pixel at position x_i. Note that this discretization is referred to as *semi-implicit* since values from both the time levels n and $n+1$ are required in the calculations.

However, in order to compute Equation 3.50, we must ensure that the gradient mag-

nitude is larger than zero in the entire image domain. When the signed distance function and frequent reinitializations are used this is not a problem because $|\nabla u| \approx 1$ can be guaranteed. However, when the bi-level representation is used, the situation is different. First, in flat image areas the gradient vanishes. We can solve this problem by evaluating only image positions with $|\nabla u| \neq 0$. In addition, even if the gradient magnitude is larger than zero at a certain image position, it might vanish in the 4-neighborhood of the pixel. Thus, a straightforward finite difference implementation such as the one above would give rise to numerical problems.

These can be avoided by employing a finite difference scheme with *harmonic averaging* [139]. Thus, replacing the term $\left(\left(\frac{b}{|\nabla u|}\right)_i^n + \left(\frac{b}{|\nabla u|}\right)_j^n\right)/2$ in Equation 3.50 by its harmonic counterpart $2/\left(\left(\frac{|\nabla u|}{b}\right)_i^n + \left(\frac{|\nabla u|}{b}\right)_j^n\right)$ yields the following semi-implicit discretization:

$$u_i^{n+1} = u_i^n + \tau \left(a_i |\nabla u|_i^n \sum_{j \in \mathcal{N}(i)} \frac{2}{\left(\frac{|\nabla u|}{b}\right)_i^n + \left(\frac{|\nabla u|}{b}\right)_j^n} \frac{u_j^{n+1} - u_i^{n+1}}{h^2} \right). \tag{3.51}$$

Note that by evaluating only image positions with $|\nabla u|_i \neq 0$, the denominator in this scheme cannot vanish. In matrix-vector notation we can rewrite this as follows:

$$u^{n+1} = u^n + \tau \left(\sum_{l \in \{x,y\}} A_l(u^n) \right) u^{n+1}. \tag{3.52}$$

The matrix $A_l(u^n) = (\hat{a}_{ijl}(u^n))$ is given by

$$\hat{a}_{ijl}(u^n) := \begin{cases} a_i |\nabla u|_i^n \frac{2}{\left(\frac{|\nabla u|}{b}\right)_i^n + \left(\frac{|\nabla u|}{b}\right)_j^n}, & j \in \mathcal{N}_l(i) \\ -a_i |\nabla u|_i^n \sum\limits_{m \in \mathcal{N}_l(i)} \frac{2}{\left(\frac{|\nabla u|}{b}\right)_i^n + \left(\frac{|\nabla u|}{b}\right)_m^n}, & j = i \\ 0, & \text{else,} \end{cases} \tag{3.53}$$

where $\mathcal{N}_l(i)$ represents the two neighboring pixels with respect to direction $l \in \{x, y\}$. Due to the semi-implicit character of the scheme, the solution u^{n+1},

$$u^{n+1} = \left(I - \tau \sum_{l \in \{x,y\}} A_l(u^n) \right)^{-1} u^n, \tag{3.54}$$

cannot be directly computed. Instead, it is necessary to solve a linear system of equations. Reformulating Equation 3.54 within the AOS scheme [142] yields

$$u^{n+1} = \frac{1}{2} \sum_{l \in \{x,y\}} \left(I - 2\tau A_l(u^n)\right)^{-1} u^n. \tag{3.55}$$

Note that the term $(I - 2\tau A_l(u^n))$ results in a strictly diagonally dominant tridiagonal linear system which can be solved very efficiently [90]. As previously said, this semi-implicit scheme is unconditionally stable, so, we can apply arbitrarily large time steps.

However, we have so far neglected the constant force term $|\nabla u|kg$ (cf. Equation 3.48). This term stems from the hyperbolic equation $\partial_t u = \pm|\nabla u|$. Consequently, in a numerical implementation the gradient has to be approximated by an upwind scheme (cf. Equation 3.33). Integrating the constant force term into Equation 3.55 is straightforward and yields for $k < 0$:

$$u^{n+1} = \frac{1}{2} \sum_{l \in \{x,y\}} \left(I - 2\tau A_l(u^n)\right)^{-1} \left(u^n + \tau|\nabla^- u|^n kg\right). \tag{3.56}$$

Since the dilation/erosion equation approximated on a grid with $h = 1$ is stable only for $\tau \leq 0.5$ [98], the constant force term limits the applicable time step. Consequently, Equation 3.56 is stable only for $|\tau kg| \leq 0.5$. However, since g is bounded by one, k is usually a small fraction of 1.0, and very large time steps ($\tau > 5.0$) degrade accuracy significantly [139, 142], this constraint is not severe.

3.4.3 Experimental Results

In this section we investigate the accuracy and the computational efficiency of the various active contour algorithms developed above. We discuss some implementation details and provide the results of a number of implicit snake implementations, namely, the full-matrix method, the narrow-band method, the AOS-based method, and the combined AOS/narrow-band method. In addition, for the purpose of comparison, we develop a simple algorithm for explicit snakes.

All algorithms are applied to three test images (cf. Figure 3.14) using a standard PC (AMD Athlon 1.4 GHz running Linux). The goal in each case is to approximate the object boundaries by the contour. We use the *single-square* image to check whether the contour is able to approximate the corners well. In the *shamrock* image the object boundary has varying curvatures, thus, the evolving contour must meet different demands. Finally, the *4-square* image is used to demonstrate the splitting capabilities of the contour under consideration.

Since the objects can be easily segregated from the background, the true object bound-

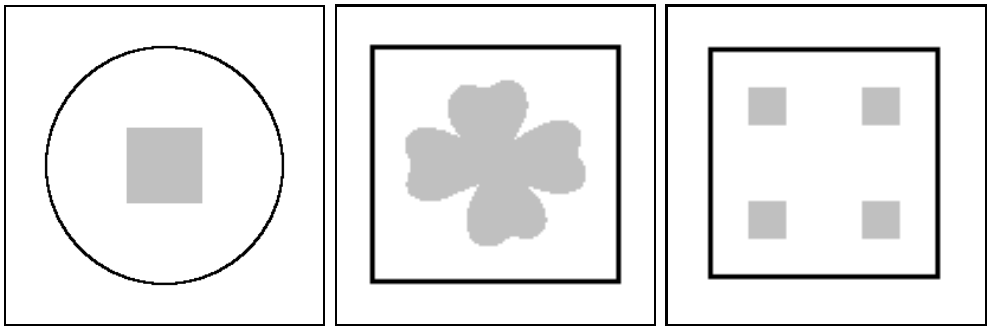

Figure 3.14: Test objects (light gray) and initial contours (black). (a) LEFT: Single-square image, 256 × 256 pixels, (b) MIDDLE: Shamrock image, 128 × 128 pixels, (c) RIGHT: 4-square image, 128 × 128 pixels.

aries are well-known and can be compared to the final contour. To this end, a simple distance measure was developed: Given a final contour and a reference contour (e.g., the true object boundary), the distance to the nearest pixel on the reference contour is calculated for each pixel on the final contour. Averaging these values yields the average distance (in pixels) between the two contours and thus the error.

3.4.3.1 Explicit Snake Implementation

Based on Section 3.3.1.3, we can formulate a simple algorithm to evolve an explicit snake. Recall that in this approach the contour is modeled explicitly by a collection of marker particles. As noted above, numerical problems may occur when two neighboring marker particles come together. Thus, we employ a periodic redistribution procedure that rearranges the marker particles on the contour. The different steps of the algorithm are as follows:

(1) Distribute equally spaced marker particles $p_i^0 = (x_i^0, y_i^0)$ with distance h on the initial contour C_0.

(2) If a neighboring marker particle pair p_i, p_j, $i \neq j$ exists with $|p_i - p_j| < \Delta s$, then interpolate the contour linearly from the current set of marker particles and redistribute marker particles with distance h.

(3) Calculate relocated contour C^{n+1} from C^n using Equation 3.15.

(4) Iterate steps 2 and 3 until stability has been attained.

To indicate a stable steady state, the following criterion is used:

$$\sum_i \sqrt{\left(x_i^{n+1} - x_i^n\right)^2 + \left(y_i^{n+1} - y_i^n\right)^2} < \epsilon. \tag{3.57}$$

We applied the preceding algorithm to the three test images *single-square*, *shamrock*, and *4-square*. In each case we ran the algorithm without the force term ($k = 0$) and with the force term ($k = -0.1$). However, we did not consider the *4-square* image in conjunction with $k = -0.1$. This constellation requires topological adaptivity which is not included in the explicit approach. The results are summarized in Table 3.1. Note that the stopping criterion was not employed. Instead, to facilitate comparisons with the implicit models that will be discussed later, we employed a fixed number of iterations and a standard parameter set. Therefore, as indicated in the distance column of the table, the final contour might differ from the true object boundary.

explicit active contour model ($\Delta s = 2.0, h = 5.0$)					
image	τ	k	iterations	CPU time	distance
single square	0.25	0.0	24800	2.15 s	1.89
single square	0.25	-0.1	3200	0.34 s	1.45
shamrock	0.25	0.0	13800	3.05 s	3.05
shamrock	0.25	-0.1	11400	1.73 s	1.73
4 squares	0.25	0.0	13200	0.79 s	5.51
4 squares	0.25	-0.1	-	-	-

Table 3.1: Performance of the explicit active contour model.

The *shamrock* image illustrates that the force term is of crucial importance for the model. A force term with $k < 0$ significantly improves the final placement of the contour. The computation times required by the explicit snake implementation will serve as a reference for the implicit snake implementations discussed in the following.

3.4.3.2 Implicit Snake Implementation: Full-matrix Method

We start our discussion on implicit snakes with their simplest implementation, namely, the full-matrix method. This method calculates the underlying PDE on the entire image domain. Consequently, we can expect poor computational efficiency.

The purpose of this section is twofold. We present computation times for different models and parameter sets. In addition, we compare the accuracy and efficiency of the signed distance initialization and the bi-level initialization as presented in Section 3.3.2.

Let us first turn to the different initialization methods. When using signed distance

initialization two situations must be considered: (1) the *construction* of the initial image u_0 and (2) *keeping* u as a signed distance function during image evolution. To obtain the initial image u_0 we applied a fast distance transform algorithm based on one-dimensional erosion [134]. In order to maintain u as a signed distance function, the magnitude of the gradient of u must be monitored. Remember that an image initialized to a signed distance function has a gradient magnitude of 1.0 at each image position. If the average gradient magnitude differs from this by a certain amount, a reinitialization process will be triggered. In our implementation we employed an auxiliary PDE for reinitialization as proposed by Peng et al. [102]. Usually a few iterations suffice to reinitialize u to a signed distance function.

The bi-level initialization is simpler. Here, the construction of the initial image u_0 is straightforward. Furthermore, a reinitialization procedure is not employed. However, numerical problems may occur in areas of flat gradients when using bi-level initialization since the denominator of the curvature term $\operatorname{div}(u/|\nabla u|)$ vanishes. This case is treated numerically by not imposing changes at these image positions.

Figure 3.15 displays the evolution of an implicit geodesic active contour on the two different initializations. We observe that the signed distance representation leads to a slightly quicker evolution than its bi-level counterpart. The final placement of the contour is identical.

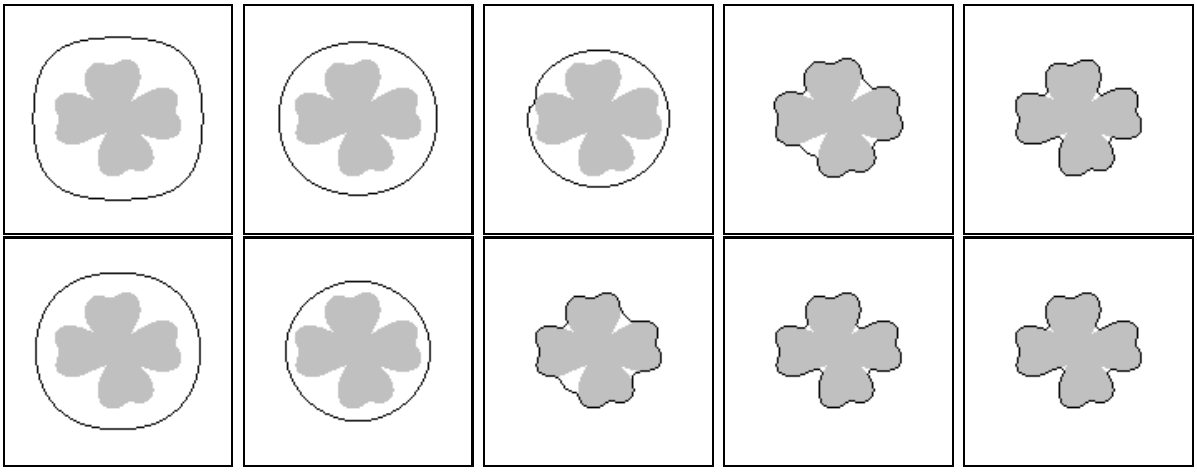

Figure 3.15: Evolution of an implicit geodesic snake under signed distance and bi-level initialization. FROM LEFT TO RIGHT: Iterations 1000, 2000, 3000, 4000, 6000. (a) TOP ROW: Bi-level initialization, (b) BOTTOM ROW: Signed distance initialization.

After these visual inspections we shall investigate these differences more precisely. For this purpose we extract two contours at every 100 iterations, one from the signed distance

image and the other from the bi-level image. These two contours are compared using the above-mentioned contour distance measure. This is done for each pair of extracted contours yielding minimum, average, and maximum distance for the entire curve evolution process. Table 3.2 summarizes the results. Each row contains the minimum, average, and maximum distance of a contour evolving on a bi-level initialization to a contour evolving on a signed distance function. We note that despite large differences at certain points in time, the average distance is still rather small. Especially when a force term is applied the results are nearly identical.

			contour distance (pixels)		
model	image	k	minimum	average	maximum
geometric	single square	0.0	0.29	1.60	4.00
geometric	single square	-0.1	0.14	0.83	1.08
geometric	shamrock	0.0	0.87	1.35	3.15
geometric	shamrock	-0.1	0.88	1.50	2.41
geometric	4 squares	0.0	0.26	0.95	2.20
geometric	4 squares	-0.1	0.56	1.37	4.05
geodesic	single square	0.0	0.02	2.81	11.77
geodesic	single square	-0.1	0.02	0.67	1.45
geodesic	shamrock	0.0	0.01	1.36	8.67
geodesic	shamrock	-0.1	0.00	0.27	2.15
geodesic	4 squares	0.0	0.05	1.65	5.97
geodesic	4 squares	-0.1	0.05	0.47	1.81

Table 3.2: Contour differences caused by initialization methods.

Table 3.3 which displays the results calculated on the test images with different models, parameters, and initializations underlines this observation. In all cases the chosen initialization method hardly influenced the final result. The signed distance function leads to a slightly higher accuracy.

Let us discuss Table 3.3 in detail. First of all, the number of iterations required in each case resulted from a simple stopping criterion. To indicate that a contour has reached a stable state we used the following stopping criterion: Every time a certain period Δt_k has elapsed the average value of the evolving image u will be calculated. E. g., when setting $\Delta t_k = 50$ and $\tau = 0.25$, the average value is computed every 200 iterations. The process stops if two consecutive measurements differ by less than an accuracy parameter α. In all experiments conducted on bi-level initializations the parameters for the stopping criterion were set to $\Delta t_k = 50$ and $\alpha = 0.01$. In order to facilitate comparisons we used the same number of iterations in the cases with signed distance initialization.

geodesic model ($\tau = 0.25$)						
image	initialization	k	iterations	CPU time	reinit.	distance
single square	bi-level	0.0	19200	288.72	-	0
single square	signed dist.	0.0	19200	555.75	9118(1)	0
single square	bi-level	-0.1	3200	46.52	-	0
single square	signed dist.	-0.1	3200	92.84	1535(1)	0
shamrock	bi-level	0.0	13800	43.88	-	0.04
shamrock	signed dist.	0.0	13800	99.97	3064(2.8)	0.04
shamrock	bi-level	-0.1	11600	47.01	-	0.04
shamrock	signed dist.	-0.1	11600	72.96	1533(3)	0.04
4 squares	bi-level	0.0	5800	9.78	-	4.32
4 squares	signed dist.	0.0	5800	51.27	2505(2.4)	4.39
4 squares	bi-level	-0.1	2600	22.70	-	0
4 squares	signed dist.	-0.1	2600	21.13	599(3.7)	0

geometric model ($\tau = 0.25$)						
image	initialization	k	iterations	CPU time	reinit.	distance
single square	bi-level	0.0	24800	461.77	-	1.00
single square	signed dist.	0.0	24800	665.02	4292(1)	0.77
single square	bi-level	-0.1	3200	37.25	-	1.00
single square	signed dist.	-0.1	3200	69.22	773(1)	1.00
shamrock	bi-level	0.0	11400	28.73	-	3.14
shamrock	signed dist.	0.0	11400	56.51	2537(1.4)	2.30
shamrock	bi-level	-0.1	15200	40.54	-	1.60
shamrock	signed dist.	-0.1	15200	65.66	1057(2.3)	1.63
4 squares	bi-level	0.0	13200	17.73	-	4.80
4 squares	signed dist.	0.0	13200	61.52	2147(1.5)	4.72
4 squares	bi-level	-0.1	5800	19.99	-	1.00
4 squares	signed dist.	-0.1	5800	28.60	710(2.4)	0.26

Table 3.3: Performance of the implicit active contour model (full-matrix method).

For the geodesic model we observe that independent of the force term, the objects in the *single-square* and the *shamrock* image could be approximated very well. However, the topological change required to approximate the four objects in the *4-square* image could be made only possible with an additional force term ($k = -0.1$).

The findings for the geometric model differ on some points. Here, the force term is necessary to guide the contour into the narrow parts of the shamrock and to enforce the topological change required for the *4-square* image. The accuracy was slightly worse than that of the geodesic model.

With regard to computational efficiency the results are rather poor compared to those of the explicit snake implementation discussed in the previous section. For instance, averaging the timing results for the single square ($k = 0$ and $k = -0.1$), the shamrock ($k = 0$ and $k = -0.1$), and the 4 squares ($k = 0$) reveals that the explicit snake is about 75 times faster than the full-matrix implementation of an implicit geometric snake (bi-level representation).

Note that maintaining the evolving image as a signed distance function requires additional computations. The column labeled "reinit." in Table 3.3 shows the number of reinitializations carried out during the image evolution. In brackets the number of iterations required to reinitialize the image to a signed distance function is quoted. As stated above, the method proposed by Peng et al. [102] needed only a few iterations to converge.

For typical image processing tasks such as object segmentation the accuracy obtained using either initialization method is sufficient. Thus, in the following we prefer the bi-level initialization since it is computationally more efficient. We should note, however, that for applications in different areas the signed distance representation is a prerequisite [125]. For instance, in the field of medical image analysis, the segmentation of the cortex using two coupled surfaces requires that the distance between the surfaces is known at any time [40].

3.4.3.3 Implicit Snake Implementation: Narrow-band Method

Let us now examine the narrow-band method as described in Section 3.4.1. We start with some implementation details, and proceed to discuss results on efficiency and accuracy.

In our approach the narrow band is obtained from a binary morphological gradient operator. Thus, image positions belonging to the narrow band are marked by one, positions outside by zero. To exploit the narrow-band information one can iterate through all image positions and check whether the corresponding narrow-band flag is set. Of course, the narrow-band information can be stored more efficiently [3, 114]. One possibility is to use run-length encoding. Consider a row of the image obtained from the morphological gradient. This row might contain runs of zeros alternating with runs of ones. Thus, to represent the narrow-band pixels within this row, we store only the start and end positions of the runs containing ones. By doing so for all rows of the gradient image, an efficient narrow band structure can be constructed.

Obviously, the computational effort required for an iteration in the curve evolution process depends on the width of the narrow band. Additionally, as the curve moves through the image, rebuilds of the narrow band become necessary. Thus, while a small

narrow band reduces the computational load within each iteration, it still may result in frequent rebuilds of the narrow band structure itself.

Table 3.4 displays the results calculated on our test images. As expected, the computation times drop as the width of the narrow band is reduced. Note that although the number of rebuilds increases these do not affect the efficiency severely.

geodesic model ($\tau = 0.25$)							
image	k	width	iterations	time (s)	time/iter. (ms)	rebuilds	distance
square	0.0	16	19200	109.49	5.70	3	0
square	0.0	4	19200	42.47	2.21	22	0
square	-0.1	16	3200	16.45	5.14	3	0
square	-0.1	4	3200	6.70	2.09	20	0
shamrock	0.0	16	13800	23.37	1.69	2	0.04
shamrock	0.0	4	13800	7.32	0.53	17	0.07
shamrock	-0.1	16	11600	22.63	1.95	2	0.04
shamrock	-0.0	4	11600	7.06	0.61	13	0.05
4 squares	0.0	16	5800	9.04	1.56	1	4.32
4 squares	0.0	4	5800	2.97	0.51	10	4.32
4 squares	-0.1	16	2600	6.03	2.32	3	0
4 squares	-0.1	4	2600	1.47	0.57	16	0

geometric model ($\tau = 0.25$)							
image	k	width	iterations	time (s)	time/iter. (ms)	rebuilds	distance
square	0.0	16	24800	116.16	4.68	3	1
square	0.0	4	24800	44.75	1.80	19	1
square	-0.1	16	3200	13.44	4.20	3	1
square	-0.1	4	3200	6.16	1.93	18	1
shamrock	0.0	16	11400	28.99	2.54	1	3.15
shamrock	0.0	4	11400	5.05	0.44	7	3.15
shamrock	-0.1	16	15200	27.56	1.81	2	1.62
shamrock	-0.1	4	15200	6.45	0.42	8	1.66
4 squares	0.0	16	13200	16.68	1.26	0	4.80
4 squares	0.0	4	13200	5.06	0.38	7	4.80
4 squares	-0.1	16	5800	9.55	1.65	2	1
4 squares	-0.1	4	5800	2.68	0.46	15	1

Table 3.4: Performance of the implicit active contour model (narrow-band method).

For greater precision, we evaluated our narrow-band construction algorithm separately. Note that binary one-dimensional dilations and erosions can be implemented very efficiently. We applied the narrow-band construction algorithm to a collection of images.

Note that the image content is irrelevant, only size matters. The results on different image sizes were as follows (the computation times in milliseconds are given in brackets): 128×128 (1.7 ms), 256×256 (6.8 ms), 512×512 (39.7 ms), 352×288 (11.4 ms), 704×576 (71.0 ms). The latter two formats occur frequently in video processing. We observe that in comparison to a single iteration of the contour algorithm (see "time/iter." column in Table 3.4), the computational cost of the narrow-band construction is greater up to a factor of four. Thus, infrequent rebuilds of the narrow band are negligible for the overall performance.

As we can see decreasing the narrow band width does not only reduces computation times but also decreases accuracy. To investigate the influence of the narrow-band implementation on accuracy we compared its results to those of the full-matrix method. For this purpose the contours of both implementations were extracted at every 100 iterations and compared by using our edge distance measure. Table 3.5 summarizes some results for different narrow-band sizes. As the narrow-band width decreases, the contours differ from those obtained by the full-matrix implementation. Note, however, that the final contour placements are almost identical to those of the full-matrix method.

				contour distance (pixels)		
model	image	k	nb width	minimum	average	maximum
geodesic	single square	0.0	16	0.00	0.05	0.32
geodesic	single square	0.0	8	0.00	0.41	2.53
geodesic	single square	0.0	4	0.00	1.63	8.87
geodesic	single square	-0.1	16	0.00	0.25	0.94
geodesic	single square	-0.1	8	0.00	0.81	2.60
geodesic	single square	-0.1	4	0.00	1.89	5.80
geometric	single square	0.0	16	0.00	0.06	0.30
geometric	single square	0.0	8	0.00	0.35	1.51
geometric	single square	0.0	4	0.00	1.37	4.84
geometric	single square	-0.1	16	0.00	0.28	0.99
geometric	single square	-0.1	8	0.00	0.86	2.48
geometric	single square	-0.1	4	0.00	2.00	5.33

Table 3.5: Contour differences caused by the narrow-band method.

We close this section by comparing the computation times of the full-matrix implementation to those gained by our narrow-band method (cf. Tables 3.3 and 3.4). Averaging over all test cases, the narrow-band method (width = 4) is about seven times faster than its full-matrix counterpart. Note, however, that this ratio depends on the size of the image and the width of the narrow band.

3.4.3.4 Implicit Snake Implementation: the AOS Scheme

In this section we cover the semi-implicit implementation of the geometric and the geodesic models as presented in Section 3.4.2. Remember that this implementation differs from the explicit full-matrix algorithm in two aspects. First, of course, the AOS-based implementation allows larger time steps. Second, the discretization of the (modified) curvature term $\operatorname{div}(\nabla u/|\nabla u|)$ (resp. $\operatorname{div}(g\nabla u/|\nabla u|)$ in the geodesic model) is different. To discriminate between those differences we discuss them separately.

Recall that in the geometric model the curvature term is given as

$$\operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) = \frac{u_{xx}u_y^2 - 2u_x u_y u_{xy} + u_{yy}u_x^2}{(u_x^2+u_y^2)^{3/2}}. \tag{3.58}$$

In the standard implementation as used in the above-mentioned full-matrix method this term is approximated straightforwardly by central differences. In the AOS-based implementation, however, a discretization based on harmonic averaging is used. Thus, the curvature term is approximated at image position i as follows:

$$\operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right)\Big|_i \approx \sum_{j\in\mathcal{N}(i)} \frac{2}{|\nabla u|_i + |\nabla u|_j}\frac{u_j - u_i}{h^2}. \tag{3.59}$$

The geodesic model employs a modified curvature term, namely,

$$\operatorname{div}\left(g\frac{\nabla u}{|\nabla u|}\right) = g\cdot\operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) + \nabla g^T\cdot\frac{\nabla u}{|\nabla u|} \tag{3.60}$$

Here, the standard implementation discretizes the first term by central differences and the second term by an upwind scheme. The AOS-based implementation is again based on harmonic averaging and reads

$$\operatorname{div}\left(g\frac{\nabla u}{|\nabla u|}\right)\Big|_i \approx \sum_{j\in\mathcal{N}(i)} \frac{2}{\left(\frac{|\nabla u|}{g}\right)_i + \left(\frac{|\nabla u|}{g}\right)_j}\frac{u_j - u_i}{h^2}. \tag{3.61}$$

To investigate the influence of the different spatial discretizations on the curve evolution we compared the standard implementation to an implementation based on harmonic averaging. In both cases we employed an explicit updating scheme. The results obtained by our usual distance measure are shown in Table 3.6. The final results obtained by the implementation based on harmonic averaging of the geodesic model are slightly worse

than those of the standard implementation. The average distance in all test cases is about one pixel larger than for the standard implementation. With respect to our segmentation task this is an acceptable result. In the case of the geometric model, however, the importance of the force term is further increased. Let us consider the results for the *4-square* image as an example. Without the force term the distance of the final contour is about seven pixels away from the original object boundary. Adding the force term significantly improves the result.

			contour distance			
model	image	k	minimum	average	maximum	distance
geodesic	single square	0.0	0.04	0.85	6.70	1
geodesic	single square	-0.1	0.10	0.53	1.46	1
geodesic	shamrock	0.0	0.02	0.95	4.26	1
geodesic	shamrock	-0.1	0.10	0.82	1.05	0.77
geodesic	4 squares	0.0	0.02	1.05	3.63	4.53
geodesic	4 squares	-0.1	0.08	0.91	1.29	1
geometric	single square	0.0	0.04	1.37	3.78	4.64
geometric	single square	-0.1	0.10	0.53	1.77	1.96
geometric	shamrock	0.0	0.02	1.21	2.04	4.41
geometric	shamrock	-0.1	0.07	0.85	1.02	2.44
geometric	4 squares	0.0	0.02	2.18	3.18	6.65
geometric	4 squares	-0.1	0.08	1.35	4.01	1.88

Table 3.6: Contour differences caused by spatial discretization using harmonic averaging.

Let us now turn to the semi-implicit implementation. To assess the influence of the time step, we compare the AOS-based implementation with different time steps to the explicit updating scheme with harmonic averaging as used above. Table 3.7 displays the corresponding results. We observe that the contours differ significantly during the evolution. In addition, we note that a larger time step results in a larger deviation. To inspect these differences visually let us consider Figure 3.16 which shows the curve evolution of an AOS-based geodesic snake in comparison to an explicit implementation. We observe that the evolution process with a time step of 0.125 reaches the object faster than the same process with a time step of 5.0. Of course, the former requires a much larger number of iterations. Thus, increasing the time step by a factor of 20 does not reduce necessarily the number of iterations at the same amount.

Consequently, in order to assess the performance of the AOS-based implementation, it is not sufficient to simply cut down the number of iterations and increase the size of the time step. Instead, we applied the same stopping criterion here as for the full-matrix

				contour distance		
model	image	k	τ	minimum	average	maximum
geodesic	single square	0.0	1.0	0.00	0.53	2.81
geodesic	single square	0.0	2.5	0.00	1.37	6.53
geodesic	single square	0.0	5.0	0.00	2.87	11.91
geodesic	single square	-0.1	1.0	0.00	1.54	3.39
geodesic	single square	-0.1	2.5	0.00	4.14	8.44
geodesic	single square	-0.1	5.0	0.00	8.69	16.25
geometric	single square	0.0	1.0	0.04	0.51	1.23
geometric	single square	0.0	2.5	0.10	1.20	2.99
geometric	single square	0.0	5.0	0.17	2.39	5.85
geometric	single square	-0.1	1.0	0.03	1.53	3.03
geometric	single square	-0.1	2.5	0.09	4.09	7.93
geometric	single square	-0.1	5.0	0.21	8.60	15.74

Table 3.7: Contour differences caused by different time steps.

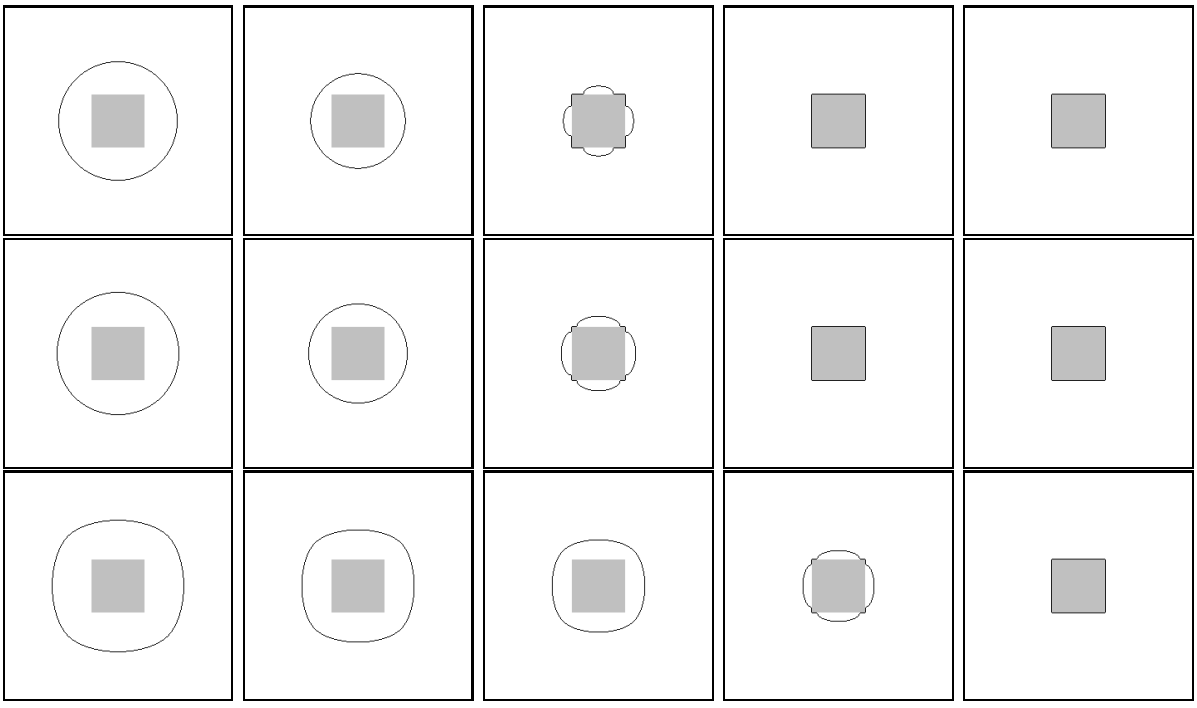

Figure 3.16: Evolution of an AOS-based geodesic snake. FROM LEFT TO RIGHT: $T = 250$, $T = 375$, $T = 500$, $T = 625$, $T = 750$. (a) TOP ROW: Explicit updating scheme with harmonic averaging ($\tau = 0.125$), (b) MIDDLE ROW: AOS scheme ($\tau = 1.0$), (c) BOTTOM ROW: AOS scheme ($\tau = 5.0$).

implementation. Table 3.8 displays the results achieved by the AOS-based implementation. Comparing these results to those calculated by the full-matrix implementation (cf. Table 3.3) we observe that the accuracy is slightly worse but still acceptable for our segmentation task. Concerning computational efficiency, the AOS-based implementation is significantly faster. Averaged over all test cases the AOS-based implementation reduces the computation time by a factor of 12.

geodesic model ($\tau = 5.0$)				
image	k	iterations	time (s)	distance
square	0.0	1290	39.63	1
square	-0.1	190	6.13	1
shamrock	0.0	340	2.32	1.01
shamrock	-0.1	130	0.89	1.02
4 squares	0.0	2190	15.33	0.49
4 squares	-0.1	140	0.95	1

geometric model ($\tau = 5.0$)				
image	k	iterations	time (s)	distance
square	0.0	1270	38.17	3.16
square	-0.1	200	6.41	1.01
shamrock	0.0	820	5.40	3.89
shamrock	-0.1	680	4.54	2.29
4 squares	0.0	620	4.13	5.67
4 squares	-0.1	550	3.67	1.01

Table 3.8: Performance of the implicit active contour model (AOS-based method).

We conclude this section by highlighting an experimental result in Table 3.8. We observe that the AOS-based implementation of the implicit geodesic model is able to detect the four objects in the *4-square* image even without the force term (average distance = 0.49) (recall that a geodesic snake implemented with an explicit updating scheme required a force term for the same task). This indicates that the AOS-based implementation is more diffusive with respect to the evolving contour. While this is desirable in the context of the *4-square* image, it may cause problems with regard to closing small gaps in the object boundaries.

As a remedy one might employ an ad hoc modification of the implicit geodesic snake by introducing an additional parameter $\beta \in [0, 1]$:

$$\frac{\partial u}{\partial t} = g(x)|\nabla u| \left(\operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right) + k \right) + \beta \nabla u^T \cdot \nabla g. \tag{3.62}$$

For $\beta = 1$ we obtain the standard geodesic snake, setting $\beta = 0$ implements the geometric active contour model. Thus, the additional parameter allows a smooth transition between both models and controls the influence of the advection term.

3.4.3.5 Implicit Snake Implementation: Narrow-band Method and AOS Scheme

As stated earlier the narrow-band method and the AOS-based implementation both significantly reduce computation times. Thus, it might prove useful to combine them. For this purpose the above-mentioned narrow-band construction algorithm is employed, and the AOS-based calculations are only performed within the narrow band. However, since the contour moves significantly faster under AOS-based iterations, we should expect that accuracy demand a larger narrow band than in the case of the explicit updating scheme.

First of all, we must refine our narrow-band structure due to the specific properties of the AOS scheme. Recall that in the AOS-based implementation an additive splitting is employed. First, new values are calculated separately along the different coordinate axes. These values are then averaged. In order to support this splitting with an appropriate data structure, we modify the storage of narrow-band information. Until now we have considered the rows of an image and stored the start and ending positions of pixel runs lying inside the band. This can be easily extended by performing the same operations along image columns [38]. Consequently, for each image dimension a narrow-band structure is created.

Let us first examine the influence of the narrow-band structure on the contour evolution process. To this end, we compare the results of the AOS-based implementation to those of AOS combined with the narrow-band method. Contours are extracted from both evolution processes at regular intervals and compared by the usual distance measure. Table 3.9 displays the results for different widths of the narrow band. As expected, accuracy decreases as the width of the narrow band is reduced.

Concerning computational efficiency, Table 3.10 summarizes the results obtained with the combined AOS/narrow-band method for the different test images. In comparison to the pure AOS-based implementation the computation times are cut by half on the average. Note that the efficiency could be further improved by reducing the width of the narrow band. However, as discussed above the AOS-based implementation demands a certain minimum width.

				contour distance		
model	image	k	nb width	minimum	average	maximum
geodesic	square	0.0	25	0.00	2.06	12.39
geodesic	square	0.0	20	0.00	2.98	12.39
geodesic	square	0.0	10	0.00	10.35	23.91
geodesic	square	-0.1	25	0.00	0.39	1.85
geodesic	square	-0.1	20	0.00	0.44	2.00
geodesic	square	-0.1	10	0.00	2.12	5.33
geometric	square	0.0	25	0.00	1.61	5.31
geometric	square	0.0	20	0.00	2.84	8.59
geometric	square	0.0	10	2.07	13.72	23.83
geometric	square	-0.1	25	0.00	0.36	1.53
geometric	square	-0.1	20	0.00	0.48	2.00
geometric	square	-0.1	10	0.00	2.20	5.16

Table 3.9: Contour differences caused by narrow-band method (AOS).

geodesic model ($\tau = 5.0$)						
image	k	width	iterations	time (s)	rebuilds	distance
square	0.0	20	1290	14.97	3	1
square	-0.1	20	190	2.12	3	1
shamrock	0.0	20	340	2.02	2	1
shamrock	-0.1	20	130	0.73	2	1
4 squares	0.0	20	2190	12.12	2	0.49
4 squares	-0.1	20	140	0.77	2	1

geometric model ($\tau = 5.0$)						
image	k	width	iterations	time (s)	rebuilds	distance
square	0.0	20	1270	13.98	3	4.33
square	-0.1	20	200	2.05	3	1.01
shamrock	0.0	20	820	5.23	1	3.73
shamrock	-0.1	20	680	3.86	1	1.93
4 squares	0.0	20	620	3.47	1	5.64
4 squares	-0.1	20	550	2.88	2	1

Table 3.10: Performance of the implicit active contour model (combined AOS/narrow-band method).

We are now in a position to summarize our experimental results obtained with different implicit snake implementations. The proposed narrow-band method and the AOS

scheme each decrease the accuracy of the segmentation slightly, however, the computational efficiency improves significantly. Averaging over all test cases, we observe that (a) the narrow-band method reduces computation time by a factor of 7, (b) the AOS scheme by a factor of 12, and (c) the combination of both by a factor of 20. In comparison to our simple explicit snake implementation, the fastest implicit implementation is about six times slower while retaining topological adaptivity.

3.5 Integration of Motion Cues

In the previous sections we discussed active contour models for grouping local information. However, for illustrative purposes we restricted ourselves to still image segmentation based on spatial edges. In the following we replace edge-based information by motion cues calculated by the algorithms developed in Chapter 2. This combination of motion cues and implicit active contour models constitutes our approach to motion segmentation.

For the sake of clarity we first discuss the combined approach under the assumption of a static camera [67, 69]. We then proceed to discuss motion segmentation under the assumption of a moving camera.

3.5.1 Motion Segmentation under the Assumption of a Static Camera

Let us recall how the integration of image features into the snake-based segmentation process works. In all models a function is employed that stops the curve evolution process near certain image features. For instance, as described above, a simple edge-based stopping function is given as

$$g(x) = \frac{1}{1 + |\nabla f_\sigma(x)|^2}. \tag{3.63}$$

By redefining the stopping function g appropriately we can integrate motion cues into the segmentation process. Let $r(x) \in [0, 1]$ denote a measure that indicates the reliability of a motion estimate $v(x) = (v_x, v_y)$ at an image position x.

Then, replacing ∇f_σ in Equation 3.63 by s_σ where s is given as

$$s(x) = \begin{cases} w_{\max}, & (r(x) \geq 1 - \epsilon) \text{ and } (|v(x)| \geq T_v) \\ 0 \quad , & \text{else} \end{cases} \tag{3.64}$$

stops the curve evolution ($g \approx 0$) upon reaching positions that coincide with "motion

pixels". Here, $w_{\max}$ denotes the maximum gray value of an image. An image position is considered to be a motion pixel if the reliability measure is close to 1.0 and the magnitude of the corresponding motion vector is larger than a certain minimum value T_v (typically T_v is set to 0.1). To compensate for camera jitter it is useful to consider only motion vector magnitudes that exceed this threshold.

Figure 3.17 illustrates our segmentation approach on a sample frame of the *hall-and-monitor* sequence. Displayed in the top row are the original frame and the motion pixel image calculated by the structure tensor with non-maximum suppression. In the bottom row of the figure the evolution of a geodesic snake is superimposed on the frame. As described above, a stopping function based on the motion pixel image interrupts the curve evolution process.

Figure 3.17: Motion segmentation under static camera. (a) TOP LEFT: Frame from the *hall-and-monitor sequence*, (b) TOP RIGHT: Motion pixel image calculated by the structure tensor with non-maximum suppression. (c) BOTTOM ROW: Evolution of a geodesic active contour superimposed on the hall-and-monitor frame.

3.5.2 Motion Segmentation under the Assumption of a Moving Camera

When acquiring a video sequence by means of a moving camera and estimating the corresponding motion vector field, motion vectors result from both the camera motion itself as well as by the movement of objects. Consequently, in order to determine the camera parameters reliably from the motion vector field, one has to discard the vectors caused by object motion. Conversely, we can identify motion vectors belonging to moving objects by eliminating those vectors which fit to the camera motion model.

In Chapter 2 we developed a robust algorithm for camera motion estimation. Let us recall its basic steps: First, reliable motion estimates are calculated from the video sequence. Of course, both camera motion and object motion contribute to the 2D vector field. Second, camera parameter estimates are calculated by a robust statistical estimator that compensates for outliers caused by object motion.

Henceforth, we are interested not in the dominant camera motion but in the motion vectors that deviate from the camera motion model. Fortunately, the least trimmed squares (LTS) estimator used to estimate the camera parameters enables us to detect and analyze outliers. Remember that the LTS estimator is given as

$$\min_{\hat{\theta}} \sum_{i=1}^{h} \left(r^2\right)_{i:n} \tag{3.65}$$

where n is the number of cases, $\hat{\theta} := \{\hat{\theta}_0, \ldots, \hat{\theta}_7\}$ denote the parameters to estimate, and $(r^2)_{1:n} \leq (r^2)_{2:n} \leq \ldots \leq (r^2)_{n:n}$ are the ordered squared residuals. By setting $h = n/2$ the "best half" of the test cases is obtained and thus the best robust properties are achieved. By considering only the "good data" (resp. the best half) we can define a robust error scale estimate $\hat{\sigma}$, namely,

$$\hat{\sigma} \quad = \quad \sqrt{\frac{1}{h} \sum_{i=1}^{h} \left(r^2\right)_{i:n}} \tag{3.66}$$

Then, considering a test case i, a large value $|r_i/\hat{\sigma}|$ indicates that the case i is identified as an outlier, and thus belongs to a moving object.

To clarify the process, we consider the following example. We assume that the camera parameters $\hat{\theta}_0, \ldots, \hat{\theta}_7$ and the error scale estimate have already been calculated by the least trimmed squares estimator. Then, for each pixel $p = (x, y)$ we proceed as follows.

The apparent position of p in the succeeding frame is given as its 2D motion vector $v(p) = (v_x, v_y)$ as $p' = (x', y') = (x + v_x, y + v_y)$. Additionally, the new position of p, $\hat{p}' = (\hat{x}', \hat{y}')$, can be predicted from the estimated camera parameters as

$$\hat{x}' = \frac{\hat{\theta}_0 x + \hat{\theta}_1 y + \hat{\theta}_2}{\hat{\theta}_6 x + \hat{\theta}_7 y + 1}, \qquad \hat{y}' = \frac{\hat{\theta}_3 x + \hat{\theta}_4 y + \hat{\theta}_5}{\hat{\theta}_6 x + \hat{\theta}_7 y + 1} \tag{3.67}$$

Thus, we can compare the residuals $r_x = x' - \hat{x}'$ and $r_y = y' - \hat{y}'$ to the error scale estimate $\hat{\sigma}$ and identify whether pixel p is an outlier or whether it fits into the camera motion model.

By redefining the stopping function g employed in the active contour models, information on outliers can be included in the segmentation process. Again, the term ∇f_σ is replaced by a function s_σ. This time, however, s is defined as

$$s(x) = \begin{cases} w_{\max}, & (r(x) \geq 1 - \epsilon) \text{ and } (|r_x(x)/\hat{\sigma}| \geq T_c \text{ or } |r_y(x)/\hat{\sigma}| \geq T_c) \\ 0 \quad , & \text{else} \end{cases} . \tag{3.68}$$

Recall that $r(x)$ indicates the reliability of a motion estimate and $r_x(x)$, $r_y(x)$ denote the residuals at position x as described above. Thus, an image position is classified as part of a moving object if the certainty of the corresponding motion estimate is close to 1.0 and if the residuals calculated from the vector components and the camera motion model are large compared to the error scale estimate. Note that in practice the threshold T_c that decides whether an estimate is an outlier or not has to be determined empirically.

In the following we illustrate our segmentation approach under camera motion by means of an example. Figure 3.18 displays a frame from the *coastguard* sequence and illustrates the different steps. Note that in this part of the sequence the camera follows the moving boat, panning to the right. The corresponding motion vector field is shown in the top row on the right. White areas indicate insufficient certainty for reliable motion calculation. The camera parameters are calculated from the motion field using the robust camera motion estimator. The resulting global camera motion field is shown in the middle row on the left, clearly indicating the pan. Using the camera parameters and the motion field, we are now able to detect outliers according to Equation 3.68. Figure 3.18 depicts the result in the middle row on the right. As said previously, the outlier detection is the basis for the active contour algorithm. The curve evolution process superimposed on the original frame is shown in the bottom row of the figure.

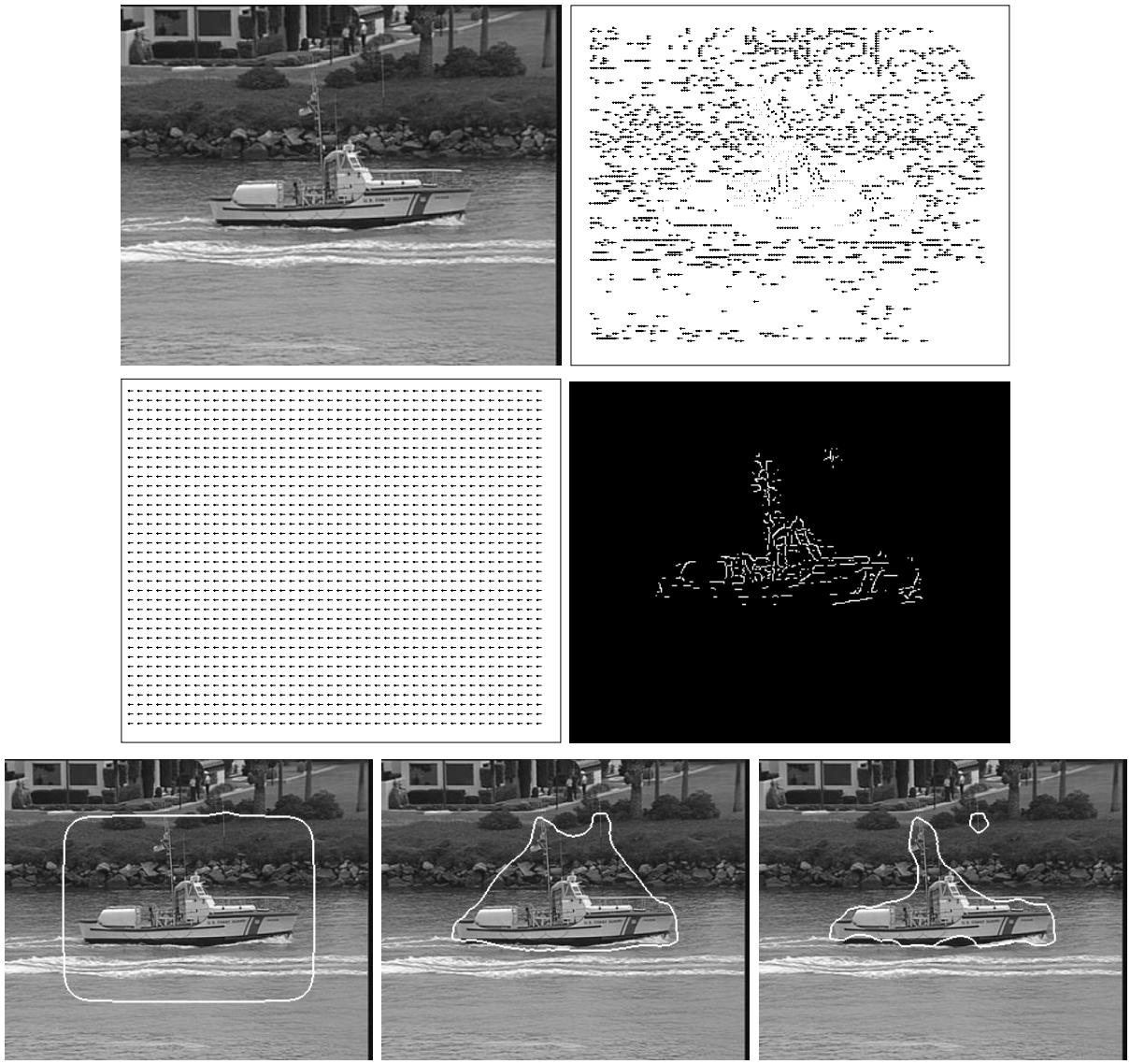

Figure 3.18: Motion segmentation under camera motion. (a) TOP LEFT: Frame from the *coastguard* sequence, (b) TOP RIGHT: Motion vector field calculated by feature-based motion estimation, (c) MIDDLE LEFT: Estimated global camera motion, (d) MIDDLE RIGHT: Outlier detection, (e) BOTTOM ROW: Evolution of a geometric active contour superimposed on the coastguard frame.

3.5.3 Experimental Results

We conclude this chapter by presenting some results obtained with our motion-based segmentation approaches. For this purpose, we applied the algorithms to four well-known video sequences:

- The *Hamburg taxi* sequence was acquired by a static camera and contains four

moving objects: three cars and a pedestrian.

- The *hall and monitor* sequence, also recorded by a static camera, contains two persons walking.

- The *coastguard* video sequence was recorded by a panning camera and shows two boats.

- The *Stefan* sequence displays a tennis player during a match. It was recorded by a camera that followed the player.

Figure 3.19 depicts the segmentation results for the *Hamburg taxi* sequence. We observe that the contour detected all four objects simultaneously. The segmentation of the taxi and the car on the left is accurate. Even the body of the pedestrian could be segregated from the background. However, due to the low contrast and a tree in the foreground, the detection of the van on the right is erroneous.

Segmentation results for the *hall-and-monitor* sequence are shown in Figure 3.20. Again, the algorithm detected the moving objects and in most cases provided a fairly accurate segmentation. However, the first two frames in the second row in the figure illustrate the limitations of our approach. Here, the left person stops and turns towards the wall. Since some parts of the person do not move during this action, they are ignored in the segmentation process.

Let us now turn to the sequences recorded by a moving camera. Figure 3.21 displays the results for the *coastguard* sequence. The achieved segmentations are promising, however, we observe some problems. Since the camera moves, we can only take feature points into account where full motion can be calculated. Remember this calculation is only possible at image positions that are surrounded by a texture changing in two dimensions. Consequently, some positions belonging to the moving objects are not selected as feature points. Consider, for instance, the bottom line of the smaller boat. Here, due to the lack of a two-dimensional structure, points are excluded from the feature correspondence process. Since the resulting gap is rather large, the active contour does not approximate the real boundary of the object correctly. To overcome this limitation, one might include region information, e. g., obtained from a spatial segmentation step.

Next, we discuss the segmentation results for the *Stefan* sequence displayed in Figure 3.22. In order to suppress small motion areas resulting from movements in the audience, we erased small regions from the image that were only a few pixels in size. Note that in the last frame of the bottom row the ball hits the net and therefore a second moving

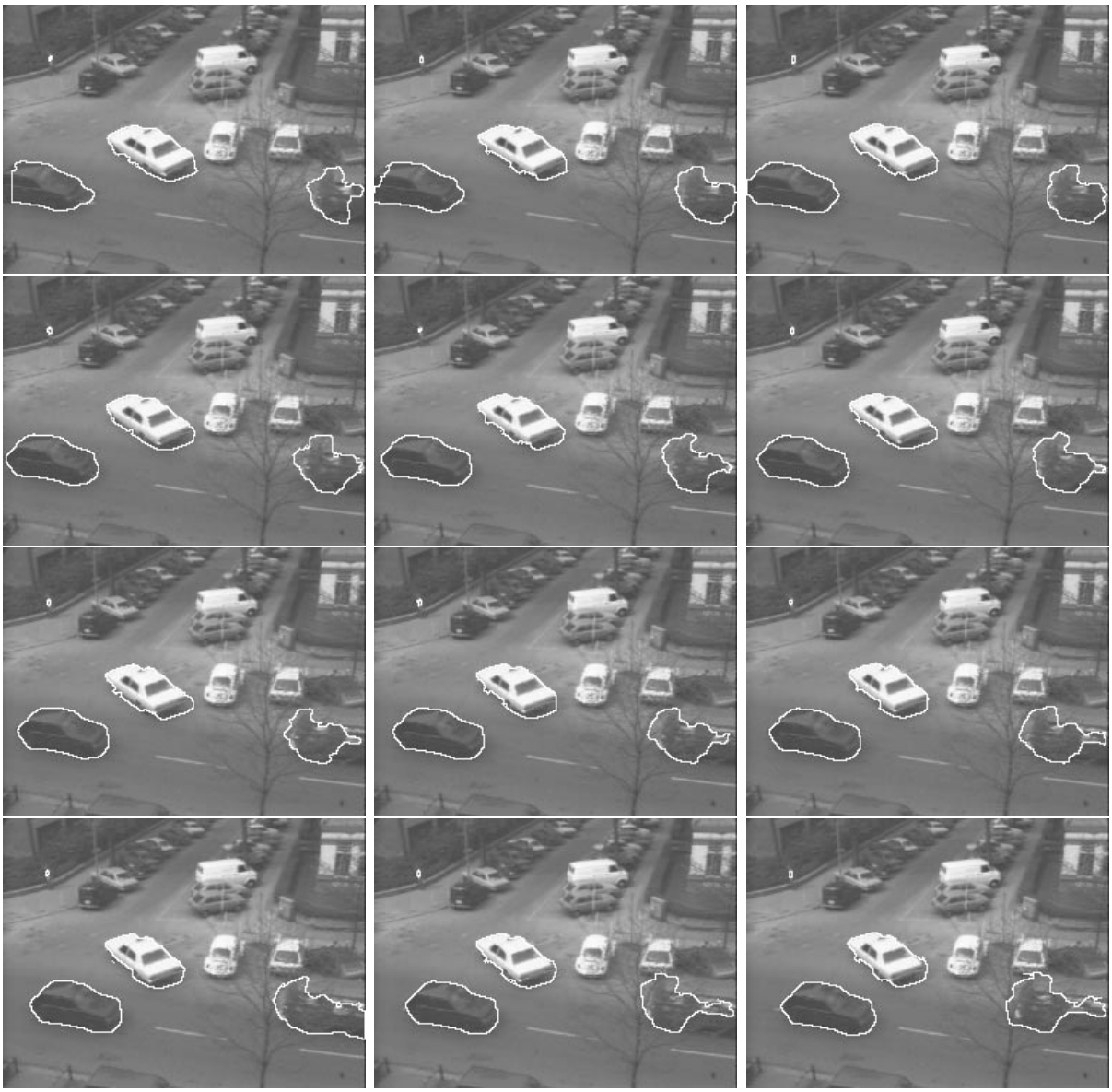

Figure 3.19: Motion segmentation of the *Hamburg taxi* sequence (Tensor-based motion estimation, implicit geodesic active contour).

object is segmented. We observe the same problem as for the *hall-and-monitor* sequence: Parts of the player that do not move are ignored in the segmentation process (e. g., see the first frame in the second row). One possibility to overcome this limitation would be the inclusion of an "object memory" into the segmentation algorithm [82].

In order to obtain quantitative results for our segmentation algorithm, we compared our results to those provided within the European COST (Coopération Européenne dans la recherche scientifique et technique) 211 framework [4]. Within this activity, a seg-

Figure 3.20: Motion segmentation of the *hall-and-monitor* sequence (Tensor-based motion estimation with non-maximum suppression, implicit geometric active contour).

mentation algorithm, the so-called COST Analysis Model (COST-AM), was developed [30, 35, 82, 83]. The COST-AM includes components for color segmentation, global motion analysis, local motion analysis, and change detection. Intermediate results from the different modules are fused by a rule processor. In addition, temporally accumulated

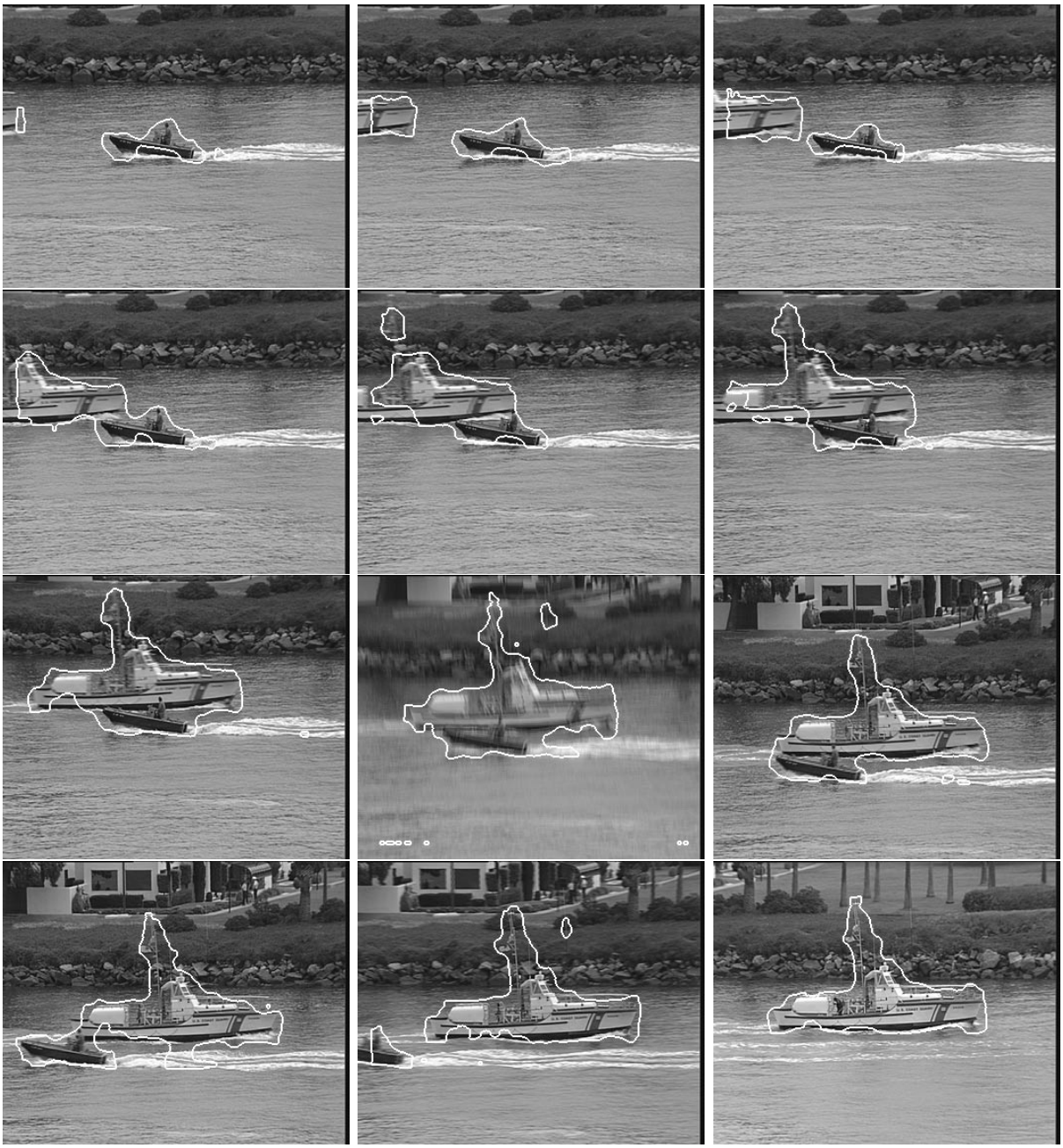

Figure 3.21: Motion segmentation of the *coastguard* sequence (Feature-based motion estimation, implicit geometric active contour).

information is used to improve the segmentation result.

To include both static and moving camera, we chose the *hall-and-monitor* and the *coastguard* sequences. For the former sequence a reference segmentation created manually was provided within the COST activity. Using this reference segmentation, we compared

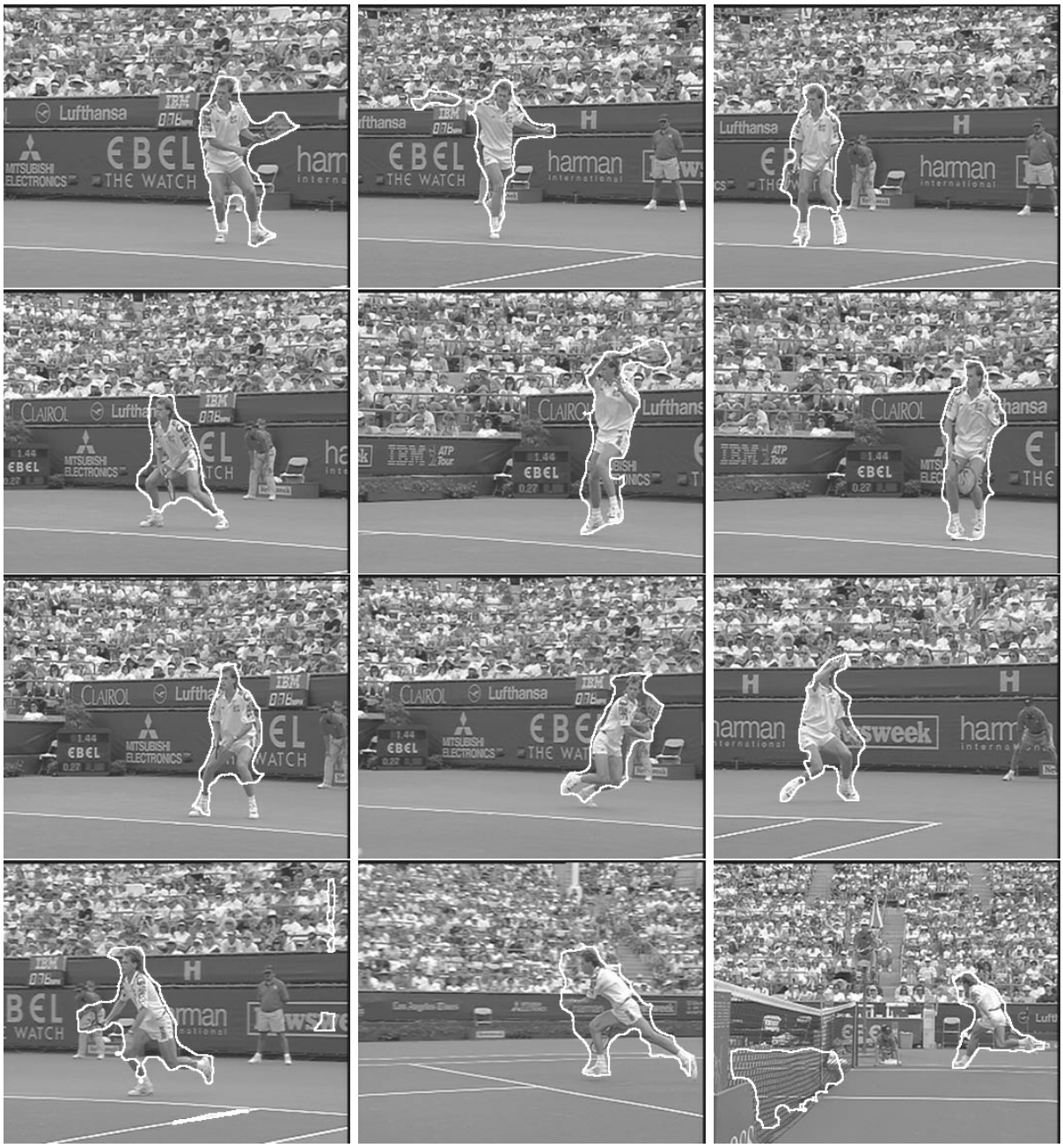

Figure 3.22: Motion segmentation of the *Stefan* sequence (Feature-based motion estimation, implicit geometric active contour).

the results of our segmentation algorithm to those calculated with the COST-AM. The comparison was carried out using the spatial quality measure (sQM) as proposed by [136]. The spatial quality measure is based on the mask error value, i. e., it accumulates pixels that deviate from the reference mask. To integrate a perceptive weighting, two error types

(missing foreground points, added background points) are distinguished. For both types, the distances of wrong pixels to the reference mask are taken into account. In order to obtain an SNR-like presentation, the sQM measure is converted to an upper-bounded logarithmic scale by using

$$\mathrm{sQM(dB)} = 10 \cdot \log\left(\frac{1}{1+\mathrm{sQM}}\right). \tag{3.69}$$

Therefore, a perfect segmentation sets the above spatial quality measure to zero, while negative values indicate segmentation errors.

Figures 3.23 and 3.24 display evaluation results for segmentations of the *hall-and-monitor* sequence obtained by the COST-AM and by our algorithm.

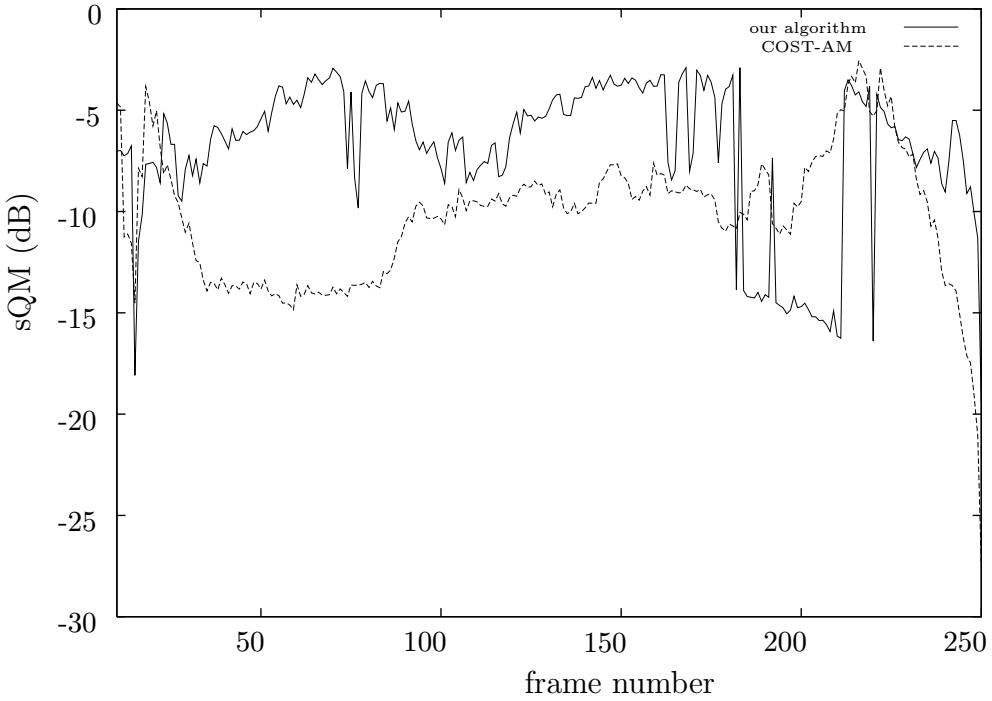

Figure 3.23: Spatial quality measure for the segmentation of the *hall-and-monitor* sequence (left person).

Figure 3.23 depicts the segmentation of the walking person on the left (cf. Figure 3.20). We observe that our segmentation outperforms the COST-AM up to frame 180. The inferior results obtained by the COST-AM are due to the temporal accumulation which clutters the object with background information. Around frame number 200, the segmentation quality of our algorithm suddenly drops. In this part of the sequence, the distance between the two walking persons is rather small. Here, the evolving contour does

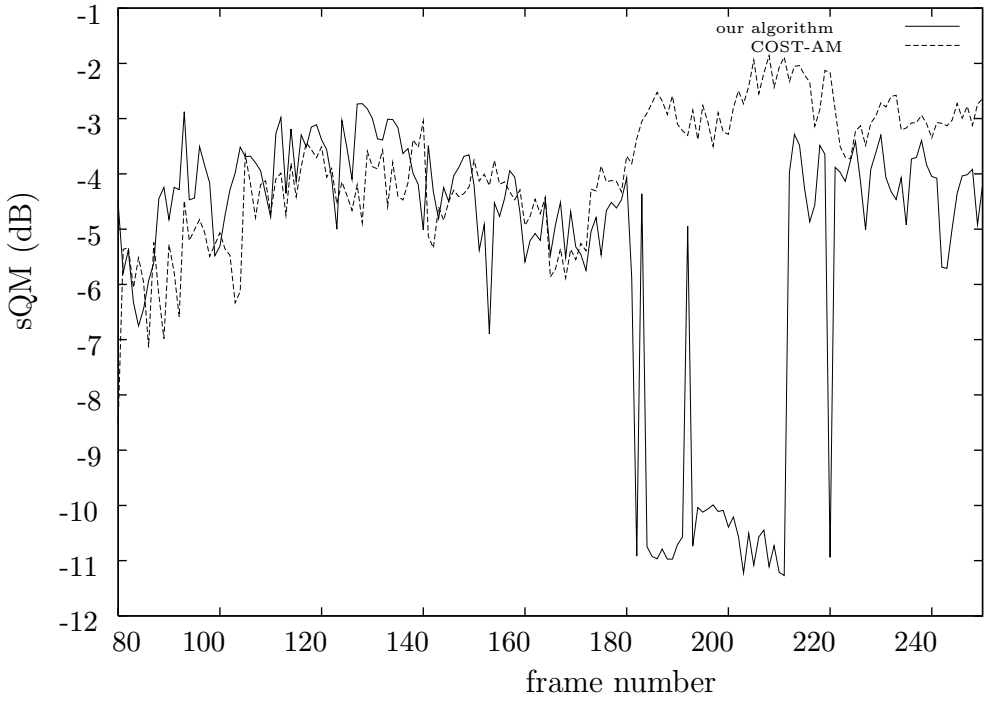

Figure 3.24: Spatial quality measure for the segmentation of the *hall-and-monitor* sequence (right person).

not split and segments the two persons as one object. Consequently, the segmentation error becomes rather large. In Figure 3.24 the evaluation with respect to the walking person on the right is displayed (cf. Figure 3.20). Here, both algorithms yield comparable results up to frame 180. Again, we observe the decreasing segmentation quality of our algorithm around frame 200 due to the above-mentioned problem.

Let us now turn to segmentation results of the *coastguard* sequence calculated by the COST-AM and by our algorithm. For this example a reference segmentation was not available. Therefore, we have to rely on visual comparisons. Figure 3.25 displays segmentations of the frames 50, 100, 150, 200. We selected those frames because they illustrate the characteristics of the different approaches. Note that for other parts of the sequence the results are similar. We observe that our algorithm provides acceptable segmentation results for each frame. However, since we do not employ temporal accumulation, the segmentation of a single frame does not benefit from results obtained on frames earlier in time. In contrast, the COST-AM requires several frames to provide acceptable segmentation results by aggregating object regions temporally. Therefore, the COST-AM results for the frames 50, 100, and 150 are rather poor. However, the segmentation results achieved for frame 200 (and also for consecutive frames) are very accurate.

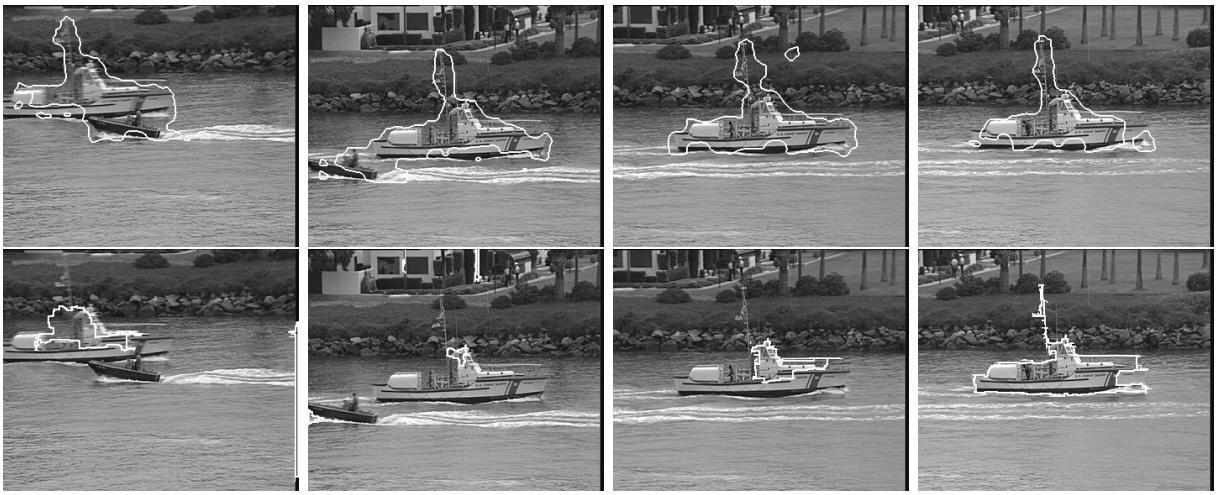

Figure 3.25: Segmentation of the *coastguard* sequence. (a) Top row: results of our algorithm, (b) Bottom row: results of the COST-AM.

Summarizing, we demonstrated the performance of our segmentation algorithms on a collection of well-known real-world sequence. For sequences recorded by a static camera, we obtained good segmentation results. In this case, all image features where normal or full motion could be determined are integrated in the segmentation procedure. In the case of a moving camera, however, only image features surrounded by an appropriate texture can be considered for motion estimation and thus for motion-based segmentation. Consequently, the segmentation results for sequences recorded by a moving camera are less accurate.

Chapter 4

Contour-based Classification of Video Objects

4.1 Introduction

The algorithms for moving object segmentation developed in the previous chapter can be used in a variety of applications ranging from video compression [52, 131] to object-based video abstracting [60]. One of the most exciting application areas is the recognition of objects appearing in video sequences. In general, object recognition can be addressed at different levels of abstraction. For instance, an object might be classifiable as a "cat" (entry level or object class), as a "Siamese cat" (subordinate level) or as "my neighbor's cat" (exemplar level or individual object) [128].

In the following, we concentrate on the recognition of video objects with respect to the object-class level and develop a system for the classification of moving objects appearing in video sequences. Since our approach relies on object shapes, an appropriate object segmentation is required. This segmentation is provided by our motion-based segmentation algorithms as developed in Chapter 3. The classification system is not restricted to a special class of objects. Instead, our purpose is to provide a generic approach that can be tailored to specific application areas.

To classify objects, two aspects have to be considered. First, a *representation* of the object is needed that captures its characteristics. For this purpose, one could consider features extracted from a 3D object model (*object-centered representation*) or rely on information gathered from two-dimensional views taken from different perspectives (*viewer-centered representation*). In both cases, within the visual feature scope, color [51,100,126], texture [44, 74], shape [16, 32, 70, 101], or motion [25, 66, 97] cues and their combina-

tion [108,119,135] can provide appropriate representations. Second, for assigning objects to different classes, class definitions are required. This might be achieved by providing a *reference set* containing manually classified objects.

Our system for object classification [67,105] consists of two major parts: (1) a database containing representations of prototypical video objects and (2) an algorithm for matching extracted video objects with the database entries. The object representation is viewer-centered and relies on shape information which is sufficient for many common objects [133]. In detail, curvature features of the contour of each two-dimensional appearance of an object in a video frame are calculated. These features are matched to those of views of prototypical video objects stored in the database. The final classification of the object is achieved by integrating the matching results for a number of successive frames. This adds reliability to the approach, since unrecognizable single object views occurring in the video sequence are insignificant with respect to the whole sequence.

The main innovation of our approach is that we consider video object retrieval rather than still image retrieval. For this purpose, we propose an accumulation process that integrates the results for a collection of frames from the video sequence. In addition, we discuss classification results for both manual and automatic segmentations.

The remainder of the chapter is structured as follows. After discussing related work, the representation of prototypical video objects within the database is described in Section 4.3. Section 4.4 discusses the matching procedure. Finally, Section 4.5 presents experimental results on a collection of real-world video sequences.

4.2 Related Work

A large body of literature is available on object recognition. A recent survey including both object-centered and viewer-centered representations can be found in [11]. With respect to viewer-centered representations, typical techniques for representing a single view rely on principal component analysis [96], shape templates [29], or shape features [87].

In our work, we focus on shape descriptions calculated with the curvature scale space (CSS) method developed by Mokhtarian et al. [1, 2, 87–89]. Recently, an extension of this technique has been selected for inclusion into the MPEG-7 standard [16]. The main advantages of the CSS method are robustness, efficiency, size invariance, and rotation invariance. Concerning robustness, small perturbations of the object outline do not disturb the classification process. This makes the CSS method appropriate when working with automatic object segmentations which might contain segmentation errors. Efficiency is

especially important in the context of video sequences, since a large number of frames has to be evaluated. In addition, classifying a large variety of objects requires a large reference set of pre-classified objects.

The CSS technique has its roots in still image retrieval. The main difference of our approach is that video objects rather than single images are to be classified. Thus, we extend the CSS technique for the processing of video sequences. In addition, we extract additional information from the object's shape and the curvature scale space to minimize ambiguities in the classification process.

While the CSS technique is appropriate for representing single object views, one has to consider the representation of complete 3D real-world objects. A possible approach is to capture views of those objects by walkarounds. Then, using the aspect graph representation [64, 65], a minimal set of prototypical views is selected from these views. For instance, a shape similarity-based aspect graph has been employed in [31].

However, to create views from walkarounds, the corresponding real-world object must be available. Also, this procedure can be very time consuming. Therefore, we follow a different approach: we select images from clip art libraries and group them into different object classes. Note that we collect arbitrary views not necessarily of one individual object, but instead capture a large variety of objects belonging to the same class. Obviously, this complicates the decision on the optimal number of views. However, since CSS-based matching can be performed very fast, additional database entries do not pose problems with respect to computational complexity.

4.3 Representation of Video Objects

As noted above, there are two ways to generate object-related information. First, one could extract features from a 3D object model, and second, one can use a collection of two-dimensional views of an object, taken from different perspectives as a basis. In cognitive psychology, a number of theories have been developed with regard to object representation in the human brain [13,76,133]. Although a general theory is not available, psychophysical evidence indicates that humans encode three-dimensional objects as multiple viewpoint-specific representations that are largely two-dimensional [127].

We adhere to this theory and store a number of different two-dimensional views for each object class, so-called object views. Furthermore, we pool different views of different objects into one object class to obtain a reliable class definition. Figure 4.1 illustrates the object class *car* by depicting several object views from this class.

Figure 4.1: A subset of the two-dimensional views representing the object class *car*.

Of the views that can be generated from different perspectives, we prefer so-called *canonical views*. A canonical view shows an object in a — with respect to human perception — typical perspective and provides a sufficient number of object characteristics to allow for rapid recognition. For instance, for a car, one possible canonical view is a slightly elevated view of the frontal and side parts of the object (see Figure 4.1). A view of the bottom of a car is generally not considered canonical.

Different sources of information are available to characterize a two-dimensional view (e. g., contour, color, texture, or motion). However, many common objects can be identified by their contours only [133]. In our approach, for each object view a few parameters are extracted from its contour and stored in conjunction with the object view's class name in a database. The parameters are calculated using a modified curvature scale space technique that will be discussed in the following.

4.3.1 Basic Curvature Scale Space Representation

The curvature scale space (CSS) technique [1, 2, 87–89] is based on curve evolution ideas, i.e., the deformation of a curve over time. Basically, it provides a multi-scale representation of the curvature zero-crossings of a closed planar contour. The CSS technique is closely related to mean curvature motion [61, 62] that has already been discussed in Section 3.3.1.1. Recall that a curve evolving under mean curvature motion is smoothed and becomes convex after a certain time period. Figure 4.2 depicts an example of an object outline evolving under mean curvature motion. We observe that the shape of the object becomes simpler as the number of iterations increases, and, finally, approaches a convex form. Thus, mean curvature motion provides a *scale-space*, i. e., a continuum of gradually simplified versions of the initial contour.

Now, the basic assumption of the CSS technique is that these simplifications occurring over time are characteristic for a shape under consideration. Consequently, one has to identify appropriate features on the evolving contour that reflect the simplification process. Since convex and concave segments of the contour merge during evolution, and therefore transitions between them vanish, inflection points (zero-crossings) of the curvature located

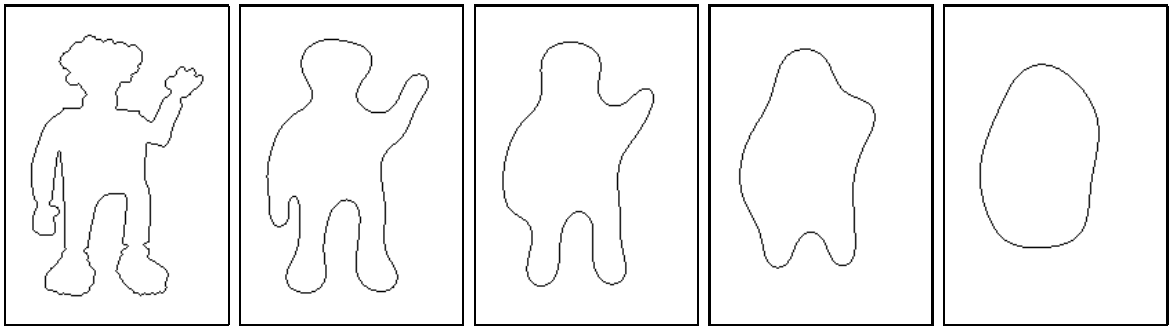

Figure 4.2: Object contour evolving under mean curvature motion. The number of iterations increases from left to right.

at these positions are a reasonable choice.

Figure 4.3 displays an example with regard to zero-crossings of the curvature. On the left the outline of an object view evolves under mean curvature motion. The small dots on the contour depict the zero-crossings of the curvature. We observe that convex and concave segments merge during iterations, and the number of inflection points decreases. Note that certain zero-crossings survive for a large number of iterations. Let us now track the position of these points on the contour during the iterations. For this purpose, the closed curve is parameterized by arc length denoted by a parameter s. Then, we are able to locate the zero-crossings of the curvature with respect to s at each iteration. From the positions of the zero-crossings at different iterations a so-called CSS image can be constructed. The CSS image shows the zero-crossings with respect to their position on the contour and the scale, i. e., the number of iterations (see Figure 4.3). We observe that significant contour properties are visible for a large number of iterations which results in high peaks in the CSS image. However, segments with rapidly changing curvatures caused by noise produce only small local maxima.

In detail, consider the closed planar parametric curve $C_0 = (x(s), y(s))$ with the normalized arc length parameter $s \in [0, 1]$ that represents the initial outline of the object view. Then, curve evolution similar to mean curvature motion can be achieved by smoothing the components $x(s)$ and $y(s)$ iteratively with a one-dimensional Gaussian kernel of width σ. The curvature κ of a point on the parametric curve can be calculated according to Equation 3.3. Consequently, a CSS image $I(s, n)$ is defined by

$$I(s, n) = \{(s, n) | \kappa(s, n) = 0\} \tag{4.1}$$

where s denotes the arc length parameter and n denotes the iterations.

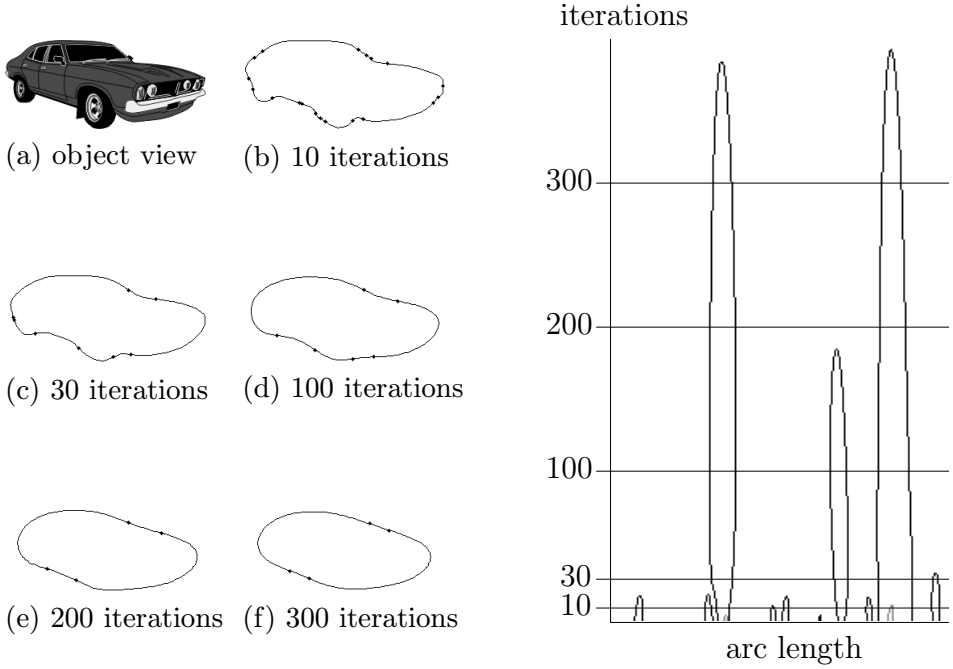

Figure 4.3: Construction of an CSS image. LEFT: (a)-(f) Smoothed contour after 10, 30, 100, 200 and 300 iterations. The small dots on the contour mark the zero crossings of the curvature. RIGHT: Resulting CSS image.

A numerical implementation of the CSS technique can employ similar techniques as those discussed in Section 3.3.1.3. Again, we discretize the parametric curve by a collection of equally distributed marker points (x_i, y_i). However, to include size invariance into the CSS technique, we sample for each contour a fixed number of equidistant contour points (in our implementation we use 200 sample points). Curve evolution can then result from mean curvature motion or the iterative Gaussian smoothing as described above.

In many cases the peaks of the CSS image provide a robust and compact representation of an object view's contour [2, 88, 89]. Note that a rotation of an object view in the image plane can be accomplished by shifting the CSS image left or right in a horizontal direction. Furthermore, a representation of a mirrored object view is obtained by mirroring the CSS image. However, due to the dependence on zero crossings of the curvature, convex object views cannot be represented appropriately with the CSS technique.

To obtain a compact representation, in the basic CSS technique a peak is represented by its height (given by the number of iterations) and its position on the contour. Furthermore, only significant peaks, i. e., those above a certain noise level, are extracted from the CSS image. For instance, for the example depicted in Figure 4.3, only four data pairs have to be stored, assuming a noise level of 30 iterations.

4.3.2 Modifications to the Curvature Scale Space Representation

A major drawback to the basic CSS technique as described in the last section is the occurrence of ambiguities. Under certain conditions shallow and deep concavities on a contour may result in peaks of the same height in the CSS image. Figure 4.4 depicts this problem: The shallow concavity of the object contour shown in (a) and the deep concavity of the object contour shown in (b) result in peaks of nearly the same height (relative difference about 1%) in the CSS images (c). Consequently, certain contours differing significantly in their visual appearance are claimed to be similar by the basic CSS technique.

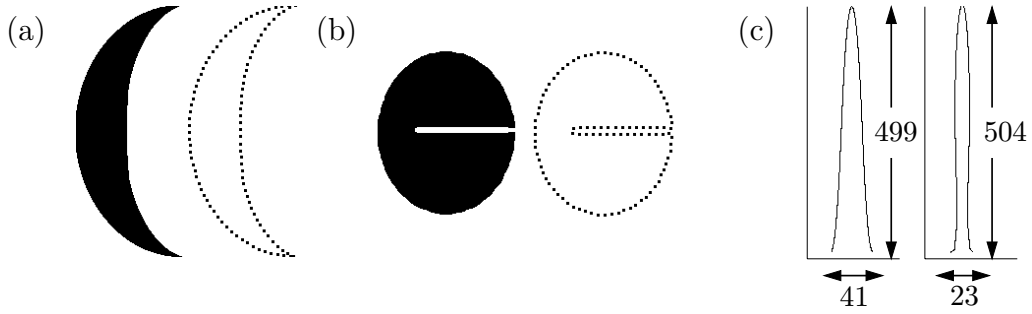

Figure 4.4: Ambiguities in CSS images. (a) shallow concavity: object view (left), contour (right) (b) deep concavity: object view (left), contour (right), (c) left: CSS image of (a), right: CSS image of (b).

As a remedy Abbasi et al. [2] presented several approaches for avoiding these ambiguities. However, the proposed strategies increase the computational cost significantly. In our extension [105], we utilize additional information already available in the CSS image. In addition to the height of a peak in the CSS image, we also extract its width at the bottom of the arc-shaped contour corresponding to the peak. As depicted in Figure 4.4c, the shallow and the deep concavity result in peaks of similar height but their widths differ significantly (the relative width difference is about 80%). Note that the width specifies the normalized arc length distance of the two zero-crossings bordering the contour segment represented by the peak in the CSS image. Thus, for each peak in the CSS image three values have to be stored: the position of the maximum, the corresponding number of iterations, and the width at the bottom of the arc-shaped contour.

Furthermore, it is possible to combine the CSS representation with other descriptors. Commonly used shape descriptors include area, perimeter, elongation, eccentricity and

circularity [16, 120]. These descriptors, consisting only of a single value, can be compared very efficiently. Consequently, they might be used within a prefiltering step to avoid subsequent computations. In our approach, we store in addition to the CSS-related information the *aspect ratio* of the shape given by the division of height and width of the shape's bounding box. Note that this descriptor is not invariant to rotation of the object's shape. While this restriction is not useful in the context of generic image retrieval, it is appropriate in our context. Recall that we populate the database by collecting canonical views, i. e., views from which an object can be easily recognized. For many video objects (e. g., cars, ships) it can be assumed that they appear in typical perspectives in the video sequence. Then, using the aspect ratio as a filter criterion includes only reasonable database entries in further matching procedures. Note that the aspect ratio is invariant to mirroring.

Let us summarize our approach to representing video objects and thus creating appropriate database entries. Each object is represented by a collection of object views. For each object view, its aspect ratio and a number of data triples consisting of position, height, and width of the CSS peaks are stored in the database. The matching algorithm described in the following section utilizes this information to compare segmented video objects with the prototypical video object views in the database.

4.4 Matching of Video Objects

Our classification of segmented video objects is based on a two-step matching process. Let us denote a segmented video object by $S = \{S_1, \ldots, S_k\}$ where $S_i, i \in \{1, \ldots, k\}$, is a single object view segmented from a frame of the video sequence. The database contains a collection of prototypical video objects represented by a number of object views. Let us denote all object views contained in the database by $P = \{P_1, \ldots, P_n\}$ where $P_j, j \in \{1, \ldots, n\}$ denotes a single prototypical object view. Note that for each object view P_j the corresponding object class is known.

In the first step of the matching process to be discussed in detail in the following section each segmented object view S_i is compared to all object views P_j in the database. Thus, for each S_i a best match and a corresponding class label is determined. However, the classification might be incorrect for several views due to segmentation errors or an insignificant object outline. Therefore, a second matching step (see Section 4.4.2) considers the different classifications. Here, the results are accumulated and a confidence value is calculated. On its basis, the object class of the object in the sequence is determined.

4.4.1 View-based Matching

In order to find the most similar object view P_j for a query object view S_i, a view-based matching algorithm is required. The general idea is to calculate distances between database views and an object view under consideration by comparing the peaks in the corresponding CSS images. Recall that those peaks are represented by triples (height, position, width). The matching algorithm has to take into account that the object shape might be mirrored or rotated compared to the best match in the database. As mentioned before, rotation of the original object corresponds to shifting the CSS image. Thus, the matching procedure needs to be executed many times until the best-matching position is found. A heuristic is used to reduce execution time. Only the most promising rotations are calculated. These are determined by shifting the CSS images so that the highest peaks are aligned. Since not all possible rotations of an object view are stored in the database, it is reasonable to compensate this during the matching process. Several requirements exist which need to be met in the matching process. For all peaks, the matched peak needs to be within a certain height, position and width range. These ranges are set via threshold parameters that have to be chosen empirically. Note that setting these threshold values is not very critical. As noted above, the aspect ratio might be employed for prefiltering.

In detail, the algorithm works as follows [105]:

(1) *Alignment:* First of all, an alignment of the peaks in the CSS images might be necessary to account for rotation or different starting positions when sampling object outlines. Recall that the rotation of an object shape corresponds to shifting the CSS image. In the following, C_1 and C_2 denote the CSS images to compare. Alignment is achieved by shifting C_2 until the position of the highest peak matches the position of the highest peak in C_1. Since there might be several peaks of similar height, a number of possible alignments are calculated. For each alignment the following step is performed.

(2) *Peak matching:* For each significant peak in C_1 we examine whether a related peak in C_2' exists. Note that C_2' denotes an aligned version of C_2 according to the previous step. Two peaks match if their height, position and width are within a certain range. If a matching peak can be determined the Euclidean distance with respect to height and position is calculated. Otherwise a distance value is obtained by multiplying the height of the peak under consideration with a penalty factor. Finally, the distance between C_1 and C_2' is given by summing up the distance values obtained for the different peaks.

(3) *Matching distance:* For each alignment position the distance between both CSS representations is calculated. In addition, to account for horizontal mirroring steps 1 and 2 are performed for a mirrored version of C_2. Finally, the matching distance of C_1 and C_2 is given by the minimum distance calculated for the different versions of C_2.

The algorithm might return a maximum value if none of the peaks can be matched within the given tolerance range. If this is the case, the two objects are considered significantly different, and therefore a match is not possible. A clear rejection helps to improve the overall results of the matching algorithm since object views which do not bear much resemblance to objects from a sequence are eliminated in this way.

4.4.2 Sequence-based Matching

The matching procedure returns a list of matches for each object view in the sequence. This list contains the differences of the object view from the sequence to object views in the database and their object class. Only the top match, i .e., the object view with the minimal difference, and its corresponding class is used for evaluation. In some cases no match at all can be found in the database. Thus, no conclusive statement can be made about the class of the object view from the sequence. Since the database does not contain all possible object views, such a result might occur frequently, depending on the object in the sequence.

All available top matches are used for accumulation. In the accumulation process, the inverse difference for each object view in the sequence to its top match is added to an entry for the specific recognized object class O_i (e. g., *car*, *people*). For each object class O_i this procedure yields one accumulated inverse distance value d_i. The sum of these accumulated values gives the total accumulated inverse difference for the sequence under consideration. In order to normalize the result, each d_i is divided by this sum, yielding normalized inverse distance values $d_i' \in [0, 1]$. The object class with d_i' larger than a reliability threshold (e. g., $d_i' > 70\%$) is considered to be the object class of the sequence.

4.5 Experimental Results

For the purpose of evaluation we created a test database containing 247 views of the object classes *cars*, *people*, and *other*. For each object class a number of images was collected from a clip art library. We chose these categories to represent a specific class of rigid

objects (*cars*) and a specific class of non-rigid objects (*people*). The third class (*other*) was introduced to increase the variability of object shapes in the database.

The object class *people* contains most objects (102 images). The contours of humans differ greatly in image sequences, e. g., the positions of arms and legs have considerable impact on the contour. Thus, to enable for recognition of query objects displaying a human, it is necessary to have a large variety of images in the database. The object class *car* is very well represented in the database. With a limited number of 48 cars most perspectives and types are covered. The object classes *other* holds 97 images. Thus, by randomly choosing an object view from the database, the probabilities for the object classes *cars*, *people*, and *other* objects are about 41%, 19%, and 40%.

The indexing of the database with about 250 objects required 30 seconds computation time on a standard personal computer[1], thus 8 CSS images per second could be calculated. The database stores for each CSS image the name of the object class, the data for the relevant peaks (height, position, width), and the aspect ratio of the original object outline.

4.5.1 Experimental Results on Manually Segmented Video Objects

First of all, we evaluated our approach on *manually* segmented video objects to indicate the performance in absence of segmentation errors. In particular, we chose two types of sequences, either containing rigid objects (cars) or non-rigid objects (walking people). Consequently, the 97 objects belonging to the class *other* complicate the classification process.

To match a sequence to the database, the CSS features of the sequence need to be calculated first. In a second step, the calculated CSS features of the sequence are matched to the pre-calculated CSS features of the object views stored in the database. The whole matching process can be done in 5 frames per second. Of course, the matching time increases linearly with additional object views added to the database.

Our classification results for the different sequences are shown in Table 4.1. For each sequence the number of frames and the number of frames where a match was possible are given. The last column indicates the classification performance.

We observe that in the case of non-rigid objects (*cars*) the classification is excellent. Since all cars enter or leave the scene, images in the first or last frames exist in which parts of the car are missing. These frames were mostly rejected which explains the unmatched

[1] AMD Athlon 700 MHz CPU running Linux

sequence	segmentation	no. of frames	matched frames	object class
car-1	manually	51	33	cars (100%)
car-2	manually	21	11	cars (87%)
car-3	manually	51	40	cars (100%)
car-4	manually	19	14	cars (100%)
car-5	manually	22	17	cars (100%)
walking-1	manually	29	26	people (71%)
walking-2	manually	13	6	people (67%)

Table 4.1: Classification results for manually segmented real-world sequences. The height, position, and width ranges are set to 20%, 20%, and 40%.

frames. In some cases, however, the contours extracted from these frames resemble object views of different classes and, thus, result in incorrect matches (see sequence *car-2*). However, the accumulation process over time compensates these mismatches. Sample images of the sequence *car-4* in conjunction with their top matches from the database are displayed in Figure 4.5.

Figure 4.5: Classification results for the *car-4* sequence. TOP ROW: Manually segmented objects views. BOTTOM ROW: Best matches from the database for the object views displayed above.

The results obtained on non-rigid objects are less convincing. Since there are a lot of deformations over time some contours in those sequences happen to have no similar representation in the database. Consequently, several mismatches occur and reduce the reliability of the classification. Including additional views for non-rigid objects, however, should improve the classification performance in these cases. Figure 4.6 displays several frames from the *walking-2* sequence and the corresponding top matches. Here, one mismatch is visible. The database contains no human in a similar pose, and the CSS peaks of the helicopter and the human are just in range of the parameters. Although the matching distance is rather large compared to the other frames of the sequence, this indicates that

ambiguities might occur within the proposed CSS method.

Figure 4.6: Classification results for the *walking-2* sequence. TOP ROW: Manually segmented objects views. BOTTOM ROW: Best matches from the database for the object views displayed above.

In addition to the recognition of the object class, it is possible to make out the perspective of the object in the frame. Comparing the top matches in Figures 4.5 and 4.6, the perspectives of the objects in the sequence and the top matches are quite similar.

4.5.2 Experimental Results on Automatically Segmented Video Objects

Let us now turn to the classification of *automatically* segmented video objects. The segmentations were calculated using our motion-based segmentation algorithms as discussed in Chapter 3. Similar to the previous section, we examine the performance for sequences containing rigid and non-rigid objects. A representative for the first category is the *Hamburg taxi* sequence already used in previous chapters. Table 4.2 displays the results obtained for the three objects (car, taxi, van) contained in this sequence.

Obviously, the classification result depends on the accuracy of the segmentation (see Figure 4.7). The car, shown in a characteristic perspective, could be classified reliably throughout the sequence. Also, the classification of the taxi is rather good. However, we should note that the contour of the taxi does not show many significant details. In fact, for a human observer it is hardly possible to identify the object class only by means of

sequence	segmentation	no. of frames	matched frames	object class
Hamburg taxi (car)	automatic	15	14	cars (92%)
Hamburg taxi (taxi)	automatic	15	11	cars (68%)
Hamburg taxi (van)	automatic	15	15	cars (29%)
walking-3	automatic	42	34	people (93%)
hall-and-monitor	automatic	195	78	people (74%)

Table 4.2: Classification results for automatically segmented real-world sequences. The height, position, and width range are set to 20%, 20%, and 40%.

the contours. Thus, several correct matches might be due to the specific structure of our test database. As expected, in the case of the van the classification fails completely. The segmentation does not provide the correct outline of the van but can only extract certain parts of it. Figure 4.7 provides top matches for the object outline of the car.

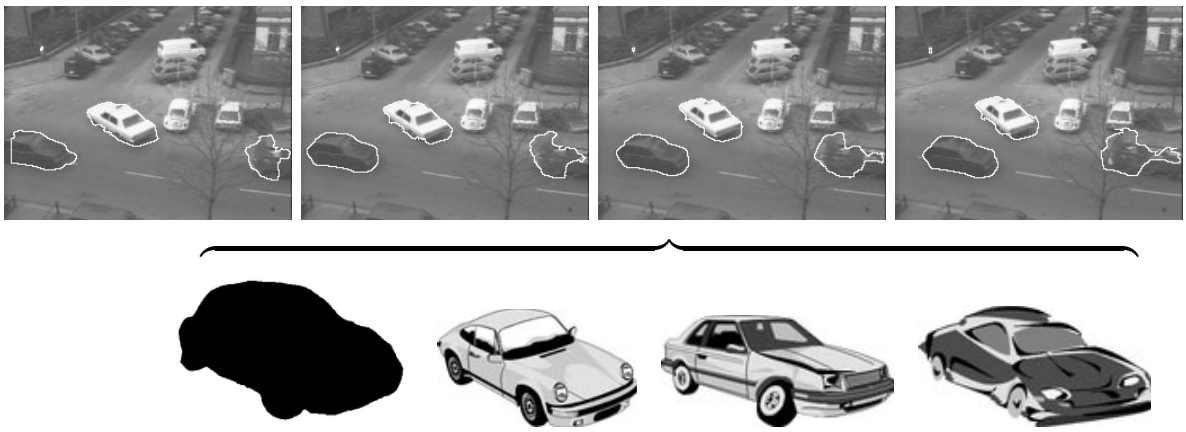

Figure 4.7: Classification results for the *Hamburg taxi* sequence. (a) TOP ROW: Automatically segmented object views, (b) BOTTOM ROW: Classification of the car (left) in the third frame, the matches 1–4 (for the same object view) are displayed.

Next, we discuss the performance on automatic segmentations of non-rigid objects. For this purpose we first examine the *walking-3* sequence. In this sequence a person walks around and changes orientation towards the camera. Only the upper part of the person is visible. Figure 4.8 displays several object segmentations in conjunction with the best matches of the database. The classification result is shown in Table 4.2. Since the object outline does not change very much and typical shapes are available in the database, the classification result is very good. Even several segmentation errors do not disturb the classification process severely. Indeed, the robustness of the CSS method against small

Figure 4.8: Classification results for the *walking-3* sequence. (a) TOP ROW: Automatically segmented objects views, (b) BOTTOM ROW: Best matches from the database for the object views displayed above.

variations in the contour allows a reliable classification.

More challenging with respect to contour deformations is the *hall-and-monitor* sequence also used in previous chapters. Again, the classification result (person walking on the left in the sequence) is shown in Table 4.2. Furthermore, Figure 4.9 depicts some example matches for the sequence. Note that only 78 out of 195 frames could be matched to object views in the database. In addition, most matching distances were rather large compared to those for the *Hamburg taxi* and the *walking-3* sequence. As in the previous section, this indicates that the variability of non-rigid objects requires additional database entries. In particular, the automatic segmentation of the *hall-and-monitor* sequence results in several object outlines whose identification pose problems even to the human observer.

Let us discuss the results as displayed in Figure 4.9 in more detail. The first segmented object view contains several segmentation errors. The segmented area is too large, and a part of the left leg is missed by the algorithm. Nevertheless, characteristic parts of the contour remain and enable a successful classification. The last two object views can not be identified correctly. Since in these cases contour features hardly suffice for a unique classification mismatches occur. The last object view displays another type of segmentation error. Due to the proximity of the two walking persons, the segmentation algorithm provides one object rather than two separate segmentations. Consequently, the matching algorithm finds an object that approximates the shape of both walking people. In this case the matching distance is dominated by the two concave parts of the shape.

Summarizing, the proposed object classification approach is able to cope with seg-

Figure 4.9: Classification results for the *hall-and-monitor* sequence. (a) TOP ROW: Automatically segmented objects views, (b) BOTTOM ROW: Best matches from the database for the object views displayed above.

mentation errors up to a certain extent. Small perturbations of the object shape are eliminated due to the scale-space approach. In addition, the temporal accumulation compensates for occasional mismatches. We obtained excellent results for rigid objects and for objects that do not deform to much over time. However, the classification of non-rigid objects requires additional considerations and would certainly benefit from an extension of the database.

Chapter 5

Conclusions and Perspectives

In the preceding chapters, algorithms for the extraction and classification of moving video objects were developed. Basically, we followed a bottom-up approach, starting at the image signal level and ending at the conceptual primitives level.

First, we proposed two local operators for estimating motion in image sequences. These operators were especially optimized to facilitate subsequent segmentation tasks. The three-dimensional structure tensor turned out as a robust motion estimator. By adding a non-maximum suppression scheme, we were able to improve the localization of a moving object's boundary significantly. This scheme is easy to implement and computationally efficient. Since gradient-based optical flow approaches such as the structure tensor are limited to small image displacements, an additional technique was required. For this purpose, a feature-based motion estimator was developed. In particular, we proposed specific reliability measures that allow to discriminate between features where normal velocity and those where full velocity can calculated. In addition, as opposed to widely used detection schemes, these measures increased the feature density. To account for camera motion and thus enable the segmentation of image sequences acquired by a moving camera, a robust camera motion estimator was developed. For this purpose we estimated global motion parameters by a robust regression method, namely least trimmed squares regression.

Second, based on our motion estimators, motion-based segmentation algorithms were developed. To group selected features into semantically meaningful regions or objects, we investigated implicit active contour models. These models provide a powerful framework that comprises numerical stability and topological adaptivity but does not require a parametrization. However, the computational complexity of standard implementations is not acceptable for video processing. Therefore, we developed fast algorithms for both the

implicit geometric and the implicit geodesic active contour model. In particular, a new narrow-band algorithm and a unified semi-implicit discretization were developed. Applying our contour algorithms to motion pixels, we proposed an efficient algorithm for the segmentation of moving objects under the assumption of a static camera. Through integration of the robust camera motion estimator into the algorithm, object segmentation is possible even when the camera moves. The performance of both segmentation algorithms was demonstrated on a collection of well-known image sequences.

Finally, video objects formed by the segmentation algorithms were classified. For this purpose, we compared the shape features of object views obtained from successive frames to a collection of pre-classified prototypical object views stored in a database. This accumulation procedure allows classification of objects even when a certain number of the extracted object views are distorted by occlusion or segmentation errors. The object classification results were twofold. For rigid objects like cars or talking heads, we obtained excellent results. For non-rigid objects like walking persons, however, the results were inferior. Non-rigid objects result in a great variety of possible shapes. Consequently, this variety needs to be reflected in the database. The advantage of our object classification approach is that a large variety of video objects can be covered without the need of a training procedure. In addition, specific databases can be created for certain classification tasks. Furthermore, the classification process is very fast since only small descriptors have to be compared. The main disadvantage of the approach is that it requires a fairly accurate segmentation of the object.

Future work may comprise a number of improvements and extensions. The segmentation approach proposed so far works by aggregating local motion information within a single frame. It might be useful to extend the integration process to 3D and perform segmentation on several frames simultaneously. For this purpose, one could extend the proposed contour algorithms to 3D and develop an *implicit active tube model* that operates like a "shrinking tube" on several consecutive frames. Using this approach, missing information could be interpolated not only in the spatial dimensions but also on the temporal axis. In addition, the segmentation algorithms can be extended by tracking object features over time. In the current implementation, parts of the object that cease to move are lost during segmentation. The situation could be improved by developing an appropriate tracking procedure.

Several extensions with regard to our classification approach are possible. So far, we have collected prototypical object views and stored them in the database on the same level. Thus, to classify an object view, each database entry has to be compared to the

query. Since the matching procedure operates very fast this does not pose problems up to a certain database size. However, in the case of very large databases, it would be useful to store prototypical object views hierarchically. Based on this hierarchy, only a certain number of general entries would have to be compared first. Then, only the entries subordinated to the best matching general view would be considered. However, creating these general entries is no trivial task. Another possible structure of the database might lead to the representation of *object behavior*. For this purpose, one could establish relations between views of the same object taken at different phases of object motion. Given such a database organization and an appropriate matching algorithm, it should be possible to determine an object's behavior, for instance, "a car turning the corner", "a person walking slowly", or "a person is running".

Establishing behavioral representations could lead to further improvements. On their basis one could consider the feedback loops within an image sequence evaluation system as depicted in Figure 1.1. For instance, when a certain motion phase of an object has been identified, the object view following chronologically might be used as a template to guide the segmentation of the next frame of the image sequence. By using object view templates, it should be possible to improve segmentation and to account for occlusion.

Let us conclude with a quotation by Hans-Hellmut Nagel, one of the pioneers in the field of computer vision. Already in the '70s he recognized the potential of image sequence analysis. Considerable progress has been made since then, however, many open issues still remain.

> *The analysis of image sequences is therefore likely to contribute to the sciences of image understanding as well as to the solution of a wide variety of applications problems.*
>
> Hans-Hellmut Nagel in 1978 [91]

Appendix A

Derivation of the Structure Tensor

Our goal is to find the unit vector n that minimizes

$$J(x) = \int\limits_{x' \in \Omega(x)} w(x - x')(\nabla f(x')^T \cdot n)^2 dx'\, dt'. \tag{A.1}$$

Denoting $\nabla f = (f_x, f_t)^T$ and $n = (n_x, n_t)^T$ we can rewrite Equation A.1

$$\begin{aligned}
J(x) &= \int\limits_{x' \in \Omega(x)} w(x - x')(\nabla f(x')^T \cdot n)(\nabla f(x')^T \cdot n) dx'\, dt' \\
&= \int\limits_{x' \in \Omega(x)} w(x - x')[(f_x n_x)^2 + 2 f_x f_t n_x n_t + (f_t n_t)^2] dx'\, dt' \\
&= \int\limits_{x' \in \Omega(x)} w(x - x')[n_x(f_x^2 n_x + f_x f_t n_t) + n_t(f_t^2 n_t + f_x f_t n_x)] dx'\, dt' \\
&= \int\limits_{x' \in \Omega(x)} w(x - x') \begin{pmatrix} f_x^2 n_x + f_x f_t n_t \\ f_x f_t n_x + f_t^2 n_t \end{pmatrix} n \\
&= \int\limits_{x' \in \Omega(x)} w(x - x') n^T \begin{pmatrix} f_x^2 & f_x f_t \\ f_x f_t & f_t^2 \end{pmatrix} n \\
&= n^T J n
\end{aligned} \tag{A.2}$$

with the *structure tensor*

$$J = \begin{bmatrix} J_{xx} & J_{xt} \\ J_{xt} & J_{tt} \end{bmatrix} \tag{A.3}$$

where the tensor elements $J_{pq}, p, q \in \{x, t\}$ are calculated from

$$J_{pq}(x, y) = \int\limits_{x' \in \Omega(x)} w(x - x') \left(\partial_p f(x', t') \partial_q f(x', t')\right) dx'\, dt'. \tag{A.4}$$

Since J is a symmetric matrix we can replace J by

$$J = UDU^T \tag{A.5}$$

where D is a diagonal matrix with the eigenvalues λ_1, λ_2 of J as elements and the columns of U are the corresponding unit eigenvectors v_1, v_2.

Consequently, Equation A.2 can be rewritten as follows.

$$\begin{aligned} J(x) &= n^T J n \\ &= n^T U D U^T n \\ &= n^T \begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix} n \\ &= \begin{pmatrix} n^T \cdot v_1 & n^T \cdot v_2 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_1^T \cdot n \\ v_2^T \cdot n \end{pmatrix} \\ &= \lambda_1 (n^T \cdot v_1)^2 + \lambda_2 (n^T \cdot v_2)^2 \end{aligned} \tag{A.6}$$

By virtue of $(n^T v_1)^2 + (n^T v_2)^2 = \cos^2 \alpha + \sin^2 \alpha = 1$, we can simplify Equation A.6 to

$$J(x) = \lambda_1 (1 - a) + \lambda_2 a \tag{A.7}$$

with $a \in [0, 1]$. Under the assumption that $\lambda_1 > \lambda_2$, $a = 1$ minimizes Equation A.7. Finally, since $v_2^T \cdot v_2 = 1$ choosing $n = v_2$ solves our minimization problem.

Bibliography

[1] S. Abbasi and F. Mokhtarian. Shape similarity retrieval under affine transform: Application to multi-view object representation and recognition. In *Proc. International Conference on Computer Vision*, pages 450–455. IEEE, 1999.

[2] S. Abbasi, F. Mokhtarian, and J. Kittler. Enhancing CSS-based shape retrieval for objects with shallow concavities. *Image and Vision Computing*, 18(3):199–211, 2000.

[3] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[4] A. A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel, and T. Sikora. Image sequence analysis for emerging interactive multimedia services — the European COST 211 framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):802–813, November 1998.

[5] A. A. Amini, S. Tehrani, and T. Weymouth. Minimizing the energy of active contours in the presence of hard constraints. In *Proc. International Conference on Computer Vision (ICCV'88, Tarpon Springs, FL, USA)*, pages 95–99, 1988.

[6] A. A. Amini, T. E. Weymouth, and R. C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, September 1990.

[7] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2:283–310, 1989.

[8] G. Aubert and P. Kornprobst. *Mathematical problems in image processing: Partial differential equations and the calculus of variations*, volume 147 of *Applied mathematical sciences*. Springer-Verlag, New York, 2002.

[9] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal on Computer Vision*, 12(1):43–77, 1994.

[10] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–467, 1995.

[11] M. Bennamoun and G. J. Mamic. *Object Recognition: Fundamentals and Case Studies*. Advances in Pattern Recognition. Springer, Berlin, Heidelberg, 2002.

[12] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[13] I. Biederman. Recognition-by-components: a theory of human image understanding. *Psychological Review*, 94:115–147, 1987.

[14] J. Bigün and G. H. Granlund. Optimal orientation detection of linear symmetry. In *Proc. IEEE International Conference on Computer Vision (ICCV'87, London, Great Britain)*, pages 433–438, Washington DC, 1987. IEEE Computer Society Press.

[15] J. Bigün, G. H. Granlund, and J. Wiklund. Multidimensional orientation estimation with applications to texture analysis and optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:775–790, 1991.

[16] M. Bober. MPEG-7 visual shape descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):716–719, 2001.

[17] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[18] P. Burt, J. Bergen, R. Hingorani, R. Kolczynski, W. Lee, A. Leung, J. Lubin, and H. Shvaytser. Object tracking with a moving camera. In *IEEE Workshop on Visual Motion*, pages 2–12, 1989.

[19] J. F. Canny. Finding edges and lines in images. Master's thesis, Massachusetts Institute of Technology, June 1983.

[20] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):579–698, 1986.

[21] V. Caselles, F. Catté, T. Coll, and F. Dibois. A geometric model for active contours. *Numerische Mathematik*, 66:1–31, 1993.

[22] V. Caselles and B. Coll. Snakes in movement. *SIAM Journal on Numerical Analysis*, 33(6):2445–2456, 1996.

[23] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997.

[24] F. Catté, P.-L. Lions, J.-M. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis*, 29(1):182–193, 1992.

[25] S.-F. Chang, W. Chen, and H. Sundaram. VideoQ: A fully automated video retrieval system using motion sketches. In *Proceedings Fourth IEEE Workshop on Applications of Computer Vision*, 1998.

[26] D. L. Chopp. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106(1):77–91, May 1993.

[27] L. D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 53(2):211–218, March 1991.

[28] L. D. Cohen and I. Cohen. Finite element methods for active contour models and balloons for 2d and 3d images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(15):1131–1147, November 1993.

[29] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Active shape models: Their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, January 1995.

[30] P. Correia and F. Pereira. Segmentation of video sequences in a video analysis framework. In *Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, pages 155–160, 1997.

[31] C. M. Cyr and B. B. Kimia. 3D object recognition using shape similarity-based aspect graph. In and , editors, *International Conference on Computer Vision*, pages 254–261, 2001.

[32] L. da Fontoura Costa and R. M. Cesar, Jr. *Shape Analysis and Classification*. CRC Press, Boca Raton, FL, September 2000.

[33] W. Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, 43:150–177, 1988.

[34] M. Fischler and R. Bolles. Random sample concensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications ACM*, 24(6):381–395, 1981.

[35] M. Gabbouj, G. Morrison, F. Alaya-Cheikh, and R. Mech. Redundancy reduction techniques and content analysis for multimedia services — the European COST 211quat action. In *Workshop on Image Analysis for Multimedia Interactive Services*, pages 69–72, Mai/June 1999.

[36] F. Glazer. Multilevel relaxation in low-level computer vision. In A. Rosenfeld, editor, *Multiresolution Image Processing and Analysis*, pages 312–320. Springer, Berlin, Heidelberg, 1984.

[37] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky. Fast geodesic active contours. In M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, editors, *Scale-space theories in computer vision*, volume 1682 of *Lecture Notes on Computer Science*, pages 34–45. Springer, Berlin, 1999.

[38] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky. Fast geodesic active contours. *IEEE Transactions on Image Processing*, 10(10):1467–1475, October 2001.

[39] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition, 1996.

[40] J. Gomes and O. Faugeras. Reconciling distance functions and level sets. *Journal of Visual Communication and Image Representation*, 11(2):209–223, 2000.

[41] G. H. Granlund. In search of a general picture processing operator. *Computer Graphics and Image Processing*, 8:155–173, 1978.

[42] G. H. Granlund and H. Knutsson. *Signal processing for computer vision*. Kluwer, 1995.

[43] M. A. Grayson. The heat equation shrinks embedded plane curves to round points. *Journal of Differential Geometry*, 26:285–314, 1987.

[44] R. Haralick, K. Shanmugam, and I. Dinstein. Texture feature for image classification. *IEEE Transactions Systems, Man and Cybernetics*, 3:610–621, 1973.

[45] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. Alvey Vision Conference*, pages 147–151, 1988.

[46] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, 2001.

[47] H. Haußecker and B. Jähne. A tensor approach for precise computation of dense displacement vector fields. In E. Paulus and F. Wahl, editors, *Proceedings Mustererkennung*, Informatik Aktuell, pages 199–208, Berlin, 1997. Springer.

[48] J. J. Helmsen, E. G. Puckett, P. Colella, and M. Dorr. Two new methods for simulating photolithography development in 3d. In G. E. Fuller, editor, *Proc. SPIE, Optical Microlithography IX*, volume 2726, pages 253–261, March 1996.

[49] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[50] T. S. Huang and R. Y. Tsai. Image sequence analysis: Motion estimation. In T. S. Huang, editor, *Image Sequence Analysis*, volume 5 of *Springer Series in Information Sciences*, pages 1–18. Springer-Verlag, Berlin, Heidelberg, 1981.

[51] M. Ioka. A method of defining the similarity of images on the basis of color information. Technical Report RT-0030, IBM Tokyo Research Laboratory, November 1989.

[52] ISO/IEC 14496-2. Information technology – coding of audio-visual objects – part 2: Visual, 1999.

[53] B. Jähne. *Spatio-Temporal Image Processing*, volume 751 of *Lecture Notes in Computer Science*. Springer, Berlin, 1993.

[54] B. Jähne. *Digital Image Processing. Concepts, Algorithms, and Scientific Applications*. Springer, Berlin, Heidelberg, 4th edition, 2000.

[55] S. Jehan-Besson, M. Barlaud, and G. Aubert. Region-based active contours for video object segmentation with camera compensation. In *Proc. IEEE International Conference on Image Processing (ICIP)*, October 2001.

[56] S. Jehan-Besson, M. Barlaud, and G. Aubert. Video object segmentation using eulerian region-based active contours. In *International Conference on Computer Vision*, July 2001.

[57] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *Proc. IEEE International Conference in Computer Vision (ICCV'87, London, UK)*, pages 259–267, June 1987.

[58] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1988.

[59] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi. Conformal curvature flows: from phase transitions to active vision. *Archive of Rational Mechanics and Analysis*, 134:275–301, 1996.

[60] C. Kim and J.-N. Hwang. An integrated scheme for object-based video abstraction. In *Proceedings ACM Multimedia*, 2000.

[61] B. B. Kimia and K. Siddiqi. Geometric heat equation and nonlinear diffusion of shapes and images. *Computer Vision and Image Understanding*, 64(3):305–322, 1996.

[62] B. B. Kimia, A. Tannenbaum, and S. Zucker. On the evolution of curves via a function of curvature. I. The classical case. *Journal of Mathematical Analysis and Applications*, 163(2):438–458, January 1992.

[63] H. Knutsson. *Filtering and Reconstruction in Image Processing.* PhD thesis, Linköping University, 1982.

[64] J. Koenderink and A. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

[65] J. Koenderink and A. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[66] G. Kollios, S. Sclaroff, and M. Betke. Motion mining: Discovering spatio-temporal patterns in databases of human motion. In *Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD 2001, Santa Barbara, CA*, May 2001.

[67] G. Kühne, S. Richter, and M. Beier. Motion-based segmentation and contour-based classification of video objects. In *Proc. ACM Multimedia 2001*, pages 41–50, 2001.

[68] G. Kühne, J. Weickert, M. Beier, and W. Effelsberg. Fast implicit active contour models. In L. Van Gool, editor, *Proc. Pattern Recognition (24th DAGM Symposium)*, volume 2449 of *Lecture Notes in Computer Science*, pages 133–140, Berlin, Heidelberg, September 2002. Springer.

[69] G. Kühne, J. Weickert, O. Schuster, and S. Richter. A tensor-driven active contour model for moving object segmentation. In *Proc. IEEE International Conference on Image Processing (ICIP)*, volume II, pages 73–76, October 2001.

[70] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, August 1998.

[71] T. Lu, P. Neittaanmäki, and X.-C. Tai. A parallel splitting up method and its application to navier-stokes equations. *Applied Mathematics Letters*, 4(2):25–29, 1991.

[72] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings 7th International Conference on Artificial Intelligence*, pages 674–679, 1981.

[73] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, February 1995.

[74] B. Manjunath, J.-R. Ohm, V. V. Vasudevan, and A. Yamada. Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):703–715, 2001.

[75] A.-R. Mansouri, B. Sirivong, and J. Konrad. Multiple motion segmentation with level sets. In *Proc. SPIE Image and Video Communications and Processing*, volume 3974, pages 584–595, January 2000.

[76] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* Freeman, San Francisco, CA, 1982.

[77] T. McInerney and D. Terzopolous. Topologically adaptable snakes. In *Proc. IEEE International Conference on Computer Vision (ICCV'95, Cambridge, MA)*, pages 840–845, June 1995.

[78] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: A survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[79] T. McInerney and D. Terzopoulos. Medical image segmentation using topologically adaptable surfaces. In *Proc. First Joint Conference of Computer Vision, Virtual Reality, and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery (CVRMed-MRCAS'97, Grenoble, France)*, volume 1205 of *Lecture Notes in Computer Science*, pages 23–32, Berlin, March 1997. Springer-Verlag.

[80] T. McInerney and D. Terzopoulos. Topology adaptive deformable surfaces for medical image volume segmentation. *IEEE Transactions on Medical Imaging*, 18(10):840–850, October 1999.

[81] T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, June 2000.

[82] R. Mech and M. Wollborn. A noise robust method for segmentation of moving objects in video sequences. In *International Conference on Acoustics, Speech and Signal Processing (Munich, Germany)*, pages 2657–2660, 1997.

[83] R. Mech and M. Wollborn. A noise robust method for 2D shape estimation of moving objects in video sequences considering a moving camera. *Signal Processing*, 66(2):203–217, April 1998.

[84] P. Meer, D. Mintz, A. Rosenfeld, and D. Y. Kim. Robust regression methods for computer vision: A review. *International Journal on Computer Vision*, 6(1):59–70, 1991.

[85] P. Meer, C. V. Stewart, and D. E. Tyler. Robust computer vision: An interdisciplinary challenge. *Computer Vision and Image Understanding*, 78(1):1–7, 2000.

[86] A. Mitiche and P. Bouthemy. Computation and analysis of image motion: A synopsis of current problems and methods. *International Journal of Computer Vision*, 19(1):29–55, 1996.

[87] F. Mokhtarian. Silhouette-based isolated object recognition through curvature scale space. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(5):539–544, 1995.

[88] F. Mokhtarian, S. Abbasi, and J. Kittler. Efficient and robust retrieval by shape content through curvature scale space. In *Proc. International Workshop on Image DataBases and MultiMedia Search*, pages 35–42, 1996.

[89] F. Mokhtarian, S. Abbasi, and J. Kittler. Robust and efficient shape indexing through curvature scale space. In *British Machine Vision Conference*, 1996.

[90] K. Morton and D. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, Cambridge, UK, 1994.

[91] H.-H. Nagel. Analysis techniques for image sequences. In *Proc. International Joint Conference on Pattern Recognition (Kyoto, Japan)*, pages 186–211, 1978.

[92] H.-H. Nagel. Image sequence analysis: What can we learn from application? In T. S. Huang, editor, *Image Sequence Analysis*, volume 5 of *Springer Series in Information Sciences*, pages 19–228. Springer-Verlag, Berlin, Heidelberg, 1981.

[93] H.-H. Nagel. Constraints for the estimation of displacement vector fields from image sequences. In *Proc. Eighth International Joint Conference on Artificial Intelligence (IJCAI'83, Karlsruhe, Germany)*, pages 945–951, August 1983.

[94] H.-H. Nagel. Image sequence evaluation: 30 years and still going strong. In A. Sanfeliu, J. Villanueva, M. Vanrell, R. Alquézar, J.-O. Eklundh, and Y. Aloimonos, editors, *Proc. International Conference on Pattern Recognition (ICPR'00, Barcelona, Spain, September 3-7)*, volume 1, pages 149–158, Los Alamitos, CA, 2000. IEEE Computer Society.

[95] H.-H. Nagel and A. Gehrke. Spatiotemporally adaptive estimation and segmentation of OF-fields. In H. Burkhardt and B. Neumann, editors, *Proc. Fifth European Conference On Computer Vision (ECCV'98, Freiburg, Germany)*, volume 2 of *Lecture Notes in Computer Science, Vol. 1407*, pages 86–102, Berlin, Heidelberg, June 1998. Springer-Verlag.

[96] S. Nayar, S. Rene, and H. Murase. Realtime 100 object recognition system. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2321–2325, 1996.

[97] C.-W. Ngo, T.-C. Pong, and H.-J. Zhang. On clustering and retrieval of video shots. In and , editors, *Proc. ACM Multimedia 2001*, pages 51–60, 2001.

[98] S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[99] N. Paragios and R. Deriche. Geodesic active contours and level sets for the detection and tracking of moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):266–280, March 2000.

[100] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proc. ACM Conference on Multimedia*, pages 65–73, 1996.

[101] T. Pavlidis. Review of algorithms for shape analysis. *Computer Graphics and Image Processing*, 7(2):243–258, April 1978.

[102] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[103] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, New York, 1992.

[104] B. M. Radig. Image region extraction of moving objects. In T. S. Huang, editor, *Image Sequence Analysis*, volume 5 of *Springer Series in Information Sciences*, pages 311–354. Springer-Verlag, Berlin, Heidelberg, 1981.

[105] S. Richter, G. Kühne, and O. Schuster. Contour-based classification of video objects. In *Proceedings of SPIE, Storage and Retrieval for Media Databases*, volume 4315, pages 608–618, Bellingham, Washington, January 2001. SPIE.

[106] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley, New York, 1987.

[107] P. J. Rousseeuw and K. Van Driesen. Computing lts regression for large data sets. *Institute of Mathematical Statistics Bulletin*, 27(6), November/December 1998.

[108] Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of Visual Communication and Image Representation*, 10:39–62, 1999.

[109] H. Scharr. *Optimal Operators in Digital Image Processing (in german)*. PhD thesis, Interdisciplinary Center for Scientific Computing, University of Heidelberg, 2000.

[110] C. Schmid, R. Mohr, and C. Bauckhage. Comparing and evaluating interest points. In *Proc. International Conference on Computer Vision (ICCV'98, Bombay, India)*, pages 230–235, January 1998.

[111] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, June 2000.

[112] J. A. Sethian. Curvature and the evolution of fronts. *Communications in Mathematical Physics*, 101:487–499, 1985.

[113] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[114] J. A. Sethian. *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monograph on Applied and Computational Mathematics. Cambridge University Press, Cambridge, UK, 1999.

[115] E. Sifakis, C. Garcia, and G. Tziritas. Bayesian level sets for image segmentation. *Journal of Visual Communication and Image Representation*, 13(1/2):44–64, March 2002.

[116] E. Sifakis, I. Grinias, and G. Tziritas. Video segmentation using fast marching and region growing algorithms. In *Workshop on Image Analysis For Multimedia Interactive Services (WIAMIS)*, May 2001.

[117] E. Sifakis, I. Grinias, and G. Tziritas. Video segmentation using fast marching and region growing algorithms. *EURASIP Journal on Applied Signal Processing*, 2002(4):379–388, April 2002.

[118] E. Sifakis and G. Tziritas. Fast marching to moving object location. In M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, editors, *Proc. International Conference on Scale Space Theories in Computer Vision (Scale-Space '99)*, volume 1682 of *Lecture Notes in Computer Science*, pages 447–452, Berlin, Heidelberg, September 1999. Springer.

[119] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[120] M. Sonka, V. Hlavác, and R. Boyle. *Image processing, analysis and machine vision*. Champman and Hall, London, 1st edition, 1993.

[121] H. Spies and H. Scharr. Accurate optical flow in noisy image sequences. In *Proc. IEEE International Conference on Computer Vision (ICCV'01, Vancouver, Canada)*, volume I, pages 587–592, 2001.

[122] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Review*, 41(3):513–537, 1999.

[123] M. Sussman and E. Fatemi. An efficient, interface preserving level set re-distancing algorithm and its application to interfacial incompressible fluid flow. *SIAM Journal on Scientific Computing*, 20(4):1165–1191, 1999.

[124] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An improved level set method for incompressible two-phase flows. *Computers and Fluids*, 27(5-6):663–680, 1998.

[125] M. Sussman, P. Smereka, and S. Osher. A level-set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114:146–159, 1994.

[126] M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[127] M. D. Tarr and H. H. Bülthoff, editors. *Object Recognition in Man, Monkey, and Machine*. MIT Press, Cambrigde, MA, 1998.

[128] M. J. Tarr. Visual pattern recognition. In A. Kadzin, editor, *Encyclopedia of Psychology*. American Psychological Association, Washington, D.C., 2000.

[129] P. Torr and D. Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3):271–300, 1997.

[130] P. Torr and A. Zisserman. Feature based methods for structure and motion estimation. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice, International Workshop on Vision Algorithms, held during ICCV '99, Corfu, Greece, September*, volume 1883 of *Lecture Notes in Computer Science*, pages 278–294, Berlin, Heidelberg, 1999. Springer.

[131] L. Torres and E. J. Delp. New trends in image and video compression. In *X European Signal Processing Conference*, September 2000.

[132] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[133] S. Ullman. *High-level Vision: Object Recognition and Visual Cognition*. MIT Press, Cambridge, MA, 1996.

[134] R. van den Boomgaard. The morphological equivalent of the Gauss convolution. *Nieuw Archief voor Wiskunde*, 10(3):219–236, November 1992.

[135] R. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Utrecht University, March 2000.

[136] P. Villegas, X. Marichal, and A. Salcedo. Objective evaluation of segmentation masks in video sequences. In *Workshop on Image Analysis For Multimedia Interactive Services (WIAMIS)*, pages 85–88, 1999.

[137] J. Weickert. *Anisotropic Diffusion in Image Processing*. European Consortium for Mathematics in Industry. Teubner, Stuttgart, 1998.

[138] J. Weickert. Coherence-enhancing diffusion of colour images. *Image and Vision Computing*, 17:201–212, 1999.

[139] J. Weickert. Application of nonlinear diffusion in image processing and computer vision. *Acta Mathematica Universitatis Comenianae*, 70(1):33–50, 2001.

[140] J. Weickert and G. Kühne. Fast methods for implicit active contour models. In S. Osher and N. Paragios, editors, *Geometric Level Set Methods in Imaging, Vision and Graphics*. Springer, Berlin, Heidelberg, 2002.

[141] J. Weickert and C. Schnörr. A theoretical framework for convex regularizers in PDE-based computation of image motion. *International Journal of Computer Vision*, 45(3):245–264, December 2001.

[142] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*, 7(3):398–410, March 1998.

[143] D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics and Image Processing*, 55(1):14–26, January 1992.

[144] M. Ye and R. M. Haralick. Optical flow from a least-trimmed squares based adaptive approach. In *Proc. International Conference on Pattern Recognition (ICPR'00)*, volume 3, pages 1052–1055, 2000.

[145] M. Ye and R. M. Haralick. Point aerial target detection and tracking - a motion-based bayesian approach. Technical report, Intelligent Systems Laboratory, Department of Electrical Engineering, University of Washington, September 2001.

[146] J. Zhang, J. Gao, and W. Lu. Image sequence segmentation using 3-D structure tensor and curve evolution. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(5):629–641, May 2001.