USC Institute for Robotics and Intelligent Systems Technical Report IRIS-02-418, December 2002.

Motion Segmentation with Accurate Boundaries - A Tensor Voting Approach

Mircea Nicolescu and Gérard Medioni
Integrated Media Systems Center
University of Southern California
Los Angeles, CA 90089-0273
{mnicoles, medioni}@iris.usc.edu

Abstract

Producing an accurate motion flow field is very difficult at motion boundaries. We present a novel, non-iterative approach for segmentation from image motion, based on two voting processes, in different dimensional spaces. By expressing the motion layers as surfaces in a 4-D space, a voting process is first used to enforce the smoothness of motion and determine an estimation of pixel velocities, motion regions and boundaries. The boundary estimation is then combined with intensity information from the original images in order to locally define a boundary tensor field. The correct boundary is inferred by a 2-D voting process within this field, that enforces the smoothness of boundaries. Finally, correct velocities are computed for the pixels near boundaries, as they are reassigned to different regions. We demonstrate our contribution by analyzing several image sequences, containing multiple types of motion.

1. Introduction

The motivation of the motion segmentation problem stems from the fact that motion regions (pixels with similar motion) usually correspond to distinct objects in the scene. Computationally, the problem is addressed by first establishing pixel correspondences between images in order to obtain velocity values at each image location. Based on their velocities, pixels are then grouped into motion regions, separated by motion boundaries, thus producing a segmentation of the image.

However, an inherent difficulty in this process is caused by the presence of the motion boundaries themselves. The very source of information used for segmentation – pixel velocities – are mostly unreliable exactly at the motion boundaries, where the segmentation takes place. The example in Figure 1, showing a truck moving from left to right over a static background, is used to illustrate the problem. From area A that appears in the first image, only half is visible in the second image, the other half being occluded by the moving region. At the opposite side, area B is still visible in the second image, but is now split into two regions, with new, un-occluded pixels in between. Even where no occlusion takes place, such as at the upper boundary, area C is also split in the second image, due to the motion between regions.

Consequently, the apparent motion around boundaries cannot be precisely determined by using any similarity criteria, since the areas being compared must have a finite extent. Moreover, it is not realistic to assume that all the wrong matches can be later removed as noise. Due to the similarity of partial areas, wrong correspondences are often assigned in a consistent manner, resulting in over-extended image regions.

The key observation is that one should not only rely on *motion cues* in order to perform motion segmentation. Examining the original images reveals a multitude of *monocular cues*, such as intensity edges, that can aid in identifying the true object boundaries. A second look at Figure 1 will confirm it.

Figure 1. Non-similarity at motion boundaries

In this context, we formulate the problem of motion analysis as a two-component process, that:

- enforces the smoothness of motion, except at its discontinuities
- enforces the smoothness of such discontinuities, aided by monocular cues

A computational methodology that successfully enforces the smoothness constraint in a consistent and unified manner, while preserving discontinuities is Tensor Voting [1]. This technique also benefits from the fact that it is non-iterative and it does not depend on critical thresholds.

In our previous work [2] we have developed a voting-based framework designed initially for sparse data, that enforces the smoothness of motion in order to extract motion layers, as smooth surfaces in the 4-D space of image coordinates and pixel velocities.

In this paper we propose a novel approach for motion segmentation from two images, by extending our 4-D framework in order to handle real data, and integrating it with a 2-D voting-based method for accurate inference of motion boundaries.

In the next subsections we give a brief review of the related work in the area, and an overview of our method. In Section 2 we examine the voting framework by first describing the Tensor Voting formalism in 2-D, then we discuss how the voting concepts are generalized and extended to the 4-D case. In Section 3 we present our approach for establishing the initial candidate matches. In Section 4 we describe the extraction of motion layers in 4-D, and in Section 5 we present the boundary inference in 2-D. Section 6 shows our experimental results, while Section 7 summarizes our contribution and provides further research directions.

1.1. Related work

Barron, Fleet, and Beauchemin [3] provide a useful review of the computational methodologies used in the motion analysis field. Optical flow techniques – such as differential methods [4], region-based matching [5], or energy-based methods [6] – rely on local, raw estimates of the optical flow field to produce a partition of the image. However, the flow estimates are very poor at motion boundaries and cannot be obtained in uniform areas.

Past approaches have investigated the use of Markov Random Fields (MRF) in handling discontinuities in the optical flow [7]. While these methods give some good results, they rely heavily on a proper spatial segmentation early in the algorithm, which will not be realistic in many cases. Another research direction uses regularization techniques, which preserve discontinuities by weakening the smoothing in areas that exhibit strong intensity gradients [8]. Here an incorrect assumption is also made, that the motion boundaries can always be detected in advance, based on intensity only.

Significant improvements have been achieved by using layered representations and the Expectation-Maximization algorithm [9][10]. There are many advantages of this formalism – mainly because it represents a natural way to incorporate motion field discontinuities, and it allows for handling occlusion relationships between different regions in the image. While these techniques provide a basis for much subsequent study, they still suffer from some major defects – the procedure requires an initialization step, which is essentially arbitrary, the algorithm is iterative, subject to stability concerns, and the description of the optical flow is parameterized and does not permit a general description as would be desirable.

Shi and Malik [11] have approached the problem of motion segmentation in terms of recursive partitioning of the spatio-temporal space through normalized cuts within a weighted graph, but no prescription is offered for deciding when the space has been adequately partitioned.

1.2. Overview of our method

In order to compute a dense velocity field and to segment the image into motion regions, we use an approach based on a *layered 4-D representation* of data, and a *voting scheme* for communication. First we establish candidate matches through a multi-scale, normalized cross-correlation procedure. Following a perceptual grouping perspective, each potential match is seen as a token characterized by four attributes – the image coordinates (x,y) in the first image, and the velocity with the components (v_x,v_y).

Tokens are encapsulated as (x,y,v_x,v_y) points in the 4-D space, this being a natural way of expressing the spatial separation of tokens according to *both* velocities and image coordinates. In general, for each pixel (x,y) there can be several candidate velocities, so each 4-D point (x,y,v_x,v_y) represents a potential match.

Within this representation, smoothness of motion is embedded in the concept of surface saliency exhibited by the data. By letting the tokens communicate their mutual affinity through voting, noisy matches are eliminated as they receive little support, and distinct moving regions are extracted as smooth, *salient surface layers*, in the 4-D space.

Although noisy correspondences are rejected as outliers, there are also wrong matches that are consistent with the correct ones. This mostly occurs at the motion boundaries, where the occluding layer is typically over-extended towards the occluded area.

In the next stage we infer the correct motion boundary by adding monocular information from the original images. First we define zones of boundary uncertainty along the margins of layers. Within these zones we create a 2-D saliency map that combines the following information: the position and overall orientation of the

layer boundary, and the strength and orientation of the intensity edges from the images. Then we enforce the smoothness and continuity of the boundary through a 2-D voting process. The true boundaries are finally extracted as the most *salient curves* within these zones.

2. The Tensor Voting framework

2.1. Voting in 2-D

The use of a voting process for feature inference from sparse and noisy data was formalized into a unified tensor framework by Medioni, Lee and Tang [1]. The input data is encoded as tensors, then support information (including proximity and smoothness of continuity) is propagated by voting. The only free parameter is the scale of analysis, which is indeed an inherent property of visual perception.

In the 2-D case, the salient features to be extracted are points and curves. Each token is encoded as a second order symmetric 2-D tensor, geometrically equivalent to an ellipse. It is described by a 2×2 eigensystem, where eigenvectors e_1 and e_2 give the ellipse orientation and eigenvalues λ_1 and λ_2 are the ellipse size. The tensor is represented as a matrix $S = \lambda_1 \cdot e_1 e_1^T + \lambda_2 \cdot e_2 e_2^T$.

An input token that represents a curve element is encoded as a *stick tensor*, where e_2 represents the curve tangent and e_1 the curve normal, while λ_1=1 and λ_2=0. An input point element is encoded as a *ball tensor*, with no preferred orientation, while λ_1=1 and λ_2=1.

The communication between tokens is performed through a voting process, where each token casts a vote at each site in its neighborhood. The size and shape of this neighborhood, and the vote strength and orientation are encapsulated in predefined voting fields (kernels), one for each feature type – there is a stick voting field and a ball voting field in the 2-D case. The fields are generated based only on the scale factor σ. Vote orientation corresponds to the smoothest local curve continuation from voter to recipient, while vote strength $VS(\vec{d})$ decays with distance $|\vec{d}|$ between them, and with curvature ρ:

$$VS(\vec{d}) = e^{-\left(\frac{|\vec{d}|^2+\rho^2}{\sigma^2}\right)} \qquad (1)$$

Figure 2(a) shows how votes are generated to build the 2-D stick field. A tensor P where curve information is locally known (illustrated by curve normal $\vec{N}_P$) casts a vote at its neighbor Q. The vote orientation is chosen so that it ensures a smooth curve continuation through a circular arc from voter P to recipient Q. To propagate the

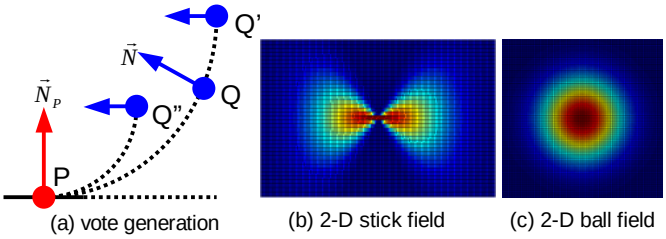

Figure 2. Voting in 2-D

curve normal $\vec{N}$ thus obtained, the vote $V_{stick}(\vec{d})$ sent from P to Q is encoded as a tensor according to:

$$V_{stick}(\vec{d}) = VS(\vec{d}) \cdot \vec{N}\vec{N}^T \qquad (2)$$

Figure 2(b) shows the 2-D stick field, with its color-coded strength. When the voter is a ball tensor, with no information known locally, the vote is generated by rotating a stick vote in the 2-D plane and integrating all contributions. The 2-D ball field is shown in Figure 2(c).

At each receiving site, the collected votes are combined through simple tensor addition, producing generic 2-D tensors. During voting, tokens that lie on a smooth curve reinforce each other, and the tensors deform according to the prevailing orientation. Each tensor encodes the local orientation of geometric features (given by the tensor orientation), and their saliency (given by the tensor shape and size). For a generic 2-D tensor, its curve saliency is given by (λ_1-λ_2), the curve normal orientation by e_1, while its point saliency is given by λ_2. Therefore, the voting process infers curves and junctions simultaneously, while also identifying outlier noise (tokens that receive very little support).

2.2. Extension to 4-D

The issues to be addressed here are the *tensorial representation* of the features in the 4-D space, the generation of *voting fields*, and the *data structures* used for vote collection. Table 1 shows all the geometric features that appear in a 4-D space and their representation as *elementary* 4-D tensors, where *n* and *t* represent normal and tangent vectors, respectively. Note that a surface in the 4-D space can be characterized by two normal vectors, or by two tangent vectors. From a *generic* 4-D tensor that results after voting, the geometric features are extracted as shown in Table 2.

The 4-D voting fields are obtained as follows. First the 4-D stick field is generated in a similar manner to the 2-D stick field (see Figure 2(a)). Then, the other three voting fields are built by integrating all the contributions obtained by rotating a 4-D stick field around appropriate

Feature	λ_1 λ_2 λ_3 λ_4	e_1 e_2 e_3 e_4	Tensor
point	1 1 1 1	Any basis	Ball
curve	1 1 1 0	n_1 n_2 n_3 t	C-Plate
surface	1 1 0 0	n_1 n_2 t_1 t_2	S-Plate
volume	1 0 0 0	n t_1 t_2 t_3	Stick

Table 1. Elementary tensors in 4-D

Feature	Saliency	Normals	Tangents
point	λ_4	none	none
curve	λ_3 - λ_4	e_1 e_2 e_3	e_4
surface	λ_2 - λ_3	e_1 e_2	e_3 e_4
volume	λ_1 - λ_2	e_1	e_2 e_3 e_4

Table 2. A generic tensor in 4-D

axes. In particular, the 4-D ball field – the only one directly used here – is generated according to:

$$V_{ball}(\vec{d}) = \int_0^{2\pi}\int\int R\, V_{stick}(R^{-1}\vec{d})\, R^T \, d\theta_{xy} d\theta_{xu} d\theta_{xv} \qquad (3)$$

where x, y, u, v are the 4-D coordinates axes and R is the rotation matrix with angles θ_{xy}, θ_{xu}, θ_{xv}.

The data structure used to store the tensors is an *approximate nearest neighbor (ANN) k-d tree* [12]. The space complexity of is $O(n)$, where n is the input size. The average time complexity of the voting process is $O(\mu n)$ where μ is the average number of tokens in the neighborhood. Therefore, in contrast to other voting techniques, such as the Hough Transform, both time and space complexities of the Tensor Voting methodology are *independent* of the dimensionality of the desired feature.

3. Generating candidate matches

We take as input two image frames that involve general motion – that is, both the camera and the objects in the scene may be moving. For illustration purposes, we give a description of our approach by using a specific example – the two images in Figure 3 are taken with a handheld moving camera, where the candy box and the background exhibit distinct motions due to their different distances from the camera.

For every pixel in the first image, the goal at this stage is to produce candidate matches in the second image. We use a normalized cross-correlation procedure, where all peaks of correlation are retained as candidates. When a peak is found, its position is also adjusted for sub-pixel

Figure 3. Candy box – input images

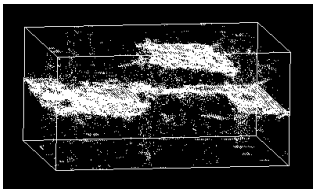

Figure 4. Matching candidates

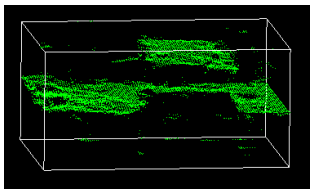

Figure 5. Selected velocities

precision according to the correlation values of its neighbors. Finally, each candidate match is represented as a (x,y,v_x,v_y) point in the 4-D space of image coordinates and pixel velocities, with respect to the first image.

Since we want to increase the likelihood of including the correct match among the candidates, we repeat this process at multiple scales, by using different correlation window sizes. Small windows have the advantage of capturing fine detail, and are effective close to the motion boundaries, but produce considerable noise in areas lacking texture or having small repetitive patterns. Larger windows generate smoother matches, but their performance degrades in large areas along motion boundaries. We have experimented with a large range of window sizes, and found that best results are obtained by using only two or three different sizes, that should include at least a very small one. Therefore, in all the examples described in this paper we used three correlation windows, with 3x3, 5x5 and 7x7 sizes.

The resulting candidates appear as a cloud of (x,y,v_x,v_y) points in the 4-D space. Figure 4 shows the candidate matches. In order to display 4-D data, the last component of each 4-D point has been dropped – the 3 dimensions shown are x and y (in the horizontal plane), and v_x (the height). The motion layers can be already perceived as their tokens are grouped in two layers surrounded by noisy matches.

Extracting statistically salient structures from such noisy data is very difficult for most existing methods. Because our voting framework is robust to considerable amounts of noise, we can afford using the multiple window sizes in order to extract the motion layers.

4. Extraction of motion layers in 4-D

Selection. Since no information is initially known, each potential match is encoded into a 4-D *ball tensor*.

Then each token casts votes by using the 4-D *ball voting field*. During voting there is strong support between tokens that lie on a smooth surface (layer), while communication between layers is reduced by the spatial separation in the 4-D space of both image coordinates and pixel velocities.

For each pixel (x,y) we retain the candidate match with the highest surface saliency $(\lambda_2-\lambda_3)$, and we reject the others as outliers. Figure 5 shows a 3-D view of the recovered matches (the height represents v_x).

Orientation refinement. In order to obtain an estimation of the layer orientations as accurate as possible, we perform an orientation refinement through another voting process, but now with the selected matches only. After voting, the normals to layers are found at each token as e_1 and e_2.

Outlier rejection. In the selection step, we kept only the most salient candidate at each pixel. However, there are pixels where all candidates are wrong, such as in areas lacking texture. Therefore now we eliminate all tokens that have received very little support. Typically we reject all tokens with surface saliency less that 10% of the average saliency of the entire set.

Densification. Since the previous step created holes (i.e., pixels where no velocity is available), we must infer them from the neighbors by using a smoothness constraint. For each pixel (x,y) without an assigned velocity we try to find the best (v_x,v_y) location at which to place a newly generated token. The candidates considered are all the discrete points (v_x,v_y) between the minimum and maximum velocities in the set, within a neighborhood of the (x,y) point. At each candidate position (x,y,v_x,v_y) we accumulate votes, according to the same Tensor Voting framework that we have used so far. After voting, the candidate token with maximal surface saliency $(\lambda_2-\lambda_3)$ is retained, and its (v_x,v_y) coordinate represent the most likely velocity at (x,y). By following this procedure at every (x,y) image location we generate a *dense velocity field*. Note that in this process, along with velocities we simultaneously infer layer orientations. A 3-D view of the dense layers is shown in Figure 6.

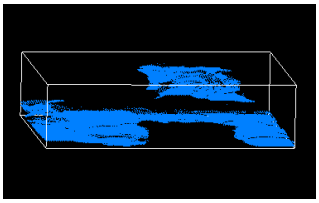

Figure 6. Dense layers

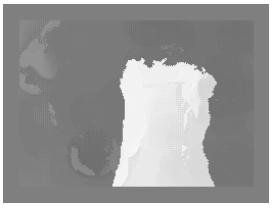

Figure 7. Layer velocities

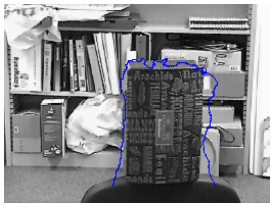

Figure 8. Layer boundaries

Segmentation. The next step is to group tokens into *regions*, by using again the smoothness constraint. We start from an arbitrary point in the image, assign a region label to it, and try to recursively propagate this label to all its image neighbors. In order to decide whether the label must be propagated, we use the smoothness of both velocity and layer orientation as a grouping criterion.

Figure 7 illustrates the recovered v_x velocities within layers (dark corresponds to low velocity), and Figure 8 shows the layer boundaries superimposed over the first image.

5. Boundary inference in 2-D

At this stage, the extracted motion layers can still be over or under-extended along the motion boundaries. This situation typically occurs in areas subject to occlusion, where the initial correlation procedure may generate wrong matches that are consistent with the correct ones, and therefore could not be rejected as outlier noise.

However, now we know how many moving objects are present in the scene and where they are. The margins of the layers give us a good estimate for the position and overall orientation of the true motion boundaries. We combine this knowledge with monocular cues (intensity edges) from the original images in order to build a boundary saliency map along the layers margins. Next we enforce the smoothness and continuity of the boundary through a 2-D voting process, and extract the true boundary as the most salient curve within the map.

This procedure is performed in two successive passes – by separately using the horizontal and vertical components of the image gradient. In the first pass, vertical edges are favored, but in fact all boundaries are found, with the exception of the ones almost horizontal. The second pass favors the horizontal edges, so that it detects the remaining ones.

5.1. The boundary saliency map

In the first pass, we start by finding the points that belong to the layer boundaries, identified by changes in region labels along horizontal lines. For each such point (x_c,y_c) we define a horizontal *zone of boundary uncertainty*, centered at (x_c,y_c). Since the over or under-extension of motion layers is usually within the limits of

the correlation window size, we chose the largest size used in correlation as the zone width. The zone height is one pixel.

Next we make use of the monocular cues by computing the image gradient (from the intensity I in first image) at each location within the zones of boundary uncertainty:

$$G_x(x,y) = I(x,y) - I(x-1,y)$$
$$G_y(x,y) = I(x,y) - I(x,y-1) \quad (4)$$

Since at this pass we are favoring vertical edges, we initialize our saliency map with the horizontal component of the gradient:

$$sal = |G_x(x,y)| \quad (5)$$

This choice is made in order not to be influenced in the analysis by purely horizontal edges, which will be detected during the second pass. Diagonal edges that exhibit a significant horizontal gradient contribute to the saliency map and they are detected in the first pass.

Finally, we incorporate our estimation of the boundary position and orientation, as resulted from motion cues, by introducing a bias towards the current layer boundaries. Within each zone, we define a weight function W that is 1 at x_c and decays exponentially by:

$$W = e^{\frac{(x-x_c)^2}{\sigma_W^2}} \quad (6)$$

where σ_W corresponds to a weight of 0.2 at the zone extremities.

The saliency map is then updated by multiplying each existing value with the corresponding weight.

5.2. Detecting the boundary

At this stage we have a saliency value and an orientation at each location within the zones of uncertainty. However, in order to extract the boundaries we need to examine how neighboring locations agree upon their information, through a voting process.

We proceed by encapsulating all the existing information within a 2-D tensorial framework. Since we have boundary orientations, at each location in the uncertainty zones we create a 2-D stick tensor, with the orientation (eigenvectors e_1 and e_2) given by the image gradient:

$e_1 = (G_x\ G_y)$ (normal to edge)
$e_2 = (-G_y\ G_x)$ (tangent to edge)

and the size taken from the saliency map:

$\lambda_1 = sal$
$\lambda_2 = 0$

The tensors then communicate through a 2-D voting process, where each tensor is influenced by the saliency and orientation of its neighbors. After voting, the curve saliency values are collected at each tensor as $(\lambda_1-\lambda_2)$ and stored back in the saliency map. Figure 9 shows the tensors after voting, with the local curve tangent given by the eigenvector e_2. The curve saliency $(\lambda_1-\lambda_2)$ is illustrated here as the length of the tangent vector, shown in red. Note that although strong texture edges are present in the uncertainty zone, after voting their saliency is diminished by the overall dominance of saliency and orientation of the correct object edges.

Figure 9. Boundary saliency map

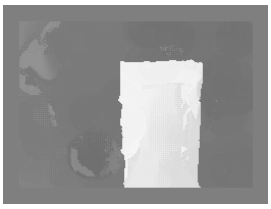

Figure 10. Refined velocities

Figure 11. Refined boundaries

The true boundaries are extracted by choosing seeds with maximum curve saliency, and growing the boundary from an uncertainty zone to the next, according to the local curve saliency and orientation.

After marking the detected boundaries, the entire process is repeated in a similar fashion, this time favoring horizontal edges, in order to detect any horizontal boundaries that have been missed during the first pass.

Finally, each zone of boundary uncertainty is revisited in order to reassign pixels to regions, according to the new boundaries. In addition to changing the region label, their velocities are recomputed in a 4-D voting process similar to the one used for densification. However, since region labels are now available, the votes are collected only from points within the same layer.

Figure 10 shows the refined velocities within layers (dark represents small velocity), and Figure 11 shows the refined motion boundary, that indeed corresponds to the actual object.

6. Results

We have also analyzed several other image sequences, and we present here the results obtained. In all

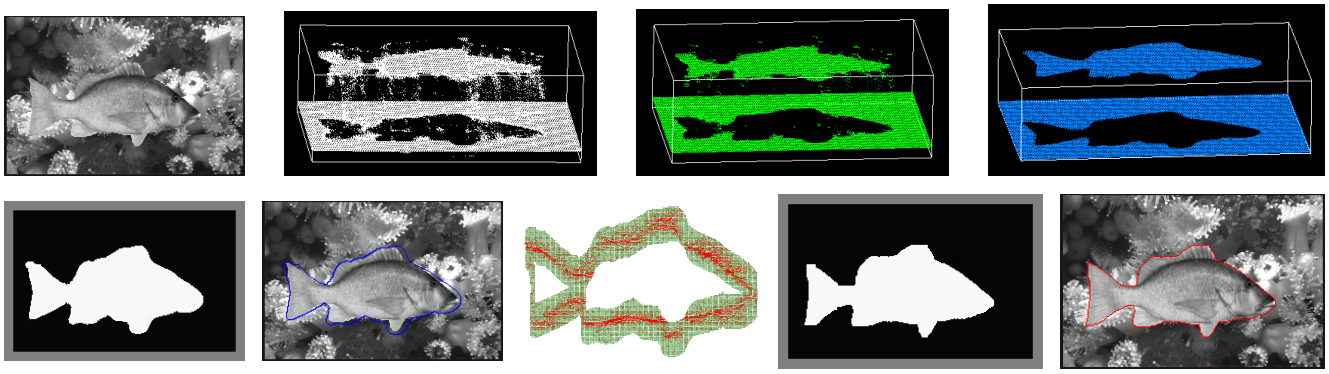

Figure 12. Fish sequence

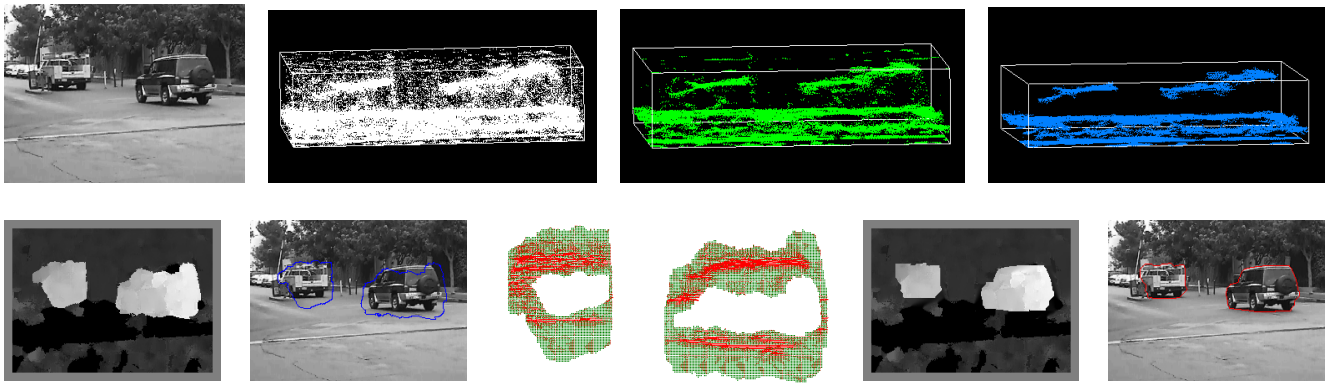

Figure 13. Barrier sequence

experiments we used three correlation windows, with 3x3, 5x5 and 7x7 sizes, and for each window we retained all peaks of correlation. Therefore each pixel in the image had at least 3 candidate matches, among which at most one was correct. For both the 4-D and 2-D voting processes, in all examples we used the same scale factor, corresponding to an image neighborhood with a radius of 16 pixels.

Fish sequence (Figure 12). To quantitatively estimate the performance of our approach we created a synthetic sequence from real images. The silhouette of a fish was cropped from its image and pasted at different locations over a second image, in order to generate a motion sequence with ground truth. The average angular error we obtained is 0.42° ± 1.2° for 100% field coverage, which is very low despite the multitude of texture edges from the cluttered background, that were competing with the true object edges. This example is also used to show that we can successfully handle more detailed and non-convex motion boundaries.

Barrier sequence (Figure 13). We analyzed the motion from two frames of a sequence showing two cars moving away from the camera. The analysis is difficult due to the large ground area with very low texture, and because the two moving objects have relatively small sizes in the image. Also note that the *image* motion is not translational – the front of each car has a lower velocity than its back. This is visible in the 3-D view of the motion layers, which appear as tilted surfaces. In fact, our framework does not make any assumption regarding the type of motion – such as translational, planar, or rigid motion. The *only* criterion used is the smoothness of *image* motion.

7. Conclusions

We have presented a novel approach for the problem of motion segmentation, by integrating motion and monocular cues into a Tensor Voting computational framework. Our methodology for extracting motion layers is based on a *layered 4-D representation* of data, and a *voting scheme* for token communication. Monocular cues represented by intensity edges in the original images are used to refine the extracted layers, through a *2-D voting process*. The *boundary saliency map* proves to be an appropriate representation for this problem. It encodes the position, orientation and strength of both the layer boundaries and image edges, all contributing to accurate inference of the motion boundaries.

Despite the high dimensionality (in the 4-D case), our voting scheme is both time and space efficient. It is non-iterative and the only free parameter is scale, which is an inherent characteristic of human vision.

We plan to extend our approach by incorporating information from multiple frames, and to study the possibility of using an adaptive scale of analysis in the voting process.

References

[1] G. Medioni, Mi-Suen Lee, Chi-Keung Tang, "A Computational Framework for Segmentation and Grouping", *Elsevier Science*, 2000.

[2] M. Nicolescu, G. Medioni, "Perceptual Grouping from Motion Cues Using Tensor Voting in 4-D", *ECCV*, 2002, vol. 3, pp. 423-437.

[3] J. Barron, D. Fleet, S. Beauchemin, "Performance of Optical Flow Techniques", *IJCV*, 1994, 12:1, pp. 43-77.

[4] H. Nagel, W. Enkelmann, "An Investigation of Smoothness Constraints for the Estimation of Displacement Vector Fields from Image Sequences", *PAMI*, 1986, vol. 8, pp. 565-593.

[5] A. Singh, "Optical Flow Computation: A Unified Perspective", *IEEE Computer Society Press*, 1992.

[6] D. Heeger, "Optical Flow using Spatiotemporal Filters", *IJCV*, 1988, vol. 1, pp. 279-302.

[7] F. Heitz, P. Bouthemy, "Multimodal Estimation of Discontinuous Optical Flow Using Markov Random Fields", *PAMI*, December 1993, 15: 12, pp. 1217-1232.

[8] S. Ghosal, "A Fast Scalable Algorithm for Discontinuous Optical Flow Estimation", *PAMI*, 1996, 18:2, pp. 181-194.

[9] S. Hsu, P. Anandan, S. Peleg, "Accurate Computation of Optical Flow by Using Layered Motion Representations", *ICPR*, 1994, pp. 743-746.

[10] Y. Weiss, "Smoothness in Layers: Motion Segmentation Using Nonparametric Mixture Estimation", *CVPR*, 1997, pp. 520-526.

[11] J. Shi, J. Malik, "Motion Segmentation and Tracking Using Normalized Cuts", *ICCV*, 1998, pp. 1154-1160.

[12] S. Arya, D. Mount, N. Netanyahu, R Silverman, A. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions", *Journal of the ACM*, 1998, 45:6, pp. 891-923.