Proceedings of the European Conference on Computer Vision, vol. III, pages 423-437, Copenhagen, Denmark, May 2002.

Perceptual Grouping from Motion Cues Using Tensor Voting in 4-D

Mircea Nicolescu and Gérard Medioni

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

Abstract. We present a novel approach for motion grouping from two frames, that recovers the dense velocity field, motion boundaries and regions, based on a 4-D Tensor Voting computational framework. Given two sparse sets of point tokens, we encode the image position and potential velocity for each token into a 4-D tensor. The voting process then enforces the motion smoothness while preserving motion discontinuities, thus selecting the correct velocity for each input point, as the most salient token. By performing an additional dense voting step we infer velocities at every pixel location, motion boundaries and regions. Using a 4-D space for this Tensor Voting approach is essential, since it allows for a spatial separation of the points according to both their velocities and image coordinates. Unlike other methods that optimize a specific objective function, our approach does not involve initialization or search in a parametric space, and therefore does not suffer from local optima or poor convergence problems. We demonstrate our method with synthetic and real images, by analyzing several difficult cases – opaque and transparent motion, rigid and non-rigid motion, curves and surfaces in motion.

1 Introduction

A traditional formulation of the motion analysis problem is the following: given two or more image frames, the goal is to determine three types of information – a *dense velocity field*, *motion boundaries*, and *regions*. Computationally, the problem can be decomposed in two processes - *matching* and *motion capture*. The *matching* process identifies the elements (tokens) in successive views that represent the same physical object, thus producing a (possibly sparse) velocity field. The *motion capture* process infers velocity vectors at every image location, thus producing a dense velocity field, and groups tokens into regions separated by motion boundaries.

Here we focus on the problem of matching and motion capture from sparse sets of point tokens in two frames. Two examples of such input are shown in Fig. 1 and Fig. 2. If the frames in each pair are presented in a properly timed succession, a certain motion of image regions is perceived from one frame to the other. However, while in one case the regions can be detected even without motion, only from monocular cues (here, different densities of points), in the other case no monocular information is available. This example shows that analysis is possible even from motion cues *only*.

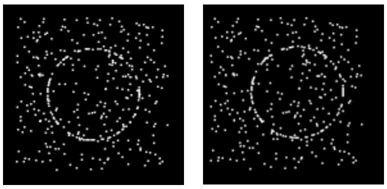


Fig. 1. Translating circle

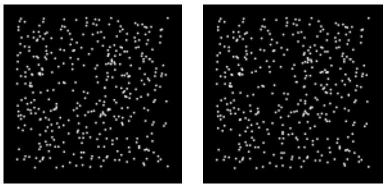


Fig. 2. Translating disk

Another interesting aspect is the fact that the human vision system not only establishes point correspondences, but also perceives *regions* in motion, although the input consists of sparse points only. This demonstrates that both processes of matching and motion capture are involved in motion analysis.

Ullman presents an excellent analysis of the correspondence problem, from both a psychological and a computational perspective [1]. Here we are following his conclusion, that correspondence formation is a low-level process which expresses mutual token affinities, and takes place prior to any 3-D interpretation. Tokens involved in matching are non-complex elements, such as points, blobs, edge and line fragments.

Optical flow techniques [2] – such as differential methods, region-based matching, energy-based or phase-based methods – 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 [3]. Another research direction uses regularization techniques, which preserve discontinuities by weakening the smoothing in areas that exhibit strong intensity gradients [4][5].

Significant improvements have been achieved more recently by using layered representations [6][7]. 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.

Establishing an association between input tokens and layers is still difficult. Some methods perform an iterative fitting of data to parametric models [8][9][10]. The difficulties involved in this estimation process range from a severe restriction in motion representation (as rigid or planar), to over-fitting and instability due to high-order parameterizations.

Other approaches employ the rigidity constraint [1] to solve the correspondence problem, but besides the inability to handle non-rigid motions, such methods require three or more frames, while we process input from two frames only.

Little et al. [11] developed a parallel algorithm for computing the optical flow by using a local voting scheme based on similarity of planar patches. However, their methodology cannot prevent motion boundary blurring due to over-smoothing and is restricted to short-range motion only.

From a computational point of view, one of the most powerful and most often used constraints is the smoothness of motion. Usually, previous techniques encounter traditional difficulties in image regions where motion is *not* smooth (i.e., around motion boundaries). To compute the velocity field, knowledge of the boundaries is required so that the smoothness constraint can be relaxed around the discontinuities. But the boundaries cannot be computed without first having determined the pixel velocities. This "chicken-and-egg" problem has lead to numerous inconsistent methods, with ad-hoc criteria introduced to account for motion discontinuities.

A computational framework that successfully enforces the smoothness constraint in a unified manner, while preserving smoothness discontinuities is Tensor Voting [12]. This approach also benefits from the fact that it is non-iterative, it does not depend on critical thresholds, does not involve initialization or search in a parametric space, and therefore it does not suffer from local optima and poor convergence problems. The first to propose using Tensor Voting to determine the velocity field were Gaucher and Medioni [13]. They employ successive steps of voting, first to determine the boundary points as the tokens with maximal motion uncertainty, and then to locally refine velocities near the boundaries by allowing communication only between tokens placed on the same side of the boundary. However, in their approach the voting communication between tokens is essentially a 2-D process that does not inhibit neighboring elements with different velocities from influencing each other.

In this paper we propose a novel approach based on a *layered 4-D representation* of data, and a *voting scheme* for token communication. Our methodology is formulated as a 4-D Tensor Voting computational framework. The position (x,y) and potential velocity (v_x,v_y) of each token are encoded as a 4-D tuple. By letting the tokens propagate their information through voting, distinct moving regions emerge as *smooth surface layers* in this 4-D space of image coordinates and pixel velocities.

In the next section we examine the voting framework by first giving an overview of the Tensor Voting formalism, then we discuss how the voting concepts are generalized and extended to the 4-D case. In Section 3 we present our approach and show how this formalism is applied to the problem of matching and motion capture. In Section 4 we present our experimental results, while Section 5 summarizes our contribution and provides further research directions.

2 Voting Framework

2.1 Tensor Voting Overview

The use of a voting process for feature inference from sparse and noisy data was introduced by Guy and Medioni [14] and then formalized into a unified tensor framework [12]. This methodology is non-iterative and robust to considerable amounts of outlier noise. The only free parameter is the scale of analysis, which is indeed an inherent property of visual perception. The input data is encoded as tensors, then support information (including proximity and smoothness of continuity) is propagated by convolution-like voting within a neighborhood.

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, which is geometrically equivalent to an ellipse. This ellipse can be described by a 2×2 eigensystem, where the eigenvectors e_1 and e_2 give the ellipse orientation and the eigenvalues λ_1 and λ_2 $(\lambda_1 \geq \lambda_2)$ represent the ellipse size. The tensor is internally represented as a matrix S:

$$S = \lambda_1 \cdot e_1 e_1^T + \lambda_2 \cdot e_2 e_2^T \tag{1}$$

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 token that represents a 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 on a single parameter – the scale factor σ. Vote orientation corresponds to the best (smoothest) possible local curve continuation from the voter to the recipient, while vote strength $VS(\vec{d})$ decays with distance $|\vec{d}|$ from voter to recipient, and with curvature ρ:

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

Fig. 3 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. The curve normal $\vec{N}$ thus obtained needs to be propagated by the vote from P to Q. The vote $V_{stick}(\vec{d})$ received from P at Q is encoded as a tensor according to (3), where $\vec{d} = Q - P$.

$$V_{stick}(\vec{d}) = VS(\vec{d}) \cdot \vec{N}\vec{N}^T \tag{3}$$

Note that vote strength at both Q’ and Q” is smaller than at Q – because Q’ is farther, and Q” requires a higher curvature than Q. Fig. 4 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, according to equation (4). The corresponding 2-D ball field is shown in Fig. 5.

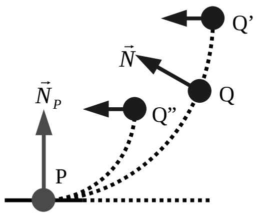


Fig. 3. Vote generation

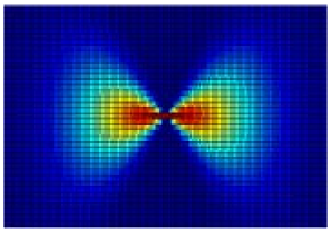


Fig. 4. 2-D stick field

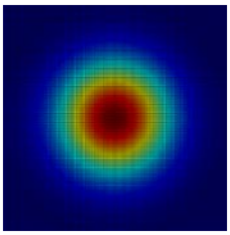


Fig. 5. 2-D ball field

$$V_{ball}(\vec{d}) = \int_0^{2\pi} R_\theta \, V_{stick}(R_\theta^{-1}\vec{d}) \, R_\theta^T \, d\theta \tag{4}$$

At each receiving site, the collected votes are combined through simple tensor addition (sum of matrices $V(\vec{d})$), thus 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 such tensor encodes the local orientation of geometric features such as curves (given by the tensor orientation), and confidence of this knowledge (also called saliency, given by the tensor shape and size). For a general tensor, its curve saliency is given by (λ_1-λ_2) and the curve normal orientation by e_1, while its point saliency is given by λ_2. After voting, each resulting tensor S in general form is decomposed into a stick and a ball component, each being weighted by the corresponding saliency:

$$S = (\lambda_1 - \lambda_2) e_1 e_1^T + \lambda_2 (e_1 e_1^T + e_2 e_2^T) \tag{5}$$

Therefore, the voting process infers curves and junctions (points) simultaneously, while also identifying outlier noise (tokens that receive very little support).

In the 3-D case, the salient features are points, curves and surfaces. A point element corresponds to a *ball tensor,* with λ_1=λ_2=λ_3=1 and no preferred orientation, a curve element is represented by a *plate tensor,* where two eigenvalues co-dominate (λ_1=1, λ_2=1, λ_3=0) and the eigenvector e_3 gives the curve tangent, and a surface element is represented by a *stick tensor*, where one eigenvalue dominates (λ_1=1, λ_2=0, λ_3=0), the surface normal is given by e_1, and e_2 and e_3 are surface tangents.

2.2 Tensor Voting in 4-D

The Tensor Voting framework is general enough to be extended to any dimension readily, except for some implementation changes, mainly for efficiency purposes [15]. The issues to be addressed are the *tensorial representation* of the features in the desired 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

Table 1. Elementary tensors in 4-D

Feature	λ_1 λ_2 λ_3 λ_4	e_1 e_2 e_3 e_4	Tensor
point	1 1 1 1	Any orthonormal 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 2. A generic tensor 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

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 can be extracted as shown in Table 2.

The voting fields are a key part of the formalism – they are responsible of the size and shape of the neighborhood where the votes are cast, and also control how the votes depend on distance and orientation. 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, as it was explained in Section 2.1 and illustrated in Fig. 3. Then, the other three voting fields are built by integrating all the contributions obtained by rotating a 4-D stick field around appropriate axes. In particular, the 4-D ball field – the only one directly used here – is generated according to:

$$V_{ball}(\vec{d}) = \int\int\limits_{0}^{2\pi}\int R_{\theta_{xy}\theta_{xu}\theta_{xv}}\, V_{stick}(R^{-1}_{\theta_{xy}\theta_{xu}\theta_{xv}}\vec{d})\, R^{T}_{\theta_{xy}\theta_{xu}\theta_{xv}}\; d\theta_{xy}d\theta_{xu}d\theta_{xv} \qquad \textbf{(6)}$$

where *x*, *y*, *u*, *v* are the 4-D coordinates axes, θ_{xy}, θ_{xu}, θ_{xv} are rotation angles in the specified planes, and the stick field corresponds to the orientation (1 0 0 0).

In the 2-D or 3-D case, the data structure used to store the tensors during vote collection was a simple 2-D grid or a red-black tree. Because we need a data structure that is gracefully scalable to higher dimensions, the solution used in our approach is an *approximate nearest neighbor (ANN) k-d tree* [16].

Since we use efficient data structures to store the tensors, the space complexity of the algorithm 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. The running time for an input of size 700 is about 20 seconds on a Pentium III (600 MHz) processor.

3 Our Approach

The main difficulties in visual motion analysis appear at motion boundaries, where velocity estimates are very poor. This happens because the problem is typically cast as a two-dimensional process. As a result, along boundaries tokens have a strong mutual affinity because they are close in the image, despite the fact that they may belong to different regions, with different velocities. Accordingly, we believe that the desirable representation should be based on a layered description, where regions in motion are represented as smooth and possibly overlapping layers.

In any method that seeks to solve the motion analysis problem, each token is characterized by four attributes – its image coordinates (x,y) and its velocity with the components (v_x,v_y). We encapsulate each token into a (x,y,v_x,v_y) tuple in the 4-D space, this being a natural way of expressing the spatial separation of tokens according to *both* velocities and image coordinates. It is especially helpful for eliminating the problem of uncertainty along motion boundaries, where although tokens are close in image space, their interaction is now inhibited by their separation in velocity space. In general, there may be several candidate velocities for each point (x,y), so each tuple (x,y,v_x,v_y) represents a (possibly wrong) candidate match.

Both matching and motion capture are based on a process of communicating the affinity between tokens. In our representation, this affinity is expressed as the token preference for being incorporated into a *smooth surface layer* in the 4-D space. A necessary condition is to enforce strong support between tokens in the same layer, and weak support across layers, or at isolated tokens.

A suitable computational framework that enforces the smoothness constraint while preserving discontinuities is Tensor Voting, here performed in the 4-D space. The affinities between tokens are embedded in the concept of surface saliency exhibited by the data. By letting the tokens propagate their information through voting, wrong matches are eliminated as they receive little support, and layers are extracted as salient smooth surfaces. Essentially, the matching problem is expressed as an *outlier rejection* process, while motion capture is performed mainly as a *layer densification* process.

We demonstrate the contribution of this work by addressing the problems of matching and motion capture. Given two sparse sets of point tokens, we first use 4-D voting to select the correct match for each input point, as the most salient token, thus producing a sparse velocity field. By using the same voting framework during the motion capture process, we then generate a dense layer representation in terms of motion boundaries and regions. We illustrate our method with both synthetic and real images, by analyzing several cases – opaque and transparent motion, rigid and non-rigid motion, curves and surfaces in motion.

3.1 Matching

We take as input two frames containing identical point tokens, in a sparse configuration. For illustration purposes, we give a step-by-step description of our approach by using a specific example – the point tokens represent an opaque **translating disk** (Fig. 2) against a static background. Later we also show how our method performs on several other examples.

Before proceeding, we need to make a brief comment on how we display the intermediate results (i.e. those in 4-D). In order to allow for a three-dimensional display, the last component of each 4-D point has been dropped, so that the 3 dimensions shown are image coordinates x and y (in the horizontal plane), and the v_x component of image velocity (the height).

We do not assume any a priori information about the scene. Candidate matches are generated as follows: in a pre-processing step, for each token in the first frame we simply create a potential match with every point in the second frame that is located within a neighborhood (whose size is given by the scale factor) of the first token. The resulted candidates appear as a cloud of (x,y,v_x,v_y) points in the 4-D space. In our translation example we have 400 input points, and by using the procedure described above we generate an average of 5.3 candidate matches per point, among which at most one is correct. Fig. 6(a) shows the candidate matches. Note that the correct matches can be already visually perceived as they are grouped in two parallel layers surrounded by noisy matches.

Since no information is initially known, each potential match is encoded into a 4-D *ball tensor* - the eigenvalues and eigenvectors are the following:

$$\begin{array}{ll} \lambda_1=1 & e_1 = (\,0\;0\;0\;1\,)^T \\ \lambda_2=1 & e_2 = (\,0\;0\;1\;0\,)^T \\ \lambda_3=1 & e_3 = (\,0\;1\;0\;0\,)^T \\ \lambda_4=1 & e_4 = (\,1\;0\;0\;0\,)^T \end{array}$$

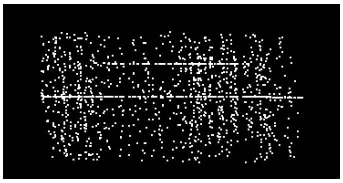
After encoding, each token casts votes in its neighborhood (given by the scale factor). This is a sparse voting process, in the sense that votes are cast only at input token locations. Votes are generated by using the 4-D *ball voting field*, where no particular orientation is preferred.

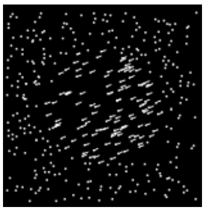
During voting there is strong support between tokens that happen to lie on a smooth surface (layer), while communication between layers is reduced by the spatial separation in the 4-D space of both image coordinates and pixel velocities. Wrong matches appear as isolated points, which receive little or no support. A measure of this support is given by the surface saliency.

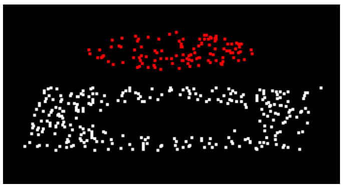
The next step is to eliminate wrong matches. For each group of tokens that have common (x,y) coordinates but different (v_x,v_y) velocities, we retain the token with the strongest surface saliency (that is, with the maximum value for λ_2-λ_3), while rejecting the others as outliers. For the translating disk example, a comparison with the ground truth shows that matching was 100% accurate - all 400 matches have been recovered correctly, despite the large amount of approximately 500% noise present. Fig. 6(b) shows the recovered sparse velocity field, while Fig. 6(c) shows a 3-D view of the recovered matches, where the height represents the v_x velocity component.

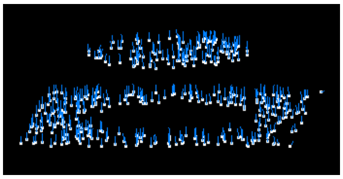
3.2 Motion Capture

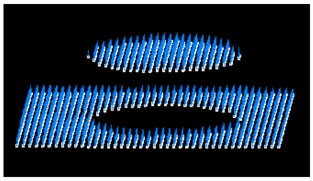
We start with a sparse velocity field described by (x,y,v_x,v_y) tuples in the 4-D space, that has been produced by the matching process. In the first stage we need to obtain an estimation of the layer orientations as accurate as possible. Although local layer orientations have been already determined as a by-product during the matching process (after voting, the eigenvectors e_1 and e_2 represent the normals to layers), they may have been corrupted by the presence of wrong correspondences.

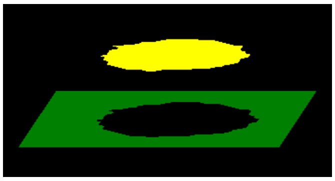

(a) Candidate matches


(b) Sparse velocity field


(c) Recovered v_x velocities


(d) Refined velocity field (sparse)


(e) Dense velocity field


(f) Regions

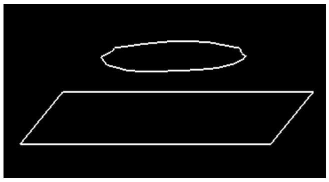

(g) Boundaries

Fig. 6. Translating disk

Therefore, we perform here an orientation refinement through another sparse voting process, but this time with the correct matches only. To this purpose, every 4-D tuple is again encoded as a *ball tensor*. After voting, the desired orientations – as normals to layers – are found at each token as the first two eigenvectors e_1 and e_2. We remind the reader that a surface in 4-D is characterized by *two* normal vectors. In Fig. 6(d) we show a 3–D view of the tokens with refined layer orientations (only one of the normals is shown at each token).

In order to attain the very goal of the motion capture problem – that is, to recover boundaries and regions as *continuous* curves and surfaces, respectively – it is necessary to first infer velocities and layer orientations at *every* image location.

The previously developed Tensor Voting framework allows for a densification procedure that extracts surfaces or curves from sparse data. However, since the algorithm is based on a marching process that grows the surface around a seed, if the surfaces are not closed they will be *over-extended*. This happens because the growing process stops only when saliency drops below a certain level, due to the decay with distance from supporting tokens.

Since in our case it is crucial to obtain accurate motion boundaries, we devised a different densification scheme. The key fact is that our 4-D space is not isotropic – we need to obtain a tensor value at every (x y) image location, but certainly not at every (v_x,v_y) location in velocity space.

In our approach, for each discrete point (x y) in the image 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 sparse tokens set, within a neighborhood of the $(x\ y)$ point. At each candidate position (x,y,v_x,v_y) we accumulate votes from the sparse tokens, according to the same Tensor Voting framework that we have used so far. The candidate token with the largest surface saliency after voting is retained, and its (v_x,v_y) coordinates represent the best 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, given by eigenvectors e_1 and e_2. In Fig. 6(e) we show a 3-D view of the dense set of tokens and their associated layer orientations.

The next step is to group tokens into *regions* that correspond to distinct moving objects, 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. Having both pieces of information available is especially helpful in situations where neighboring pixels have very similar velocities, and yet they must belong to different regions. Most methods that are based only on velocity discontinuities would fail on these cases. We will show such an example later. After assigning region labels to every token, for illustration purposes we perform a triangularization of each of the regions detected. The resulting surfaces are presented in Fig. 6(f).

Finally, we have implemented a method to extract the *motion boundary* for each region as a "partially convex hull". The process is controlled by only one parameter – the scale factor – that determines the perceived level of detail (that is, the departure from the actual convex hull). The resulting boundary curves are shown in Fig. 6(g).

4 Results

The case illustrated so far may be considered too simple since the only motion involved is translation. However, no assumption – such as translational, planar, or rigid motion – has been made. The *only* criterion used is the smoothness of *image* motion. To support this argument, we show next that our approach also performs very well for several other configurations.

Expanding disk (Fig. 7). The input consists of two sets of 400 point tokens each, representing an opaque disk in expansion against a static background. The average number of candidate matches per point is 6.1. Comparing the resulting matches with the true motion shows that only 1 match among 400 is wrong. This example demonstrates that, without special handling, our framework can easily accommodate non-rigid image motion.

Rotating disk – translating background (Fig. 8). The input consists of two sets of 400 point tokens each, representing an opaque rotating disk against a translating background. The average number of candidate matches per point is 5.8. After processing, only 2 matches among 400 are wrong. This is a very difficult case even for human vision, due to the fact that around the left extremity of the disk the two motions (of the disk and the background) are almost identical. In that part of the image there are points on different moving objects that are not separated, even in the

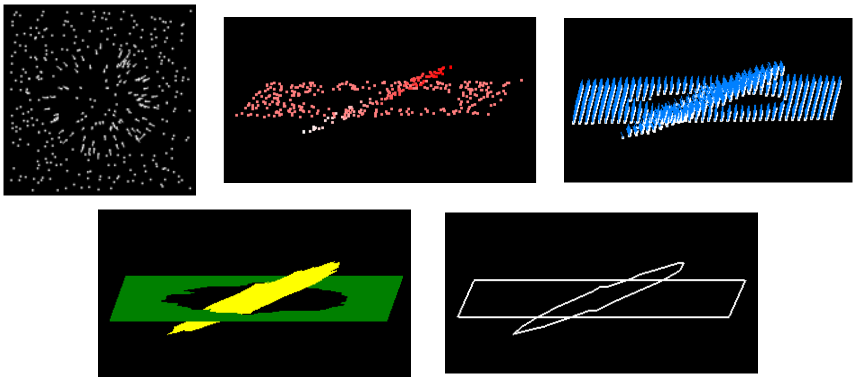


Fig. 7. Expanding disk

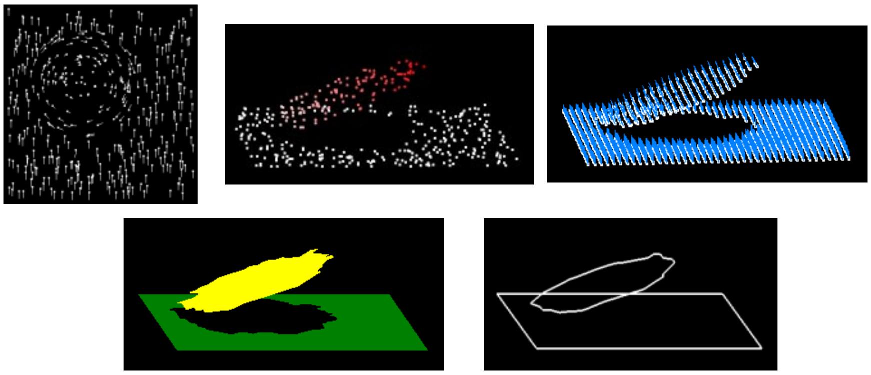


Fig. 8. Rotating disk – translating background

4-D space. In spite of this inherent ambiguity, our method is still able to accurately recover velocities, regions and boundaries. The key fact is that we rely not only on the 4-D positions, but also on the local layer orientations that are still different and therefore provide a good affinity measure.

Rotating ellipse (Fig. 9). The input consists of two sets of 100 point tokens each, representing a rotating ellipse. The average number of candidate matches per point is 5.9. After processing, all 100 matches have been correctly recovered. Many methods would fail on this example (used in the literature to illustrate the aperture effect) – one clear difficulty is that at the points where the rotated ellipse "intersects" the original one the velocity could be wrongly estimated as zero.

Transparent motion (Fig. 10). The input consists of two sets of 500 point tokens each, representing a transparent disk in translation against a static background. The average number of candidate matches per point is 8.9. After processing, all 100 matches have been correctly recovered. This example is extremely relevant to illustrate the power of our approach. If the analysis had been performed in a two-dimensional space, it would have failed because the two motion layers are superimposed in 2-D. In our framework, using the 4-D space provides a very natural separation between layers, separation that is consistent with human perception. For

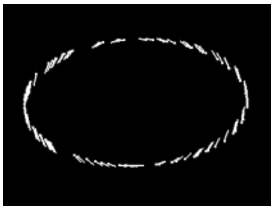


Fig. 9. Rotating ellipse

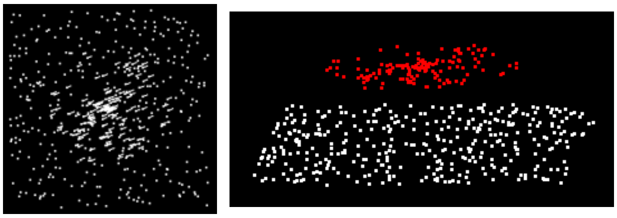


Fig. 10. Transparent motion

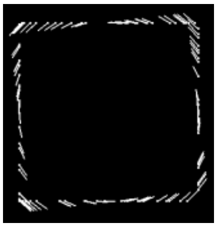


Fig. 11. Rotating square

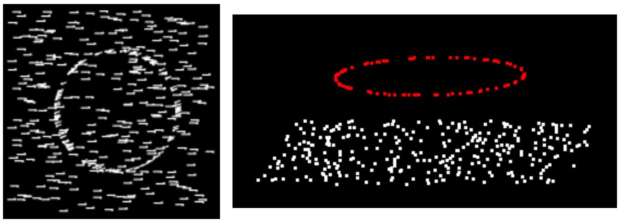


Fig. 12. Translating circle

our matching method, the transparent motion does not appear as a special case and it does not create any more difficulties than the opaque motion.

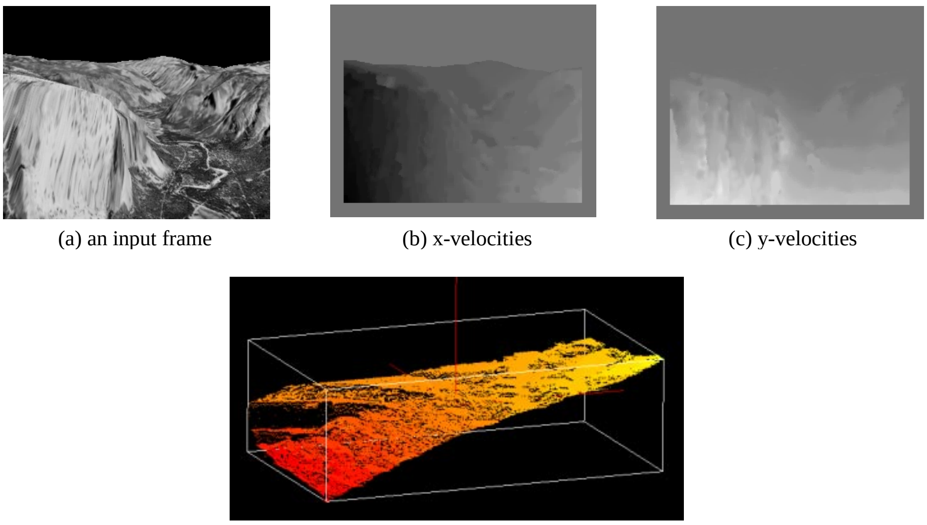
Rotating square (Fig. 11). The input consists of two sets of 100 point tokens each, representing a rotating square. The average number of candidate matches per point is 5.7. After processing, all 100 matches have been correctly recovered. This example is similar to the rotating ellipse and shows that the presence of non-smooth curves does not produce additional difficulty for our methodology.

Translating circle (Fig. 12). The input consists of two sets of 400 point tokens each, representing a translating circle against a static background. The average number of candidate matches per point is 6. After processing, all 100 matches have been correctly recovered. This example shows that we can successfully handle both curves and surfaces in motion.

So far we have presented only cases where no monocular information (such as intensity) is available, and the entire analysis has been performed based on motion cues *only*. Human vision is able to handle these cases remarkably well, and their study is fundamental for understanding the motion analysis process. Nevertheless they are very difficult from a computational perspective – most existing methods cannot handle such examples in a consistent and unified manner.

To further validate our approach we have also analyzed several standard image sequences, where both monocular and motion cues are available. In order to incorporate monocular information into our framework, we only needed to change the pre-processing step where candidate matches are generated. We ran a simple intensity-based cross-correlation procedure, and we retained all peaks of correlation as candidate matches. The rest of our framework remains unchanged.

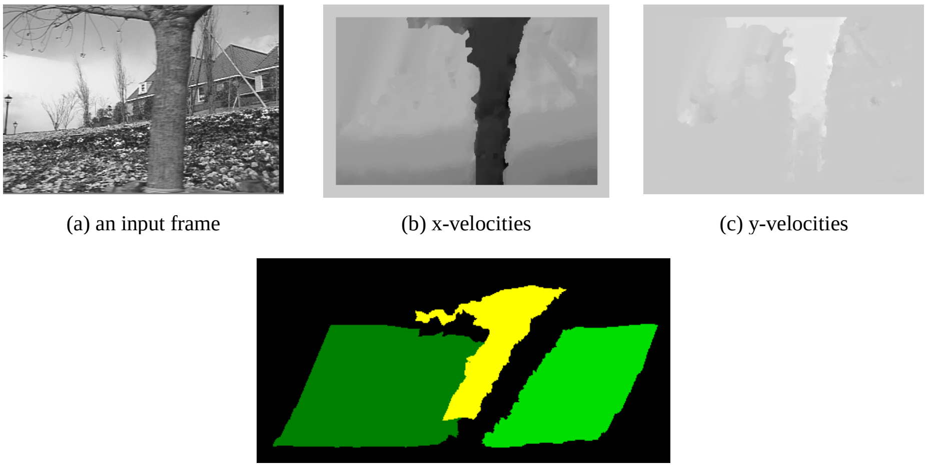
Yosemite sequence (Fig. 13). We analyzed the motion from two frames of the Yosemite sequence (without the sky) to quantitatively estimate the performance of our approach. The average angular error obtained is 3.74° ± 4.3° for 100% field coverage, result which is comparable with those in the literature [2]. This example also shows that our method successfully recovers non-planar motion layers.



(a) an input frame (b) x-velocities (c) y-velocities

(d) motion layer (x-velocities)

Fig. 13. Yosemite



(a) an input frame (b) x-velocities (c) y-velocities

(d) regions

Fig. 14. Flower Garden

Flower Garden sequence (Fig. 14). For a qualitative estimation, we also analyzed the motion from two frames of the Flower Garden sequence. It is worth mentioning that wrong candidates generated due to occlusion are corrected during the densification step.

Scale sensitivity. Since the only parameter involved in our voting framework is the scale factor that defines the voting fields, we analyzed how it influences the quality of the analysis. We ran our algorithm on the expanding disk example for a large range of

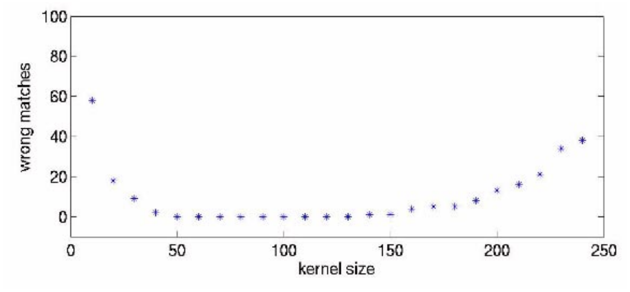


Fig. 15. Scale factor influence

scale values and we found that the method is remarkably robust to varying scale factors. Fig. 15 shows the number of wrong matches (for an input of 400 points) obtained for different values of the voting field size. Comparatively, the image size is 200 by 200. Note that when the field is too small, tokens do not communicate any more.

5 Conclusions and Future Work

We have presented a novel approach for the problem of perceptual grouping from motion cues, based on a *layered 4-D representation* of data, and a *voting scheme* for token communication. Our methodology is formulated as a 4-D Tensor Voting computational framework.

The moving regions are conceptually represented by smooth layers in the 4-D space of image coordinates and pixel velocities. Within this data representation, we employed a voting scheme for token affinity communication. Token affinities are expressed by their preference for being incorporated into smooth surfaces, as statistically salient features. Communication between sites is performed by tensor voting. From a possibly sparse input consisting of identical point tokens in two frames, without any a priori knowledge of the motion model we determine a dense representation in terms of accurate velocities, motion boundaries and regions, by enforcing the smoothness constraint while preserving motion discontinuities.

Using a 4-D space for our Tensor Voting approach is essential, since it allows for a spatial separation of the points according to both their velocities and image coordinates. Consequently, the proposed framework allows tokens from the same layer to strongly support each other, while inhibiting influence from other layers, or from isolated tokens.

Despite the high dimensionality, our voting scheme is both time and space efficient. Its complexity depends on the input size only. Our approach does not involve initialization or search in a parametric space, and therefore does not suffer from local optima or poor convergence problems. The only free parameter is scale, which is an inherent characteristic of human vision, and its setting is not critical.

We demonstrated the contributions of this work by analyzing several cases – opaque and transparent motion, rigid and non-rigid motion, curves and surfaces in motion. We showed that our method successfully addresses the difficult problem of motion analysis from motion cues *only*, and is also able to incorporate the use of monocular cues that are present in real images.

We plan to extend our approach for real image sequences by using a more elaborate procedure for generating the initial candidates, rather than a simple cross-correlation technique. Other research directions include studying the occlusion relationships and incorporating information from multiple frames.

Acknowledgements. This research has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152, and by National Science Foundation Grant 9811883**.**

References

[1] S. Ullman, “The Interpretation of Visual Motion”, *MIT Press*, 1979.
[2] J. Barron, D. Fleet, S. Beauchemin, “Performance of Optical Flow Techniques”, *IJCV*, 1994, 12:1, pp. 43-77.
[3] F. Heitz, P. Bouthemy, “Multimodal Estimation of Discontinuous Optical Flow Using Markov Random Fields”, *PAMI*, December 1993, 15: 12, pp. 1217-1232.
[4] S. Ghosal, “A Fast Scalable Algorithm for Discontinuous Optical Flow Estimation”, *PAMI*, 1996, 18:2, pp. 181-194.
[5] R. Deriche, P. Kornprobst, G. Aubert, “Optical Flow Estimation while Preserving its Discontinuities: A Variational Approach”, *ACCV*, 1995, pp. 290-295.
[6] S. Hsu, P. Anandan, S. Peleg, “Accurate Computation of Optical Flow by Using Layered Motion Representations”, *ICPR*, 1994, pp. 743-746.
[7] A. Jepson, M. Black, “Mixture Models for Optical Flow Computation”, *CVPR*, 1993, pp. 760-761.
[8] M. Irani, S. Peleg, “Image Sequence Enhancement Using Multiple Motions Analysis”, *CVPR*, 1992, pp. 216-221.
[9] J. Wang, E. Adelson, “Representing Moving Images with Layers”, *IEEE Trans. On Image Processing Special Issue: Image Sequence Compression*, 1994, 3:5, pp. 625-638.
[10] Y. Weiss, “Smoothness in Layers: Motion Segmentation Using Nonparametric Mixture Estimation”, *CVPR*, 1997, pp. 520-526.
[11] J. Little, H. Bulthoff, T. Poggio, “Parallel Optical Flow Using Local Voting”, *ICCV*, 1988, pp. 454-459.
[12] G. Medioni, M.-S. Lee, C.-K. Tang, “A Computational Framework for Segmentation and Grouping”, *Elsevier Science*, 2000.
[13] L. Gaucher, G. Medioni, “Accurate Motion Flow Estimation with Discontinuities”, *ICCV*, 1999, pp. 695-702.
[14] G. Guy, G. Medioni, “Inference of Surfaces, 3-D Curves and Junctions from Sparse, Noisy 3-D Data”, *IEEE Trans. PAMI*, 1997, 19: 11, pp. 1265-1277.
[15] C.-K. Tang, G. Medioni, M.-S. Lee, “Epipolar Geometry Estimation by Tensor Voting in 8D”, *ICCV*, 1999, pp. 502-509.
[16] 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.