Layered 4D Representation and Voting for Grouping from Motion

Mircea Nicolescu, *Student Member*, *IEEE Computer Society*, and Gérard Medioni, *Fellow*, *IEEE*

Abstract—We address the problem of perceptual grouping from motion cues by formulating it as a motion layers inference from a sparse and noisy point set in a 4D space. Our approach is based on a *layered 4D representation* of data, and a *voting scheme* for token communication, within a tensor voting computational framework. Given two sparse sets of point tokens, the image position and potential velocity of each token are encoded into a 4D tensor. By enforcing the smoothness of motion through a voting process, the correct velocity is selected for each input point as the most salient token. An additional dense voting step allows for the inference of a dense representation in terms of pixel velocities, motion regions, and boundaries. Using a 4D 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 most other methods that optimize certain objective functions, our approach is noniterative 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 nonrigid motion, curves and surfaces in motion.

Index Terms—Motion analysis, perceptual grouping, tensor voting.

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 entity, 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*.

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, that takes place prior to any 3D interpretation. Tokens involved in matching are noncomplex elements, such as points, blobs, edge, and line fragments. In our approach, we only study the case where the input consists of identical point tokens.

Barron et al. [2] provide a useful review of the computational methodologies used in the motion analysis field. Optical flow techniques—such as differential methods [3], [4], [5], [6], region-based matching [7], [8], [9], or energy-based methods [10]—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 [11], [12], [13], [14]. 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 [15], [16]. 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 [17], [18], [19], [20], [21], [22]. 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

• *The authors are with the Institute for Robotics and Intelligent Systems and with the Integrated Media Systems Center, University of Southern California, Powell Hall 204, 3737 Watt Way, Los Angeles, CA 90089-0273. E-mail: {mnicoles, medioni}@iris.usc.edu.*

Manuscript received 11 Dec. 2001; revised 1 July 2002; accepted 18 Aug. 2002.
Recommended for acceptance by D. Jacobs and M. Lindenbaum.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 117710.

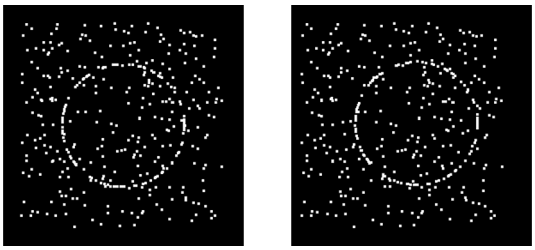


Fig. 1. Translating circle.

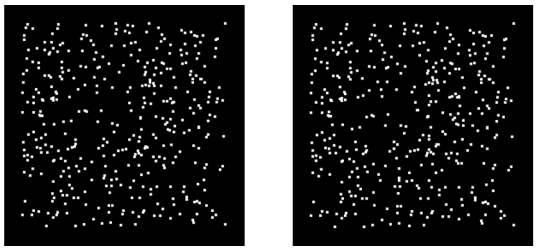


Fig. 2. Translating disk.

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. Some methods perform an iterative fitting of data to parametric models [23], [24]. 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.

An example of using basis set methods (in the form of steerable flow fields) is the work of Fleet et al. [25]. The results are good, but the use of a gradient descent solution is heavily dependent on initial conditions and parameters governing movement in the coefficient space.

Shi and Malik [26] 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.

Wu et al. [27] have applied wavelet techniques to the problem of optical flow determination. While their results are fairly adequate, motion discontinuities are modeled poorly due to oversmoothing. The presence of iteration in finding the solution also leaves open the possibility of instability.

Little et al. [28] 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 oversmoothing 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 discontinuities is tensor voting [29]. This approach also benefits from the fact that it is noniterative and it does not depend on critical thresholds. The first to propose using tensor voting to determine the velocity field were Gaucher and Medioni [30]. 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 2D 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 4D representation* of data and a *voting scheme* for token communication. Our methodology is formulated as a 4D tensor voting computational framework. The position $(x\ y)$ and potential velocity $(v_x\ v_y)$ of each token are encoded as a 4D tuple. By letting the tokens propagate their information through voting, distinct moving regions emerge as *smooth surface layers* in this 4D 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 4D case. In Section 3, we present our approach for the problem of matching and motion capture. Section 4 describes 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 [31] and then formalized into a unified tensor framework [29]. This methodology is noniterative 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 voting within a neighborhood.

In the 2D case, the salient features to be extracted are points and curves. Each token is encoded as a second order symmetric 2D tensor, which is geometrically equivalent to an ellipse. It is described by a 2×2 eigensystem, where the eigenvectors e_1 and e_2 give the ellipse orientation and the

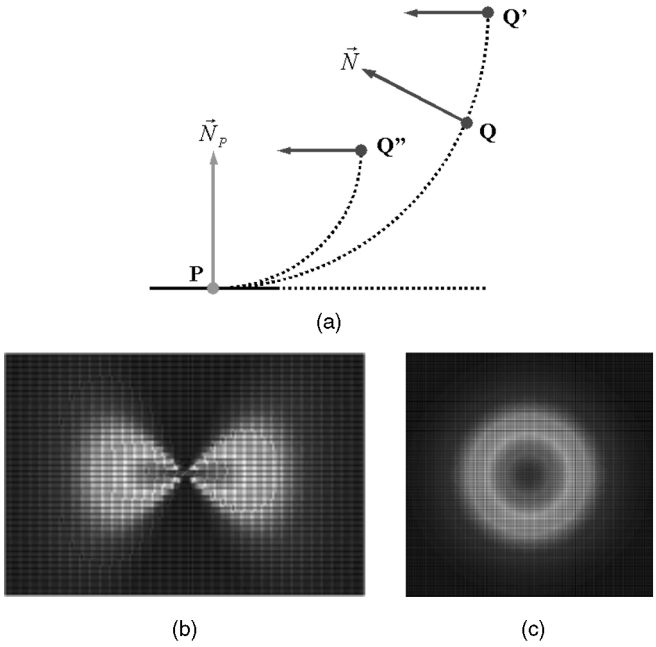


Fig. 3. Voting in 2D. (a) Vote generation. (b) Stick field. (c) Ball field.

eigenvalues λ_1 and $\lambda_2 (\lambda_1 \geq \lambda_2)$ give the ellipse size. The tensor is internally represented as a matrix S:

$$S = \lambda_1 \cdot e_1 e_1^T + \lambda_2 \cdot e_2 e_2^T. \quad (1)$$

An input token that represents a curve element is encoded as a *stick tensor*, where e_2 is the curve tangent and e_1 the curve normal, while $\lambda_1 = 1$ and $\lambda_2 = 0$. An input token that represents a point element is encoded as a *ball tensor*, with no preferred orientation, while $\lambda_1 = 1$ and $\lambda_2 = 1$.

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 2D case. The fields are generated based on a single parameter—the scale factor σ. Vote orientation corresponds to the best (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:

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

Fig. 3a shows how votes are generated to build the 2D 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 curve normal $\vec{N}$ thus obtained, the vote $V_{stick}(\vec{d})$ sent from P to Q is encoded as a tensor according to (3), where $\vec{d} = Q - P$.

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

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. 3b shows the 2D 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 2D plane and integrating all contributions according to (4). The corresponding 2D ball field is shown in Fig. 3c.

$$V_{ball}(\vec{d}) = \int_0^{2\pi} R_\theta V_{stick}\left(R_\theta^{-1}\vec{d}\right) R_\theta^T d\theta. \quad (4)$$

At each receiving site, the collected votes are combined through simple tensor addition (sum of matrices $V(\vec{d})$), thus producing generic 2D 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 confidence of this knowledge (also called saliency, given by the tensor shape and size). For a generic 2D tensor, its curve saliency is given by $(\lambda_1 - \lambda_2)$ 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:

$$S = (\lambda_1 - \lambda_2) e_1 e_1^T + \lambda_2 \left(e_1 e_1^T + e_2 e_2^T\right). \quad (5)$$

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

In the 3D case, the salient features are points, curves and surfaces [29]. A point element corresponds to a *ball tensor*, with $\lambda_1 = \lambda_2 = \lambda_3 = 1$ and no preferred orientation, a curve element is represented by a *plate tensor*, where two eigenvalues codominate ($\lambda_1 = 1,\ \lambda_2 = 1,\ \lambda_3 = 0$) and the eigenvector e_3 gives the curve tangent, and a surface element is represented by a *stick tensor*, where one eigenvalue dominates ($\lambda_1 = 1,\ \lambda_2 = 0,\ \lambda_3 = 0$), and the surface normal is given by e_1.

2.2 Tensor Voting in 4D

The tensor voting framework is general enough to be extended to any dimension readily, except for some implementation changes, mainly for efficiency purposes. The issues to be addressed here are the *tensorial representation* of the features in the 4D 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 4D space and their representation as *elementary* 4D tensors, where n and t represent normal and tangent vectors, respectively. Note that a surface in the 4D space can be characterized by two normal vectors, or by two tangent vectors. From a *generic* 4D 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 for the size and shape of the neighborhood where the votes are cast and also control how the votes depend on distance and orientation. The 4D voting fields are obtained as follows: First, the 4D stick field is generated in a similar manner to the 2D 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

TABLE 1
Elementary Tensors in 4D

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 4D

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

obtained by rotating a 4D stick field around appropriate axes. In particular, the 4D ball field—the only one directly used here—is generated according to:

$$V_{ball}(\vec{d}) = \int\int_0^{2\pi}\int R_{\theta_{xy}\theta_{xu}\theta_{xv}} V_{stick}\left(R^{-1}_{\theta_{xy}\theta_{xu}\theta_{xv}}\vec{d}\right) R^{T}_{\theta_{xy}\theta_{xu}\theta_{xv}} d\theta_{xy} d\theta_{xu} d\theta_{xv}, \quad (6)$$

where x, y, u, v are the 4D 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 2D or 3D case, the data structure used to store the tensors during vote collection was a simple 2D 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* [32]. 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 2D 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 them into a $(x\ y\ v_x\ v_y)$ tuple in the 4D 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) potential 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 4D 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 4D 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 4D 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 nonrigid 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** against a static background. The input frames are shown in Fig. 4a.

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

Candidate matches are generated as follows: In a preprocessing step, for each token in the first frame, we

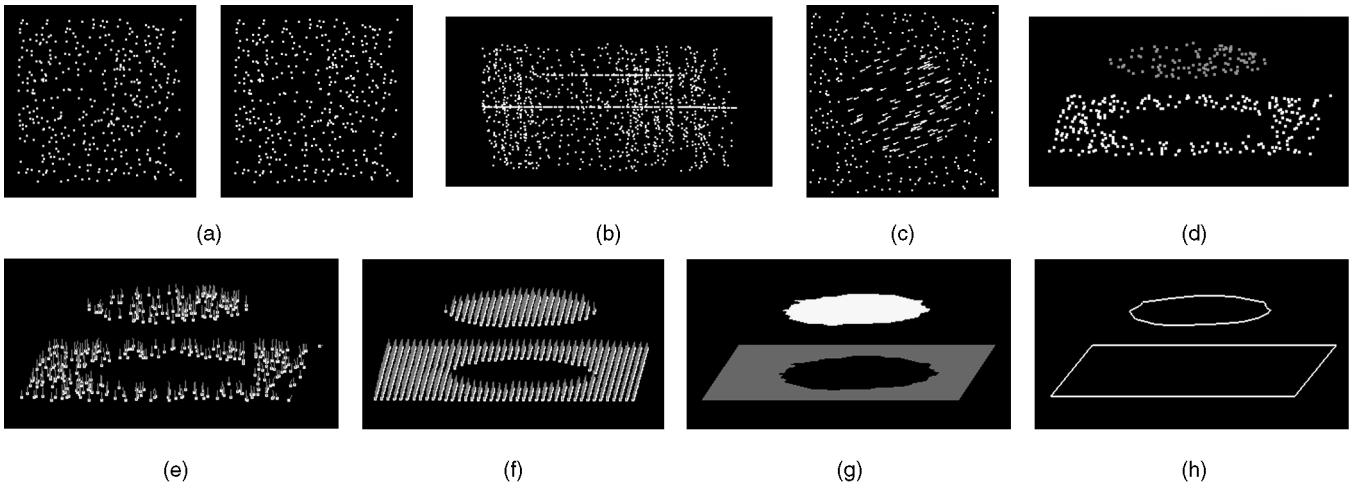


Fig. 4. Translating disk. (a) Input frames. (b) Candidate matches. (c) Sparse velocity field. (d) Recovered v_x velocities. (e) Refined velocity field (sparse). (f) Dense velocity field. (g) Regions. (h) Boundaries.

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 4D 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. 4b 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 4D *ball tensor*—the eigenvalues and eigenvectors are the following:

$$\lambda_1 = 1 \qquad e_1 = (0\ 0\ 0\ 1)^T,$$

$$\lambda_2 = 1 \qquad e_2 = (0\ 0\ 1\ 0)^T,$$

$$\lambda_3 = 1 \qquad e_3 = (0\ 1\ 0\ 0)^T,$$

$$\lambda_4 = 1 \qquad e_4 = (1\ 0\ 0\ 0)^T.$$

After encoding, each token casts votes in a sparse voting process, in the sense that votes are sent only at input token locations. Votes are generated by using the 4D *ball voting field*, where no particular orientation is preferred.

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 4D 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 $\lambda_2 - \lambda_3$), while rejecting the others as outliers. For the translating disk example, a comparison with the ground truth shows that matching was 100 percent accurate—all 400 matches have been recovered correctly, despite the large amount of approximately 500 percent noise present. Fig. 4c shows the recovered sparse velocity field, while Fig. 4d shows a 3D view of the recovered matches (the height represents the v_x velocity component).

3.2 Motion Capture

We start with a sparse velocity field described by the $(x\ y\ v_x\ v_y)$ tuples in the 4D space, that have 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 already been 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.

Therefore, we perform an orientation refinement through another sparse voting process, but this time with the correct matches only. To this purpose, every 4D 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 4D is characterized by *two* normal vectors. In Fig. 4e, we show a 3D 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. Therefore, we must obtain appropriate tensor values at every pixel $(x\ y)$. There may be several tensors with the same $(x\ y)$ but with different $(v_x\ v_y)$ since overlapping layers are present in the case of transparent motion.

The densification process is illustrated in Fig. 5. For each pixel $(x\ y)$, we try to find the best $(v_x\ v_y)$ locations at which to place the newly generated tokens. 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

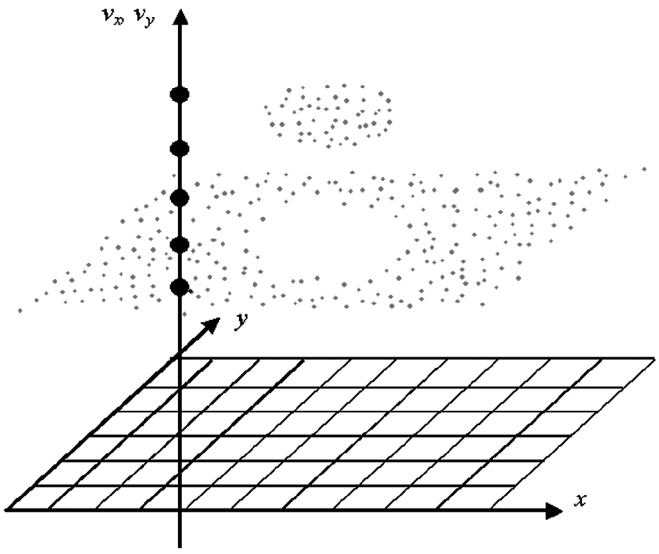


Fig. 5. Densification.

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. After voting, the candidate tokens whose surface saliencies $(\lambda_2 - \lambda_3)$ are locally maximal are retained, and their $(v_x\ v_y)$ coordinates represent the most likely velocities 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. 4f, we show a 3D 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. 4g.

Finally, we have implemented a method to extract the *motion boundary* for each region as a "partially convex hull." We start at some arbitrary point S on the boundary—for example, the point with the largest x coordinate in the region. From there, the boundary curve is grown so that at every current point C, the curve is *locally convex*. For the current point C, the next boundary point N is chosen so that all the points within a neighborhood of C are inside (to the right of) the boundary found so far, including the segment CN. The process is controlled by only one parameter—the scale factor—that gives the size of the neighborhood and, thus, determines the perceived level of detail (that is, the departure from the actual convex hull). The resulting boundary curves are shown in Fig. 4h.

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.

4.1 Using Motion Cues Only

Expanding disk (Fig. 6). 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 one match among 400 has been incorrectly recovered. This example demonstrates that, without special handling, our framework can easily accommodate nonrigid image motion.

Rotating disk—translating background (Fig. 7). 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 two 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

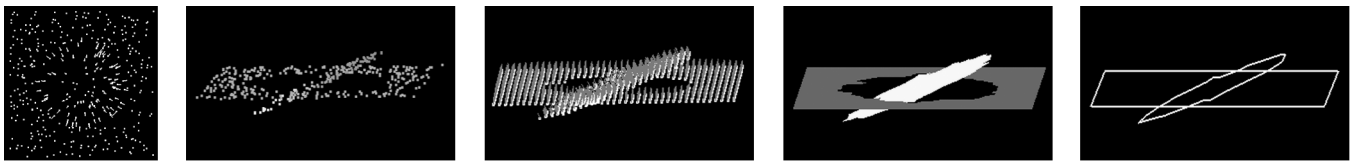


Fig. 6. Expanding disk.

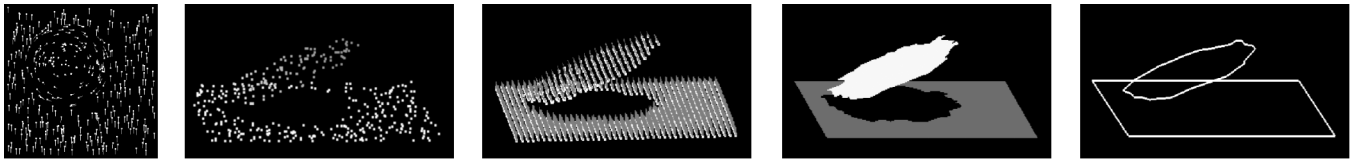


Fig. 7. Rotating disk—translating background.

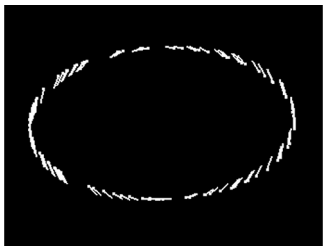


Fig. 8. Rotating ellipse.

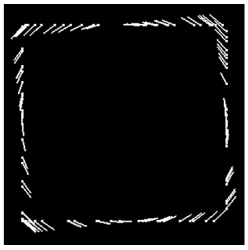


Fig. 9. Rotating square.

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 4D 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 4D positions, but also on the local layer orientations that are still different and, therefore, provide a good affinity measure.

Rotating ellipse (Fig. 8). 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 difficulty is that at the points where the rotated ellipse "intersects" the original one the velocity could be wrongly estimated as zero.

Rotating square (Fig. 9). 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 nonsmooth curves does not produce additional difficulty for our methodology.

Translating circle (Fig. 10). 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 six. After processing, all 100 matches have been correctly recovered. This example shows that we can successfully handle both curves and surfaces in motion.

Fig. 10. Translating circle.

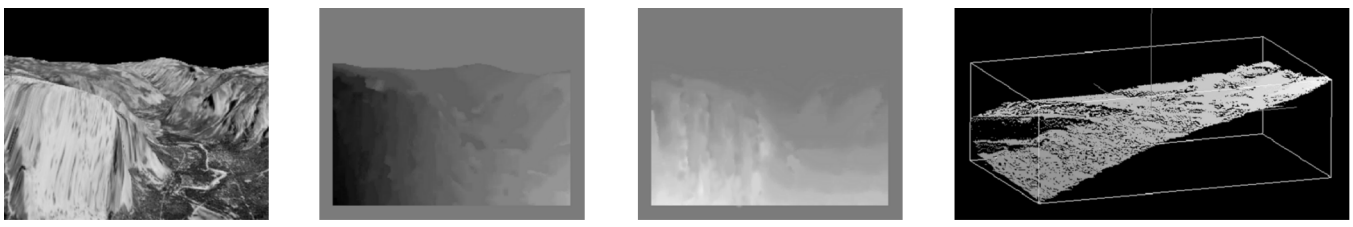
4.2 Incorporating Intensity Information

So far, we have only presented 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 preprocessing 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. 11). We analyzed the motion from two frames of the Yosemite sequence (without the sky) to quantitatively estimate the performance of our approach. Although this is an artificial fly-through sequence, it uses real images as texture for the valley model. The average angular error obtained is $3.74^\circ \pm 4.3^\circ$ for 100 percent field coverage, result which compares favorably to those in the literature—in Table 3, we show the results of other methods, as reproduced from [2]. This example also shows that our method successfully recovers nonplanar motion layers.

Flower Garden sequence (Fig. 12). For a qualitative estimation, we also analyzed the motion from two frames of the Flower Garden sequence. It is worth mentioning that outliers representing wrong candidates around the tree boundary are corrected during the densification step. In



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

Fig. 11. Yosemite. (a) An input frame. (b) X-velocities. (c) Y-velocities. (d) Motion layer (x-velocities).

TABLE 3
Yosemite Sequence Results from Barron et al. [2]

Technique	Average Error	Standard Deviation	Density
Horn and Schunck	22.58°	19.73°	100%
Lucas and Kanade ($\lambda_2 \geq 1.0$)	5.20°	9.45°	35.1%
Lucas and Kanade ($\lambda_2 \geq 5.0$)	3.55°	7.11°	8.8%
Uras et al. (unthresholded)	16.45°	21.02°	100%
Uras et al. ($det(H) \geq 1.0$)	5.97°	11.74°	23.4%
Uras et al. ($det(H) \geq 2.0$)	3.75°	3.44°	6.1%
Anandan	15.54°	13.46°	100%
Heeger	11.74°	19.0°	44.8%
Fleet and Jepson ($\tau = 1.25$)	4.95°	12.39°	30.6%
Fleet and Jepson ($\tau = 2.5$)	4.29°	11.24°	34.1%

order to improve the result, we currently investigate the use of multiple frames and a more elaborate procedure with subpixel precision for the densification and segmentation steps. Since there is no ground truth for this sequence, we illustrate the error by showing the difference image between the input frames registered using the computed motion.

4.3 Handling Reflections and Transparency

Since our framework allows for overlapping motion layers, it can successfully handle images containing reflections and transparency. Here, we consider the image $I(x, y, t)$ at time t as a combination of two patterns A and B, which have independent motions a and b:

$$I(x, y, t) = A^{ta} + B^{tb}, \tag{7}$$

where A^{ta} denotes pattern A transformed by motion ta.

In order to obtain the dominant motion (assume it is a), we run a cross-correlation procedure, followed by a step of voting as described in the Matching section, to eliminate noisy matches. Next, we use a "nulling" method [33], [34], to estimate the remaining motion b. The pattern component A with velocity a is removed from the sequence by moving each frame with a, then subtracting it from the following frame. The resulting difference images are:

$$\begin{aligned} D_k &= I(x, y, k+1) - I^a(x, y, k) \\ &= \left(A^{(k+1)a} + B^{(k+1)b}\right) - \left(A^{(k+1)a} + B^{kb+a}\right) \\ &= (B^b - B^a)^{kb}. \end{aligned} \tag{8}$$

Assuming that we have three frames, the difference images are $D_0 = (B^b - B^a)$ and $D_1 = (B^b - B^a)^b$, which show a pattern $(B^b - B^a)$ moving with a single motion b. We use the same method—cross-correlation followed by voting—to determine motion b from frames D_0 and D_1.

Finally, we put together the two sets of 4D tokens with velocities a and b, and run a step of dense voting and grouping (as described in the Motion Capture section) on the entire set. This process also fills any holes in the layers, which may have been produced by the noisy matches elimination. Note that at this point the motions have been determined, but without separating the image patterns. In order to better show our results, based on the known motions, we recover the two image patterns by solving a least-squares problem, as described in [34].

Transparent motion sequence (Fig. 13). We analyzed the motion from three frames captured with a moving camera, showing a face reflected in a framed picture. We show one of the input frames in Fig. 13a, and the two separated image patterns in Fig. 13b and Fig. 13c.

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

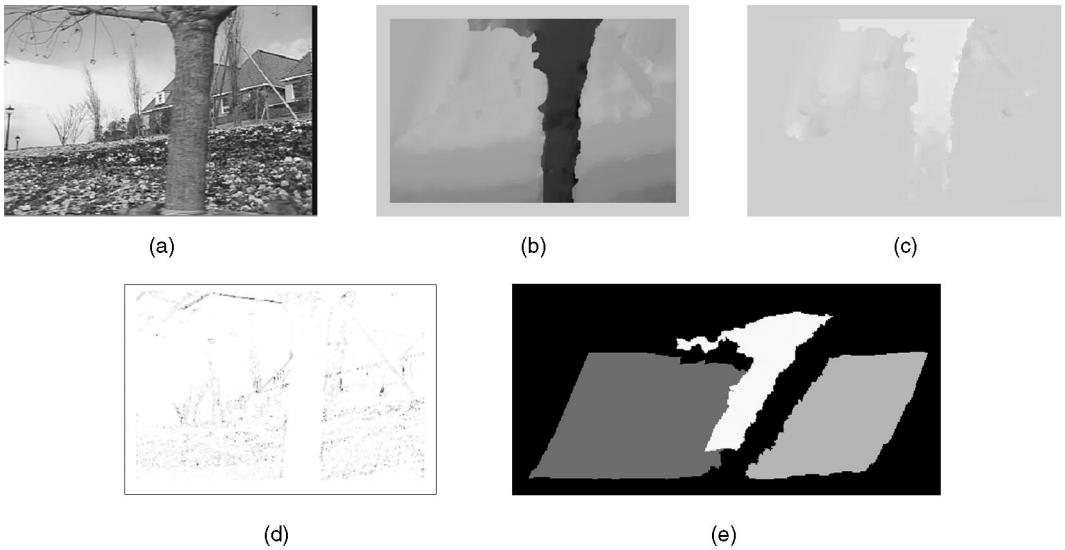


Fig. 12. Flower Garden. (a) An input frame. (b) X-velocities. (c) Y-velocities. (d) Difference error. (e) Regions.

(a) (b) (c)

Fig. 13. Transparent motion. (a) An input frame. (b) Background. (c) Foreground.

example for a large range of scale values and we found that the method is remarkably robust to varying scale factors. Fig. 14 shows the number of wrong matches (for an input of 400 points) obtained for different values of the voting field size (in pixels). 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 4D representation* of data, and a *voting scheme* for token communication. Our methodology is formulated as a 4D tensor voting computational framework.

The moving regions are conceptually represented by smooth layers in the 4D 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 4D 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.

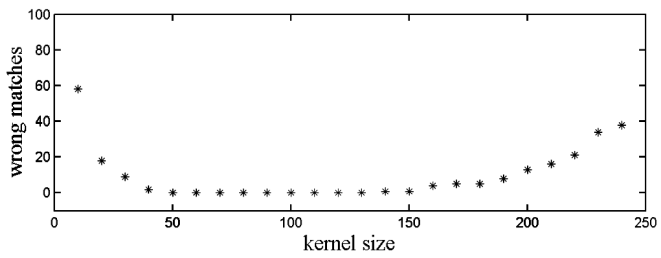


Fig. 14. Scale factor influence.

Despite the high dimensionality, our voting scheme is both time and space efficient. It 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 nonrigid motion, curves, and surfaces in motion. We showed that our method successfully addresses the difficult problem of grouping 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.

Acknowledgments

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, and S. Beauchemin, "Performance of Optical Flow Techniques," *Int'l J. Visual Computing,* vol. 12, no. 1, pp. 43-77, 1994.
[3] B. Horn and B. Schunck, "Determining Optical Flow," *Artificial Intelligence,* vol. 17, pp. 185-203, 1981.
[4] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proc. DARPA Image Understanding Workshop,* pp. 121-130, 1981.
[5] E. Simoncelli, E. Adelson, and D. Heeger, "Probability Distributions of Optical Flow," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 310-315, 1991.
[6] H. Nagel and W. Enkelmann, "An Investigation of Smoothness Constraints for the Estimation of Displacement Vector Fields from Image Sequences," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 8, pp. 565-593, 1986.
[7] P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *Int'l J. Visual Computing,* vol. 2, pp. 283-310, 1989.
[8] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. Comm.,* vol. 31, pp. 532-540, 1983.
[9] A. Singh, *Optical Flow Computation: A Unified Perspective.* IEEE CS Press, 1992.

[10] D. Heeger, "Optical Flow using Spatiotemporal Filters," *Int'l J. Visual Computing,* vol. 1, pp. 279-302, 1988.
[11] F. Heitz and P. Bouthemy, "Multimodal Estimation of Discontinuous Optical Flow Using Markov Random Fields," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 12, pp. 1217-1232, Dec. 1993.
[12] M. Gelgon and P. Bouthemy, "A Region-Level Graph Labeling Approach to Motion-Based Segmentation," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 514-519, 1997.
[13] C. Kervrann and F. Heitz, "A Markov Random Field Model-Based Approach to Unsupervised Texture Segmentation Using Local and Global Spatial Statistics," *IEEE Trans. Image Processing,* vol. 4, no. 6, pp. 856-862, 1995.
[14] Y. Boykov, O. Veksler, and R. Zabih, "Markov Random Fields with Efficient Aproximations," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 648-655, 1998.
[15] S. Ghosal, "A Fast Scalable Algorithm for Discontinuous Optical Flow Estimation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 2, pp. 181-194, Feb. 1996.
[16] R. Deriche, P. Kornprobst, and G. Aubert, "Optical Flow Estimation while Preserving its Discontinuities: A Variational Approach," *Proc. Asian Conf. Computer Vision,* pp. 290-295, 1995.
[17] T. Darrell and A. Pentland, "Robust Estimation of a Multilayered Motion Representation," *Proc. IEEE Workshop Visual Motion,* pp. 173-178, 1991.
[18] S. Ayer and H. Sawhney, "Layered Representation of Motion Video Using Robust Maximum Likelihood Estimation of Mixture Models and MDL Encoding," *Proc. Int'l Conf. Computer Vision,* pp. 777-784, 1995.
[19] G. McLachlan and K. Basford, *Mixture Models Inference and Applications to Clustering.* Marcel Dekker, Inc., 1988.
[20] S. Hsu, P. Anandan, and S. Peleg, "Accurate Computation of Optical Flow by Using Layered Motion Representations," *Proc. Int'l Conf. Pattern Recognition,* pp. 743-746, 1994.
[21] A. Jepson and M. Black, "Mixture Models for Optical Flow Computation," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 760-761, 1993.
[22] Y. Weiss, "Smoothness in Layers: Motion Segmentation Using Nonparametric Mixture Estimation," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 520-526, 1997.
[23] M. Irani and S. Peleg, "Image Sequence Enhancement Using Multiple Motions Analysis," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 216-221, 1992.
[24] J. Wang and E. Adelson, "Representing Moving Images with Layers," *IEEE Trans. Image Processing Special Issue: Image Sequence Compression,* vol. 3, no. 5, pp. 625-638, 1994.
[25] D. Fleet, M. Black, and A. Jepson, "Motion Feature Detection Using Steerable Flow Fields," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 274-281, 1998.
[26] J. Shi and J. Malik, "Motion Segmentation and Tracking Using Normalized Cuts," *Proc. Int'l Conf. Computer Vision,* pp. 1154-1160, 1998.
[27] Y. Wu, T. Kanade, J. Cohn, and C. Li, "Optical Flow Estimation Using Wavelet Motion Model," *Proc. Int'l Conf. Computer Vision,* pp. 992-998, 1998.
[28] J. Little, H. Bulthoff, and T. Poggio, "Parallel Optical Flow Using Local Voting," *Proc. Int'l Conf. Computer Vision,* pp. 454-459, 1988.
[29] G. Medioni, M.-S. Lee, and C.-K. Tang, *A Computational Framework for Segmentation and Grouping.* Elsevier Science, 2000.
[30] L. Gaucher and G. Medioni, "Accurate Motion Flow Estimation with Discontinuities," *Proc. Int'l Conf. Computer Vision,* pp. 695-702, 1999.
[31] G. Guy and G. Medioni, "Inference of Surfaces, 3D Curves and Junctions from Sparse, Noisy 3D Data," *IEEE Trans. Pattern Analysis and Machine Itelligence,* vol. 19, no. 11, pp. 1265-1277, Nov. 1997.
[32] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," *J. ACM,* vol. 45, no. 6, pp. 891-923, 1998.
[33] J. Bergen, P. Burt, R. Hingorani, and S. Peleg, "A Three-Frame Algorithm for Estimating Two-Component Image Motion," *IEEE Trans. Pattern Analysis and Machine Itelligence,* vol 14, no. 9, pp. 886-896, Sept. 1992.
[34] R. Szeliski, S. Avidan, and P. Anandan, "Layer Extraction from Multiple Images Containing Reflections and Transparency," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 246-253, 2000.

Mircea Nicolescu received the Diploma de Licenta in computer science from the Polytechnic University Bucharest, Romania, in 1995, and the MS degree in computer science from the University of Southern California (USC), Los Angeles, in 1999. He is currently a PhD candidate in computer science, and a graduate research assistant at the Computer Vision Laboratory of the Institute of Robotics and Intelligent Systems, and the Integrated Media Systems Center, at USC. His research interests include panoramic video systems, real-time segmentation and tracking, motion analysis, and perceptual grouping. In 1999, he received the USC Academic Achievements Award and in 2002 the Best Student Paper Award at the International Conference on Pattern Recognition in Quebec City, Canada. He is a student member of the IEEE Computer Society.

Gérard Medioni received the Diplôme d' Ingénieur Civil from the Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1977, and the MS and PhD degrees in computer science from the University of Southern California, Los Angeles, in 1980 and 1983, respectively. He has been with the University of Southern California (USC) in Los Angeles, since 1983, where he is currently a professor of computer science and electrical engineering, codirector of the Computer Vision Laboratory, and chairman of the Computer Science Department. He was a visiting scientist at INRIA Sophia Antipolis in 1993 and the Chief Technical Officer of Geometrix, Inc. during his sabbatical leave in 2000. His research interests cover a broad spectrum of the computer vision field and he has studied techniques for edge detection, perceptual grouping, shape description, stereo analysis, range image understanding, image to map correspondence, object recognition, and image sequence analysis. He has published more than 100 papers in conference proceedings and journals. Dr. Medioni is a fellow of the IEEE and a fellow of the IAPR. He has served on the program committees of many major vision conferences and was program chairman of the 1991 IEEE Computer Vision and Pattern Recognition Conference in Maui, program cochairman of the 1995 IEEE Symposium on Computer Vision held in Coral Gables, Florida, general cochair of the 1997 IEEE Computer Vision and Pattern Recognition Conference in Puerto Rico, program cochair of the 1998 International Conference on Pattern Recognition held in Brisbane, Australia, and general cochairman of the 2001 IEEE Computer Vision and Pattern Recognition Conference in Kauai. Professor Medioni is an associate editor of the *Pattern Recognition and Image Analysis* journal and one of the North American editors for the *Image and Vision Computing* journal.