Occam Algorithms for Computing Visual Motion

Haim (Shvaytser) Schweitzer, *Member, IEEE*

***Abstract*—The standard approach to computing motion relies on pixel correspondence. Computational schemes impose additional constraints, such as smoothness and continuity of the motion vector field, even though these are not directly related to pixel correspondence. This paper proposes an alternative to the multiple constraints approach. By drawing analogy with machine learning, motion is computed as a function that accurately predicts frames. The Occam-Razor principle suggests that among all functions that accurately predict the second frame from the first frame, the best predictor is the "simplest," and simplicity can be rigorously defined in terms of encoding length. An implementation of a practical algorithm is described. Experiments with real video sequences verify the algorithm assumptions by showing that motion in typical sequences can be accurately described in terms of a few parameters. Our particular choice of predictors produces results that compare very favorably with other image flow algorithms in terms of accuracy and compactness. It may, however, be too constrained to enable accurate recovery of 3D motion and structure.**

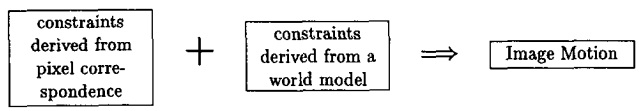
***Index Terms*—Image motion, optic flow, video compression, machine learning, Occam algorithms.**

I. Introduction

Let I_1, I_2 be two pictures of a 3D world scene. If $I_1(x, y)$ and $I_2(x + \Delta x, y + \Delta y)$ are projections of the same object point then we say that the motion between I_1 and I_2 at the point (x, y) is $(\Delta x, \Delta y)$. Motion is an important source of visual information. Motion estimates are used in a variety of applications, ranging from high level tasks such as the recovery of 3D structure (e.g., [1], [28], [11]) to low level tasks such as image compression (e.g., [10], [15]). The design of accurate and efficient algorithms for computing motion has been one of the most active research areas in computer vision for at least a decade.

Several motion estimation algorithms address pixel correspondence directly. They first identify a set of local features in each frame and then try to compute a match between these features [19], [18], [16]. Unfortunately, these direct techniques do not produce a dense vector field, and their complexity may grow exponentially in the number of feature points.

Most recent motion algorithms take a different approach. They treat motion estimation as an optimization problem, defined in terms of multiple constraints, each of which is insufficient by itself. (It is known that the constraints implied by pixel correspondence are not enough to uniquely determine the motion [12], and additional constraints derived from a world model must be introduced.) The general framework of these algorithms is shown in the following diagram:



Constraints derived from pixel correspondence are based on the assumption that the intensity of corresponding pixels is similar. Constraints derived from a world model usually take the form of "smoothness" constraints, and are sometimes used as regularization terms in a minimization procedure. In some cases the additional constraints are explicitly stated [17], [12], [20], and in other cases they are hidden in the computational scheme. Examples of the latter case are multi-resolution techniques [28], [2], [5], [4].

A. A Criticism of the Multiple Constraints Approach

Multiple constraints techniques have a methodological problem: they *define* the solution to the ill-posed correspondence problem in terms of their proposed set of constraints. Since each technique gives an optimal (or near optimal) solution to a slightly different problem, the results are difficult to compare. Even though some of these algorithms perform very well, none claim to be computing the *exact* image motion. Instead, they claim to produce approximations, but without explicitly stating what quantities are being approximated.

Thus, the only accurate way to compare the relative merit of these algorithms is by experiments with a known ground truth [3]. Unfortunately, the ground truth is only known in very simple and regular cases. The difficulty of determining what multiple constraints techniques approximate is best exemplified by the fact that these techniques do not perform significantly better when given unrestricted computational resources.

B. Our Approach

The observation that led to the algorithm presented here is that the computation of motion as a solution to the ill-posed correspondence problem can be formulated as learning from examples, or inductive inference. Roughly speaking, we view the frames of a video sequence as a set of examples. The learning task is to come up with a method to accurately predict unknown frames. We formulate the problem of motion estimation as a search for a function that can accurately predict frames. Given two frames of a motion sequence, this function is to be chosen according to two criteria:

1) It accurately predicts the second frame from the first frame.
2) The predictor is simple.

As in the work of [24], [8], [7] the complexity of a predictor can be defined as its size in a fixed encoding of all possible predictors. This fixed encoding is chosen in a way that reflects

Manuscript received Dec. 2, 1992; revised May 9, 1995.

The author is with the Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, PO Box 830688, Richardson, TX 75083-0688; e-mail: haim@utdallas.edu.

IEEECS Log Number P95127.

a world model, which assumes a scene with several objects moving in a simple motion. Specifically, the predictor complexity is defined as the encoding size of a segmentation of the frames into moving objects and the motion of these objects. Even though an exact segmentation may be just as difficult to compute as the motion itself, it may not be necessary. As was shown in [8], [7], near optimal encodings are also near optimal predictors. Motivated by the ID3 learning algorithm [23] we compute an approximate segmentation by a greedy algorithm that maintains a binary tree structure of rectangular patches. The complexity of this representation can be measured by the number of leaves in the tree.

The idea that segmentation should be performed simultaneously with the motion estimation is not new. See for example [14] and the arguments in Anandan's paper [2] that discontinuities must be detected simultaneously with the flow computation. Our approach is different in that we consider the goal to be a simple (short) encoding of predictors and not a meaningful segmentation. According to this criterion *a crude segmentation with compact encoding should be more useful for computing accurate motion than a very accurate segmentation that does not have a compact encoding*. This is the major difference between our approach and previously proposed motion estimation techniques.

The seemingly paradoxical statement in the last paragraph needs some clarification. The point made is not that accurate segmentation is bad, but that it cannot be computed reliably from the given data. Thus, algorithms that attempt to compute accurate segmentation that have no compact encoding must make some arbitrary decisions that may lead to bad estimates. On the other hand, the compactness of the predictor is an a priori indication of its ability to accurately predict frames.

C. Evaluation Criteria

An algorithm based on the Occam-Razor principle was implemented. It was tested on real video images with several alternative assumptions on the type of uniform motion inside each homogeneous (rectangular) patch. These include translation rotation and scale, the general linear transformation, and linear transformation combined with grey level scaling. The algorithm performance was evaluated according to the following criteria:

- Confirming the assumptions of the world model. It was experimentally observed that a good predictor for a typical 512 × 512 image requires approximately 100 rectangles, with the motion inside each rectangle specified by 4-8 numbers. Using a smart tree structure, the encoding of such predictors requires less than 1,000 numbers. (This should be compared to 262,144, the number of pixels in a 512 × 512 image.)
- Comparison with other algorithms.
  - — Prediction results were compared to Anandan's algorithm [2] and to the block motion estimation algorithm used in MPEG.
  - — Compactness of the encoding was compared to the compactness of MPEG motion descriptors [15].
  - — Motion accuracy was compared to results reported by Barron et al. [3] on synthetic data.

These comparisons show that our algorithm compares favorably with other motion estimation algorithms in terms of accuracy and compactness of the motion representation. Thus, the algorithms shows great potential for applications related to video encoding and compression. On the other hand, the assumptions made on the uniformity of motion inside each rectangular patch may not enable accurate recovery of *local* 3D structure.

II. On The Definition of Optic Flow, Image Motion, and Predictors

A digitized video sequence is a discrete sample in time and space of a continuous intensity function $I(x, y, t)$. For a constant t, the two dimensional function $I(x, y, t)$ is called *a frame*. For constant x, y, t, the intensity value $I(x, y, t)$ is called *a pixel*.

The values of $I(x, y, t)$ are usually obtained from projections of a 3D scene into a viewing plane. These projections induce a natural correspondence between pixels on different frames: *There is a correspondence between $I(x_1, y_1, t_1)$ and $I(x_2, y_2, t_2)$ $(t_1 \neq t_2)$ if they are both projections of the same 3D point.* The correspondence between two frames is a two dimensional vector field given in terms of the two functions (U_x, U_y). They are defined by the following relation:

There is a correspondence between the pixels $I(x + U_x(x, y, t_1, t_2), y + U_y(x, y, t_1, t_2), t_1)$ and $I(x, y, t_2)$ for all x, y, t_1, t_2.

Using the frame $I_0(x, y) = I(x, y, 0)$ as a reference frame, the *image motion* is defined as the vector field given by the two functions $U_x(x, y, t)$, $U_y(x, y, t)$ such that:

For all x, y, t[1] there is a correspondence between the pixels $I_0(x + U_x(x, y, t), y + U_y(x, y, t))$ and $I(x, y, t)$.

Assuming that U_x, U_y have partial time derivatives, the *optic flow at $t_0 = 0$* is the two dimensional vector field given by the two functions V_x, V_y, and defined as:

$$V_x(x,y) = \lim_{t\to 0} \frac{\partial U_x(x,y,t)}{\partial t}, \qquad V_y(x,y) = \lim_{t\to 0} \frac{\partial U_y(x,y,t)}{\partial t}.$$

Assuming no acceleration:

$$U_x(x, y, t) = tV_x(x, y),\ U_y(x, y, t) = tV_y(x, y).$$

Most of the computational schemes rely on the assumption of constant intensity of corresponding pixels:

Corresponding pixels have the same (or nearly the same) intensity.

An immediate consequence of this assumption is that the entire sequence $I(x, y, t)$ is specified (or at least well approximated) by the single frame $I_0(x, y)$ and the image motion:

$$I(x, y, t) \approx I_0(x + U_x(x, y, t), y + U_y(x, y, t)), \tag{1}$$

and with the additional assumption of no acceleration:

$$I(x, y, t) \approx I_0(x + tV_x(x, y), y + tV_y(x, y)). \tag{2}$$

1. The definition can be relaxed to include a subset of all x, y, t.

Equation (2) shows that *under the assumptions of constant intensity and no acceleration* the values of $I(x, y, t)$ can be accurately predicted from a reference frame and its optic flow. The common argument used to justify these assumptions is that in natural scenes they (approximately) hold over short time periods. This, however, may not be accurate. An intensity change may be caused, for example, by an instantaneous change in illumination conditions, and rotation, however small, always involves acceleration. The effects of rotation and illumination change cannot be accounted for by a predictor of the type described in (2).

A crucial point in our work is the argument that the prediction property of the optic flow can be used to *define* a generalization of optic flow, which may not depend so heavily on the above assumptions.

DEFINITION. *A function P is a predictor for the sequence $I(x, y, t)$ if*:

$$I(x, y, t) \approx P(x, y, t).$$

When (2) holds, optic flow can be used to form a predictor. Other predictors are described in Section IV.B.

III. Motion Estimation Viewed as Learning

In the previous section motion was defined in terms of the projected 3D scene. The problem of computing motion from two (or any finite number of) frames is clearly ill-posed. Previously suggested motion algorithms treated motion estimation as an optimization problem, defined in terms of multiple constraints. Additional constraints were derived from a world model.

A similar approach of solving ill posed problems by introducing a world model is found in machine learning. Learning algorithms get as input a set of examples and attempt to discover some regularity that can be used to predict future (unknown) examples. The performance of a learning algorithm is measured by the accuracy of the predictor. Recent advances in computational learning theory distinguish between two learning strategies [8], [7]:

- *Consistent algorithms.* If it is known that according to a world model the set of accurate predictors of future examples is highly constrained, learning can be achieved by computing a predictor that is consistent with a small set of training examples.
- *Occam algorithms.* If the set of accurate predictors of examples is not highly constrained, a predictor consistent with a small set of training examples may not perform well on other examples. To achieve accurate learning the predictor must be chosen as both consistent with the set of training examples and "simple." The complexity of a predictor is defined as its length in a fixed (but otherwise arbitrary) encoding of all predictors.

In computational learning theory, the constrained nature of the set of predictors is defined in terms of the VC dimension [8]. A consistent algorithm is an algorithm that is guaranteed to produce (with high probability) a predictor that accurately predicts the values of random training examples. An Occam algorithm is an algorithm that is guaranteed to produce (with high probability) an accurate predictor for the training examples with encoding that is significantly shorter than the original data.

Viewed from this perspective, previously suggested motion estimation algorithms fall under the category of consistent algorithms. By contrast, the algorithm presented here is an Occam algorithm. Instead of proposing a set of constraints to determine a solution to the ill-posed correspondence problem we propose an encoding scheme for the predictors. We then search for a predictor with *compact* encoding.

A. Occam Predictors as Motion Estimation Algorithms

The analogy with learning suggests the following framework for designing an Occam algorithm for motion estimation:

1) Determine the type of predictor that the algorithm is required to produce. An example is given in (2), where the predictor is given in terms of the optic flow.
2) Determine an encoding scheme for all predictors of the type defined in 1).
3) Show (by theoretical arguments or by experiments) that the encoding scheme in 2) is expected to produce significantly shorter descriptions of the data (input frames) than the data itself.
4) Design a minimization algorithm that gets, as input, two (or more) frames, and produces, as output, a minimum length—or near minimum length—predictor.

IV. The Proposed Algorithm

Let $P(x, y, t)$ be the predictor for the sequence $I(x, y, t)$. The prediction error at time t is given by:

$$E = \Sigma_{x,y} \| P(x, y, t) - I(x, y, t) \|.$$

The standard world model assumes a scene with several rigid objects, each moving in a simple motion. Therefore, the most natural encoding of a predictor appears to be a segmented scene, with different motion parameters associated with each segment (image patch). Let R be a partition of the scene into segments. The error E can now be written as:

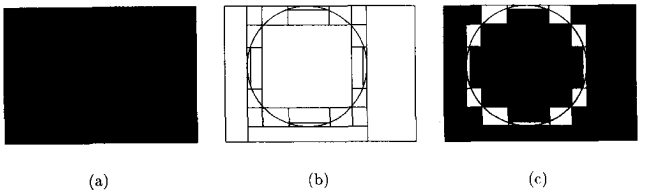
$$E = \Sigma_{r \in R} e_r, \qquad e_r = \Sigma \| P_r(x, y, t) - I(x, y, t) \|, \tag{3}$$

where the second summation is over all pixels in the image segment r, and P_r is the predictor function over the image segment r. The algorithm computes the segmentation R and the predictors P_r for all $r \in R$.

A. The Proposed Encoding Scheme

Our goal is not only to minimize the error E in (3) but also to minimize the encoding size of the predictor. If each individual function P_r can be compactly encoded, the goal of the minimization is to minimize the number of image segments while keeping E small. Unfortunately, even if the number of moving objects is small (say one), an encoding of this type is not guaranteed to be short. The difficulty lies in the encoding of object boundaries. Since we are only interested in compactness and not in a "meaningful" segmentation, we propose to replace a segmentation into arbitrarily complex image regions

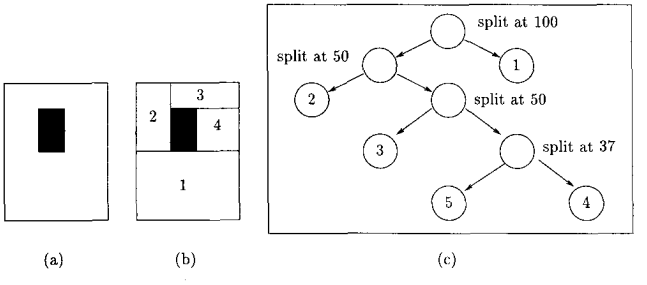
with a segmentation into patches that can be described by simple curves. One of the simplest choices is to use axis parallel rectangular patches. Each rectangular patch is determined by its location (two numbers) and its dimensions (two numbers). Some image segments may require many rectangles for approximate segmentation, as illustrated in Fig. 1.



(a) (b) (c)

Fig 1. Segmentation of a circle in terms of axis parallel rectangular patches. (a) a circle. (b) possible segmentation by 23 rectangles. (c) the white areas are not error free.

Segmenting the scene into rectangles with uniform motion is, by itself, a difficult task. The recursive algorithm that we use produces a slightly different representation of rectangles, where each rectangle is described by a single number. This representation, known in VLSI design as "a slicing floorplan" [27], is created by recursively splitting a single rectangle into two rectangles. The data structure is essentially a variation of a quad tree [25]. We use the convention that the algorithm always splits the larger side of a rectangle. An example is shown in Fig. 2. Fig. 2a shows a single rectangle. Fig. 2b shows the same rectangle embedded in a floorplan, and Fig. 2c shows the slicing floorplan tree. The floorplan scheme requires five rectangles to describe the same rectangular patch, but these rectangles are described in terms of only four numbers, as shown in the tree picture.



(a) (b) (c)

Fig. 2. Rectangles specified by a floorplan tree. (a) a rectangular patch. (b) the corresponding floorplan. (c) the slicing floorplan tree of (b).

B. The Proposed Predictors

As mentioned in the introduction, predictors that are based only on optic flow are limited. Specifically, a predictor of the type described by (2) cannot be used to describe changes caused by rotation and/or illumination change. Rotation can be described by the more general predictor given by (1); however, even (1) cannot account for illumination changes. (It was argued in [22] that variations in illumination may dominate the changes caused by the motion itself.) An obvious generalization of (1) is a predictor of the following type:

$$P_r(x, y, t) = \alpha_t \cdot I_0(x + U_x(x, y, t), y + U_y(x, y, t)) + \beta_t, \quad (4)$$

where P_r is the predictor over the rectangle r, $I_0(x, y)$ is the reference frame, α_t, β_t are time-dependent constants, and the functions U_x, U_y depend on a small number of parameters.

While it is may seem natural to extend translation and rotation over time by linear interpolation, it is not so obvious how to interpolate other parameters, such as scale change or the parameter α_t in (4). In all cases we have used the following rule of a thumb:

> Write the predictor in a "natural" representation. If a parameter has the value 0 when predicting the reference frame (e.g., the parameter β_t in (4)), extend it over time by linear interpolation. If it has the value 1 when predicting the reference frame and it must be positive (e.g., the parameter α_t in (4)), extend it over time by exponential interpolation.

We implemented and investigated the following cases:

PREDICTOR A. Pure translation, rotation, and scale. Here, $\alpha_t = 1$, $\beta_t = 0$, and the predictor in each rectangle is defined in terms of four constant parameters: Δx, Δy, θ, and S. In matrix notation:

$$\begin{pmatrix} U_x(x,y,t) \\ U_y(x,y,t) \end{pmatrix} = S_t \begin{pmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x_t \\ \Delta y_t \end{pmatrix},$$

where:

$$\Delta x_t = t\Delta x, \qquad \Delta y_t = t\Delta y,$$
$$\theta_t = t\theta, \qquad S_t = e^{tS}.$$

PREDICTOR B. The general linear transformation. Here, $\alpha_t = 1$, $\beta_t = 0$, and the predictor in each patch is defined in terms of six constant parameters. If the transformation is written in the standard linear form as:

$$\begin{pmatrix} U_x(x,y,t) \\ U_y(x,y,t) \end{pmatrix} = \begin{pmatrix} a_t x + b_t y + c_t \\ d_t x + e_t y + f_t \end{pmatrix}$$

it is not clear how to interpolate the six parameters over time. We use the equivalent representation given in terms of the parameters: Δx, Δy, θ, $S1$, $S2$, and q.

$$\begin{pmatrix} U_x(x,y,t) \\ U_y(x,y,t) \end{pmatrix} = \begin{pmatrix} S1_t & q_t \\ q_t & S2_t \end{pmatrix} \begin{pmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x_t \\ \Delta y_t \end{pmatrix} \quad (5)$$

where:

$$\Delta x_t = t\Delta x, \quad \Delta y_t = t\Delta y,$$
$$\theta_t = t\theta, \quad q_t = tq,$$
$$S1_t = e^{tS1}, \quad S2_t = e^{tS2}.$$

PREDICTOR C. The general linear transformation combined with grey level scaling. This predictor is defined in terms of eight parameters: Δx, Δy, θ, $S1$, $S2$, q, α, and β. The functions U_x, U_y are defined in terms of Δx, Δy, θ, $S1$, $S2$, q, as in (5), and α_t, β_t are given by:

$$\alpha_t = e^{t\alpha}, \qquad \beta_t = t\beta.$$

V. The Minimization Procedure

A. The Recursive Split Algorithm

We describe the minimization procedure for a pair of frames taken at times 0, t. Obviously, there are many predictors of the type described in Sections IV.A and IV.B that accurately predict the second frame from the first frame. For example, one can choose all rectangles to be of size 1×1 (a single pixel), and then the predictors are reduced to the type described by (2). The Occam-Razor principle suggests that among all predictors that accurately predict the second frame from the first frame, the "simplest" is the best choice for predicting other frames. Using the encoding length as a measure of simplicity we need a procedure that can find an accurate predictor while minimizing the number of rectangular patches.

In formulating the minimization as a search problem we assume that when a rectangle is given, it is possible to compute the optimal parameters of the predictors of Section IV.B. The actual algorithm that does that is described in Section V.B. The precise minimization problem can be stated in two alternative ways. Let E be the error in approximating the second frame as given by (3), and let n stand for the number of axis parallel rectangles in the partitioning.

PROBLEM 1. For a given E, minimize n.

PROBLEM 2. For a given n, minimize E.

We describe a greedy algorithm for approximating the solution to Problem 2. The algorithm is described here in an analogous way to heuristic search procedures [21].

The algorithm maintains a list R of rectangular patches. Let e_r be the error in predicting the second frame from the first frame over the rectangle r, as given by (3).

The Recursive Split Algorithm

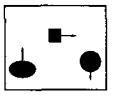
Start with R containing a single rectangular patch that covers the entire frame.

Repeat $n - 1$ times Steps 1), 2), 3):

1) Search R for the rectangle r with the largest error e_r, and remove it from R.
2) Split r into two rectangles r_1, r_2 such that $e_{r_1} + e_{r_2}$ is minimized.
3) Add r_1, r_2 to R.

Other criteria for choosing a rectangle at Step 1, or for stopping before $n - 1$ iterations are completed may also be used. Notice that the rectangles created by this algorithm can be described as the leaves of a floorplan tree. To simplify the search for the optimal split in Step 2 our implementation always splits the larger side of the rectangle r.

EXAMPLE. Assume that we are given frames with a circle, an ellipse, and a rectangle, with the true motion as described by the arrows in the following figure:



The algorithm starts with a single rectangle:

$R = \{ r = $  $\}$

The first split must be vertical and cannot produce zero error. Assume that it creates the following two rectangles:

$R = \{ r_1 = $  , $r_2 =$  $\}$

For the next split, assume r_1 is chosen. Since it is possible to split r_1 horizontally into two patches that give zero error, the next split gives:

$R = \{ r_{11} =$  , $r_{12} =$  , $r_2 =$  $\}$

At this point, $e_{r11} = 0$ and $e_{r12} = 0$, so the algorithm must choose r_2 next. As before, a horizontal split reduces the error over r_2 to 0 and we have:

$R = \{ r_{11} =$  , $r_{12} =$, $r_{21} =$, $r_{22} =$  $\}$.

Now all the rectangles in R have zero error and additional splits cannot reduce the total error. The floorplan tree is shown if Fig. 3. Notice that the result is not the optimal segmentation into three segments.

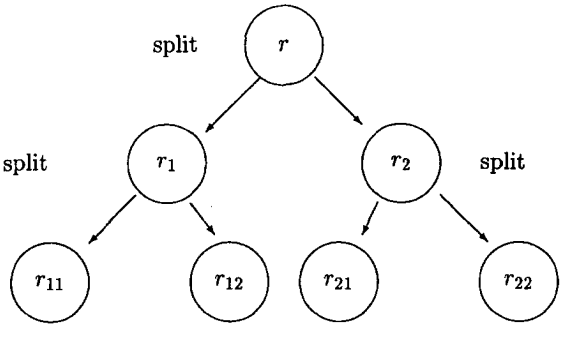


Fig. 3. The slicing floorplan tree of the example.

The recursive split algorithm is similar to standard split and merge techniques (e.g., [13]), but it never attempts to merge rectangles. Merging may break the tree structure and reduce it to a random collection of rectangles that are more difficult to encode.

B. Optimal Splitting of a Single Rectangle

In this section, we describe an implementation of Step 2 of the recursive split algorithm.

Taking the least squares error of the approximation given by (4) over a rectangular patch r we have:

$$e_r = \sum_{x,y \in r} (\alpha_t \cdot I_0(x + U_x(x, y, t), y + U_y(x, y, t)) + \beta_t - I(x, y, t))^2 \quad (6)$$

Define:

$$m_0 = \Sigma 1,$$
$$m_1 = \Sigma I_0(x + U_x(x, y, t), y + U_y(x, y, t)), \quad M_1 = \Sigma I(x, y, t),$$
$$m_2 = \Sigma I_0^2(x + U_x(x, y, t), y + U_y(x, y, t)), \quad M_2 = \Sigma I^2(x, y, t),$$
$$Q = \Sigma I_0(x + U_x(x, y, t), y + U_y(x, y, t)) \cdot I(x, y, t),$$

where the summations are over all x, y in a rectangle r. In terms of these quantities the error e_r of (6) can be written as:

$$e_r = \alpha^2 m_2 + \beta^2 m_0 + M_2 - 2(\alpha\beta m_1 - \alpha Q - \beta M_1). \quad (7)$$

Since this is a quadratic function in α, β, the error e_r is minimized for α, β that are solutions to the following system:

$$\alpha m_2 - \beta m_1 = Q$$
$$-\alpha m_1 + \beta m_0 = M_1 \quad (8)$$

The error that needs to be minimized in Step 2 of the recursive split algorithm is:

$$e = e_{r_1} + e_{r_2} \quad (9)$$

Thus, in the case of Predictor A (see Section IV.B) the error e is a function of nine parameters: Δx^{r1}, Δy^{r1}, θ^{r1}, S^{r1}, Δx^{r2}, Δy^{r2}, θ^{r2}, S^{r2}, z, where Δx^{r1}, Δy^{r1}, θ^{r1}, and S^{r1}, are the parameters of Predictor A over r_1, and Δx^{r2}, Δy^{r2}, θ^{r2}, and S^{r2} are the parameters of Predictor A over r_2. The parameter z is the coordinate where r is split into r_1, r_2 (we only consider splitting the larger side of r). These parameters are illustrated in Fig. 4. Similarly, if Predictor B is used, the error e is expressed as a function of 13 parameters. Since (8) can always be solved to determine the optimal α, β and these values can be substituted back into (7), the error e for Predictor C can also be expressed as a function of 13 parameters.

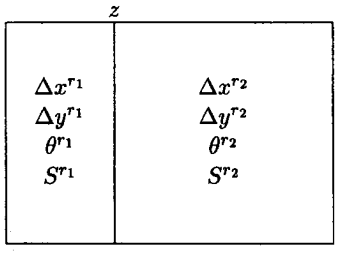


Fig. 4. The error parameters for Predictor A.

Bergen et al. [6] have proposed a clever hierarchical minimization technique for fitting an affine surface to a single rectangular patch. This suggests a minimization procedure that iterates between solving for surface parameters and the one-dimensional search for the optimal value of z. Unfortunately, our experiments show that this approach does not work well in our case. It appears that the surface fit technique of Bergen et al. is only useful when the minimized error can be made very small (this is the case when there is a *single* dominant moving object).

Since the number of free parameters that determine the error e is small (nine for Predictor A and 13 for predictors B and C), the minimum can be computed by standard numerical minimization techniques. We have implemented the Parallel Planes (PARP) algorithm [9], a well known multi-dimensional minimization techniques. PARP is a variation of conjugate gradients that does not require gradients. Instead, it uses several one-dimensional line searches that can be implemented by computing function values. See [9] for details. See also the work of Srinivasan and Rao [26], where a variation of PARP is used for computing translation (minimization of a function of two variables).

PARP is guaranteed to find a (local) minimum in one iteration if the error function (as given by (9)) is quadratic. It is also known to give a good approximation to the minimum when the error function is nearly quadratic, and its rate of convergence is similar to Newton's method. Our experiments show that in typical video sequences the error (as given by (9)) is minimized by a single PARP iteration. This can be taken as an experimental evidence that the error surface is, indeed, nearly quadratic.

C. Implementation Details and Complexity

Line minimization requires three function values if the function is (assumed to be) quadratic. A single iteration of the PARP algorithm requires $\binom{m}{2}$ line minimizations, where m is the number of variables. Since a change in the error parameters of e_{r_1} does not affect the error e_{r_2} when the value of z is unchanged (see Fig. 4), it is possible to reduce the PARP computation of a single iteration to $\binom{m}{2} + m$ line minimizations over r_1, $\binom{m}{2} + m$ line minimizations over r_2, and a single line minimization over r, where m is the number of error parameters in a single rectangle. This complexity is the same as computing $\binom{m}{2} + m + 1$ line minimizations on r.

This gives a total of 45 computations of function values for Predictor A (m = 4), and 84 computations of function values for Predictors B, C (m = 6). A function value is the error e_r (as a function of the unknown parameters) as given by (3). It can be computed fast over small rectangles, but it is more expensive to compute over large rectangles. A conservative estimate can be obtained from the assumption that the floorplan tree is a full binary tree. If the predictor is given in terms of n rectangles on a full slicing floorplan tree, a single function computation in each rectangle is as expensive as $\log_2 n$ function computations on the surrounding rectangle. With $n \approx 100$, this adds a factor of about seven, which brings the number of function values needed for Predictor A to 315, and for Predictors B, C to 588.

We conclude that the complexity grows logarithmically in the number of rectangles, and linearly in the number of pixels. Even though these asymptotic results may be the best possible, the constant factor is quite large. Actual run time of our partially optimized code takes about 15 minutes for a pair of 512 × 512 frames on a Sun workstation. On the other hand, since the time complexity is dominated by the performance of the error computation over large rectangular patches, standard

approximation techniques such as multiresolution or random sampling may be used to reduce the constant factor. Work in this direction is currently in progress.

VI. Experimental Results

This section describes experiments on real and simulated video sequences. The results verify the algorithms assumptions. They show that accurate predictors can be described in terms of a small number of parameters. Since most motion algorithms produce image flow (actually displacements) as output, we have extracted the displacement component as given by the functions U_x, U_y of Section IV.B. Specifically, the displacements shown were computed by:

$$\Delta x\,(x, y) = U_x(x, y, 1) - U_x(x, y, 0),$$
$$\Delta y\,(x, y) = U_y(x, y, 1) - U_y(x, y, 0).$$

Prediction error is displayed by taking frame difference and adding a constant value of 127.

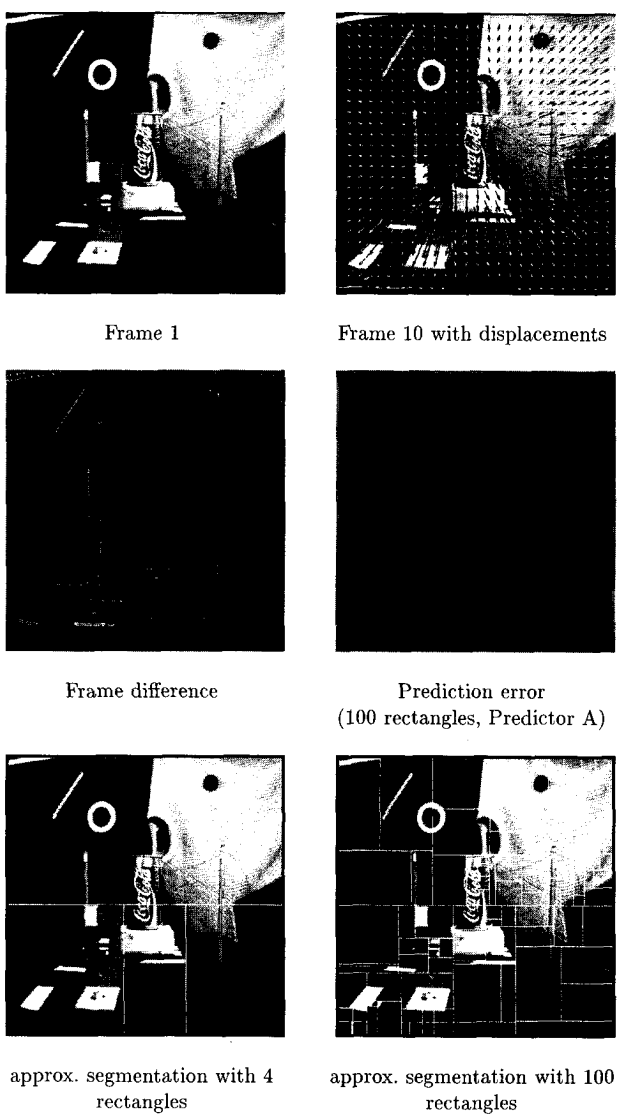


Figure 5: motion estimation and segmentation of the coke sequence

A. The Coke Sequence

The results of applying our algorithm to the coke sequence[2] (512 × 512 pixels) are shown in Fig. 5. The imagery contains a can (the FOE is centered on the can), and the camera axis is aligned with the straight translatory axis of motion. There are two pencils with wires drawn between them, a background with a varigated background, a sweater with a rougher texture, and other objects. Predictor A was applied to frames 1 and 10, and the displacements obtained with 100 rectangles are superimposed on frame 10. The results of the computed motion in the bottom left corner are clearly wrong, but most other displacements appear to be right in terms of both arrow direction and arrow length. The intermediate segmentation into four rectangles is also shown. Notice that these results were obtained with Predictor A, which assumes planar translation, rotation and scale in each rectangle. See Barron et al. [3] for the flow produced by other algorithms when applied to the same sequence.

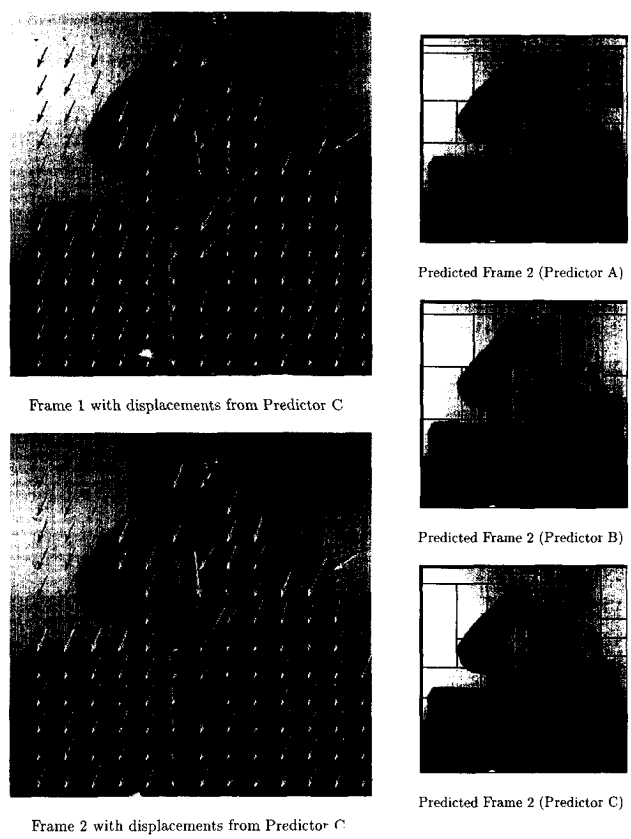


Fig. 6. The robot sequence—results of the three predictors.

B. The Robot Sequence

The results of the three predictors applied to the robot sequence (412 × 412 pixels) are shown in Fig. 6. The amount of motion between the two frames is quite large here, and many

2. The coke sequence is available from the Vision List Archives via anonymous ftp to ftp.teleos.com. This imagery was collected at NASA Ames Research Center and is provided courtesy of Dr. Banavar Sridhar.

details are lost, especially with Predictor A (e.g., the left hand). Predictors B and C perform much better. The displacements obtained from Predictor C are shown superimposed on frames 1 and 2. Notice that the same arrows are shown in both frames, so that what is pointed to by the arrow tail in frame 1 should be pointed to by the arrow head in frame 2. While there are some obvious misses, the overall results appear to be very good. For example, observe the arrow pointing to the middle of the forehead, the arrow pointing to the left side of the mouth, the arrow that points to the top of the left hand, etc.

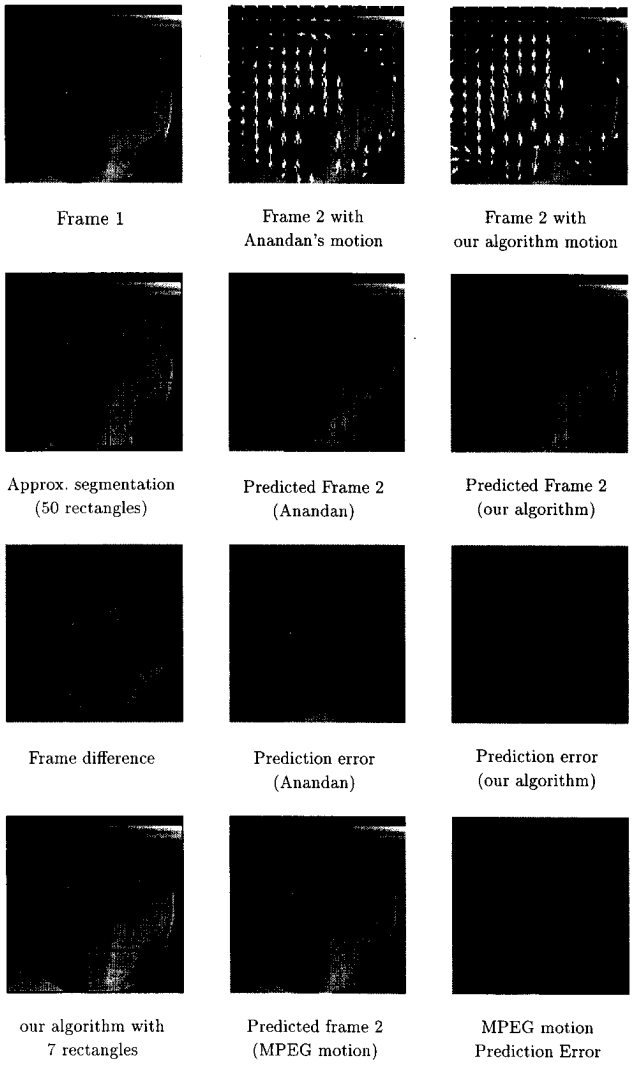


Fig. 7. Comparison with results obtained by Anandan's algorithm and the MPEG-1 encoding.

C. Comparison with Anandan's Algorithm

The code implementation of Anandan's algorithm as described in [2] can be obtained from the image library of the VISIONS group at the University of Massachusetts at Amherst. (It can be obtained via anonymous ftp to cicero.cs.umass.edu.) Anandan's code comes with two frames of a soccer ball sequence, shown in Fig. 7. The frame size is 128×128 pixels. Even though the displacements obtained by Anandan's algorithm look "cleaner," they appear to be less accurate and give worse prediction than the results obtained by our algorithm. The results shown were obtained with Predictor B and 50 rectangles.

D. Comparison with MPEG Motion Encoding

MPEG implementations estimate translational motion over 16×16 fixed size rectangles [15]. These estimates with 0.5 pixel accuracy can be computed by brute force search. We have implemented the MPEG motion algorithm and compared its results to ours. An example is shown in Fig. 7. The results shown in row three were obtained with 50 rectangles, compared with the 64 rectangles used by MPEG. Our algorithm achieves least squares error prediction comparable to MPEG with as few as seven rectangles (Predictor B), as shown in the bottom row. The corresponding motion can be encoded in 42 numbers, compared with the 128 numbers that are needed by MPEG. (This comparison ignores the quantization of these numbers, and their representation in a small number of bits.)

E. Accuracy on Synthetic Data

A recent paper by Barron et al. [3] gives empirical comparison of many existing motion estimation methods. The results on synthetic image sequences are compared according to the following error measure: Let (u, v) be the true motion at a point, and let (U, V) be the estimated motion at that point. The error is given by the following arc cosine:

$$\arccos \frac{uU+uV+1}{\sqrt{\left(u^2+v^2+1\right)\left(U^2+V^2+1\right)}}.$$

For each method the paper lists the average error angle, standard deviation, and density (i.e., the percentage of pixels for which the algorithm produces motion estimates). The test data and related software are available via anonymous ftp to ftp.csd.uwo.ca.

The results of applying our algorithm to the "Translating Tree" and the "Diverging Tree" (Frame 20) are shown in Fig. 8. The error measures are:

	Average Error Angle	Standard Deviation	Density	Min Err	Max Err
Translating Tree	0.23	0.31	100%	0	2.8
Diverging Tree	2.3	1.6	100%	0	13.6

In both cases these results are the *best* among the dense methods that produce estimates for *all* pixels. They tie for first in the translating tree case among all methods, and trail only the Fleet and Jepson and the modified Horn and Schunck in the diverging tree case. (However, these techniques produce measurements only for 28%–61% of the pixels.)

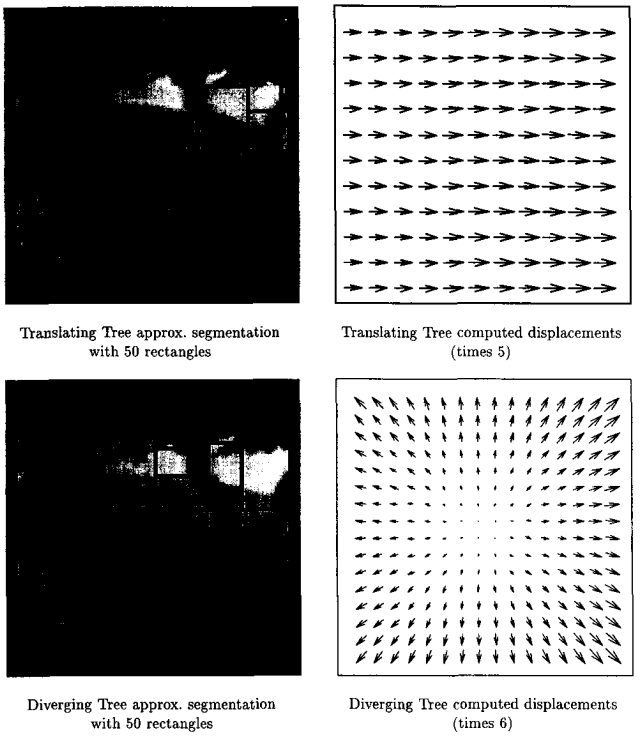


Fig. 8. Results on synthetic data.

F. High Resolution Video Sequences

Two examples of high resolution video sequences are shown in Figs. 9 and 10. (These sequences were obtained by anonymous ftp from ftp.ipl.rpi.edu.) Both results were computed with Predictor B.

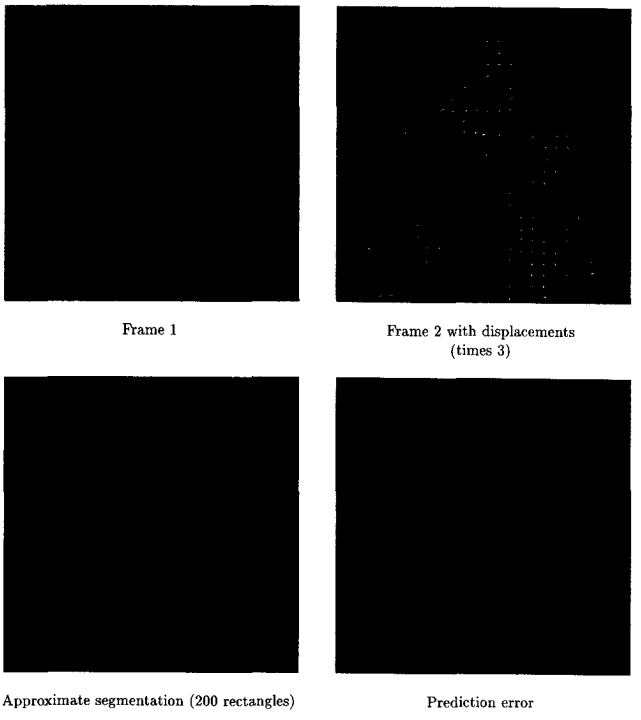


Fig. 9. Results for the surfside sequence.

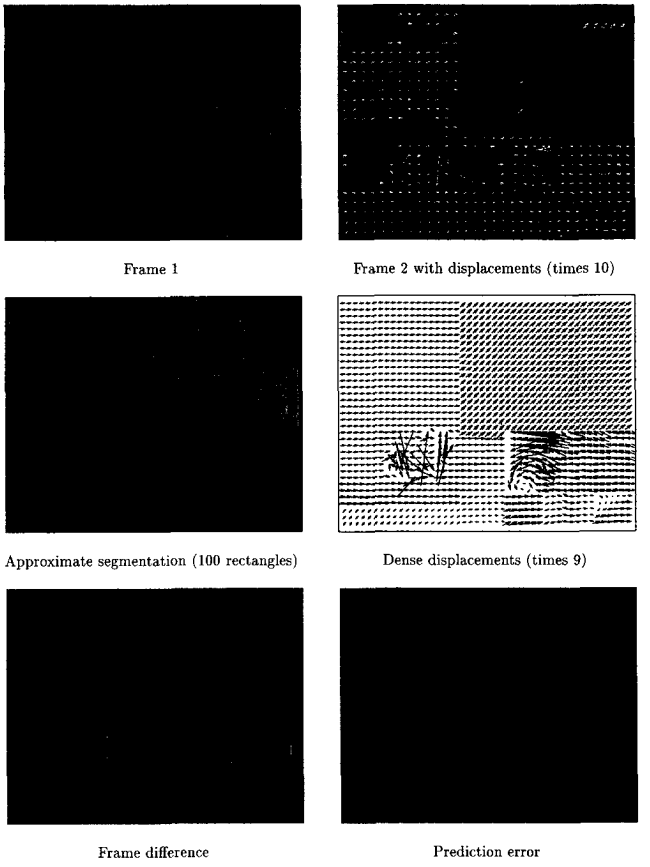


Fig. 10. Results for the CalTrain sequence.

The Surfside sequence is of size 512×512. It should be considered a "worst case" example, since the moving objects (waves and people) are non-rigid. Even though our assumptions about simple motion in each rectangle clearly do not hold in this case, the results indicate that they still give a good approximation to the true motion.

The CalTrain sequence is of size 400×512, and the motion in most parts is dominated by planar translation of large rigid objects. Most motion algorithms do well in these cases, except for areas near motion boundaries. The results produced here show accurate motion even near motion boundaries. Notice also how the algorithm refines the segmentation in areas of the scene with the more complex motion.

VII. Concluding Remarks

This paper proposes an alternative to the classic view of motion estimation as an optimization problem, defined in terms of multiple constraints. Instead, we argue that motion estimation can be viewed as a search for an accurate predictor of frames. In this framework, the inherent ambiguity in determining the right predictor is solved not by introducing additional constraints, but by the Occam-Razor principle, which suggests that the best predictor is the simplest. Simplicity can be defined in terms of the encoding length of the predictor. The effectiveness of the approach was demonstrated by a new motion estimation algorithm. The algorithm computes simulta-

neously motion estimates and a segmentation of the scene into rectangular patches.

The analogy between learning and motion estimation that led to the development of the proposed algorithm provides a firm theoretical basis. Still, our implementation depends on many assumptions that were verified experimentally. A partial list includes the following assumptions:

- Typical frames of video sequences can be segmented into a small number of rectangular patches with uniform motion. This assumption is much stronger than the assumption that a typical scene is composed of a few moving objects.
- The grey levels change over time in typical video sequences can be modeled by the predictors suggested in Section IV.B.
- The greedy recursive split algorithm of Section V.A produces a sufficient minimization of prediction error to enable accurate prediction.
- The prediction error as given by (9) can be locally approximated by a quadratic.

As a practical algorithm, our implementation of the Occam predictor in terms of the PARP minimization algorithm has several advantages over other motion estimation techniques. The three major advantages are the compactness of the representation, the partial segmentation, and the fact that the minimized error can be given as an arbitrary function. Thus, for example, it is possible to take the error as the sum of prediction errors over multiple frames, or over multiple color components. An apparent disadvantage of our implementation when compared to multi-resolution techniques (such as [4], [2], [5]) is its time complexity. However, as mentioned in Section V.C, it is very likely that the algorithm complexity can be reduced by multi-resolution or random sampling techniques.

The compact representation of motion produced by the Occam algorithm is of special importance for video coding and compression applications. On the other hand, our choice of predictors may not enable accurate recovery of 3D motion and structure. For example, approximating motion by pure translation rotation and scale implies motion of a rigid object parallel to the viewing plane, so that accurate 3D information cannot be computed.

ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation under Grant IRI-9309135.

REFERENCES

[1] G. Adiv, "Determining three-dimensional motion and structure from optical flow generated by several moving objects," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 7, pp. 384–401, 1985.

[2] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *Int'l J. Computer Vision*, vol. 2, pp. 283–310, 1989.

[3] J.L. Barron, D.J. Fleet, and S.S. Beauchemin, "Performance of optical flow techniques," *Int'l J. Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[4] J.R. Bergen and E.H. Adelson, "Hierarchical, computationally efficient motion estimation algorithm," *J. Optical Soc. Am.* A, vol. 4, no. 13, p. 48, 1987.

[5] J.R. Bergen, P. Anandan, K.J. Hanna, and R.Hingorani, "Hierarchical model-based motion estimation," *Computer Vision (ECCV '92)*, G. Sandini, ed., no. 588 in Lecture Notes in Computer Science, pp. 237–252. Springer-Verlag, 1992.

[6] J.R. Bergen, P.J. Burt, R. Hingorani, and S. Peleg, "A three-frame algorithm for estimating two-component image motion," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, pp. 886–896, Sept. 1992.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, pp. 377–380, 1987.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension, *J. ACM*, vol. 36, no. 4, pp. 929–965, Oct. 1989.

[9] M. Hestenes, *Conjugate Direction Methods in Optimization*. New York: Springer-Verlag, 1980.

[10] T. Hidaka and K. Ozawa, "Subjective assesment of redundancy-reduced moving images for interactive applications: Test methodology and report," *Signal Processing, Image Comm.*, vol. 2, no. 2, Aug. 1990.

[11] B.K. Horn, *Robot Vision*, MIT Press, 1986.

[12] B.K. Horn and B.G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[13] S.L. Horowitz and T. Pavlidis, "Picture segmentation by a tree traversal algorithm," *JACM*, vol. 23, pp. 368–388, 1976.

[14] M. Irani, B. Rousso, and S. Peleg, "Detecting and tracking multiple moving objects using temporal integration," *Computer Vision (ECCV'92)*, G. Sandini, ed., no. 588 in Lecture Notes in Computer Science, pp. 282–287. Springer-Verlag, 1992.

[15] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Comm. ACM*, vol. 34, no. 4, pp. 46–58, 1991.

[16] C.H. Lee and T. Huang, "Finding point correspondences and determining motion of a rigid object from two weak perspective views," *Proc. IEEE Conf. Computer Vision and Pattern Recognition, (CVPR '88)*, pp. 398–403, 1988.

[17] V. Markandey and B.E. Flinchbaugh, "Multispectal constraints for optical flow computation, *Proc. Third Int'l Conf. Computer Vision (ICCV'90)*, pp. 38–41, Dec. 1990.

[18] H.H. Nagel, "On change detection and displacement vector estimation in image sequences," *Pattern Recognition Letters*, vol. 1, pp. 55–59, 1982.

[19] H.H. Nagel, "Displacement vectors derived from second order intensity variations in image sequences," *Computer Vision, Graphics, and Image Processing*, pp. 85–117, 1983.

[20] H.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, no. 5, pp. 565–593, 1986.

[21] J. Pearl, *"Heuristics: Intelligent Search Strategies for Computer*. Reading, Mass.: Addison-Wesley, 1984.

[22] A. Pentland, "Photometric motion," *Proc. Third Int'l Conf. Computer Vision (ICCV'90)*, pp. 178–187, Dec. 1990.

[23] J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

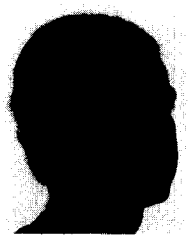
[24] J. Rissanen, "Universal coding, information, prediction, and estimation," *IEEE Trans. Information Theory*, vol. 30, no. 4, pp. 629–636, 1984.

[25] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, Mass.: Addison-Wesley, 1990.

[26] R. Srinivasan and R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Trans. Comm.*, vol. 33, no. 8, pp. 888–896, Aug. 1985.

[27] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Information and Control*, vol. 57, pp. 91–101, 1983.

[28] J. Weng, N. Ahuja, and T.S. Huang, "Matching two perspective views," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 8, pp. 806–825, Aug. 1992.



Haim (Shvaytser) Schweitzer received the BSc degree from Tel-Aviv University, Israel, in 1982, and the PhD degree from the Hebrew University of Jerusalem, Israel, in 1986. He subsequently served as a Weizmann postdoctorate at the University of Texas at Austin, Columbia University, and Cornell University, and as a member of the technical staff at DSRC SRI in Princeton, N.J. Dr. Schweitzer has been an assistant professor at the University of Texas at Dallas since January 1991. His research interests are in artificial intelligence, with emphases on computer vision, computational learning, and artificial neural networks.