# Fast Gradient Methods Based on Global Motion Estimation for Video Compression

Yosi Keller and Amir Averbuch

*Abstract*—**This paper presents a fast global motion estimation (GME) algorithm based on gradient methods (GM), which can be used for real-time applications, such as in MPEG4 video compression. This approach improves the existing state-of-the-art GME algorithms by introducing two major modifications: first, only a small subset (down to 3%) of the original image pixels is used in the estimation process. Second, an interpolation-free formulation of the basic GM is derived, further decreasing the computational complexity. Experimental results show no loss of GME accuracy and compression efficiency compared to the MPEG-4 verification model, while reducing the computation complexity of the GME by a factor of 20.**

*Index Terms*—**Global motion estimation, gradient methods, MPEG-4, sprites, video coding.**

## I. Introduction

MPEG-4 is a new video compression standard providing core technologies for efficient storage, transmission, and manipulation of video data in multimedia environments [11], [20]. Motion estimation algorithms calculate the motion between successive video frames and predict the current frame from previously transmitted frames using the motion information [7]. Global motion estimation (GME) algorithms estimate a single parametric motion model [9] for the whole frame, which can be used within MPEG-4 to produce either *static* or *dynamic sprites*. Static sprites [17] are mosaics containing the visual information of the objects which were visible over the sequence. While various mosaic generation algorithms were developed [3], [4], [6], [14], their applicability to general purpose video compression applications is limited by the significant delay incurred by frames accumulation and mosaic image coding (as intra frames) [18]. Furthermore, the 8-parameters projective motion model used by the MPEG-4 coding standard is suitable for a restricted range of camera motions [4]. Thus, each static sprite can be only used for a single short video segment. Therefore, this paper concentrates on dynamic sprites while its results are also applicable to static sprites. The dynamic sprite [21] coding scheme utilized by the MPEG-4 verification model, estimates the motion between consecutive frames using a six-parameters affine motion model. A sprite is generated every time step by warping the previous frame according to the motion parameters and used as a reference frame [19]. Further improvement is achieved by first estimating both the global and local motions (using block matching) and then coding each macro-block using the motion estimation mode which results in a lower prediction error [15], [16].

A comprehensive comparative survey by Barron *et al.* [1] found the family of gradient-based motion estimation methods (GM), originally proposed by Horn and Schunck [2], to perform especially well. The purpose of the GM algorithm is to estimate the parameters vector $p$ associated with the *parametric image registration* problem [3]. A critical implementation issue concerning the GME, is its significant computational complexity, making it useless for real-time encoding application, especially when implemented on low-power devices such as PDAs and cellular phones. This paper offers two modifications to the GM algorithm, reducing its computational complexity by 20 times. First, only a small, selective subset of the image pixels named *dominant pixels*, is used by the GM algorithm. Second, the interpolation-free formulation of the GM algorithm (IFGM) [13] allows for further complexity reduction. These two algorithms are complexity-wise complementary: each of them reduces the complexity of a different component within the original GM algorithm. Experimental results demonstrate the significant complexity reduction while maintaining the GME accuracy and video compression efficiency.

The regular GM based GME algorithm is presented in Section II, while the selective-integration-based GM (SIGM) and IFGM are introduced in Sections III and IV, respectively. The resulting algorithm, fast GM, is presented in Section V, while experimental results are given in Section VI.

## II. Gradient-Method-Based Motion Estimation

Let the two images $I_1(x, y)$ and $I_2(x, y)$ be related to each other by a parametric coordinates transform: $x_i^{(2)} = f_1(p, x_i^{(1)}, y_i^{(1)})$, $y_i^{(2)} = f_2(p, x_i^{(1)}, y_i^{(1)})$, where $(x_i^{(1)}, y_i^{(1)})$ and $(x_i^{(2)}, y_i^{(2)})$ are the corresponding locations under the transform in $I_1$ and $I_2$, respectively; $p$ is the parameters set, and $f_1$ and $f_2$ are defining functions; $i$ is just the label of a location. For example, for the affine motion model

$$\begin{aligned} x_i^{(2)} &= a \cdot x_i^{(1)} + b \cdot y_i^{(1)} + c \\ y_i^{(2)} &= d \cdot x_i^{(1)} + e \cdot y_i^{(1)} + f \\ p &= (a, b, c, d, e, f). \end{aligned} \tag{2.1}$$

GM methodology [8], [10] estimates the motion parameters $p$ by minimizing the intensity discrepancies between the input

Manuscript received March 16, 2002; revised September 9, 2002. This paper was recommended by Associate Editor L. Onural.

Y. Keller is with the Computer Science Department, The Technion, Haifa 32000, Israel (e-mail: keller@post.tau.ac.il).

A. Averbuch is with the Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel (e-mail: amir@math.tau.ac.il).

Digital Object Identifier 10.1109/TCSVT.2003.811360

images as

$$p^* = \arg\min_p \cdot \sum_{(x_i^{(1)},\, y_i^{(1)}) \in S} \left(I_1\left(x_i^{(1)}, y_i^{(1)}\right) - I_2\left(x_i^{(2)}, y_i^{(2)}\right)\right)^2 \quad (2.2)$$

where $S$ is the set of coordinates over which the minimization is carried.

The solution of (2.2) is based on a pixel-wise first-order Taylor expansion of $I_1$ in terms of $I_2$ around $p = 0$ as

$$\begin{aligned} I_1\left(x_i^{(1)}, y_i^{(1)}\right) &= I_2\left(x_i^{(2)}\left(0, x_i^{(1)}, y_i^{(1)}\right), y_i^{(2)}\left(0, x_i^{(1)}, y_i^{(1)}\right)\right) + \sum_{Pj \in p} \\ &\cdot \left.\frac{\partial I_2\left(x_i^{(2)}\left(p, x_i^{(1)}, y_i^{(1)}\right), y_i^{(2)}\left(p, x_i^{(1)}, y_i^{(1)}\right)\right)}{\partial p_j}\right|_{p=0} p_j. \end{aligned} \quad (2.3)$$

Equation (2.3) can be considered the linearization step of a Gauss–Newton nonlinear minimization [22] of (2.2). By gathering the pixel-wise equations similar to (2.3), an equation set is formed and solved for $p$ [3], [5]

$$p = \left(H^T H\right)^{-1} H^T I_t \quad (2.4)$$

where

$$I_t = ((I_t)_1 \cdots (I_t)_n)^T \quad (2.5)$$

and

$$\begin{aligned} (I_t)_i =& I_1\left(x_i^{(1)}, y_i^{(1)}\right) \\ &- I_2\left(x_i^{(2)}\left(p, x_i^{(1)}, y_i^{(1)}\right), y_i^{(2)}\left(p, x_i^{(1)}, y_i^{(1)}\right)\right). \end{aligned} \quad (2.6)$$

The partial derivatives according to the motion parameters are calculated using the derivative chain rule

$$\begin{aligned} H_{i,j} =& \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j} \quad (2.7) \\ =& \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial x_i^{(2)}} \frac{\partial x_i^{(2)}\left(x_i^{(1)}, y_i^{(1)}, p\right)}{\partial p_j} \\ &+ \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial y_i^{(2)}} \frac{\partial y_i^{(2)}\left(x_i^{(1)}, y_i^{(1)}, p\right)}{\partial p_j} \quad (2.8) \end{aligned}$$

where $(x_i^{(2)}, y_i^{(2)})$ are related to the parametric motion model used. For the affine motion model we get

Due to the nonlinear nature of (2.2), it is solved iteratively as in (2.4). The basic GM iteration, which is marked as "*single iteration*," and the iterative refinement phase are presented in Fig. 1.

In order to improve the convergence properties, a standard Gaussian pyramid [3] is constructed using scaling factors of two or three [6], [10]. Hence, the GM algorithm starts at the coarsest resolution scale of the pyramid, then follows the subsequent levels in a coarse-to-fine approach. At each resolution scale, (2.4) is iterated until a maximal number of iterations is reached or the magnitude of the update of translation parameters reaches a predetermined threshold. Finally, when the procedure stops at the finest resolution scale, the final motion parameters are obtained. In video-compression applications, the induced relative motion is usually small, therefore, less than ten iterations are needed for an accurate registration and convergence. In situations where larger motion is anticipated, a bootstrapping procedure can be applied [9].

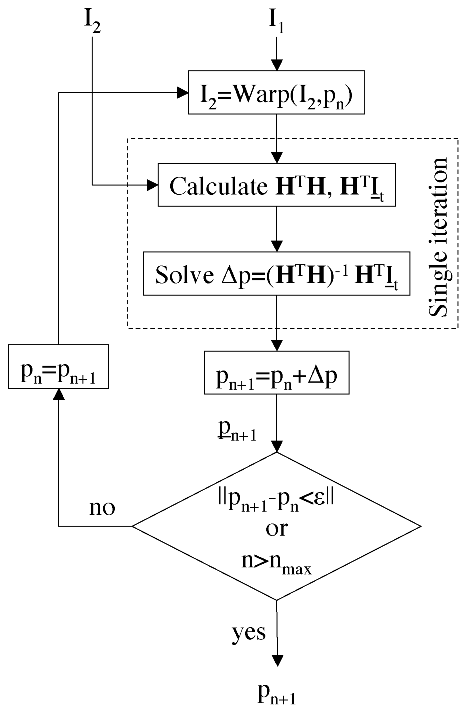


Fig. 1. Block diagram of the basic and iterative GM formulations. For $n = 0$, $p_0$ is given as an initial guess and $\Delta p$ is the iterative update after each iteration.

## III. SIGM

The evaluation of (3.4) at each GM iteration uses $S$, the complete set of pixels common to $I_1$ and $I_2$. Hence, the resulting equation is highly overdetermined. At QCIF[1] frame size (176 $\times$ 144 pixels) there are 25 344 equations, as opposed to six or eight unknown parameters related to affine and perspective motion models, respectively. The SIGM evaluates (3.4) using a small subset (down to 3%) of $S$. This set is denoted $\hat{S}$. Following the GM scheme in Section III-A, the SIGM algorithm uses a multiresolution pyramid of the input images, where the pixel set $\hat{S}$ is chosen at the coarsest resolution scale, and tracked using a coarse-to-fine formulation. The pixel subset selection process is described in Section III-A. No other modifications are made to the procedure in Section II. It is used in the initialization and

[1]An industry acronym for Quarter Common Intermediate Format, a video-conferencing format that specifies data rates of 30 frames per second (fps), with each frame containing 144 lines and 176 pixels per line. This is one fourth the resolution of full CIF.

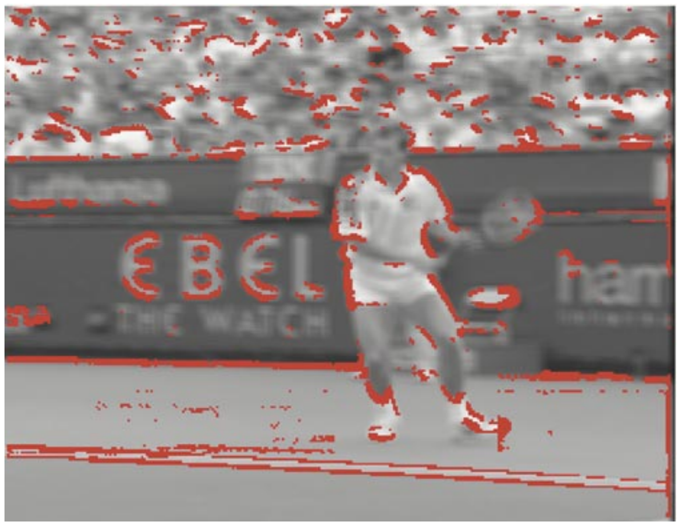


Fig. 2. Example of the locations of the subset of pixels used for the global motion computation. The pixel set is superimposed on the first frame of the Stefan video sequence.

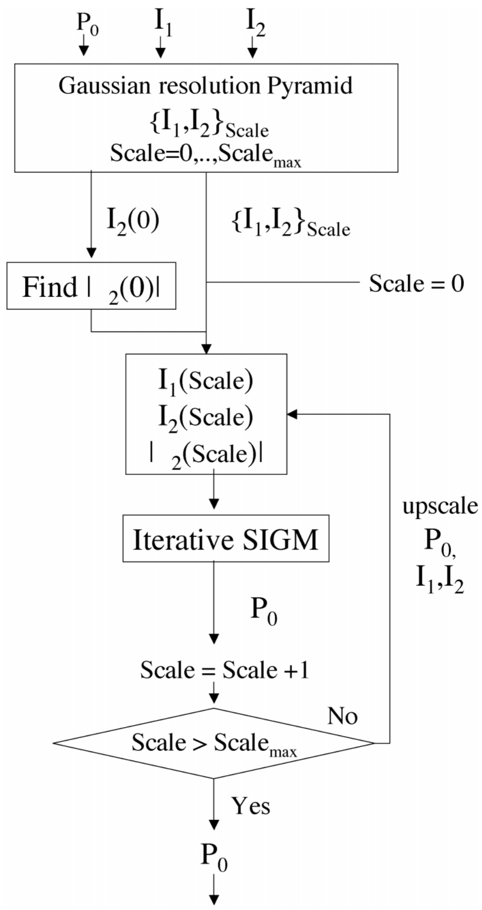


Fig. 3. SIGM multiscale image-registration flowchart. The scheme utilizes the iterative registration presented in Fig. 4.

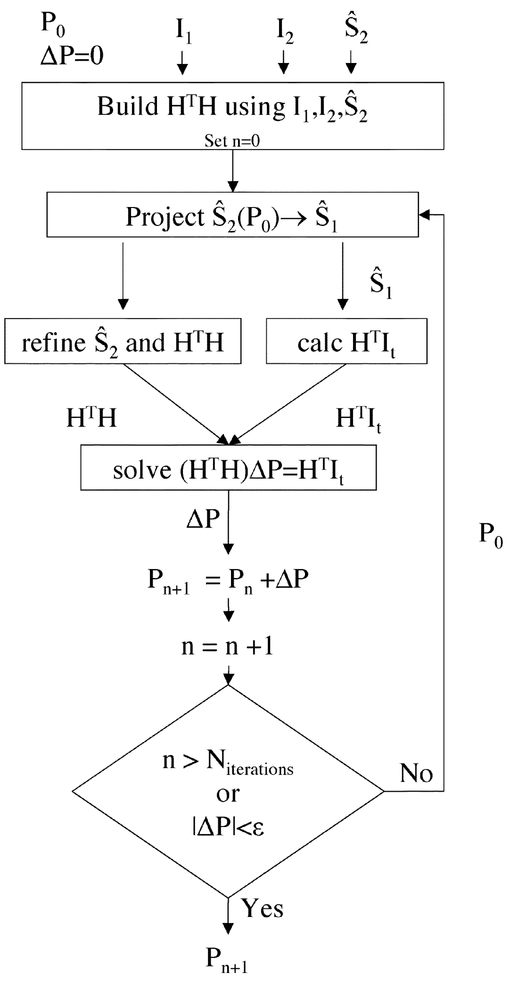


Fig. 4. Flowchart of the SIGM iterative image registration, following the procedure described in Section III-C.

multiscale embedding of the SIGM, which is presented in Section III-B and illustrated in Fig. 3. The iterative formulation is described in Section III-C and Fig. 4.

## A. Pixel Subset Selection

The pixel subset $\hat{S}$ is chosen by finding the pixels having the largest gradient magnitude. In order to avoid numerical instabilities caused by the concentration of the subset $\hat{S}$ in small image regions. The image is divided into 100 subregions, where the top 10% are chosen at each subregion. A similar approach was used for feature tracking by Dellaert *et al.* [12], while Wei *et al.* [23] improved the GM robustness by using selective integration where the selection criterion was based on temporal gradient sorting. This procedure was also used by [9]. Fig. 2 illustrates the proposed pixel selection process, which was applied to the first frame of the Stefan sequence. The uniform distribution of the feature points is evident.

## B. SIGM Multiscale Scheme

This section presents the multiscale SIGM registration scheme which uses the iterative refinement algorithm in Section III-C

1) Similar to Section II, a resolution pyramid of the input images $I_1$ and $I_2$ is constructed.

2) At the lowest resolution level $\mathrm{Scale} = 0$, $N$ pixels in $I_2$, having the largest gradient magnitude are added to the pixel set $\hat{S}_2(\mathrm{Scale})$, following the procedure described in Section III-A and Fig. 2.
3) The pixel set $\hat{S}_2(\mathrm{Scale})$ is used as an input to the iterative refinement algorithm in Section III-C, where the initial estimate of the motion can be either set to zero or calculated according to the motion of the previous frames.
4) The pixel set $\hat{S}_2(\mathrm{Scale})$ and the result of step 2 are upscaled and used as an input to the calculation of steps 2 at a higher resolution scale, until the original image size is reached.

## C. SIGM Iterative Refinement

At each resolution scale, the initial estimate of the registration is refined using the following procedure.

1) At the first iteration in each resolution scale $(n = 0)$, the matrix $(H^T H)$ is calculated according to (3.6) and (3.7) using the pixel set $\widehat{S_2}$.
2) The pixel set $\widehat{S_1}$, which is a warping of $I_1$ toward the pixel set $\widehat{S_2}$, is calculated using the inverse of the current estimate $p_n^{-1}$, $n \geq 0$. For $n = 0$, $p_0$ is given as input.
3) If a pixel in $\widehat{S_2}$, does not have a corresponding pixel in $I_1$, (its coordinates fall outside the $I_1$ frame) it is extracted from $\widehat{S_2}$ and its contribution to the matrix $(H^T H)$ is subtracted.
4) The vector $(H^T I_t)$ is calculated according to $\widehat{S_1}$ and $\widehat{S_2}$.
5) Equation (3.4) is solved for $\Delta p$ using $(H^T H)$ and $(H^T I_t)$ calculated above.
6) $\Delta p$, the outcome of step 2, is used to update the solution

$$p_{n+1} = \Delta p + p_n, \qquad n \geq 0.$$

7) Step 2 is repeated until one of the following stopping criteria is met:
   a) At most $N_{\max}$ iterations were performed

   or

   b) The translation parameters within the updated term $\Delta p$ reach a predetermined threshold which corresponds to the required registration accuracy. A threshold of 0.1 pixel was used as a practical limit to the motion estimation's accuracy [13].
8) The process is stopped if the translation parameters within the updated term $\Delta p$ reach a predetermined threshold which corresponds to the required registration accuracy. A threshold of 0.1 pixel was used as a practical limit to the motion estimation accuracy [13].

## D. Complexity Analysis

Next we provide an estimation of the GM algorithm's complexity and a comparison to the SIGM. Let $C_{\mathrm{GM(Scale)}}$ be the total complexity of a regular GM at a certain resolution scale $\mathrm{Scale}$, then

$$C_{\mathrm{GM(Scale)}} = K_{\mathrm{GM}} |S_2| N_{\mathrm{Iterations}} \tag{3.1}$$

where

| | |
|---|---|
| $K_{\mathrm{GM}}$ | number of operations per pixel; |
| $\lvert S_2\rvert$ | number of pixels in the set $S_2$; |
| $N_{\mathrm{Iterations}}$ | number of iterations per resolution scale. |

Hence, the total complexity of the iterative multiscale process becomes

$$\begin{aligned} C_{\mathrm{GM}} &= \sum_{m=0}^{N_{\mathrm{scales}}-1} C_{\mathrm{GM}}(m) \\ &= \sum_{m=0}^{N_{\mathrm{scales}}-1} (\mathrm{ScaleStep}^2)^m \cdot C_{\mathrm{GM}}(0) \\ &= N_{\mathrm{Iterations}} \cdot K_{\mathrm{GM}} \cdot |S_2(0)| \\ &\quad \cdot \frac{(\mathrm{ScaleStep})^{2N_{\mathrm{scales}}} - 1}{(\mathrm{ScaleStep})^2 - 1} \end{aligned}$$

| | |
|---|---|
| $C_{\mathrm{GM}}(0)$ | GM complexity at the coarsest resolution scale; |
| $\mathrm{ScaleStep}$ | resolution scale step (the image's downscaling factor in each dimension); |
| $\mathrm{ScaleStep}^2$ | the ratio of the number of pixels between successive resolution scales. |
| $\lvert S_2(0)\rvert$ | the number of pixel used for the GM estimation at the lowest resolution scale. |

Therefore, the SIGM significantly reduces the GM complexity.

1) The matrix $(H^T H)_{\mathrm{SIGM}}$ is calculated using a small subset of pixels $\widehat{S_2}$, which is much smaller than the pixel set $S_2$ used by the GM in Section II. The complexity reduction is

$$\frac{C_{(H^T H)_{\mathrm{SIGM}}}}{C_{(H^T H)_{\mathrm{GM}}}} = \frac{\left|\widehat{S_2}\right|}{|S_2|}. \tag{3.2}$$

2) $I_2$ remains static throughout the iterative solution process ($I_1$ is being warped). Hence, there is no need to recalculate the matrix $(H^T H)_{\mathrm{SIGM}}$ in each iteration. It has to be calculated just once at the first iteration of each resolution scale.

In the SIGM case, the matrix $(H^T H)$ has to be estimated just once, while $(H^T I_t)$ has to be evaluated at each iteration

$$\begin{aligned} C_{\mathrm{SIGM(Scale)}} = K_{\mathrm{GM}}^{(H^T H)} \left|\widehat{S_2}\right| \left(1 + \Delta\widehat{S_2}\right) \\ + K_{\mathrm{GM}}^{(H^T I_t)} \left|\widehat{S_2}\right| N_{\mathrm{Iterations}}, \qquad \forall rs \end{aligned} \tag{3.3}$$

where

| | |
|---|---|
| $K_{\mathrm{GM}}^{(H^T H)}$ | complexity of estimating $(H^T H)$ using a single pixel; |
| $K_{\mathrm{GM}}^{(H^T I_t)}$ | complexity of estimating $(H^T I_t)$ using a single pixel; |
| $\Delta\widehat{S_2}$ | the relative change in the size of the set $\widehat{S_2}$ due to pixels in $S_2$ that do not have a corresponding pixel in $I_1$; |
| rs | resolution scale. |

The total SIGM complexity becomes

$$C_{\mathrm{SIGM}} = \sum_{m=0}^{N_{\mathrm{scales}}-1} C_{\mathrm{SIGM}}(m). \tag{3.4}$$

Following (3.3), $C_{\text{SIGM}}$ is not a function of the resolution scale $Scale$; therefore, we have

$$\begin{aligned} C_{\text{SIGM}} &= \sum_{m=0}^{N_{\text{scales}}-1} C_{\text{SIGM}}(0) \\ &= C_{\text{GM}}(0) \cdot N_{\text{scales}}. \end{aligned} \tag{3.5}$$

For a typical global motion estimation in a video sequence (320 × 240) using three dyadic resolution scales ($N_{\text{scales}} = 2$, $\text{ScaleStep} = 2$) and ten iterations ($N_{\text{Iterations}} = 10$), we get

$$\begin{aligned} |S_2(0)| &= 320 \times 240/4 = 19\,200 \\ \left|\widehat{S_2}(0)\right| &= |S_2(0)| \cdot 10\% = 1920 \\ \Delta\widehat{S_2} &= 10\%, \\ K_{\text{GM}}^{(H^T I_t)} &= (\text{no. motion parameters}) \\ &\quad + (4 \text{ multiplications to interpolate } I_t) \\ K_{\text{GM}}^{(H^T H)} &= \text{no. multiplications needed to evaluate} \\ &\quad K_{\text{GM}}^{(H^T H)}, \text{ since } (H^T H) \text{ is symmetric.} \end{aligned} \tag{3.6}$$

The complexity analysis shown in Table I demonstrates a significant complexity reduction achieved by the SIGM especially for the advanced motion models, such as the affine and the projective. It should be noted that for the affine motion model, we have

$$K_{\text{GM}}^{(H^T I_t)} \cdot N_{\text{Iterations}} \gg K_{\text{GM}}^{(H^T H)}. \tag{3.7}$$

Hence, the affine motion estimation complexity is dominated by the complexity of evaluating $H^T I_t$, which can be reduced using the interpolation-free GM algorithm described in Section IV.

## IV. Interpolation-Free Motion Estimation

In order to reduce the complexity of the GM algorithms, we reformulate (4.3) such that no warping is needed for the evaluation of $I_1(x_i^{(1)}, y_i^{(1)})$ while maintaining the same accuracy [13]. We start by rewriting the regular GM formulation for the one-dimensional (1-D) case in Section IV-A. Then, the 1-D interpolation-free reformulation for translational motion is presented in Section IV-A, while its extension to two-dimensional (2-D) and general motion models is shown in Section IV-C.

### *A. 1-D Gradient Methods Formulation*

We consider the registration of two 1-D discrete signals $I_1(x)$ and $I_2(x)$ sharing some common interval. Using a 1-D formulation of Section II, we get

$$I_1\left(x_i^{(1)}\right) = I_2\left(x_i^{(2)}\right) \tag{4.1}$$

where $x_i^{(1)}$ and $x_i^{(2)}$ are the coordinates of the $i$th sample common to $I_1$ and $I_2$.

Assuming relative translational 1-D motion, we have

$$x_i^{(1)} = x_i^{(2)} + \Delta x. \tag{4.2}$$

TABLE I
PERFORMANCE COMPARISON BETWEEN THE SIGM AND GM SHOWING THE SIGNIFICANT COMPUTATIONAL COMPLEXITY REDUCTION ACHIEVED BY THE SIGM

| | Translation | Affine | Projective |
|---|---|---|---|
| $\left(K_{GM}^{(\underline{\underline{H}}^t\underline{\underline{H}})}, K_{GM}^{(\underline{\underline{H}}^t\underline{I_t})}, K_{GM}\right)$ | $(3, 7, 10)$ | $(21, 10, 31)$ | $(46, 12, 58)$ |
| $GM$ | $9.6 \cdot 10^6$ | $30 \cdot 10^6$ | $56 \cdot 10^6$ |
| $SIGM$ | $14 \cdot 10^4$ | $23 \cdot 10^4$ | $33 \cdot 10^4$ |
| Complexity gain: $\frac{C_{GM}}{C_{SIGM}}$ | $\approx 70$ | $\approx 130$ | $\approx 170$ |

Similar to Section II, (4.1) is solved by expanding $I_2$ in a first-order Taylor expansion and solving for $\Delta x$

$$I_1\left(x_i^{(1)}\right) = I_2\left(x_i^{(2)}\right) + \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x}\Delta x \tag{4.3}$$

where $I_1(x_i^{(1)})$ is evaluated at nonintegral grid points using interpolation.

By gathering the pixel-wise equations

$$\underbrace{I_1\left(x_i^{(1)}\right) - I_2\left(x_i^{(2)}\right)}_{=\widehat{I_t}} = \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x}\Delta x \tag{4.4}$$

an equation set is formulated

$$Hp = I_t \tag{4.5}$$

and solved in the least-square sense similar to (4.4).

### *B. 1-D Interpolation-Free Gradient Methods Formulation*

Next we reformulate (4.3) to account for nonintegral coordinate values. Let $x_i^{(1)}$ be the initial estimate of the $i$th common pixel, at a certain GM iteration. Then

$$x_i^{(1)} \triangleq \left\lfloor x_i^{(1)} \right\rfloor + \varepsilon_x, \qquad x_i^{(1)}, \varepsilon_x > 0. \tag{4.6}$$

Thus, instead of evaluating $I_1$ at $x = x_i^{(1)}$, we evaluate it at $x = \hat{x}$

$$\hat{x}_i^{(1)} \triangleq \begin{cases} \left\lfloor x_i^{(1)} \right\rfloor & |\varepsilon_x| \leqslant 0.5 \\ \left\lfloor x_i^{(1)} \right\rfloor + 1 & |\varepsilon_x| > 0.5 \end{cases} \tag{4.7}$$

$$I_1\left(\hat{x}_i^{(1)}\right) = I_2\left(x_i^{(2)}\right) + \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x}\widehat{\Delta x} \tag{4.8}$$

where

$$\widehat{\Delta x} \triangleq \Delta x + \widehat{\varepsilon_x} \tag{4.9}$$

and $\widehat{\varepsilon_x}$ is the coordinate shift

$$\widehat{\varepsilon_x} \triangleq \hat{x}_i^{(1)} - x_i^{(1)}. \tag{4.10}$$

Substituting (4.9) into (4.8), we get

$$\begin{aligned} I_1\left(\hat{x}_i^{(1)}\right) &= I_2\left(x_i^{(2)}\right) + \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x} \cdot \widehat{\varepsilon_x} \\ &= I_2\left(x_i^{(2)}\right) + \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x} \cdot \Delta x + \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x} \cdot \widehat{\varepsilon_x} \end{aligned} \tag{4.11}$$

$$\underbrace{I_1\left(\hat{x}_i^{(1)}\right) - I_2\left(x_i^{(2)}\right) - \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x} \cdot \widehat{\varepsilon_x}}_{=\widehat{I_t}} = \frac{\partial I_2\left(x_i^{(2)}\right)}{\partial x} \cdot \Delta x. \tag{4.12}$$

By comparing (4.3) and (4.12), we note that the left-hand-side of (4.3) uses two floating-point multiplications to interpolate the value of $I_1(x_i^{(1)})$ at nonintegral coordinates, while the interpolation-free formulation in (4.12) uses single floating-point multiplication.

## *C. Generalization to 2-D Signals and General Motion Models*

Equation (4.3) is reformulated to estimate the input image at integral coordinates

$$\begin{aligned} I_1(\hat{x}, \hat{y}) = I_2\left(x_i^{(2)}, y_i^{(2)}\right) &+ \sum_{\substack{p_j \in p \\ p_j \notin \Delta x, \Delta y}} \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j} p_j \\ &+ \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta x)} \widehat{\Delta x} + \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta y)} \widehat{\Delta y} \end{aligned} \tag{4.13}$$

where the shifted coordinates $(\hat{x}, \hat{y})$ are defined similarly to (4.9)

$$\begin{aligned} \hat{x} &\triangleq \begin{cases} \left\lfloor x_i^{(1)} \right\rfloor & |\varepsilon_x| \leqslant 0.5 \\ \left\lfloor x_i^{(1)} \right\rfloor + 1 & |\varepsilon_x| > 0.5 \end{cases} \\ \hat{y} &\triangleq \begin{cases} \left\lfloor y_i^{(1)} \right\rfloor & |\varepsilon_y| \leqslant 0.5 \\ \left\lfloor y_i^{(1)} \right\rfloor + 1 & |\varepsilon_y| > 0.5 \end{cases} \end{aligned} \tag{4.14}$$

and $(\widehat{\Delta x}, \widehat{\Delta y})$ is defined similarly to (4.9)

$$\begin{aligned} \widehat{\Delta x} &\triangleq \Delta x + \hat{x} - \left\lfloor x_i^{(1)} \right\rfloor \\ \widehat{\Delta y} &\triangleq \Delta y + \hat{y} - \left\lfloor y_i^{(1)} \right\rfloor . \end{aligned} \tag{4.15}$$

Substituting (4.14) and (4.15) into (4.13), we get

$$\begin{aligned} I_1(\hat{x}, \hat{y}) &- I_2\left(x_i^{(2)}, y_i^{(2)}\right) \\ &= \sum_{\substack{p_j \in p \\ p_j \notin \Delta x, \Delta y}} \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j} p_j \\ &\quad + \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta x)} \left(\Delta x + \hat{x} - \left\lfloor x_i^{(1)} \right\rfloor\right) \\ &\quad + \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta y)} \left(\Delta y + \hat{y} - \left\lfloor y_i^{(1)} \right\rfloor\right) \\ &= \sum_{p_j \in p} \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j} p_j \\ &\quad + \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta x)} \left(\hat{x} - \left\lfloor x_i^{(1)} \right\rfloor\right) \\ &\quad + \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta y)} \left(\hat{y} - \left\lfloor y_i^{(1)} \right\rfloor\right) \end{aligned} \tag{4.16}$$

and

$$\widehat{I_t} = \sum_{p_j \in p} \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j} p_j \tag{4.17}$$

where

$$\begin{aligned} \widehat{I_t} \triangleq I_1(\hat{x}, \hat{y}) - I_2\left(x_i^{(2)}, y_i^{(2)}\right) &- \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta x)} \left(\hat{x} - \left\lfloor x_i^{(1)} \right\rfloor\right) \\ &- \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta y)} \left(\hat{y} - \left\lfloor y_i^{(1)} \right\rfloor\right) . \end{aligned} \tag{4.18}$$

## *D. Algorithm Flow*

1) The matrix $(H^T H)_{\mathrm{IFGM}}$ is identical to $(H^T H)_{\mathrm{GM}}$ computed in the regular GM algorithm using $I_2$. The computation follows (4.7)

$$\left(H^T H\right)_{k,j}^{I_2} = \sum_i \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_k} \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial p_j}. \tag{4.19}$$

2) $I_{ti}$ is calculated according to (4.6) and it is shifted according to (4.18)

$$\begin{aligned} \left(\widehat{I_t}\right)_i = (I_t)_i &- \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta x)} \left(\hat{x} - \left\lfloor x_i^{(1)} \right\rfloor\right) \\ &- \frac{\partial I_2\left(x_i^{(2)}, y_i^{(2)}\right)}{\partial (\Delta y)} \left(\hat{y} - \left\lfloor y_i^{(1)} \right\rfloor\right) . \end{aligned} \tag{4.20}$$

3) Equation (4.20) is solved for $p$

$$\left(H^T H\right) p = H^T \widehat{I_t}. \tag{4.21}$$

4) The interpolation-free GM returns $p$ as its result.

## *E. Complexity Analysis*

Following (4.18) and Table II, we established that the interpolation-free GM reduced the complexity related to the evaluations the vector $(H^T I_t)$ in (4.4). Two floating-point multiplications per entry are needed, rather than four. This improve-

TABLE II
PERFORMANCE COMPARISON OF AFFINE MOTION. THE REGULAR GM WAS TAKEN AS THE REFERENCE CPU COMPLEXITY. THE COMPLEXITY RELATED TO THE CALCULATION OF $H^T I_t$ IS REDUCED BY 20%

| | $C_{\underline{\underline{H}}^t \underline{I_t}}$ | $C_{\underline{\underline{H}}^t \underline{I_t}}$ $N_{Param} = 6, N_{Iterations} = 10$ $\lvert S_2 \rvert = 176 \times 144, \frac{\lvert \widehat{S_2} \rvert}{\lvert S_2 \rvert} = 10\%$ |
|---|---|---|
| GM | $\lvert S_2 \rvert N_{Iterations} (N_{Param} + 4)$ | $2.5 \cdot 10^6$ |
| IFGM | $\lvert S_2 \rvert N_{Iterations} (N_{Param} + 2)$ | $2.0 \cdot 10^6$ |
| $\frac{C_{IFGM}}{C_{GM}}$ | $\frac{N_{Param}+2}{N_{Param}+4}$ | 0.8 |

TABLE III
PERFORMANCE COMPARISON BETWEEN THE SIGM AND GM: APPROXIMATED NUMBER OF MULTIPLICATIONS NEEDED FOR THE SCENARIO INTRODUCED IN TABLE I

| | Affine | Projective |
|---|---|---|
| $GM$ | $30 \cdot 10^6$ | $56 \cdot 10^6$ |
| $SIGM$ | $23 \cdot 10^4$ | $33 \cdot 10^4$ |
| $Fast\ GM$ | $20 \cdot 10^4$ | $29 \cdot 10^4$ |
| $\frac{C_{Fast\ GM}}{C_{GM}}$ | 0.7% | 0.5% |
| $\frac{C_{Fast\ GM}}{C_{SIGM}}$ | 87% | 88% |

ment is insignificant when the regular GM is used, since its complexity is dominated by $K_{\text{GM}}^{(H^T H)}$. However, when the IFGM is implemented together with the SIGM, a significant additional improvement is achieved.

## V. FAST GM ALGORITHM

In order to reduce the complexity of the GM algorithm, the formulations presented in Sections III and IV were integrated into the fast GM algorithm.

1) A multiscale pyramid is built following Section II.
2) Utilizing the SIGM algorithm (Section III), a set of points $\widehat{S_2}$, is selected in the image $I_2$ at the coarsest resolution scale.
3) Starting with the coarsest resolution scale, (4.4) is solved iteratively according to the IFGM in Section IV-D.
4) The iterative and multiscale refinement were conducted according to the SIGM, where the motion estimation results and pixel set $\widehat{S_2}$, were propagated through the resolution pyramid in a coarse-to-fine manner.

Thus, the evaluation of $H^T H$ is optimized by the SIGM while the evaluation of $H^T I_t$ is optimized by the IFGM. The complexity estimation presented in Table III, shows the overall improvement, provided by the Fast GM, to be of $O(100)$. The complexity was estimated for the test case introduced in Table I. Most of the improvement is achieved by the SIGM, while the IFGM accomplishes an additional improvement of 10%–15%.

## VI. EXPERIMENTAL RESULTS

Affine motion is utilized by the MPEG-4 video compression standard as its main GME motion model. In order to explore the Fast GM coding and its associated complexity performances, the proposed scheme was integrated into the MPEG-4 Verification Model software [11] and compared to the GME algorithm implemented in it, while no other modifications were made. Simulations have been carried out using the sequences shown in Fig. 5: Mobile, Stefan, and Coastguard at CIF size ($352 \times 240$). Both the fast GM and regular GM used the three-step initialization method described in [9]. Two resolution scales were constructed using a three-tap filter, as follows:

$$\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

and the spatial derivatives were approximated using the mask

$$\begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix}.$$

The iterative termination threshold at each resolution scale was a translation update of $10^{-1}$ or at most ten iterations. The pixel subset $|\widehat{S_2}|$ used by the fast GM was 10% of the pixels in the lowest resolution scale, which amounts to 2.5% of the pixels at the original resolution scale. The test sequences were encoded in interframe mode (IPPPPP ...) and two fixed quantizer sizes $Q = 10$ and $Q = 31$. $Q = 31$ corresponds to measuring the Global motion compensation (GMC) error directly, while $Q = 10$ relates to a typical compression scenario. The coding efficiency results of the MPEG-4 using the Fast GM were compared to those of the regular GM and the non-GMC compression mode. The results shown in Fig. 6 and Table IV establish that the fast GM achieves the same coding efficiency as the regular GM while exhibiting 20 times less computational complexity. In both quantizer settings, the difference in the compressed frame size was no more than 2%–3%. In Fig. 7(a), frames 175–190 and 230–270 present a situation where the use of GME substantially improves the compression ratio as the encoded frame sizes of the GME modes (red and blue graphs) are considerably smaller then those of the non-GME mode (green graph). In these subsequences, significant global motion occurs within the Stefan sequence as the player rushes to the net. Thus, the GME is able to smooth the bit rate needed to encode the whole sequence at a fixed quantizer (quality). In the Coastguard sequence [Fig. 6(b)], the improvement caused by the GME is less significant, as the global motion in this sequence is a slow uniform translational motion. This type of motions are efficiently compressed by the non-GME mode, since the MPEG-4 standard utilizes differential coding of motion vectors [11], [20]. In particular, Fig. 7(b) presents a subsequence of the Coastguard, where there is no significant global motion, yet the fast GM achieves similar results to the regular GM using 20-fold less computational complexity. Similar results can be observed in Fig. 6(c) (Mobile sequence), where a dominant, very slow translational background motion exists. For the Stefan and Coastguard sequences, the average compression ratio was improved, while the compression of Mobile sequence resulted in a decrease of the compression ratio. This occurs since the GME mode is switched on\off on a macroblock basis within the MPEG-4 verification model, based on their reconstruction error. In this procedure, the actual coding overhead of the GME parameters is not taken into account. Therefore, this problem can be easily overcome by using either a two-pass encoding process or a GME switching algorithm taking into account the coding overhead. Table V presents the experimental timing results of the proposed algorithm recorded. The execution time of the GME module were measured directly using the built-in

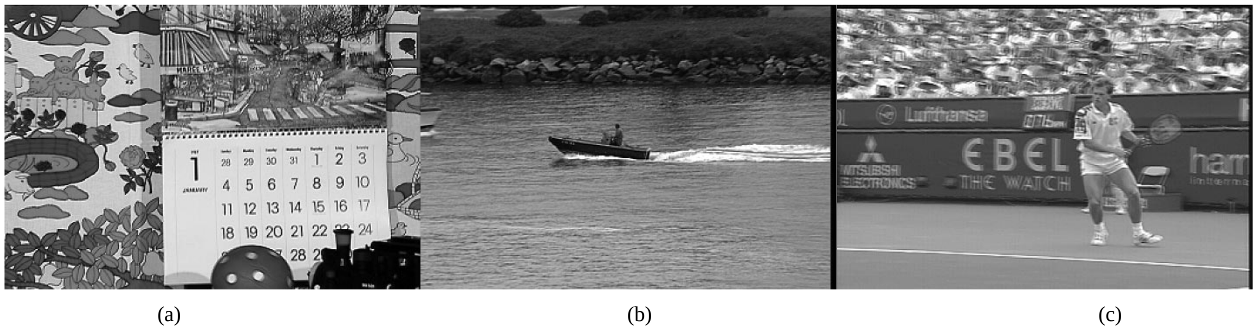


Fig. 5. Video sequences used for the registration accuracy tests for (a) Mobile, (b) Coastguard, and (c) Stefan.

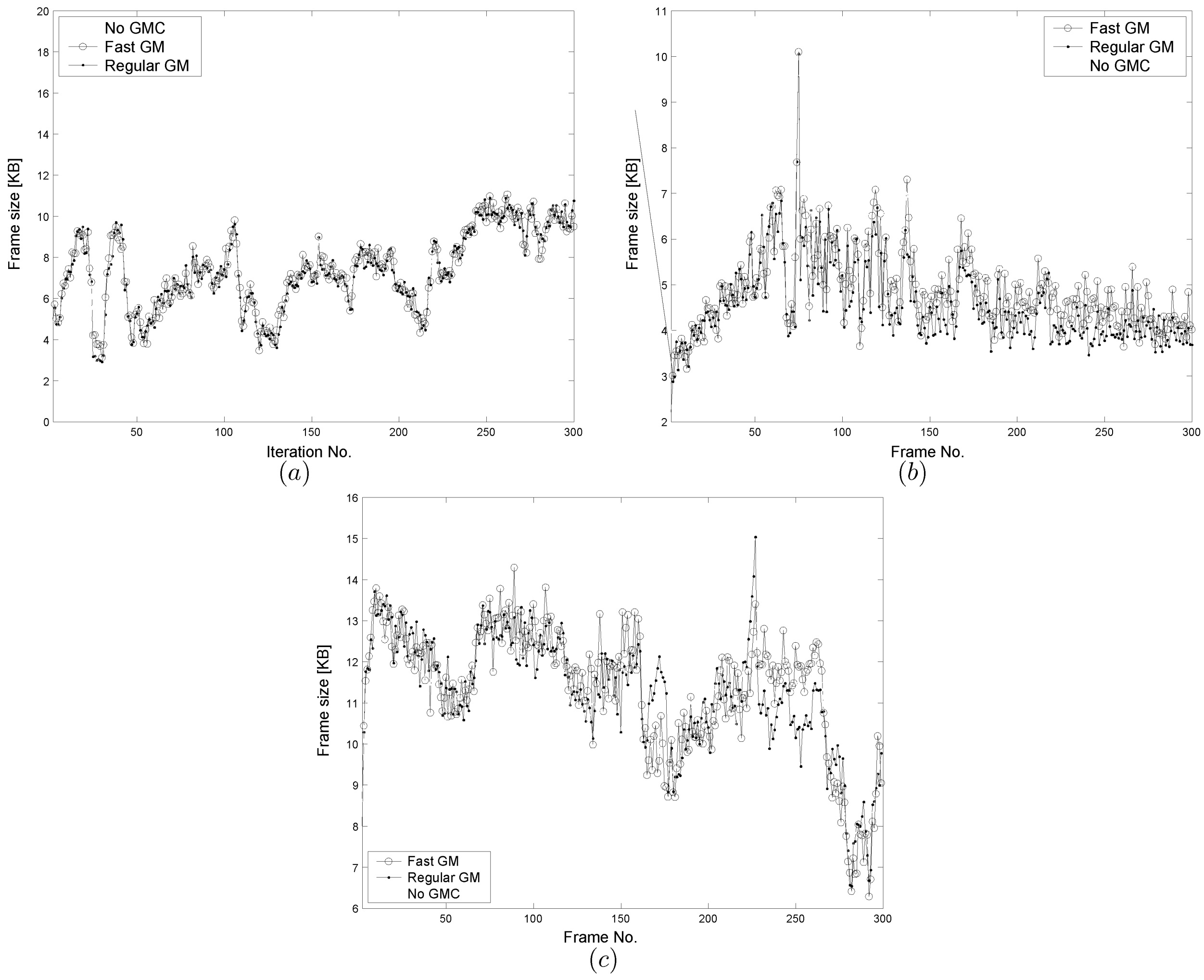
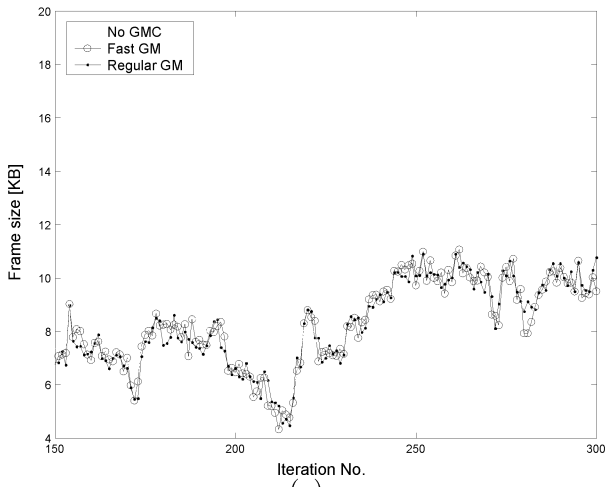
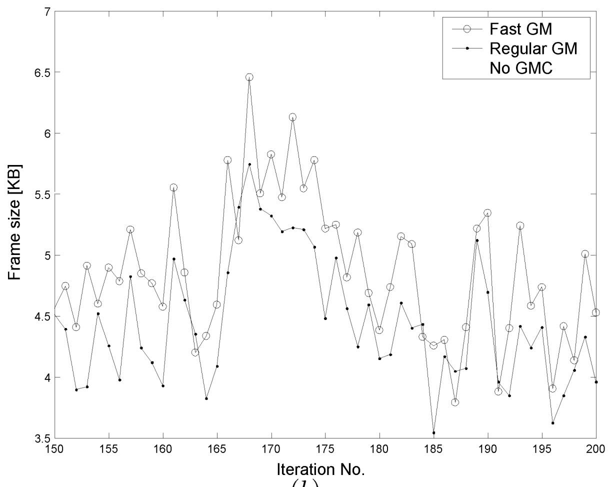


Fig. 6. MPEG4 Compression results using the Fast GM, regular GM and non-GMC compression modes for (a) Stefan, (b) Coastguard, and (c) Mobile sequences. The sequences were encoded using $Q = 31$, thus the compression results are directly related to the GMC efficiency. The fast GM provides compression results very similar to the regular GM, while utilizing 20 times less computational complexity.

Microsoft Visual C++ profiler [25]. Overall compression performance was measured by a stop-watch over a 300-frame period for each sequence. Our implementation uses standard C++ without any assembly-level optimizations and in order to avoid the performance bias caused by disk and memory caching [24], each test was run ten times and the results were averaged. The results show that the proposed GME module achieves the expected computational complexity gains presented in Table IV. The overall performance was only improved by a factor of approximately four, since other components within the compression algorithm became complexity-wise dominant. Several sizes of $|\widehat{S_2}|$ were tested; for $|\widehat{S_2}|/|S_2| = 20\%$, the compression results were identical to the results presented above, while for smaller pixel set sizes $(|\widehat{S_2}|/|S_2| = 5\%, 2\%)$ a decrease of the compression ratio was noticed. A Pentium PC P800 MHz was used for the timing experiments.



$(a)$



$(b)$

Fig. 7. MPEG4 Compression results for subsequences within the sequences in Fig. 6. (a) In this subsequence of Stefan, a global motion occurs as the player rushes to the net. Using GME results in a significant reduction in the encoded frame size. (b) A subsequence of Fig. 6(b), due to significant occlusion in the scene the improvement caused of the fast GM is 10% less than the regular GM, but the computational complexity is less by 20 fold.

TABLE IV
AVERAGE SIZES OF THE COMPRESSED VIDEO FRAMES. THE FAST GM ACHIEVES THE SAME CODING EFFICIENCY AS THE REGULAR GM. THE BOTTOM LINE SHOWS THE EXPECTED COMPUTATIONAL COMPLEXITY GAIN ACHIEVED BY USING THE FAST GM INSTEAD OF THE REGULAR GM

| | Mean frame size[KByte] | | | | | |
|---|---|---|---|---|---|---|
| | Q=31 | | | Q=10 | | |
| | Stefan | Coastguard | Mobile | Stefan | Coastguard | Mobile |
| GM | 7.3732 | 4.6155 | 10.13 | 37.35 | 21.84 | 66.22 |
| Fast GM | 7.4322 | 4.9330 | 10.3 | 37.42 | 22.12 | 66.15 |
| No GM | 8.6437 | 5.7127 | 11.3 | 42.18 | 21.73 | 65.76 |
| Complexity gain | 20 | 19.81 | 33 | 20 | 19.81 | 33 |

TABLE V
GLOBAL MOTION ESTIMATION (GME) TIMING DATA: THE TIME COLUMN INDICATES THE AVERAGE TIME SPEND IN THE GLOBAL ESTIMATION FUNCTION PER FRAME. THE SECOND COLUMN PRESENTS THE NUMBER OF FRAME PER SECOND ARCHIVED USING EACH GME MODE

| | Stefan $352 \times 240$ | | Stefan $176 \times 144$ | | Coastguard $352 \times 288$ | | Mobile $352 \times 288$ | |
|---|---|---|---|---|---|---|---|---|
| | time $[ms]$ | fps | time $[ms]$ | fps | time $[ms]$ | fps | time $[ms]$ | fps |
| GM | 617 | 1.5 | 194 | 4.41 | 612.14 | 1.36 | 1046 | 1.20 |
| Fast GM | 31.82 | 6.67 | 11.7 | 13 | 32 | 5.45 | 32 | 5.35 |
| Gain | 19.4 | 4.44 | 16.6 | 2.94 | 19.1 | 4.0 | 32.66 | 4.45 |

We conclude that in sequences where a significant global motion is present, the fast GM exhibits compression results similar to regular GM, while using significantly less (up to 33 fold) less computational complexity. In subsequences where using GME does not result in compression ratio improvement, the fast GM may produce lower compression ratios compared to the regular GM upto 10%.

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a new motion estimation algorithm based on gradient methods, which significantly reduces the complexity of state-of-the-art GME algorithms while achieving similar accuracy. This property makes it especially suitable for low-power GME applications such as video compression. The theoretical complexity analysis is back up by experimental results, which were obtained using the MPEG-4 Verification Model. The algorithm can also be used to accelerate image mosaics production and virtual reality applications [3]. In order to avoid compression ratio reductions due to sequences with no inherent global motion, we intend to develop a better algorithm for choosing when to use the GME mode.

## REFERENCES

[1] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *Int. J. Comput. Vis.*, vol. 12, pp. 43–77, 1994.
[2] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artific. Intell.*, vol. 17, pp. 185–203, 1981.
[3] M. Irani and S. Peleg, "Motion analysis for image enhancement: Resolution, occlusion and transparency," *J. Vis. Commun. Image Repres.*, vol. 4, no. 4, pp. 324–335, Dec. 1993.
[4] M. Irani, P. Anandan, and S. Hsu, "Mosaic based representations of video sequences and their applications," in *Proc. Int. Conf. Computer Vision*, Boston, MA, June 1995, pp. 605–611.
[5] M. Irani, B. Rousso, and S. Peleg, "Computing occluding and transparent motions," *Int. J. Comput. Vis.*, vol. 12, no. 1, pp. 5–16, 1994.

[6] R. Szeliski, “Video mosaics for virtual environments,” *IEEE Comput. Graph. Appl.*, vol. 16, no. 2, pp. 22–30, 1996.

[7] A. Tekalp, *Digital Video Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[8] B. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *Proc. DARPA, Image Understanding Workshop*, Apr. 1981, pp. 121–130.

[9] F. Dufaux and J. Konrad, “Efficient, robust, and fast global motion estimation for video coding,” *IEEE Trans. Image Processing*, vol. 9, pp. 497–501, Mar. 2000.

[10] M. Irani and P. Anandan, “All about direct methods,” in *Int.Workshop on Vision Algorithms*, Corfu, Greece, Sept. 1999.

[11] S. Fukunaga *et al.*, “MPEG-4 video verification model version 16.0,” ISO/IEC document JTC1/SC29/WG11 N3312, Mar. 2000.

[12] F. Dellaert and R. Collins, “Fast image-based tracking by selective pixel integration,” in *ICCV 99 Workshop on Frame-RateVision*, Corfu, Greece, Sept. 1999.

[13] A. Averbuch and Y. Keller, “Warp free gradient methods based image registration,” Tel-Aviv University, Tel Aviv, Israel, Tech. Rep. TBD, Nov. 2001.

[14] A. Smolić and T. Wiegand, “High-resolution video mosaicing,” in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, Thessaloniki, Greece, Sept. 2001.

[15] E. Steinbach, T. Wiegand, and B. Girod, “Using multiple global motion models for improved block-based video coding,” in *Proc. IEEE Int. Conf. Image Processing, ICIP-99*, vol. 2, Kobe, Japan, Oct. 1999, pp. 56–60.

[16] H. Sawhney and S. Ayer, “Compact representation of videos through dominant and multiple motion estimation,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, no. 8, pp. 814–830, Aug. 1996.

[17] A. Smolic, T. Sikora, and J.-R. Ohm, “Long-term global motion estimation and its application for sprite coding, content description, and segmentation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 1227–1242, Dec. 1999.

[18] N. Grammalidis, D. Beletsiotis, and M. Strintzis, “Sprite generation and coding in multiview image sequences,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 302–311, Mar. 2000.

[19] F. Dufaux, “Results for video coding using dynamic sprite (Core Experiment N3),” ISO/IEC JTC1/SC29/WG11 MPEG-4 meeting, Maceio, Brazil, Tech. Rep. M1458, , Nov. 1996.

[20] T. Sikora, “The MPEG-4 video standard verification model,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 19–31, Feb. 1997.

[21] F. Dufaux, “Background mosaicking,” ISO/IECJTC1/SC29/WG11, MPEG96/M0653, Munich, Germany, Jan. 1996.

[22] P. Gill, *Practical Optimization*: Academic Press, 1982.

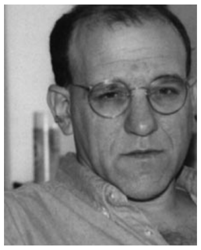
[23] Q. Wei, H. J. Zhang, and Y. Zhong, “A pre-analysis method for robust global motion estimation,” in *Proc. IEEE Int. Conf. Image Processing, ICIP-99*, vol. 2, Kobe, Japan, Oct. 1999, pp. 26–30.

[24] R. Cutler and L. Davis, “Developing real-time computer vision applications for intel pentium III based windows NT workstations,” in *FRAME-RATE: Frame-rate Applications, Methods and Experiences With Regularly Available Technology and Equipment [in Conjunction With IEEE Int. Conf. Computer Vision (ICCV)]*, Kerkyra, Greece, Sept. 1999.

[25] Microsoft Developer Network. (2002). [Online]. Available: http://msdn.microsoft.com.

**Yosi Keller** received the B.Sc. degree in electrical engineering in 1994 from The Technion-Israel Institute of Technology, Haifa, Israel, and the M.Sc and Ph.D degrees in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 1998 and 2003, respectively.

From 1994 to 1998, he was a Research and Development Officer in the Israeli Intelligence Force. His research interests include computational methods and their applications to motion estimation, video analysis, tracking, image enhancement, and restoration.



**Amir Averbuch** was born in Tel Aviv, Israel. He received the B.Sc. and M.Sc. degrees in mathematics from the Hebrew University, Jerusalem, Israel in 1971 and 1975, respectively, and the Ph.D. degree in computer science from Columbia University, New York, in 1983.

During 1966–1970 and 1973–1976, he served in the Israeli Defense Forces. From 1976 to 1986, he was a Research Staff Member in the Department of Computer Science, IBM T.J. Watson Research Center, Yorktown Heights, NY. In 1987, he joined the Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, where he is now a Professor of computer science. His research interests include wavelets, signal/image processing, multiresolution analysis, numerical computation for the solutions of PDEs, scientific computing (fast algorithms), and parallel and supercomputing (software and algorithms).