2-D MOTION ESTIMATION WITH HIERARCHICAL CONTENT-BASED MESHES

Ghassan Al-Regib and Yucel Altunbasak

Center for Signal and Image Processing
Georgia Institute of Technology
Atlanta, GA 30332-0250

ABSTRACT

Two-dimensional mesh-based models provide a good alternative to motion estimation and compensation. The estimation of the best motion vectors at the node-point motion vectors is a challenging task. To this effect, Nakaya *et al.* proposed a hexagonal matching procedure. Toklu *et al.* improved the hexagonal search algorithm in terms of both motion estimation accuracy and computational complexity by employing a hierarchy of regular meshes. Recognizing the limitations of regular meshes, Van Beek *et al.* extended Toklu's work by utilizing content-based meshes. Here, we provide an alternative hierarchical motion estimation method with content-based meshes where hierarchical representations are employed for images as well as meshes in order to provide further improvements in computational complexity as well as motion accuracy.

1. INTRODUCTION

Two-dimensional (2-D) dense motion estimation methods can be classified as block-based, optical-flow equation based, pel-recursive, and Bayesian methods [1]. Although, block-based dense motion compensation yields high peak signal-to-noise ratio (PSNR), the corresponding dense motion field usually contains several outliers (because of the aperture problem and lack of overall smoothness constraints). Optical-flow equation based methods yield smoother dense motion fields, but they do not always perform well in terms of PSNR. Joint Bayesian motion estimation and segmentation methods have the ability to provide both piece-wise smooth motion fields and excellent PSNR performance. However, they are generally time-consuming and parameter-dependent.

Two-dimensional mesh-based models provide a good alternative to motion estimation and compensation as reported by several researchers [2, 3, 4]. They are simple enough for fast implementations (especially with the help of graphics cards), but powerful enough for describing the motion content accurately.

In 2-D mesh-based methods, motion-compensation within each mesh element (patch) is accomplished by a spatial transformation (affine, bilinear, etc.) whose parameters can be computed from node-point motion vectors. Estimation of motion vectors around each node independently (*e.g.*, by block-matching or deformable block matching) is usually not desirable because: i) motion vectors do not represent the entire 2-D dense motion field, and ii) motion vectors may cross each other (especially around small patches) destroying the connectivity of the mesh. Hence, search-based solutions to node-point motion estimation with triangular and/or quadrilateral meshes and spatial transformations have been proposed [2, 3]. In particular, Nakaya *et al.* [2] proposed a hexagonal matching procedure where the motion vector at a node-point is estimated by iterative local minimization of the prediction error. Toklu *et al.* [5] extended this method by employing a hierarchy of regular meshes such that motion estimation with a coarse mesh provide initialization for the next (finer) level of the mesh. However, both Nakaya and Toklu utilized regular meshes. Regular meshes are obtained by dividing the image area into equal size triangular or rectangular elements, hence they may not have the ability to reflect the scene content; *i.e.*, a single mesh element may contain multiple motions. As a result, hierarchical quad-tree meshes are proposed, in which patches that yield high motion-compensation error are successively subdivided. A more fundamental approach to overcome the problem of mesh elements with more than one motion is to employ a content-based mesh, which is not limited to a pseudo-regular hierarchical structure. Content-based meshes aim to match boundaries of patches with important scene features [3, 4]. In [4], Altunbasak *et al.* proposed a computationally efficient algorithm for content-based 2-D triangular mesh design. Van Beek *et al.* [6] extended Toklu's work by employing coarse-to-fine content-based meshes in motion estimation. But, although both Toklu's and Van Beek's work employ coarse-to-fine meshes, the image/frame size is kept the same, *i.e.*, number of nodes at each level of the mesh hierarchy changes while all meshes are *still* laid on the *original* image since no hierarchical image representation is employed. Here, we provide an alternative hierarchical motion estimation method with content-based meshes that is superior to aforementioned method in terms of both performance and complexity. A comparison among various 2-D mesh based motion estimation methods is summarized in Table 1.

Method	Motion Est.	Mesh-Type	Mesh Hierarchy	Image Size/Hierarchy
Nakaya *et al.* [2]	Search	Regular	None	N/A
Wang *et al.* [3]	Optimization	Content-based	None	N/A
Altunbasak *et al.* [4]	Closed-form	Content-based	None	N/A
Toklu *et al.* [5]	Search	Regular	Coarse-to-fine	Constant size
Van Beek *et al.* [6]	Search	Content-based	Coarse-to-fine	Constant size
Proposed	Search	Content-based	Hierarchical	Var. size/Gaus. Pyr.

Table 1. Comparison of various 2-D mesh-based motion estimation algorithms.

Section 2 describes the hierarchical method proposed by [6] while the proposed method are described in section 3. Section 4 describes the coarse-to-fine and fine-to-coarse mesh generation algorithms employed in the two implementations of the proposed method, respectively. The results are discussed in section 5.

2. BACKGROUND

The hierarchical content-based motion estimation method proposed by Van Beek *et al.* [6] is especially relevant within the context of

this paper, hence, it will be briefly described. This method inherits its hierarchical nature from the process of generating a fine mesh from a coarse mesh. The fine mesh represents the scene motion more accurately since new nodes are introduced. This is illustrated in Figure 1 where the mesh on the left, M_{l+1}, represents the coarse mesh, while the one on the right, M_l, denotes the fine mesh. In M_{l+1} there are nine node-points labeled as $n_0 ... n_8$ that are retained in M_l with more node-points introduced by the mesh refinement algorithm. These new node-points are shown in a gray-color dots whereas the new edges appear as dash lines. The motion vectors calculated by the generalized polygonal search procedure [2] at the node-points of the coarse mesh M_{l+1} are used in initializing newly added node-points in the fine mesh M_l. In the rest of the paper, this method is referred to as method-I.

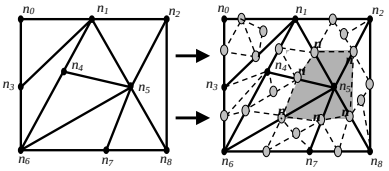

Fig. 1. Generating a fine 2-D mesh from a coarse mesh while the image is kept spatially at the same size.

3. METHODOLOGY

Although the above method outperforms the single-level tracking in terms of motion vector accuracy and computational complexity, it *still* incurs high computational load. To further reduce the computation time, the proposed method constructs a hierarchical/pyramid representation for both the images and the corresponding meshes. Accordingly, the image at each level will be a blurred and downsampled version of the image in the previous level. Similarly, the 2-D mesh at each level will be a coarse version of the one in the previous level. As a result, the computation time is significantly reduced since the patches and the search ranges are small in size at the coarsest level of the pyramid. Another advantage raises from the fact that motion estimation algorithms are less prone to be trapped at the local minima when applied to coarse images.

In this paper, two different implementations of this method are proposed. They differ in the level of hierarchy at which the mesh is designed as well as the method of constructing mesh levels. These two implementations are discussed in the following two subsections, respectively.

3.1. Implementation I

As mentioned earlier, the proposed method constructs a hierarchical representation for both the image and the 2-D mesh. This is valid in both implementations. However, in the first implementation, the 2-D mesh is designed at the coarsest level. Then, the 2-D mesh in the next finer level is constructed from the coarser mesh by a coarse-to-fine mesh generation algorithm that is described in section 4.1.

Figure 2 illustrates this implementation for a specific example of two-level hierarchy for simplicity, though it can easily be generalized for any number of levels. An outline of this implementation is as follows:

1. Generate a coarse image, I_1, by blurring and downsampling the original image, I_0.
2. *Design a mesh*, M_1, for the coarse image, I_1.
3. Compute the motion vectors at the node-points in M_1.
4. Generate a fine mesh, M_0, by *refining* M_1 using the coarse-to-fine mesh generation algorithm.

After this step, the motion vectors for the node-points in M_0 needs to be initialized by the node-point motion vectors estimated at level 1. Assume that we are interested in finding the motion vectors for a node-point with coordinates *(x,y)* in I_0. This can be achieved as follows:

1. Find the corresponding coordinates of the node-point in I_1, which are equal to $(\frac{x}{2}, \frac{y}{2})$.
2. Locate the patch, P_i, in I_1 where the point $(\frac{x}{2}, \frac{y}{2})$ lies in.
3. Calculate the affine parameters (for P_i) from the motion vectors of the vertices of the patch P_i.
4. Compute the motion vector for the point $(\frac{x}{2}, \frac{y}{2})$ from the affine parameters estimated in step 3. To calculate the motion vector for the node-point (x, y) in M_0, scale the computed motion vectors by a factor of 2 to compensate for the downsampling effect.

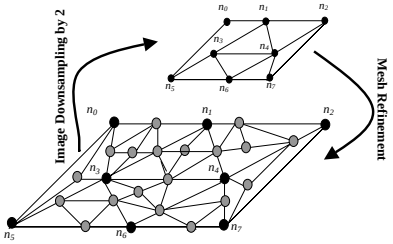

Fig. 2. Illustration of the implementation-I for a 2-level hierarchy.

3.2. Implementation II

While the first implementation designs the 2-D mesh at the coarsest level, this implementation designs the mesh at the finest level. Similar to the first implementation, the tracking algorithm still is applied on the coarsest level. However, while a coarse-to-fine *mesh generation* algorithm is needed for the first implementation, here a fine-to-coarse *mesh reduction* algorithm is employed.

The process of getting a coarse mesh from a fine one is modified from progressive-mesh methods, which are commonly employed in 3-D computer graphics [7]. This technique will be discussed in details in section 4.2. Figure 3 illustrates this implementation for a specific example of *2*-level hierarchy. The outline of this implementation is as follows:

1. Generate a coarse image, I_1, by blurring and downsampling the original image, I_0.
2. *Design a 2-D mesh*, M_0, for the original image, I_0.
3. *Construct a coarse mesh* M_1 from M_0 by applying the fine-to-coarse mesh reduction algorithm. This is indicated by the right-hand arrow that heads up the hierarchy in Figure 3, while in the first implementation this arrow heads down the hierarchy as in Figure 2.

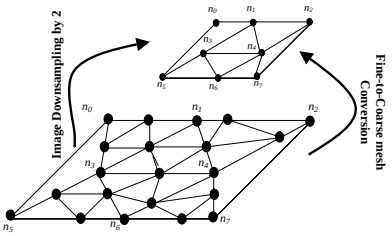

Fig. 3. Illustration of the implementation-II for a 2-level hierarchy.

4. MESH CONSTRUCTION

As mentioned earlier, the two implementations of the proposed method differ in the level of hierarchy at which the mesh is designed as well as the method of constructing mesh levels. While the first implementation employs a mesh refining algorithm, the second one uses a fine-to-coarse mesh generation algorithm. These two algorithms are described in this section.

4.1. Coarse-to-fine mesh construction

Here, we modify the 2-D content-based triangular mesh design previously proposed by Altunbasak *et al.* [4]. The algorithm retains the selected node points from the coarser mesh level, and further selects non-uniformly spaced node-points at each level. The procedure also aims to partition the image into triangles in such a way that a predefined function of the displaced frame difference (DFD) within each patch attains approximately the same value. An outline of the algorithm is as follows:

1. Generate an L level Gaussian image pyramid by blurring and down-sampling the original image pair. Let $l = 0 \ldots L-1$ denote levels of the hierarchy, where $l = 0$ represents the original images. Set $l = L - 1$.

2. Label all pixels as "unmarked."

3. Scale the locations of all node-points selected at the level $l + 1$ by a factor of 2. Include these points in the list of selected node points at the level l.

4. Compute the average displaced frame difference $DFD_{avg}[l]$ given by

$$DFD_{avg}[l] = \frac{\sum_{(x,y)} DFD(x,y)}{K[l]} \qquad (1)$$

   where the summation is over all the image pixels at the level l, and $K[l]$ is the number of node points at that level.

5. Grow a region about all selected node points until $\sum DFD(x,y)$ in this region is greater than $DFD_{avg}[l]$. Label all pixels within this region as "marked."

6. If the number of retained nodes (from the coarser level of the hierarchy) is less than $K[l]$, then select more node-points as follows:

   (a) Compute a cost function $C(x,y)$ associated with each unmarked pixel as a predefined function of spatio-temporal intensity gradients [4]. The cost function includes terms that are functions of both spatial and temporal intensity gradients so that selected node points, hence the boundaries of the patches, coincide with spatial edges and motion edges.

   (b) Find the unmarked pixel with the highest $C(x,y)$ which is not closer to any other previously selected node point than a prespecified distance. Label this point as a node point.

   (c) Grow a region about this node point until $\sum DFD(x,y)$ in this region is greater than $DFD_{avg}[l]$. Label all pixels within this region as "marked."

   (d) Go to 6(c) until a desired number of node points, $K[l]$, are selected.

7. Given the selected node points at the level l, apply a triangulation procedure (e.g., Delauney triangulation [8]) to obtain a content-based mesh.

8. Decrease l by 1. Unless $l < 0$ go to step 2.

4.2. Fine-to-coarse mesh construction

The fine mesh construction algorithm used in the second implementation depends on two related operations: node-split and edge-collapse, which are shown in Figure 4. These two operations are the 2-D mesh equivalent operations to the 3-D mesh ones, which are called: vertex-split and edge collapse, respectively. Practically, a series of node-split operations will produce a fine mesh from a coarse one while a series of edge-collapse operations will result in a coarse mesh from a fine mesh. The following algorithm outlines the main steps in constructing levels of coarse meshes from a fine one.

1. Start with a fine mesh M_0.
2. Apply the edge-collapse transform, $ecol_i(n_s, n_t)$, on the edge connecting the nodes n_s and n_t, where n_s and n_t are internal nodes of the mesh. A node is internal if it does not lie on the frame/image edges. The index i in $ecol_i$ indicates the level at which this edge-collapse command is taking place. Such collapse results in vanishing the adjacent patches (faces) $\{n_s, n_t, n_l\}$ and $\{n_t, n_s, n_r\}$.
3. The choice of the edge as well as the new location of the new node, with connectivity-K and A scalar attributes, depends on minimizing the following energy function:

$$E_K = \min_{N,A} E_{dist}(N) + E_{spring}(N) + E_{scalar}(N,A)$$

   where, E_{dist} measures the total distance of all the new points from the mesh, E_{spring} is used to regularize this optimization problem and $E_{scalar}(N,A)$ measures the accuracy of the scalar attributes such as intensity. $E_{scalar}(N,A)$ calculates the cost in terms of attributes of the new vertices (intensity and motion vector). This energy function is minimized over all edges and the edge with the minimum E is collapsed at this $ecol_i$ transform. The reader is referred to [7] for more information on this energy function minimization for 3-D meshes.

4. If *the total number of nodes > number of external nodes in* M_i, then repeat step 2 for another set of nodes. Otherwise, stop the algorithm and the resulting mesh is the coarsest mesh.

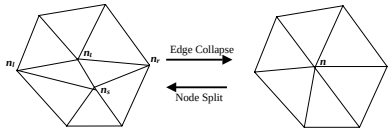

Fig. 4. Constructing fine meshes from a given coarse mesh and visa versa.

5. COMPARISON AND RESULTS

We have designed experiments to compare the proposed method (method II) with the algorithm in [6] (method I) with the video sequence "Flower&Garden". The results are documented in Table 2. The parameters used in these experiments are also tabulated in the same table. The number of nodes at each level of hierarchy is provided in the fifth column where the first number in the array represents the number of nodes at the finest level while the last number in the array represents the number of nodes at the coarsest level. The number of nodes at each level is kept the same for both methods. The search size in method I is increased by a factor of 2 in both horizontal and vertical directions at each level of the hierarchy since it does not utilize an image pyramid representation. Without increasing the search size, method I would be at a disadvantage. The PSNR of the motion compensated frame difference is calculated and tabulated in the second column for a pair of frames in each sequence. The PSNR before motion compensation is 11.31. The execution time is given in the third column. No effort has been directed to optimize algorithms in terms of execution speed.

The theoretical computational complexity for both methods is given in Table 3, where N by N is the original image size, P is the number of nodes at the finest level, R by R is the search range at the finest mesh level, I is the number of iterations at the hexagonal matching procedure, S is the search step size at the finest mesh level, and L is the number of levels. This table provides approximate number of addition operations involved in the motion estimation stage. Since the affine warping operations along a horizontal line can be computed using additions, the multiplication operations is negligible. In the calculation for Method-I, the search range R is assumed to be increased at each level by a factor of 2. Similarly, the step size S is increased by the same ratio. The number of nodes P is reduced by 4 at each level for both methods. This table assumes a regular mesh as opposed to content-based mesh to make the calculations tractable. From the results in Table **??** and the comparisons in Table 3, we can claim that the proposed method performs equally or slightly better in terms of PSNR and runs significantly faster. The latter is a direct consequence of pyramid image representation.

In conclusion, the theoretical discussion as well as the experimental results show that the proposed method has three main advantages as compared to method I. These advantages are valid in both implementations. First, the pyramid/hierarchical image representation will reduce the computational complexity considerably. Node-point motion estimation computational complexity is mainly a function of the number of nodes, patch sizes and the search range. Second, since the image is blurred and downsampled at higher levels, the proposed method is likely to lock onto more global motions at these levels, rather than being trapped by local motions. This is reflected in the experimental results in Table 2 where increasing the number of level from three to five increases the PSNR. Third, hierarchical image representation utilized in method II will decrease the execution time for content-based mesh design/modification process at higher levels, which is obvious from the results stated in the third column of Table 2.

Method	PSNR	Time in sec	♯ of Lev.	♯ of Nodes	Search Size	Step Size
I	14.23	45.16	3	[100, 16, 1]	[2, 4, 8]	[0.5, 1.5, 4.0]
II	14.35	35.51	3	[100, 16, 1]	[2, 2, 2]	[0.5, 0.75, 1.0]
I	14.34	157.75	3	[100, 16, 1]	[4, 8, 16]	[0.5, 1.5, 4.0]
II	15.96	125.81	3	[100, 16, 1]	[4, 4, 4]	[0.5, 0.75, 1.0]
I	12.94	57.84	5	[100, 64, 36, 16, 1]	[2, 4, 8, 16, 32]	[0.5, 1.5, 4.0, 10.0, 24.0]
II	16.96	36.13	5	[100, 64, 36, 16, 1]	[2, 2, 2, 2, 2]	[0.5, 0.75, 1.0, 1.25, 1.5]
I	14.72	196.20	5	[100, 64, 36, 16, 1]	[4, 8, 16, 32, 64]	[0.5, 1.5, 4.0, 10.0, 24.0]
II	17.19	127.52	5	[100, 64, 36, 16, 1]	[4, 4, 4, 4, 4]	[0.5, 0.75, 1.0, 1.25, 1.5]

Table 2. Experimental results for the sequence "Flower&Garden" (352×240).

	N. of additions
Method I	$\sum_{l=0}^{L-1} 9N^2\frac{R^2}{S^2}I$
Method II	$\sum_{l=0}^{L-1} 9N^2\frac{R^2}{S^2}\frac{1}{2^l}I$

Table 3. Theoretical comparison results.

6. REFERENCES

[1] A. M. Tekalp, *Digital Video Processing*, Prentice Hall, 1995.

[2] Y. Nakaya and H. Harashima, "Motion compensation based on spatial transformations," *IEEE Trans. Circ. and Syst for Video Tech.*, vol. 4, pp. 339–356, June 1994.

[3] Y. Wang and O. Lee, "Active mesh - a feature seeking and tracking image sequence representation scheme," *IEEE Trans. Image Proc.*, vol. 3, pp. 610–624, Sept. 1994.

[4] Y. Altunbasak and A. M. Tekalp, "Occlusion-adaptive, content-based 2-d mesh design and tracking for object-based video coding," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1270–1280, September 1997.

[5] C. Toklu, A. T. Erdem, M. I. Sezan, and A. M. Tekalp, "Tracking motion and intensity variations using hierarchical 2d mesh modeling for synthetic object transfiguration," *Graphical models and image processing*, vol. 58, no. 6, pp. 553–573, Nov 1996.

[6] P. V. Beek, A. M. Tekalp, N. Zhuang, I. Celasun, and M. Xia, "Hierarchical 2d mesh representation, tracking, and compression for object-based video," *IEEE transactions on circuits and systems for video technology*, vol. 9, no. 2, pp. 353–369, Mar 1999.

[7] H. Hoppe, "Progressive meshes," in *Proceedings ACM SIGGRAPH'96*, 1996, pp. 99–108.

[8] J. R. Shewchuk, "Triangle: Engineering a 2d quality mesh generator and delaunay triangulators," in *Proceedings first workshop on Applied Computational Geometry*, PA, USA, 1996, pp. 124–133.