ON MACROBLOCK PARTITION FOR MOTION COMPENSATION

Edson M. Hung, Ricardo L. De Queiroz
Universidade de Brasilia, Brazil, mintsu@image.unb.br, queiroz@ieee.org
Debargha Mukherjee
Hewlett-Packard Laboratories, USA, debargha.mukherjee@hp.com

ABSTRACT

In the H.264/AVC video coding standard, motion compensation can be performed by partitioning macroblocks into square or rectangular sub-macroblocks in a quadtree decomposition. This paper studies a motion compensation method using wedges, i.e. partitioning macroblocks or sub-macroblocks into two regions by an arbitrary line segment. This technique allows the shapes of the divided regions to better match the boundaries between moving objects. However, there are a large number of ways to slice a block and searching exhaustively over all of them would be an extremely computer-intensive task. Thus, we propose a fast algorithm which detects the predominant edge orientations within a block in order to pre-select candidate wedge lines. Finally a comparison among macroblock partition methods is performed, which points to the higher performance of the wedge partition method.

Index Terms— Quadtrees, wedges, motion compensation

1. INTRODUCTION

Typically, the temporal variations in video sequences occur due to camera or object motion. By compensating for this motion, better video image prediction can be obtained with relatively compact motion information, thereby enabling video compression algorithms to significantly reduce the amount of data needed to transmit sequences with acceptable quality. Hence, motion compensation is critical for obtaining good performance from video compression systems [1].

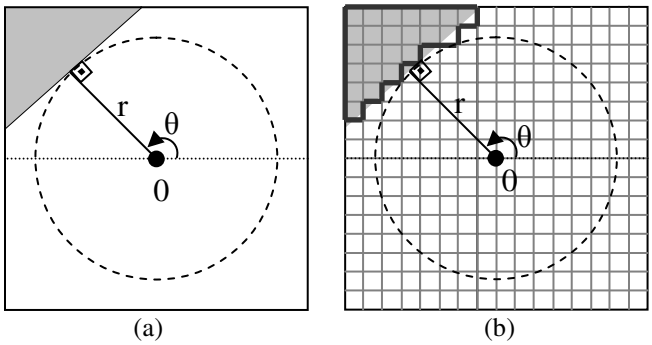
In the newest international video coding standard [2], H.264 or MPEG-4 Part 10, or AVC, the coding process operates on units of macroblocks of 16×16 pixels. When motion compensation is performed in inter-frame mode coding, the macroblock can be divided hierarchically into rectangular or square regions, up to a minimum of 4×4 pixels, in a quadtree-like decomposition structure.

In this paper, we consider enhanced motion compensation by incorporating the additional flexibility of slicing each block into wedges following arbitrary straight line segments. The general technique is referred to as wedge motion compensation. The technique is proposed as a means for improving the performance of motion compensation used in conventional hybrid video coding methods such as H.264. In a parallel work to our own research efforts, Kondo et al. [3] recently proposed a motion compensation technique using sliced blocks with one line segment in a hybrid video codec based on H.26L JM-1 [4]. They obtained an improvement of 5% in bit rate in comparison with H.26L. The authors imposed a limitation that the slice passes through specific points on the macroblock's boundary. In this paper, the slice line can pass through any position of the block. We will also show how to choose a set of potentials wedge partitions based on edge detection and a linear regression into wedgelet notation. The goal is to decrease computational complexity.

When one partitions the macroblock, there is more resolution for the motion estimation, but one spends more bits to convey the partition information to the decoder. In this paper, we address two basic questions. Shall we partition macroblocks? What are the most advantageous partitions?

2. MOTION COMPENSATION USING WEDGES

A multiscale wedgelet [5] framework can be a first step towards explicitly capturing geometric structure in images. In order to perform motion compensation based on this framework, we consider wedges obtained from an $N \times N$ block by splitting it into two regions along a line, with orientation angle θ and distance r from the block's center (see Fig. 1).



(a) (b)

Fig. 1. Wedgelet polar coordinate notation: (a) continuous (b) discrete.

Motion estimation is performed independently for each region, generating two motion vectors for the block. In general, a line segment (slice line) used in block division can pass through any position in the block, and there will potentially be a large number of partitions, leading to a correnspondingly large increase in motion estimation computational complexity. Therefore, a fast search method is needed, which will be discussed in the next section.

The set of all wedge partitions in an $N\times N$ block is generated by sampling the space of θ and r at suitable intervals, yielding a dictionary of N_w entries. Note that N_w depends on design parameters such as increment precision for θ and r.

Wedge partitioning, as described, may be carried over multiple scales in conjunction with quadtree partitioning. The overall scheme is described as follows:

Start from a macroblock size of $N\times N$ where $N = 16$ typically. For each $N\times N$ block, select one of the following:

1. Do not partition at all and send a single motion vector for the entire block. Terminate the tree. Always choose this option if $N = N_{\min}$, a minimum block size, typically 4.
2. Partition the block using a wedge partition to obtain two wedges with two associated motion vectors. Terminate the tree.
3. Divide the block into four $N/2\times N/2$ sub-blocks, and do partition selection for each sub-block.

In each macroblock, ideally, all possible partition strategies are to be tested and only the one that leads to the lowest rate-distortion-based cost function,

$$J(p) = D(p) + \lambda R(p), \qquad (1)$$

shall be chosen. Here, p refers to the overall macroblock partition strategy, while $R(p)$ means bits per macroblock and $D(p)$ is the distortion (e.g. MSE) corresponding to the partition strategy p. Finally, λ is a Lagrangian multiplier.

For compression, the encoder has to inform the decoder which macroblock partition strategy was used. Let $B(p)$ be the number of bits spent to encode p. We also have to encode the motion vectors associated with partition strategy p, thus spending $V(p)$ bits, plus the residual after compensation, spending $R_{res}(p)$ bits. Hence,

$$R(p) = V(p) + B(p) + R_{res}(p) \qquad (2)$$

The residual error is compressed with conventional transform based coding just for comparison purposes. Furthermore, for a fixed quantization matrix, D does not change much with the partition strategy. Therefore, regarding D as constant the following cost is obtained

$$J(p) = R(p) \qquad (3)$$

i.e. minimize rate for a fixed distortion.

3. FAST WEDGE MOTION COMPENSATION

Since the number of partition strategies in a macroblock can be very large, it is essential to consider viable fast search methods for the best partition strategy. To this end, at any scale, instead of testing all wedge partition possibilities within a block, we propose to pre-select a smaller subset of suitable block partitions.

Since motion compensation using wedge partitioning is expected to work better for blocks that fall along boundaries of moving objects, it is reasonable to assume that pre-selecting partitions based on the image edges may also achieve satisfactory results. In particular, within an $N\times N$ block, we first use an edge detection algorithm such as Canny or Sobel [6], followed by a thresholding operation. This yields a two-dimensional logical array of the same size as the block indicating edges. Next, the pixels with positive response are mapped into Euclidian pixel coordinates yielding $(x\ ,\ y)$ pairs, and a linear regression (based on least square error) is applied to these points. The result is a general equation of a line: $y = ax + b$, or $x = c$ (when the regression is not a function). The line is converted to polar coordinates and is associated to a wedge partition. Let $(x_0\ ,\ y_0)$ be the block center, N be the block size and θ_0 be an offset that depends on the sign of $ax_0 + y_0 + b$, such that $\theta_0 = 0°$ when $ax_0 + y_0 + b$ is positive and $\theta_0 = 180°$ otherwise. We then find starting points (seeds) for r and θ as:

$$r_{seed} = \left| \frac{ax_0 + y_0 + b}{\sqrt{a^2 + 1}} \right| \qquad (4)$$

$$\theta_{seed} = \arctan(a)\frac{180°}{\pi} + \theta_0 \qquad (5)$$

Case the regression results in $x = c$, we use:

$$r_{seed} = \left| \frac{N}{2} - c \right| \qquad (6)$$

$$\theta_{seed} = \theta_0 \qquad (7)$$

where θ_0 is an offset that depends on the sign of $N/2 - c$, such that $\theta_0 = 90°$ if $N/2 + c > 0$, and $\theta_0 = 270°$ otherwise.

From the starting seeds r_{seed} and θ_{seed}, a set of wedges is pre-selected by slightly changing parameters r and θ within the ranges $[r_{seed} - \Delta r,\ r_{seed} + \Delta r]$ and $[\theta_{seed} - \Delta\theta,\ \theta_{seed} + \Delta\theta]$ with increments δr and $\delta\theta$ respectively. The process is illustrated in Fig.2.

This approach is efficient for blocks with a single well defined (moving) straight edge, but might fail for multiple moving edges or textures. Therefore, a canonical quadtree partition is also performed. The motion estimation and the partition search are then restricted to only the pre-selected set of wedges, saving much of the search computation.

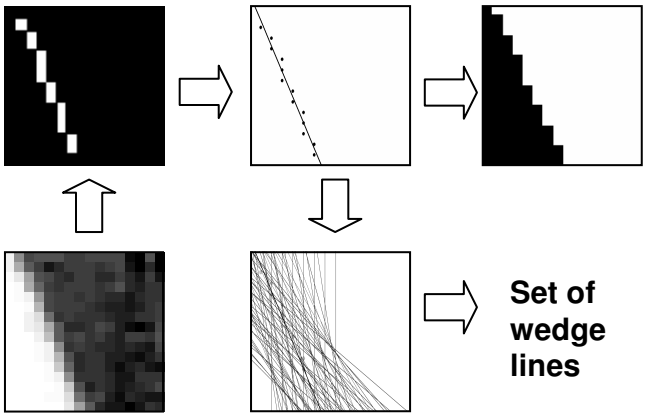


Fig. 2: Selecting the set of wedge lines for fast macroblock partition.

4. COMPARISON AMONG MACROBLOCK PARTITION METHODS

In order to understand the effectiveness and limitations of the proposed wedge-based macroblock partitioning scheme, we compare various macroblock partition strategies for 16×16-pixel macroblocks. In all the results that follow, single-pixel precision[1] and a search window range of 16×16 pixels were used.

The following strategies are compared:

- Traditional motion estimation with no macroblock partitioning.
- Quadtree partitioning.
- Wedge partitioning.
- Fast wedge partitioning.

With the traditional integral macroblock motion estimation (see Fig. 3), we only use a single motion vector per macroblock.

With quadtree partition, as in Fig. 4, each macroblock can be partitioned into either two halves of 8×16 or 16×8 pixels, or four sub-macroblocks of 8×8 pixels. Each 8×8 pixels sub-macroblock can be further partitioned into either two halves of 4×8 pixels or 8×4 pixels, or four 4×4 blocks. Hence, there are numerous partition possibilities, with a different number of motion vectors per macroblock for each.

In the wedge partition method, a 16×16 macroblock can be either not split at all, or split into two wedged blocks as depicted, or into four 8×8 sub-macroblocks. Further, each 8×8 sub-macroblock in the last case can be either not split, or split into two wedged blocks or four 4×4 blocks. There is, thus, a variable number of motion vectors per macroblock. The partition decisions are made based on minimizing the R-D cost. In our experiments, the wedge dictionary was determined using a span of $0 \le r < N/2$, where N is the block size, incremented with steps of $\delta r = 1$, and for $0° \le \theta < 360°$ with $\delta\theta = 1°$. After removing redundancies a dictionary of 2012 wedge partitions for 16×16 pixels macroblocks and 340 partitions for 8×8 pixels sub-macroblocks was obtained. A prefix code is used to convey the partition strategy. The wedge partitions at 16×16 or 8×8 scales, if used, are coded with an entropy coder with equi-probable symbols except for those canonical (halves) partitions. A typical partition map is shown Fig. 5. Note that in Foreman's hat, nose and face the wedges fit to the boundaries.

The fast wedge motion compensation technique is performed with the same wedge dictionaries as described before, and also the same codification is used. The main difference is the pre-selection of which wedge partitions would have motion vectors computed. We used values of $\Delta r = 3$, $\delta r = 1$, $\Delta\theta = 25$, $\delta\theta = 5$, yielding a total of 77 wedge partitions to search for each macroblock and sub-macroblock. This is a large reduction compared to the dictionary size of 2012 or 340 depending on the block size, as discussed above. When a sufficient number of edge pixels are not found in a block, or the error after the regression is too large, regular quadtree partitioning is used within that block. Partitioning examples for the strategy are shown in Fig. 6.

Figure 7 shows rate distortion curves comparing the four methods, the improvement in bit rate is about 4.7% for the wedge partition method and 3.1% for the fast wedge partition technique compared to quadtree partition. Wedge slicing methods use up to 20% less bits than the traditional method with no partition. In terms of PSNR, both techniques yield a gain of about 1.0dB over the traditional method. Compared to AVC-like quadtree partitioning the gain obtained by wedge partitioning is lower. In particular, full wedge partitioning outperforms quadtree by 0.3dB. The fast wedge partition method yields a gain of about 0.2dB over the quadtree-only approach.

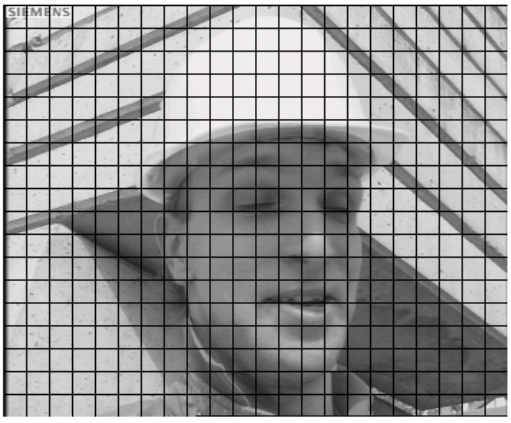


Fig. 3: Motion compensation without macroblock partitioning.

[1] Our comparison tests among macroblock partition schemes using half pixel and quarter pixel precision did not qualitatively differ from the single pixel precision case.

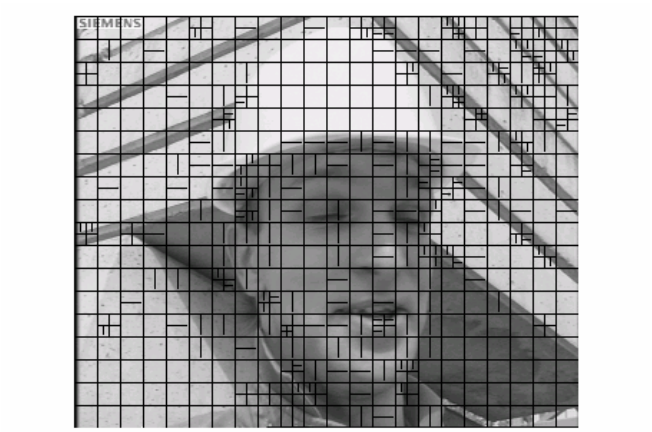


Fig. 4: Motion compensation with quadtree partitioning.

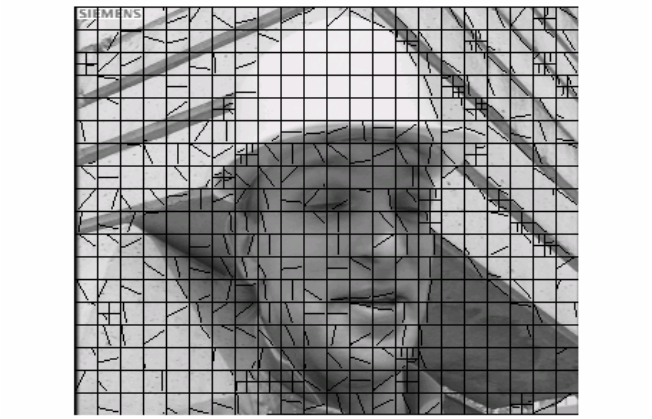


Fig. 5: Motion compensation using the complete wedge partitioning.

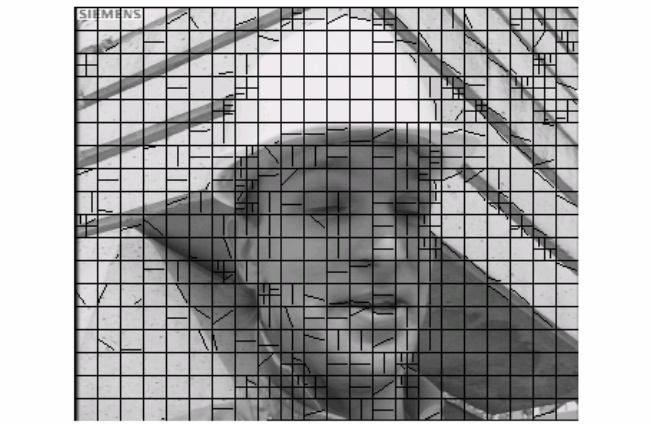


Fig. 6: Motion compensation using fast wedge partitioning.

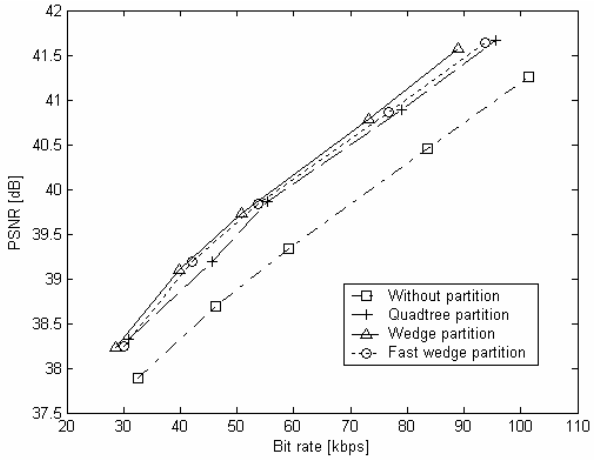


Fig. 7: Rate distortion curves comparing partition block techniques.

5. CONCLUSIONS

In this paper we compare partition blocks techniques for motion compensation, and we also propose a wedge based motion compensation technique. The wedge dictionary size can be scaled by the encoder or application. The larger the dictionary size, the more options we have. However, the more partitions there are, the more expensive it is to encode the partition choice and the more complex compression becomes. We developed a faster partition technique for the wedge case based on edge analysis that dramatically reduces the block partition complexity without any significant impact to the quality.

6. REFERENCES

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

[2] "Advanced Video Coding for Generic Audiovisual Services," *JVT of ISO/IEC MPEG & ITU-T VCEG*, Mar. 2005.

[3] S. Kondo, H. Sasai, "A Motion Compensation Technique Using Sliced Blocks In Hybrid Video Coding," *in IEEE Int. Conf. on Image Proc. – ICIP '05*, Genova, Italy, Sept. 2005.

[4] "Joint Model Number 1 (JM-1)", JVT-A003, *JVT of ISO/IEC MPEG & ITU-T VCEG*, Jan. 2002.

[5] D. L. Donoho, "Wedgelets: Nearly-minimax estimation of edges," *Annals of Stat.*, vol. 27, pp. 859-897, 1999.

[6] W. K. Pratt. "Digital Image Processing: PKIS Inside," Third Edition, John Wiley & Sons, New. York, 2002.