MOTION-COMPENSATION USING VARIABLE-SIZE BLOCK-MATCHING WITH BINARY PARTITION TREES

Marc Servais and Theo Vlachos

CVSSP, University of Surrey
Guildford, GU2 7XH, United Kingdom
m.servais@surrey.ac.uk

Thomas Davies

BBC Research and Development
Kingswood Warren, Tadworth,
Surrey, KT20 6NP, United Kingdom

ABSTRACT

A new approach to Variable Size Block Matching is proposed, based on the binary partitioning of blocks. If a particular block does not allow for accurate motion compensation, then it is split into two using the horizontal or vertical line that achieves the maximum reduction in motion compensation error. This method causes partitioning to occur along motion boundaries, thus substantially reducing blocking artifacts. In addition, small blocks are placed in regions of complex motion, while large blocks cover regions of uniform motion. The proposed technique provides significant gains in picture quality of 1.5 to 3.0 dB, when compared to Fixed Size Block Matching at the same total rate.

1. INTRODUCTION

Motion compensation techniques are an important part of almost all video codecs since they provide an effective way of exploiting the temporal redundancy between frames in an image sequence. Traditionally, Fixed Size Block Matching (FSBM) has been used to determine the motion of a block in the current frame relative to the reference frame(s) [1]. In FSBM, the blocks are a fixed size (e.g. 16×16) and are laid out in a regular grid spanning the frame. One side effect of FSBM is the introduction of blocking artifacts. These occur when a block covers an area where two or more types of motion are present, in which case it is not possible to represent the motion with just one motion vector. This can result in a large motion compensation error within such a block, as well as causing sharp spatial edges between neighbouring blocks in the motion compensated picture.

Chan *et al* [2] introduced Variable Size Block Matching (VSBM), which allows for small blocks to cover areas of complex motion, while regions of uniform motion are spanned by large blocks. Their method partitions the frame into variable size blocks using a binary tree approach. Sullivan and Baker [3] proposed a related VSBM method which

employs Lagrangian optimisation. This produces an optimal quad-tree, which minimises the distortion for a given bit-rate. Rhee *et al* also used a VSBM approach to find the quad-tree that achieves the minimum error for a given number of blocks [4]. Note that in each of these methods, a block is divided into either halves or quarters of equal area.

The H.263 and MPEG-4 coding standards were the first to allow variable size blocks (16×16 or 8×8). In addition, H.264/AVC provides several different macroblock partitioning modes, ranging from 16×16 to 4×4 blocks.

This paper describes a VSBM approach which allows the binary splitting of blocks. However, blocks are not necessarily split into two equal halves. Instead, blocks are split using the horizontal or vertical line that achieves the maximum reduction in motion compensation error. This allows for partitioning a scene along motion boundaries, which enables effective motion compensation using relatively few blocks. As a result, significant gains in rate-distortion performance are possible.

2. MOTION COMPENSATION ERROR SURFACES

In traditional block matching, the goal is to minimise the error between a block in the current frame and a displaced block in the reference frame. The error is usually measured in terms of either the sum of absolute error (SAE) or the sum of squared error (SSE). In effect, motion estimation amounts to finding the location of the minimum value on the error surface. Because they will prove useful later (in Sections 3 and 4), the process of generating motion compensation error surfaces is discussed below in more detail.

For each block, an error surface $E_{b,f}(u, v)$ is calculated, i.e. the SSE when block number b is motion compensated by translating the corresponding block in reference frame number f a distance (u, v) . For all points (x, y) in block b , the SSE between the current frame, I , and a reference frame, I_f , is calculated according to the equation:

$$E_{b,f}(u, v) = \sum_{(x,y) \in \text{Block } b} [I(x, y) - I_f(x + u, y + v)]^2$$

This work has been funded by BBC Research and Development and the Centre for Vision Speech and Signal Processing, University of Surrey.

Fig. 1. Generating motion compensation error surfaces using block matching (relative to the two reference frames).

Fig. 2. Partitioning blocks using error minimisation

For a given block, the translational motion vector (measured to pixel accuracy) can be found by determining the vector (u, v) that minimises $\mathbf{E}_{b,f}(u, v)$. Note that if two reference frames are used (e.g. one past and one future), then two error surfaces per block need to be calculated.

Figure 1 shows the position of a block in the current frame, as well as two typical error surfaces obtained when performing matching using the two reference frames.

3. BLOCK PARTITIONING

When a block contains regions with different motion characteristics, it is difficult to represent the motion accurately using just one motion vector. It is therefore desirable to partition the block into child blocks in such a way that the child blocks contain regions of homogeneous motion. How-

ever, deciding on how (and where) to split a block can be a very complex process, since searching the space of all possible partitions would be computationally expensive. It was therefore decided to adopt a simple approach to block partitioning, as discussed below.

Binary partitioning is used, which allows for a block to be split into two children. This is not a significant constraint, since these children can themselves be divided into two at a later stage. In order to simplify the partitioning process, straight horizontal or vertical lines are used to divide a block into two. While other forms of splitting (e.g. using diagonal or curved lines) may sometimes be more appropriate (in terms of fitting motion boundaries), the use of straight lines for block partitioning makes the process manageable. (If there are fewer options available for splitting, the range of options can be searched more quickly and also coded more efficiently.) One further simplification is to only allow partitioning of the *larger* dimension of a block. Thus if a block has width $>$ height, it is split by a vertical line, otherwise it is partitioned using a horizontal line.

Given the above constraints on permissible block partitions, the goal is to select the partition which allows for the maximum reduction in motion compensation error. The process for determining the optimal partition for a block of height h pixels and width w pixels is outlined below and accompanied by an illustration in Figure 2.

- Divide the block into long thin (non-overlapping) strips that are one pixel thick. If $w > h$ then there are w strips of size $1 \times h$ orientated vertically, and if $h \geq w$ then there are h strips of size $w \times 1$ orientated horizontally. Let $N = \max(w, h)$ be the number of strips.
- For each strip (s_i), calculate its two error surfaces as shown in Section 2: $\mathbf{E}_{s_i,f_A}(u, v)$ and $\mathbf{E}_{s_i,f_Z}(u, v)$, where f_A and f_Z are the two reference frames.
- Consider a partition between strips s_n and s_{n+1} that divides the block into two sets of strips. Then the first half comprises the set $C_1(n) = \{s_1, \dots, s_n\}$, and the second half consists of the set of strips $C_2(n) = \{s_{n+1}, \dots, s_N\}$.
- Error surfaces can then be calculated for each of the two halves by simply adding the error surfaces of their constituent strips:

$$\mathbf{E}_{C_1(n),f}(u, v) = \sum_{s_i=1}^n \mathbf{E}_{s_i,f}(u, v)$$

$$\mathbf{E}_{C_2(n),f}(u, v) = \sum_{s_i=n+1}^N \mathbf{E}_{s_i,f}(u, v)$$

- Then the total minimum SSE for the block when partitioning it between strips s_n and s_{n+1} is given by:

$$E_p(n) = \min \{ \min(\mathbf{E}_{C_1(n),f_A}), \min(\mathbf{E}_{C_1(n),f_Z}) \} + \min \{ \min(\mathbf{E}_{C_2(n),f_A}), \min(\mathbf{E}_{C_2(n),f_Z}) \}$$

The optimal partition point is given by n' , the value of n that minimises $E_p(n)$ (with $1 \leq n < N$). If $E_p(n)$ is constant for all n , then n' is assigned a value of $\lfloor N/2 \rfloor$. In the worst case scenario, $E_p(n')$ will equal the original minimum SSE of the block (in which case the block is simply split down the middle). However, if the motion in the block is non-uniform, $E_p(n')$ will typically have a value less than the original minimum SSE. This means that splitting the block into two child blocks $C_1(n')$ and $C_2(n')$ will yield a reduction in motion compensation error.

4. GENERATING A BINARY PARTITION TREE

The previous section described the process for splitting individual blocks based on minimising motion compensation error. This can be applied iteratively to an entire image in order to generate a block structure that allows for effective motion compensation using relatively few blocks. The strategy used is a split and merge one, where more blocks are split than required, followed by subsequent re-merging of certain blocks. (The blocks chosen to be re-merged are the ones that provide the smallest reductions in SSE when being split). The process operates as follows:

- To generate a binary partition tree, start with one block that is the size of the frame. This block is the root of the binary tree, and to start with is also its only entry.
- Find the block in the tree which has the largest minimum SSE - i.e. the block with the largest value of $E_{min}(b_i) = \min\{\min(E_{b_i, f_A}), \min(E_{b_i, f_Z})\}$, where b_i is the block index. Partition this block using the method described in Section 3 to create two new blocks. These two blocks are added to the tree as children of the partitioned block.
- The previous step is repeated while the number of *leaf* nodes in the tree (i.e. blocks without children) is less than $1.25N_B$, where N_B is the target number of blocks.
- Consider a pair of sibling blocks (b_{s_1} and b_{s_2}), both of which are leaf nodes and share a common parent, b_p . The loss in error that results from pruning these two siblings (i.e. performing motion compensation using the parent block instead of the two child blocks) is given by:

$$E_{Prune}(b_{s_1}, b_{s_2}) = E_{min}(b_p) - E_{min}(b_{s_1}) - E_{min}(b_{s_2}).$$
- Calculate E_{Prune} for each pair of siblings that are also both leaf nodes. Merge the pair which yields the smallest value of E_{Prune} . The two leaf nodes are thus pruned from the partition tree, and their parent becomes a leaf node.
- The previous step is repeated until the number of *leaf* nodes in the tree equals N_B . The end result is a frame consisting of these N_B variable-sized blocks.

Figure 3 shows an example of using a binary partition tree to divide a frame into variable size blocks according to the method outlined above. The background (which has uniform motion) is spanned by relatively large blocks, while

(a) 5 blocks, PSNR_Y = 35 dB

(b) 84 blocks, PSNR_Y = 40 dB

Fig. 3. An example of the growth of a Binary Partition Tree for frame 50 of *Mother & Daughter* (motion compensated from original frames 48 and 52). Each partition minimises the motion compensation SSE for the block that it splits.

smaller blocks cover regions of more complex motion.

Coding the binary partition tree is fairly straightforward, and comprises two components. The first component is the shape of the tree, where each node has two possibilities: either it is not a leaf (in which case it has exactly two children) or it is (in which case it has no children). For a tree with N_B leaf blocks, $N_B - 1$ bits are required to code this information. The second component that is required relates to *where* blocks with children are to be partitioned. (Recall that blocks can be partitioned along their longer dimension.) This information is encoded using arithmetic coding, with a probability model that assumes that splitting a block near the middle is more likely than towards the edges.

5. MOTION ESTIMATION

Motion Estimation plays a critical part in the block partitioning stage, since error surfaces are calculated for each block. Because each block's error surfaces are known, its motion vector (and choice of reference frame) can easily be determined by finding the location of the minimum error for that block. This provides the block's motion vector to pixel accuracy. In order to refine the accuracy to a sub-pixel level, a search is performed in the neighbourhood of the (pixel-accurate) motion vector.

Motion vectors and reference frame selection information are then coded for each block as follows: First, (leaf) blocks are sorted in raster scan order according to the position of their top left corners. This allows for a fixed ordering of blocks, which is reproducible when decoding. For each block, the motion vector difference is entropy coded - relative to the motion vectors of those neighbouring blocks already coded. Similarly, the reference frame used for each block is also predicted from those of neighbouring blocks. (Arithmetic coding is used for entropy coding both the motion and reference frame information.)

Fig. 4. Rate-Distortion plots for *Foreman* and *Stefan*. Results have been averaged over frames 10 to 40 for both sequences. The points plotted show the results when setting the target number of blocks to: $\{50, 99, 198, 396, 594, 792\}$ (regular blocks) and $\{50, 99, 198, 396, 594\}$ (variable-size).

6. RESULTS AND CONCLUSION

The performance of the proposed variable-size block matching approach was tested on 30 frames of the *Foreman* and *Stefan* CIF sequences. For a given target number of blocks, each frame (I_t) was motion compensated from two *original* frames (I_{t-2} and I_{t+2}), a distance of two frames away in time (before and after). For each frame the tree structure was encoded, along with the motion vectors (and choice of reference frame). For comparative purposes, the same tests were performed using fixed size (square) blocks. In this case only the motion information was encoded, since the block structure is regular for a given number of blocks.

Figure 4 illustrates the rate-distortion performance of the two methods. It can be seen that the proposed method provides substantial gains (1.5 - 3.0 dB) over regular block matching, despite the overhead required to represent the structure of the block partition tree. Some subjective results are presented in Figure 5. They show that the proposed method helps to significantly reduce blocking artifacts, which occur when blocks straddle motion boundaries.

The proposed VSBM technique has demonstrated promising results (both objectively and subjectively), and is currently being integrated into a video codec. The main advantage offered by the method is that it enables blocks to be partitioned along motion boundaries. This is done in such a way that each block splitting maximises the reduction in motion compensation error for that block (subject to the constraints of straight-line partitioning).

(a) 396 regular 16×16 blocks

(b) 198 variable-size blocks

(c) After motion compensation using the blocks in (a). Motion Rate: 0.034 bpp

(d) After motion compensation using blocks in (b). Structure + Motion Rate: 0.036 bpp

(e) Motion compensated residual for (c). $\text{PSNR}_Y = 34.6$ dB

(f) Motion compensated residual for (d). $\text{PSNR}_Y = 38.5$ dB

Fig. 5. Frame 32 of *Foreman* motion compensated from original frames 30 and 34. ($\frac{1}{4}$ -pel motion vector accuracy.)

7. REFERENCES

- [1] J R Jain and A K Jain, "Displacement Measurement and its Application to Interframe Image Coding," *IEEE Trans. on Comm.*, vol. COM-29, no. 12, pp. 1799, December 1981.
- [2] M H Chan, Y B Yu, and A G Constantinides, "Variable Size Block Matching Motion Compensation with Applications to Video Coding," *IEE Proc.*, vol. 137, pp. 205, August 1990.
- [3] Gary J Sullivan and Richard L Baker, "Efficient Quadtree Coding of Images and Video," *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 327-331, May 1994.
- [4] Injong Rhee, Graham R Martin, S Muthukrishnan, and Roger A Packwood, "Quadtree-Structured Variable-Size Block-Matching Motion Estimation with Minimal Error," *IEEE Trans Cts. Sys. Video Tech.*, vol. 10, pp. 42, Feb. 2000.