Adaptive Motion Compensation Without Blocking Artifacts

Timothy B. Terriberry

Mozilla, Mountain View, USA

ABSTRACT

The Block Matching Algorithms used in most popular video codec standards introduce blocking artifacts which must be removed via residual coding or deblocking filters. Alternative transform stages that do not cause blocking artifacts, such as lapped transforms or wavelets, require motion compensation methods that do not produce blocking artifacts, since they are expensive to remove. We design a new Overlapped Block Motion Compensation (OBMC) scheme that avoids these artifacts while allowing adaptive blending window sizes. This has the potential to show significant visual quality improvements over traditional OBMC.

1. INTRODUCTION

Most modern video codecs still use straightforward Block Matching Algorithms (BMA)[1] to perform motion compensation, despite their simplicity. As long as the size of the prediction units are larger than the size of the transform units, then any blocking artifacts introduced by the prediction will line up with the block boundaries of a standard DCT, leaving little motivation to avoid them. However, transform stages that do not introduce blocking artifacts, such as lapped transforms or wavelets, require motion compensation methods that are themselves free from blocking artifacts. We propose a new adaptive subdivision scheme for Overlapped Block Motion Compensation (OBMC) that retains the use of large blending windows in the blocks that are not subdivided. Our scheme uses simple window functions that can be computed on the fly, rather than stored in large tables, and admits a dynamic programming algorithm to optimize the subdivision levels with a reasonable complexity.

2. BACKGROUND

Historically, the two main motion compensation methods that avoid blocking artifacts have been Overlapped Block Motion Compensation (OBMC) and Control Grid Interpolation (CGI).

2.1 Overlapped Block Motion Compensation

OBMC was introduced by Watanabe and Singhal expressly to improve prediction near block boundaries.[2] It uses a piecewise constant motion model estimated over a regular grid, like BMA. Blocking artifacts are avoided by weighting the predictions around each grid point with a smoothly varying window function. The prediction gain is due to the blending of several samples of the reference image, which is typically highly spatially correlated. This compensates for motion uncertainty away from the grid points, provided there are no sharp edges.[3,4] The primary disadvantage of OBMC is its tendency to blur sharp features.[5]

Katto and Ohta report an 0.4 dB improvement in SNR of the prediction over BMA by simply using OBMC with motion vectors (MVs) estimated by BMA.[6] Kuo and Kuo report improvements between 0.5 and 0.8 dB vs. whole-pel BMA, and an additional 0.2 to 0.5 dB gain from local iterative refinement of the MVs.[7] The different MV accuracies are important, as Girod points out that the blending used in half-pel BMA has noise-reducing properties similar to those of a multi-hypothesis prediction method like OBMC.[8]

Copyright 2014-2015 Mozilla Foundation. This work is licensed under CC-BY 3.0.
Send correspondence to Timothy B. Terriberry <tterribe@xiph.org>.

2.2 Control Grid Interpolation

CGI, introduced by Sullivan and Baker, interpolates the MVs between grid points to produce a single predictor, which helps avoid the over-smoothing problems of OBMC.[9] It interpolates the MVs between grid points to produce a single predictor, which helps avoid the over-smoothing problems of OBMC. However, when the real motion is discontinuous, such as near an occluding edge, gross warping errors can occur. Related methods use triangular patches[10] or content-adaptive meshes,[11] with affine or projective motion models[12] and varying increases in computational requirements. CGI is difficult to parallelize without complex hardware, such as the texture units in modern GPUs. CGI also has strong 2-D dependencies between adjacent MVs. It shows little to no objective improvements over BMA[9,13] unless the MVs are optimized with expensive searches.[14] Therefore, we do not consider CGI further.

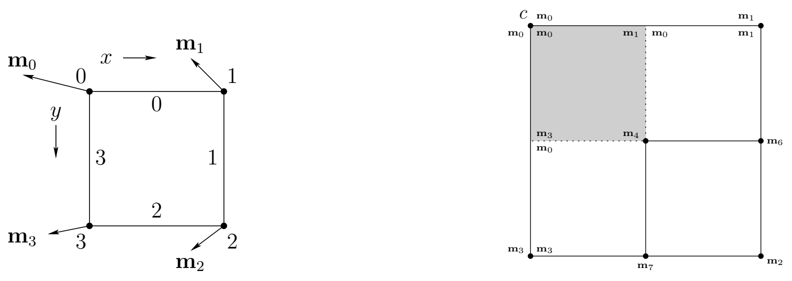
3. ADAPTIVE SUBDIVISION

Varying the block size has long been known to improve the rate-distortion performance of BMA.[15] Zhang *et al.* used variable block sizes with OBMC and achieved improvements of 0.2 to 0.9 dB in SNR over variable size BMA at 60 kbit/s.[16] To avoid blocking artifacts, they subdivide the blocks in the decoded quadtree until none of the interpolation kernels overlap more than one adjacent block in a given direction and then assign the same MV to all of the subdivided blocks. A similar, though simpler, approach was taken in Dirac,[17] where the blocks are always subdivided down to the smallest size. Zhang *et al.*'s results suggest that keeping the blending windows as large as possible improves performance.

We use a bilinear window function since neither the raised cosine nor trapezoidal windows have been shown to be consistently better,[4] and the bilinear window leads to a simpler subdivision scheme. Orchard and Sullivan propose adapting the window function to a sequence to optimize prediction gain, but when iterative motion estimation is used, the advantage over a bilinear window is minimal.[3]

We use square blocks with an origin in the upper-left corner and x and y coordinates that vary between 0 and 1. The vertices and edges of the blocks are indexed in a clockwise manner, as illustrated in Figure 1a. The bilinear weights for each vertex used in blending the predictors are w_k, defined as:

$$w_0 = (1-x)\cdot(1-y) \qquad w_1 = x\cdot(1-y) \qquad w_2 = x\cdot y \qquad w_3 = (1-x)\cdot y\ .$$



(a) Vertex and edge indices for a block. (b) Interpolation setup for unsplit edges.

Figure 1: Interpolation setup for a normal block and for a quadrant resulting from stage one subdivision. c marks the exterior corner, which receives the extra blending weight along the top edge.

Let I be the reference image, and for simplicity denote the predictor $I(x + \mathbf{m}_{k,x}, y + \mathbf{m}_{k,y})$ for the pixel at (x, y) with motion vector $\mathbf{m}_k$ as simply $I(\mathbf{m}_k)$. In a regular grid, with unique motion vectors defined at all four corners of a block, we predict the interior of the block using

$$I(\mathbf{m}_0)\cdot w_0 + I(\mathbf{m}_1)\cdot w_1 + I(\mathbf{m}_2)\cdot w_2 + I(\mathbf{m}_3)\cdot w_3\ . \tag{1}$$

In order to extend OBMC to handle variable block sizes while maintaining continuity along the edges, we start by imposing the restriction that two adjacent blocks differ by at most one level of subdivision, which greatly

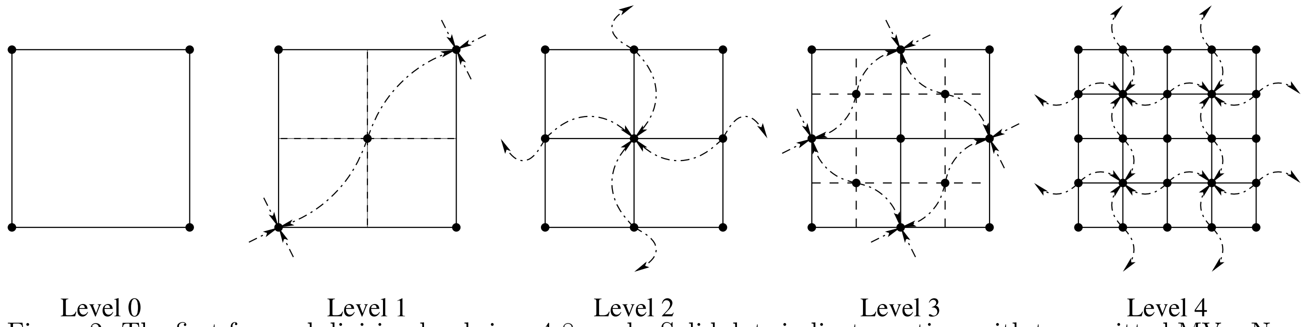


Figure 2: The first four subdivision levels in a 4-8 mesh. Solid dots indicate vertices with transmitted MVs. New vertices at each level have arrows pointing to their two parents. Vertices from the previous level have arrows from each of their four children. In both cases, some may lie in an adjacent block.

simplifies the problem. To do this, we borrow a data structure from the related graphics problem of surface simplification, called the semi-regular 4-8 mesh.[18,19] This is normally used to represent a hierarchical triangle mesh, but we use it for variable-sized quadrilaterals.

The vertices in the 4-8 mesh are arranged in a quadtree, where subdivision proceeds in two stages, as illustrated in Figure 2. In the first stage, a new vertex is added to the center of a quadrilateral. This subdivides the quadrilateral into four *quadrants*, but does not add any additional vertices to the edges. Such edges are called *unsplit*. In the second stage, each of the quadrilateral's edges are split and connected to the center vertex, forming four new quadrilaterals. One useful property of this dual-stage subdivision is that the number of vertices in the mesh merely doubles after each stage, instead of quadrupling as it would under normal quadtree subdivision. This provides more fine-grained control over the number of MVs transmitted.

This is accomplished by assigning every vertex two parents in the tree, which are the two adjacent nodes from the immediately preceding subdivision level. A vertex may not be added to the mesh until both of its parents are present. This ensures adjacent blocks never differ by more than one level of subdivision.

Therefore, we need only specify how to handle a block that has undergone stage one subdivision, but still has one or more unsplit edges, as illustrated in Figure 1b. Such a block is divided into quadrants, and each is interpolated separately using a modified version of Eq. 1.

The same two MVs are used along the unsplit edge(s) as before, but we shift some of the weight used for blending from the middle of the edge to the exterior corner. More precisely, the weights w_k are replaced by modified weights s_k. For example, if c is the index of the vertex in the exterior corner, $\oplus$ denotes addition modulo four, and $c \oplus 1$ is the index of the corner bisecting the unsplit edge, then

$$s_c = w_c + \frac{1}{2} w_{c\oplus 1} \qquad\qquad s_{c\oplus 1} = \frac{1}{2} w_{c\oplus 1} \ .$$

The remaining weights are unchanged. A similar modification is used if it is $c \oplus 3$ that lies on the unsplit edge. The modifications are cumulative. That is, if both $c \oplus 1$ and $c \oplus 3$ lie on unsplit edges,

$$s_c = w_c + \frac{1}{2} w_{c\oplus 1} + \frac{1}{2} w_{c\oplus 3} \qquad s_{c\oplus 1} = \frac{1}{2} w_{c\oplus 1} \qquad s_{c\oplus 3} = \frac{1}{2} w_{c\oplus 3} \qquad s_{c\oplus 2} = w_{c\oplus 2} \ .$$

This strategy clearly matches an adjacent block along an unsplit edge, but careful examination will verify that it also matches other quadrants along the interior edges. Each weight can be evaluated with finite differences at the cost of one add per pixel, plus setup overhead. The blending can be done with three multiplies per pixel by taking advantage of the fact the weights sum to one, just as with regular bilinear interpolation. The mesh itself may require more vertices than the unconstrained meshes of previous work to achieve a given level of subdivision in a local area, but requires fewer bits to encode the subdivision itself, simply because there are fewer admissible meshes. As long as a $(0, 0)$ MV residual can be efficiently encoded, the worst-case rate of the 4-8 mesh should be close to that of a similar, unconstrained mesh.

This process can be extended to handle blocks that differ by more than one level of subdivision, so long as the edge between them remains entirely unsplit. For example, to handle block sizes that differ by a factor of four, instead of shifting half the blending weight from one vertex to the other, one simply needs to shift $\frac{1}{4}$, $\frac{1}{2}$, or $\frac{3}{4}$ of the weight, depending on the location of the block along the unsplit edge. However, the 4-8 mesh is no longer suitable for describing which vertices can appear in the mesh, and some modifications to the adaptive subdivision algorithm in Section 4.2 are required. We have not yet implemented these extensions.

4. IMPLEMENTATION AND MOTION ESTIMATION

The algorithms in Section 3 have been implemented in the Daala video codec,[20] since it uses lapped transforms and we would like to avoid blocking artifacts in the motion compensation. It uses OBMC to produce a complete *motion compensated reference frame*, which is then itself lapped and transformed (in both the encoder and decoder) to make it available as a frequency-domain predictor for the transform stage. The full source code, including all of the OBMC work described in this paper is available in the project git repository.[21]

To reduce aliasing, the reference frames are first upsampled by a factor of two in each dimension using a Wiener filter.[22] This corrects for aliasing at the half-pel locations, where errors are the largest.[23] Samples at finer fractional displacements (up to eighth pel) are bilinearly interpolated from the upsampled image.

Luma blocks are square with sizes ranging from 16×16 to 4×4. At the coarsest resolution, the control points are placed in the center of each 16×16 block, and an extra ring of control points is added outside the image with MVs fixed at $(0, 0)$. This accomplishes several things. It makes the minimum number of transmitted MVs the same as in BMA, it aligns the interpolation kernels with basis functions of the lapped transforms at the largest block size, and it simplifies boundary cases when decoding the adaptive mesh.

We perform rate-distortion (R-D) optimization during motion estimation to balance the error attainable against the number of bits required to achieve it.[24] These two metrics are related by a Lagrangian multiplier in the cost function

$$J = D + \lambda R$$

For a particular choice of λ, a minimal value of J represents a point on the optimal operational rate-distortion curve: i.e., the best achievable distortion for a given rate, or vice versa. The value of λ required to achieve a given rate can be found with a bisection search, or a reasonable estimate may be obtained directly from the target quantizer setting.[25]

Since residual errors after motion compensation are reasonably well-approximated by an exponential distribution, we use the Sum of Absolute Differences (SAD) in the luma plane as our distortion metric.[26] We approximate the rate of small MV components (with a magnitude less than 3 after subtracting a predictor) using statistics from the previous frame, plus one sign bit for each non-zero component. Larger MV components have an additional non-adaptive rate cost added that increases logarithmically with the MV magnitude. The rate term for each vertex also includes a bit for each flag that indicates the presence of a child (2 per vertex on average). The real motion information is adaptively arithmetic encoded, but these approximations avoid having to update the rate term for every vertex every time a single vertex is changed.

We use a median-of-four predictor for almost every motion vector, as illustrated in Figure 3. The middle two values of each MV component are averaged, rounding towards even values. The only exception is if a predictor lies inside a 16×16 block that comes after the current one in raster order, in which case we use the median of the three remaining predictors. This occurs for the level 2 and 4 vertices on the right and bottom block edges. Excluding this predictor allows the mesh to be encoded one block at a time, instead of level-by-level. This permits lower latency in the encoder, and it increases cache coherency and decreases memory footprint in the decoder.

The number of combinations of subdivision levels and MVs available make finding a globally optimal set of parameters impractical. The problem of finding optimal subdivision levels alone is known to be NP-hard.[27] The estimation procedure outlined here attempts to balance speed with compression performance, though it could certainly be improved by future research. It proceeds in several stages.

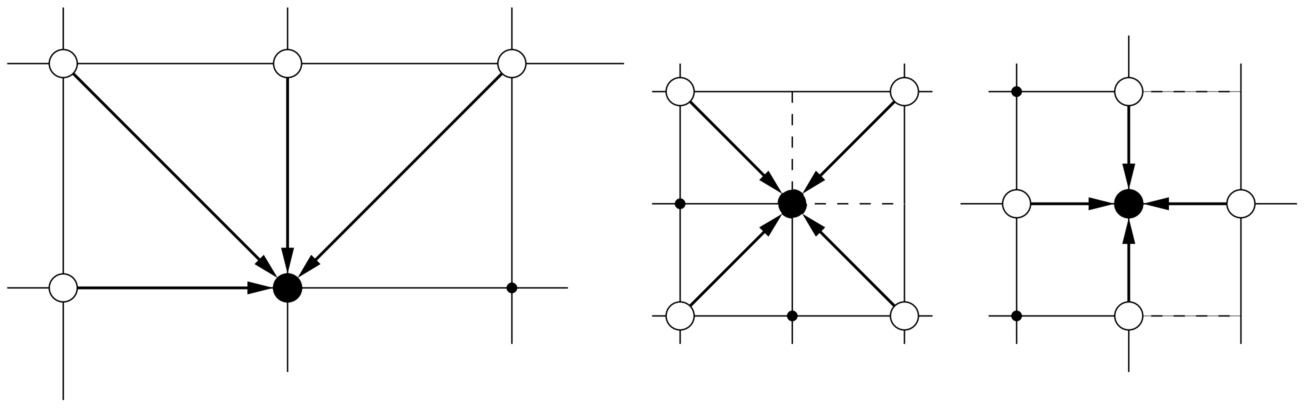


Figure 3: The predictors used for the MVs at each level of the mesh. Except at level 0, these are all ancestors of that MV, and thus are required to be present.

4.1 Initial Estimates

First, a EPZS^2-style algorithm is used to produce an initial MV estimate at each subdivision level using BMA.[28] This algorithm computes several MV candidates using spatial and temporal neighbors and assuming constant speed or acceleration. The candidates are ranked by reliability and the search terminates early if one is found with a SAD below a dynamically chosen threshold. Otherwise, a local gradient search is performed using a square pattern around the best candidate vector. The thresholds ensure extra computation time is spent only on blocks whose predictor can reasonably be expected to improve.

Tourapis does not use a Lagrangian cost function, relying solely on the correlation among predictors and the termination thresholds to achieve smoothly varying, easily compressible MVs. However, incorporating a rate term greatly improves the quality of the result. Therefore we use J to select the best candidate at each stage. We continue to perform threshold checks on the raw SAD values, however.

At level 0, 16×16 blocks centered on the control points are used. Levels 1 and 2 use 8×8 blocks, while 3 and 4 use 4×4 blocks. As the mesh is subdivided, existing vertices do not have their MVs re-estimated with the smaller block sizes, even though the area that MV influences is reduced. The larger support used by previous searches yields more reliable estimates, and the accuracy of BMA is already highest at the block center.[4] All MVs are estimated only up to whole-pel accuracy at this stage.

4.2 Adaptive Subdivision

The second step fixes the mesh subdivision. During this stage, the SAD for each block is computed using OBMC instead of BMA. The MVs produced by the previous stage are held fixed in this one; only the mesh subdivision level changes.

The extra subdivision required to add a vertex to the 4-8 mesh is similar to the implicit subdivision used by Zhang *et al.* in their variable block size OBMC scheme.[16] The difference is that we optimize over and encode such subdivision explicitly. We use a global R-D optimization strategy with general mesh decimations, as proposed by Balmelli.[29] This is a greedy approach that starts with a full mesh and successively decimates vertices. Restricting decimation candidates to the leaves of the mesh often produces sequences where the distortion goes *down* as the rate is decreased, clearly indicating that the previous rate allocation was not optimal. General mesh decimations, on the other hand, allow any vertex to be removed at a given step, not just the leaves. If a non-leaf is decimated, all of its children are decimated as well. This helps smooth out non-monotonicities in the distortion measure during the decimation process, especially at low rates.

The following notation is needed to describe the algorithm. The current mesh is denoted by M, and M_v is the *merging domain* of v in M: the set of vertices in M that must be removed to remove v. This includes v and all of its undecimated children. Additionally, the variation $\Delta \underline{u}(M_v)$ contains the pairs $(\Delta D(M_v), \Delta R(M_v))$: the change in distortion (SAD) and rate caused by removing M_v from M. We also refer to the change in SAD in a single block b caused by removing a single vertex v as $\Delta D_b(v)$. Finally, A_v is the set of ancestors of v in M. Some minor additions to Balmelli's original algorithm are made to handle the fact that distortion is measured over squares, not triangles. The steps of the algorithm are:

1. For all v, compute $\Delta\underline{u}(M_v)$.
2. Do
   (a) $v^* \leftarrow \text{argmin}_{v\in M} -\frac{\Delta D(M_v)}{\Delta R(M_v)}$
   (b) If $-\frac{\Delta D(M_{v^*})}{\Delta R(M_{v^*})} > \lambda$, stop.
   (c) For all $w \in M_{v^*}$, sorted by depth from deepest to shallowest:
      i. For all $a \in A_w$, $\Delta\underline{u}(M_a) \leftarrow \Delta\underline{u}(M_a) - \Delta\underline{u}(M_w)$
      ii. Remove w from the mesh.
      iii. If w was on an even level, then for each adjacent block b with a $w' \in M$ such that w' lies on the same level as w:
         A. $\delta D \leftarrow$ ($\Delta D_b(w')$ for that block after decimating w) $-$ ($\Delta D_b(w')$ for that block before decimating w)
         B. For all $a \in \{w'\} \cup A_{w'} \setminus A_w$, $\Delta D(M_a) \leftarrow \Delta D(M_a) + \delta D$

These steps ensure that $\Delta\underline{u}(M_v)$ contains the up-to-date changes in the global rate and distortion after each merging domain is decimated. This update process properly accounts for overlapping merging domains due to an inclusion-exclusion principle; see Balmelli for details.[29] Step 2(c)iii handles the case of decimating one corner of a block, w, when the opposite corner, w', remains. This changes $\Delta D_b(w')$, the cost of decimating the opposite corner, and that change must be propagated to each merging domain to which w' belongs. No change needs to be made to the common ancestors of w and w' however: once $\Delta D(M_{w'})$ is updated, the normal update process is sufficient. The addition of these extra steps does not affect the computational complexity of the algorithm, which is $\Theta(n \log n)$, where n is the size of the initial mesh.

The distortion measurements needed to initialize and update $\Delta\underline{u}(M_v)$ can be computed once, in advance, by computing the SAD value of each block in all sizes and with all possible combinations of unsplit edges. All told, each pixel in the image is used in exactly nine SAD computations (one for the largest block size, with no unsplit edges, and four for each additional block size). Also, since the mesh only undergoes four levels of subdivision, there are only a small number of unique merging domains and ancestor sets. These can be computed offline and stored in tables to simplify the decimation process. To compute the set difference $A_{w'} \setminus A_w$, we note that w and w' share a single common parent, p. The common ancestors of w and w' are now formed by the set $\{p\} \cup A_p$, meaning one can add δD to the nodes in $A_{w'}$ and then subtract it from the nodes in $\{p\} \cup A_p$ to effect the set difference in Step 2(c)iiiB. Alternatively, one could simply use a larger set of lookup tables.

4.3 Iterative Refinement

The next stage uses the iterated dynamic programming (DP) proposed by Chen and Willson to refine the MVs, accounting for their interdependencies.[14] In this scheme, a single row (resp. column) of MVs is optimized at a time using a Viterbi trellis,[30] while the rest remain fixed. If there is no direct block edge between two consecutive MVs in a row (resp. column) then the trellis stops, and a new one is started. This continues until the entire row (resp. column) has been examined. The process is then repeated until the total change in Lagrangian cost J falls below a given threshold.

4.3.1 Rate and Distortion Changes

We use the *change* in rate and distortion to compute the cost of each path in the trellis. A single MV can influence the distortion of as many as 12 neighboring blocks. Only the ones to the left (resp. above) are added to the current cost of each path. When the following MV is chosen, an additional 2 to 8 blocks may be added. If necessary, the blocks to the right (resp. below) are added after the last MV in the trellis.

Unfortunately, the rate of a MV depends on the values of the MVs used to predict it. Chen and Willson assume MVs use 1-D differential coding, as in MPEG-1, but modern video codecs can use several neighboring MVs to predict the current one. With our prediction scheme, several (not necessarily consecutive) MVs on the DP path may be used to predict a given MV, and the corresponding change in rate is not known until a MV has been chosen for all of them.

If we were to consider all possible combinations of candidates for the predictors, the number of trellis edges would increase by several orders of magnitude. This seems excessively wasteful, since as long as the changes to the MVs are small, the median operation ensures only one of them is likely to have any influence on the predicted value in the first place. Instead, we immediately compute the rate change in each predicted vector—excluding those that themselves lie further along the DP path, since we do not yet know what MV will be encoded. We do this assuming all MVs not already considered by the DP remain fixed, and add the change to the cost of the current path. If changing a subsequent MV on the path causes the rate of one of these predicted MVs to change again, the new rate change is used from then on.

Because we essentially discard a large number of trellis states of limited utility, we might theoretically discard the path that does not change any MVs, even though its true cost is lower than the ones we keep. Thus, as a safety precaution, we check the final cost of the best path, and do not apply it if it is greater than zero. This does occur in practice, but very rarely.

Other, simpler alternatives to this approach are also possible. For example, we tried only considering rate changes for MVs on the actual DP path, which is much like Chen and Willson's approach. However, on frames with complex motion, we have seen dramatic improvements in visual quality and motion field smoothness by properly accounting for *all* of the rate changes. This is because a level 0 MV, for example, may be used to predict up to 20 other MVs, only 6 of which lie on a given DP path. In a dense mesh, the rate changes off the path may dominate the ones on it.

4.3.2 Complexity Reduction

Chen and Willson showed that using a logarithmic search instead of an exhaustive one for the DP resulted in an average PSNR loss of only 0.05 dB and an average MV bitrate increase of 55 bits per frame. We take an even more aggressive approach, and replace the logarithmic search with a diamond search.[31] Because the complexity of a given DP chain increases quadratically in the number of MV candidates at each node, reducing the candidate count can give a substantial performance boost.

The logarithmic search uses a 9-candidate square pattern in each stage. The displacements used in the pattern shrink by a factor of two in each phase. Chen and Willson used a 3-phase search to achieve an effective search radius of $\pm 7 \times 7$ pixels. In our framework, this requires examining $3 \times 9^2 = 243$ trellis edges per MV for one full iteration. However, the large displacement patterns only alter a small number of MVs that have usually been grossly mis-estimated. They are even likely to cause further mis-estimation in areas with repeated structure or a lack of texture, which makes further refinement less effective.

One alternative is to simply discard them, and only perform the square pattern search with one-pixel displacements. The chance of being trapped in a local minimum is increased, but three times as many iterations can be performed in the same amount of time. On the other hand, a small diamond search pattern has only 5 candidates, making it even more attractive.

The diamond search (DS) algorithm has several problems when used for BMA. It fails to identify strong global motions, and at higher resolutions produces non-smooth motion fields due to an increase in large, uniform areas. However, EPZS[2] does an excellent job of identifying global motion, and the use of a Lagrangian cost function helps ensure easily compressible MVs. Although the performance advantage of DS over a square search is slight in BMA, in a DP context, the small diamond only requires examining $5^2 = 25$ trellis edges per MV per

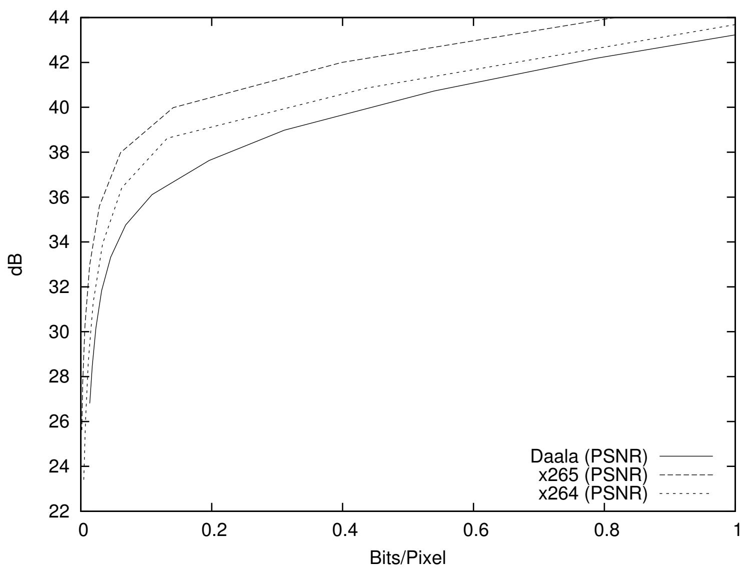


Figure 4: PSNR performance of Daala and the most popular open-source HEVC and H.264 encoders.

iteration. This allows more than nine times as many iterations as the full logarithmic search in the same amount of time.

The computational complexity is still relatively high. In a single iteration, each of the four edges of a block are traversed exactly once by a DP path, during which its SAD is evaluated 25 times, for a total of 100 SAD calculations per block. This is nearly as many as full search BMA with a $\pm 6 \times 6$ window, and computing our blended predictors already has higher complexity. Thus it is not suitable for a real-time implementation.

4.4 Sub-pel Refinement

The same small diamond-pattern search can be used to refine the motion vectors to sub-pel precision. Our implementation supports half-, quarter-, or eighth-pel resolution MVs. First, the DP process is iterated with the small diamond and half-pel displacements until the change in Lagrangian cost J for an iteration falls below a given threshold.

Finer resolutions are only used if they provide an overall R-D benefit, which is tested on a frame-by-frame basis. First, iteration is done with quarter-pel displacements, followed, if successful, by eighth-pel. If the decrease in SAD from the finer resolution MVs cannot balance the (approximately) 2 bit per MV increase in bitrate, then the coarser vectors are used instead.

5. RESULTS

We tested the implementation from Section 4[*] on 19 sequences that range in resolution between 416×240 and 2560×1600 pixels in size and are 48 to 60 frames in length. We also encoded the same sequences using x265[†] (using `--preset slow`) and x264[‡] (using `--preset placebo`), and plot their objective quality in terms of PSNR, PSNR-HVS-M,[34] and FastSSIM (a low-compelxity version of multiscale SSIM).[35]

Since Daala includes a number of features which degrade PSNR but improve visual quality as part of the bitstream, such as non-flat quantization matrices, activity masking, and various other parameter tunings, it is no surprise that Figure 4 shows that it does not currently match HEVC or H.264 in PSNR performance. However, on PSNR-HVS-M in Figure 5, which anecdotally we find does a good job of indicating the quality of clean edges, Daala shows an advantage at high rates. Meanwhile, on FastSSIM in Figure 6, which we find does a good job

[*]Commit `bf0663a65926` in the Daala git repository.[21]
[†]Changeset `8952:35d086074bb5` in the x265 Mercurial repository.[32]
[‡]Commit `ea0ca51e9432` in the x264 git repository.[33]

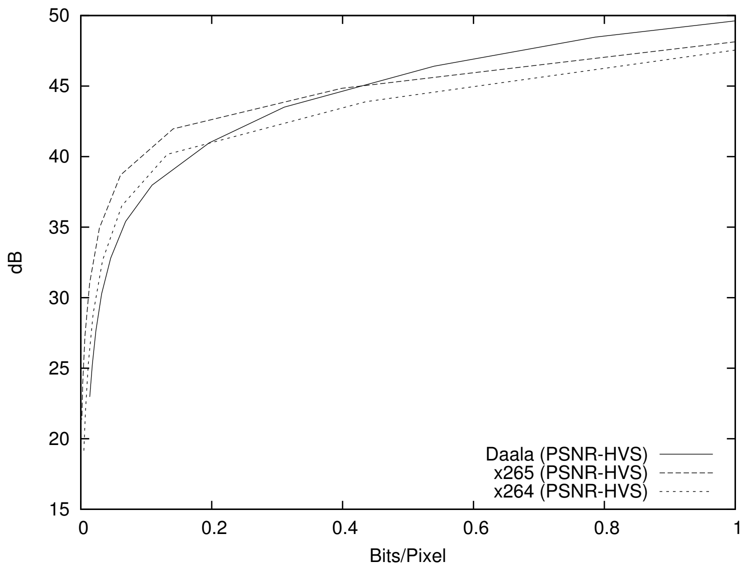


Figure 5: PSNR-HVS-M performance of Daala and the most popular open-source HEVC and H.264 encoders.

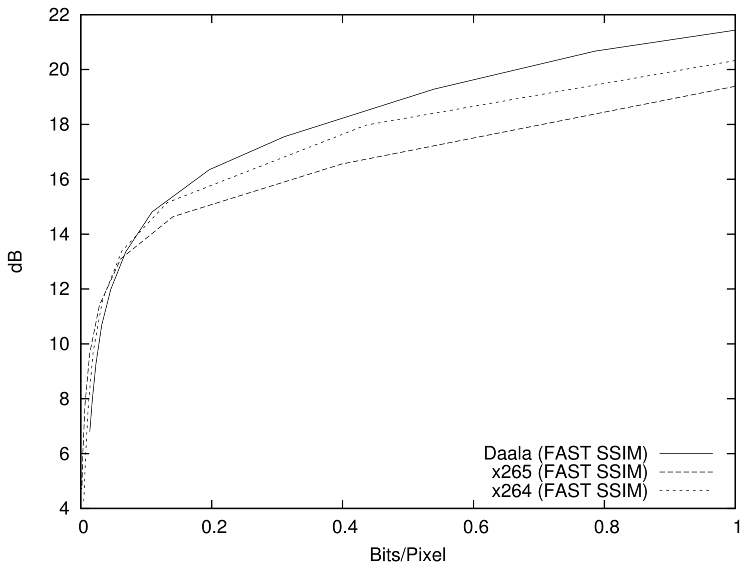


Figure 6: FastSSIM performance of Daala and the most popular open-source HEVC and H.264 encoders.

of indicating how much texture energy is preserved, Daala shows improved performance at almost all rates. Performance at the low end is likely limited by the lack of blocks larger than 16×16 and the lack of B-frames, but extending Daala to support these things is straightforward.

REFERENCES

[1] Dufaux, F. and Moscheni, F., "Motion estimation techniques for digital TV: A review and a new contribution," *Proceedings of the IEEE* **83**, 858–876 (June 1995).

[2] Watanabe, H. and Singhal, S., "Windowed motion compensation," in [*SPIE Visual Communications and Image Processing '91*], **1605**, 582–589 (Nov. 1991).

[3] Orchard, M. T. and Sullivan, G. J., "Overlapped block motion compensation: An estimation-theoretic approach," *IEEE Transactions on Image Processing* **3**, 693–699 (Sept. 1994).

[4] Zheng, W., Shishikui, Y., Naemura, M., Kanatsugu, Y., and Itoh, S., "Analysis of space-dependent characteristics of motion-compensated frame differences based on a statistical motion distribution model," *IEEE Transactions on Image Processing* **11**, 377–386 (Apr. 2002).

[5] Ishwar, P., *On Spatial Adaptation of Smoothness of Motion and Intensity Fields in Video Coding*, Master's thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign (Dec. 1998).

[6] Katto, J. and Ohta, M., "An analytical framework for overlapped motion compensation," in [*Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP-95)*], **4**, 2189–2192 (May 1995).

[7] Kuo, T. and Kuo, C.-C. J., "A hybrid BMC/OBMC motion compensation scheme," in [*Proceedings of the International Conference on Image Processing (ICIP'97)*], **2**, 795–798 (Oct. 1997).

[8] Girod, B., "Efficiency analysis of multihypothesis motion-compensated prediction for video coding," *IEEE Transactions on Image Processing* **9**, 173–183 (Feb. 2000).

[9] Sullivan, G. J. and Baker, R. L., "Motion compensation for video compression using control grid interpolation," in [*IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-91)*], **4**, 2713–2716 (May 1991).

[10] Nakaya, Y. and Harashima, H., "An iterative motion estimation method using triangular patches for motion compensation," in [*Proceedings of the SPIE–Volume 1605, Visual Communications and Image Processing '91: Visual Communication*], Tzuo, K.-H. and Koga, T., eds., 546–557 (Nov. 1991).

[11] Altunbasak, Y., Tekalp, A. M., and Bozdagi, G., "Two-dimensional object-based coding using a content-based mesh and affine motion parameterization," in [*Proceedings of the IEEE International Conference on Image Processing (ICIP'95)*], **2**, 394–397 (Oct. 1995).

[12] Wang, Y. and Lee, O., "Use of two-dimensional deformable mesh structures for video coding, part I—the synthesis problem: Mesh-based function approximation and mapping," *IEEE Transactions on Circuits and Systems for Video Technology* **6**, 636–646 (Dec. 1996).

[13] Altunbasak, Y. and Tekalp, A. M., "A hybrid video codec with block-based and mesh-based motion compensation modes," *International Journal of Imaging Systems and Technology* **9**(4), 248–256 (1998).

[14] Chen, M. C. and Willson, Jr., A. N., "Motion-vector optimization of control grid interpolation and overlapped block motion compensation using iterated dynamic programming," *IEEE Transactions on Image Processing* **9**, 1145–1157 (July 2000).

[15] Lee, J., "Optimal quadtree for variable block size motion estimation," in [*Proceedings of the IEEE International Conference on Image Processing*], **3**, 480–483 (Oct. 1995).

[16] Zhang, J., Ahmad, M. O., and Swamy, M. N. S., "New windowing techniques for variable-size block motion compensation," *IEE Proceedings—Vision, Image, and Signal Processing* **145**, 399–407 (Dec. 1998).

[17] "Dirac specification." http://diracvideo.org/download/specification/dirac-spec-2.2.3.pdf.

[18] Velho, L. and Gomes, J., "Variable resolution 4-k meshes: Concepts and applications," *Computer Graphics Forum* **19**, 195–212 (Dec. 2000).

[19] Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., and Mineev-Weinstein, M. B., "ROAMing terrain: Real-time optimally adapting meshes," in [*Proceedings of the 8th IEEE Conference on Visualization (VIS'97)*], 81–88 (1997).

[20] "Daala website." https://xiph.org/daala/.

[21] "Daala git repository." https://git.xiph.org/daala.git.

[22] Werner, O., "Drift analysis and drift reduction for multiresolution hybrid video coding," *Signal Processing: Image Communication* **8**, 387–400 (July 1996).

[23] Wedi, T. and Musmann, H. G., "Motion- and aliasing-compensated prediction for hybrid video coding," *IEEE Transactions on Circuits and Systems for Video Technology* **13**, 577–586 (July 2003).

[24] Ortega, A. and Ramchandran, K., "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine* **15**, 23–50 (Nov. 1998).

[25] Sullivan, G. J. and Wiegand, T., "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine* **15**, 74–90 (Nov. 1998).

[26] Sebe, N., Lew, M. S., and Huijsmans, D. P., "Toward improved ranking metrics," *IEEE Tranactions on Pattern Analysis and Machine Intelligence* **22**, 1132–1143 (Oct. 2000).

[27] Agarwal, P. K. and Suri, S., "Surface approximation and geometric partitions," in [*Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*], 24–33 (Jan. 1994).

[28] Tourapis, A. M., "Enhanced predictive zonal search for single and multiple frame motion estimation," in [*Proceedings of the SPIE: Visual Communications and Image Processing (VCIP'02)*], Kuo, C.-C. J., ed., **4671**, 1069–1079 (Jan. 2002).

[29] Balmelli, L., *Rate-Distortion Optimal Mesh Simplification for Communications*, PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland (2001).

[30] Forney, Jr., G. D., "The Viterbi algorithm," *Proceedings of the IEEE* **61**, 268–278 (Mar. 1973).

[31] Zhu, S. and Ma, K.-K., "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing* **9**, 287–290 (Feb. 2000).

[32] "x265 Mercurial repository." `https://bitbucket.org/multicoreware/x265`.

[33] "x264 git repository." `git://git.videolan.org/x264.git`.

[34] Ponomarenko, N., Silvestri, F., Egiazarian, K., Carli, M., Astola, J., and Lukin, V., "On between-coefficient contrast masking of DCT basis functions," in [*CD-ROM Proceedings of the* 3^{rd} *International Workshop on Video Processing and Quality Metrics for Consumer Electronics (VPQM '07)*], (Jan. 2007).

[35] Chen, M.-J. and Bovik, A. C., "Fast structural similarity index algorithm," *Journal of Real-Time Image Processing* **6**, 281–287 (Dec. 2011).