Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 7, 2016

T. Terriberry
Mozilla Corporation

July 6, 2015

Overlapped Block Motion Compensation for NETVC
draft-terriberry-netvc-obmc-00

Abstract

This document proposes a scheme for overlapped block motion compensation that could be incorporated into a next-generation video codec. The scheme described is that currently used by Xiph's Daala project, which supports variable block sizes without introducing any discontinuities at block boundaries. This makes the scheme suitable for use with lapped transforms or other techniques where encoding such discontinuities is expensive.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Most current video codecs still use straightforward Block Matching Algorithms (BMA) to perform motion compensation, despite their simplicity. These algorithms simply copy a block of pixels from a reference frame, possibly after applying a sub-pixel (subpel) filter to allow for increased motion resolution. When the motion vectors (MVs) of two adjacent blocks differ, a discontinuity is likely to be created along the block edge. These discontinuities are expensive to correct with transform stages that do not themselves have discontinuities along block edges, such as lapped transforms [I-D.egge-videocodec-tdlt]. Even with a more traditional block-based DCT as the transform stage, the creation of these discontinuities requires that some residual be coded to correct them (and to activate loop filtering along these edges) and requires that the size of a transform block used to code that residual be no larger than the size of a prediction block (or they will suffer the same efficiency problem as lapped transforms in correcting them)

Overlapped Block Motion Compensation (OBMC) avoids discontinuities on the block edges by copying slightly larger blocks of pixels, and blending them together with those of neighboring blocks, in an overlapping fashion. Under the assumption that pixels in the reference frames are highly spatially correlated, this blending compensates for motion uncertainty at the pixels farthest from the estimated MVs. This improves the accuracy of the prediction near block edges, making the expected error more uniform across a block, and improving coding performance over a similar BMA scheme (with

fixed-size blocks) by 0.4 dB [KO95] to 1.0 dB [KK97], depending on the search strategy used.

Non-overlapped BMA schemes that support varying the block size give much better compression than fixed-size schemes [Lee95]. Although previous formats such as Dirac use OBMC, it has always been with a (small) fixed blending window size. The size of a block might vary from, say, 4x4 to 16x16 pixels, with each block given a single MV, but the overlap with neighboring blocks remains fixed, limited by the smallest supported block size to, say, 2 pixels on either side of an edge (the exact sizes in Dirac are configurable, but do not vary within a frame). This is essentially equivalent to performing prediction for the entire frame at the smallest block size (4x4) with an efficient scheme for specifying that the same MV be used for many of the blocks.

We propose a subdivision scheme for OBMC that supports adaptive blending window sizes, allowing much larger blending windows in blocks that are not subdivided, which previous research has suggested should improve prediction performance compared to fixed-size windows [ZAS98]. Our scheme uses simple window functions that can be computed on the fly, rather than stored in large tables, allowing it to scale to very large block sizes. It admits a dynamic programming algorithm to optimize the subdivision levels with a reasonable complexity.

2. Adaptive Subdivision OBMC

Traditional BMA schemes and previous OBMC schemes have a one-to-one correspondence between blocks and MVs, with each block having a single MV. That MV is most reliable in the center of the block, where the prediction error is generally the smallest [ZSNKI02]. Instead, we use a grid of MVs at the corners of blocks. With a fixed-size grid, away from the edges of a frame, this difference is mostly academic: equivalent to shifting block boundaries by half the size of a block in each direction. However, with variable-sized blocks, the distinction becomes more relevant: there is no longer a one-to-one correspondence between blocks and MVs. Under the scheme where MVs correspond to the center of a block, splitting a block removes the old MV at the center of the old block and produces new MVs at the centers of the new blocks. Under the scheme where MVs belong to the corners, splitting a block retains the MVs at the existing corners (corresponding to the same motion as before), but may add new MVs at the new block corners.

We use square blocks with an origin in the upper-left corner and x and y coordinates that vary between 0 and 1 within a block. The

vertices and edges of a block are indexed in a clockwise manner, as illustrated in Figure 1.

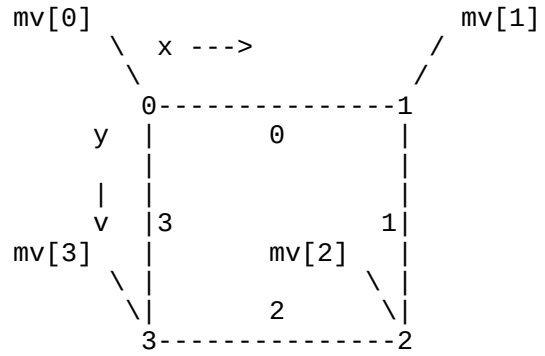


Figure 1: Vertex and Edge Indices for a Block

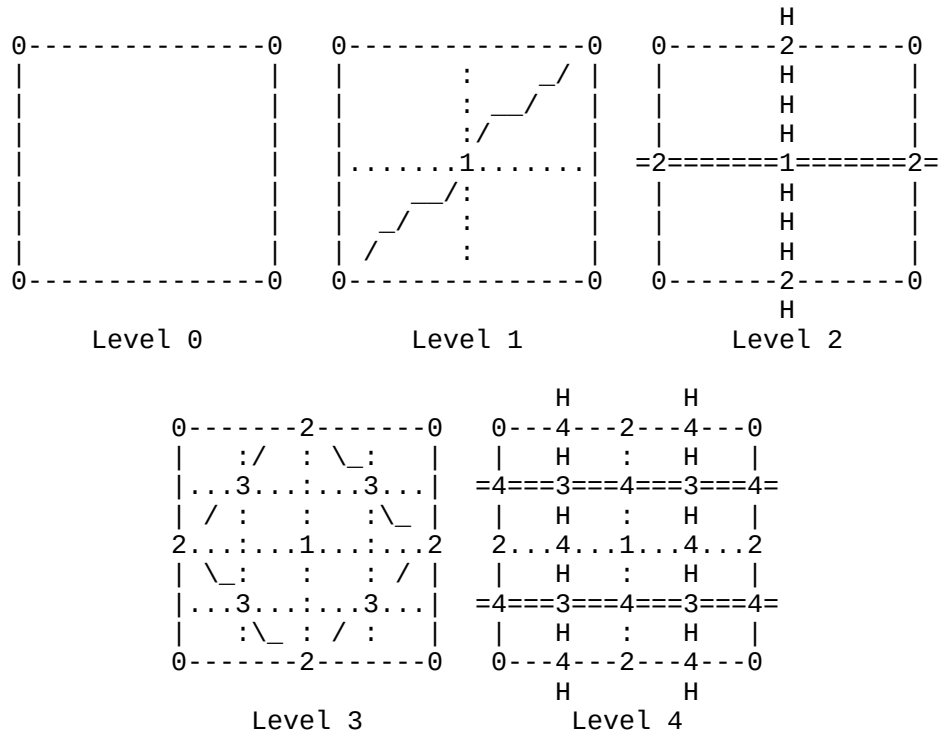
In a block with MVs at all four corners, we use normal bilinear weights to blend the predictions from each MV. The bilinear weights for each vertex, w[k], at a pixel location (x, y) are defined as

```
w[0] = (1 - x)*(1 - y)
w[1] = x*(1 - y)
w[2] = x*y
w[3] = (1 - x)*y
```

Let "I" be the reference image, and for simplicity denote the predictor I[x + mv[k].x, y + mv[k].y] for the pixel at location (x, y) with motion vector mv[k] as simply I(mv[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(mv[0])*w[0] + I(mv[1])*w[1] + I(mv[2])*w[2] + I(mv[3])*w[3]
```

In order to extend OBMC to handle variable block sizes while maintaining continuity along the edges, we start by imposing the restriction that the size of two adjacent blocks differ by at most a factor of two, which greatly simplifies the problem. To do this, we borrow a data structure from the related graphics problem of surface simplification, the semi-regular 4-8 mesh [VG00]. This is normally used to represent subdivisions in a triangle mesh, but we use it for variable-sized quadrilaterals.



The first four subdivision levels in a 4-8 mesh. Numbers indicate vertices with transmitted MVs. Diagonal lines (on odd levels) and double lines (on even levels) connect each new vertex to its two parents at the previous level (in some cases, this parent may lie in an adjacent block). Dotted lines indicate additional block boundaries.

Figure 2: Subdivision Levels in a 4-8 Mesh

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

To ensure that the size of two adjacent blocks differs by no more than a factor of two, we assign every vertex two parents in the mesh, which are the two adjacent vertices from the immediately preceding subdivision level. A vertex may not be added to the mesh until both of its parents are present. That is, a level 2 vertex may not be added to an edge until the blocks on either side have had a level 1 vertex added, and a level 3 vertex may not be added to the center of a block until both of the level 2 vertices have been added to its corners, and so on.

Therefore, we need only specify how to handle a block that has undergone phase one subdivision, but still has one or more unsplit edges, as illustrated in Figure 3. Such a block is divided into quadrants, and each is interpolated separately using a modified version of the previous bilinear weights.

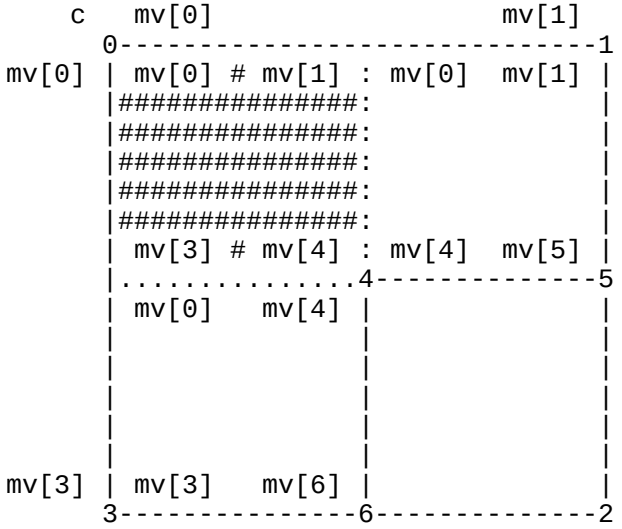


Figure 3: Interpolation Setup for Unsplit Edges

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, (+) denotes addition modulo four, and c (+) 1 is the index of the corner bisecting the unsplit edge (the top edge in the figure), then

$$s[c] = w[c] + 0.5*w[c \oplus 1]$$
$$s[c \oplus 1] = 0.5*w[c \oplus 1]$$

The remaining weights are unchanged. A similar modification is used if it is c (+) 3 that lies on the unsplit edge. The modifications are cumulative. That is, if both c (+) 1 and c (+) 3 lie on unsplit edges (as in the hashed region in Figure 3),

```
      s[c] = w[c] + 0.5*w[c (+) 1] + 0.5*w[c (+) 3]
s[c (+) 1] = 0.5*w[c (+) 1]
s[c (+) 3] = 0.5*w[c (+) 3]
s[c (+) 2] = w[c (+) 2]
```

This defintiion of the blending weights clearly matches an adjacent block along an unsplit edge, regardless of whether or not that block has been split. 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 that the weights sum to one, just as with regular bilinear interpolation.

The mesh itself may require more vertices than an unconstrained mesh 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 admissable 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 1/4, 1/2, or 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 ot the adaptive subdivision algorithm in Section 3.2 are required. We have not yet implemented these extensions.

3. Implementation and Motion Estimation

The algorithms in Section 2 have been implemented in the Daala video codec [Daala-website]. We use them to produce a complete "motion compensated reference frame", which is then lapped and transformed (in both the encoder and decoder) [I-D.egge-videocodec-tdlt] to make it available as a frequency-domain predictor for the transform stage [I-D.valin-netvc-pvq]. The full source code, including all of the OBMC work described in this draft is available in the project git repository at [1].

Luma blocks are square with sizes ranging from 32x32 to 4x4. The corners of the MV blocks are aligned with the corners of the transform blocks. An earlier design had the MV blocks offset from the transform blocks, so that MVs remained in the center of the transform blocks at the coarsest level, with an extra ring of implicit (0, 0) MVs around the frame (to keep the minimum number of transmitted MVs the same as with BMA). However, we found that there was essentially no performance difference between the two approaches (see commit 461310929fc5). Some things are simpler with the current approach (all of the special cases for the implicit (0, 0) MVs go away), and some things are more complicated, but most of the complications are confined to the computation of MV predictors.

The encoder performs rate-distortion (R-D) optimization during motion estimation to balance the prediction error (D) attainable against the number of bits required to achieve it (R), e.g., minimizing

$$J = D + \lambda R$$

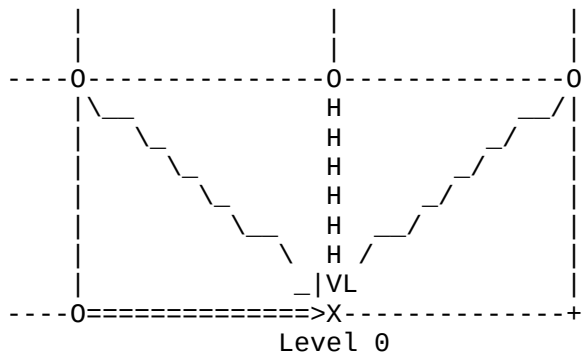
The value of lambda is obtained directly from the target quantizer setting.

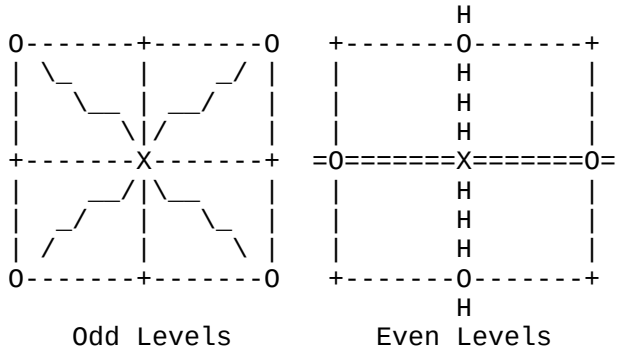
We use the Sum of Absolute Differences (SAD) in the luma plane as our distortion metric for the first three stage of the search, and use the Sum of Absolute Transformed Difference (SATD) during a final subpel refinement stage (with an appropriate adjustment to lambda).

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 [I-D.terriberry-netvc-codingtools], but these approximations avoid having to update the rate term for every MV every time a single MV is changed.

We use a median-of-four predictor for almost every MV, as illustrated in Figure 4. The middle two values of each MV component are averaged, rounding towards even values. There are two exceptions. If an MV required for prediction lies outside the frame, a (0, 0) MV is substituted in its place. If an MV required for prediction lies inside a 32x32 block that comes after the current one in raster order, then that MV is ignored and we use the median of the three remaining MVs. This occurs when predicting MVs on even levels that lie on the right or bottom edges of a 32x32 block. MVs on the top

and left edges of the frame are considered to belong to the 32x32 block below or to the right, respectively (that is, the corresponding MV in that block is not ignored).





Key:
X - MV being predicted
O - MV used for prediction. Except at level 0, these are all ancestors of the MV being predicted, and thus are required to be present.
+ - MV grid point not used for prediction (might not be coded)

Figure 4: The Predictors Used for MVs at Each Level

The current bitstream encodes MVs level-by-level for the entire frame. It is expected at some point that this will be migrated to code all the MVs for a single 32x32 block at a time. This is the reason for excluding predictors outside of the current 32x32 block.

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 [AS94]. The estimation procedure outlined below attempts to

balance speed with compression performance, though it could certainly be improved with future research.

3.1. Initial Estimates

First we produce an initial MV estimate at each point in the fully-subdivided grid using BMA. We compute several MV candidates from spatial and temporal neighbors and assuming constant speed or acceleration. The candidates are grouped into sets by reliability and the search terminates early if the best candidate from a set has 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 be reasonably expected to improve. Although we look solely at SAD to determine whether to continue the search, the candidates themselves are ranked using the full R-D cost metric, J.

Level 0 searches using (non-overlapped) 32x32 blocks centered on the corresponding grid points, while the next two levels use 16x16 blocks, the next two levels 8x8, and so on. MVs are estimated from the coarsest levels to the finest, to allow for the accurate computation of MV predictors used in the rate estimates. As the mesh is subdivided, existing grid points do not have their MVs re-estimated with smaller block sizes, even though the area that those MVs would influence in a grid subdivided to that level is reduced. All MVs are estimated only up to whole-pel accuracy at this stage.

3.2. Adaptive Subdivision

The second stage of motion estimation fixes the mesh subdivision. During this stage, the SAD for each block is computed using full OBMC instead of BMA. The MVs produced in 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 [ZAS98]. 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 Balmeli [Bal01]. This is a greedy approach that starts with a full mesh and successively decimates vertices. Restricting decimation candidates to the leaves of the mesh can frequently produce sequences where decimating a MV (reducing rate) causes distortion to go _down_, clearly indicating that the previous rate allocation was not optimal. General mesh decimations, on the other hand, allow any MV 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 used 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 dU(M_v) contains the pairs (dD(M_v), dR(M_v)): the change in distortion (SAD) and rate (bits) 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 dD_b(v). Finall, 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 dU(M_v).

2. Do

   (a) Let v* be the value of v in M for which -dD(M_v)/dR(M_v) is the smallest.

   (b) If -dD(M_v*)/dR(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, subtract dU(M_w) from dU(M_a).

      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. Let d be change in dD_b(w') before and after decimating w.

         B. For all w in {w'} U A_w' \ A_w, add d to dD(M_a).

These steps ensure that dU(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 [Bal01]. Step 2(c)iii handles the case of decimating one corner of a block, w, when the opposite corner, w', remains. This changes dD_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 dD(M_w') is updated, the normal update process that will be executed when w' is decimated 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 dU(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 13 SAD computations (one for the largest block size, with no unsplit edges, and for for each additional block size). Also, since the mesh only undergoes six 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' \ 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} U A_p, meaning one can add d to the nodes in A_w' and then subtract it from the nodes in {p} U A_p to effect the set difference in Step 2(c)iiiB. Alternatively, one could simply use a larger set of lookup tables.

3.3. Iterative Refinement

The next stage uses the iterated dynamic programming (DP) proposed by Chen and Willson to refine the MVs, accounting for their interdependencies [CW00]. In this scheme, a single row (resp. column) of MVs is optimized at a time using a Viterbi trellis [For73], 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.

3.3.1. 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. 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 or two of them are 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 rate changes. This is because a level 0 MV, for example, may be used to predict up to 24 other MVs, only 8 of which lie on a given DP path. In a dense mesh, the rate changes off the path may dominate the ones on it.

3.3.2. 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. 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 +/- 7x7 pixels. In our framework, this requires examining 3x9**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 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. This allows more than nine times as many iterations as the full logarithmic search in the same amount of time. We support using the diamond search pattern, the square search pattern, and the square search pattern with logarithmic step sizes with various complexity settings, but by default run with only the diamond pattern with a single step size. The logarithmic square pattern search saves between 0.6% and 1.2% rate on metrics, but adds 50% to the total encode time.

The computational complexity of this iterative refinement 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 evaluted 25 times, for a total of 100 SAD calculations per block. This is nearly as many as full search BMA with a +/- 6x6 window, and computing our blended predictors already has higher complexity. Thus it is not suitable for a real-time implementation, but it can easily be disabled, or even lighter-weight versions designed.

3.3.3. Subpel Refinement

The same samll diamond-pattern search can be used to refine the motion vectors to subpel precision. A square pattern is also supported at the highest complexity level, and saves an additional 0.5% to 0.7% on bitrate, but is half the speed of the default settings. Our implementation supports half-, quarter-, or eigth-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

eigth-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.

Subpel interpolation is performed using a separable 6-tap polyphase filter bank. Only eight filters are currently used, one for each subpel offset at eigth-pel resolution. If chroma is decimated (for 4:2:0 video) and eigth-pel MVs are used, then the MV is divided by two and rounded to the nearest even value to select an appropriate subpel filter.

4. References

4.1. Informative References

[I-D.egge-videocodec-tdlt]
Egge, N. and T. Terriberry, "Time Domain Lapped Transforms for Video Coding", draft-egge-videocodec-tdlt-01 (work in progress), March 2015.

[I-D.terriberry-netvc-codingtools]
Terriberry, T., "Coding Tools for a Next Generation Video Codec", draft-terriberry-netvc-codingtools-00 (work in progress), June 2015.

[I-D.valin-netvc-pvq]
Valin, J., "Pyramid Vector Quantization for Video Coding", draft-valin-netvc-pvq-00 (work in progress), June 2015.

[AS94] Agarwal, P. and S. Suri, "Surface Approximation and Geometric Partitions", Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA’94) pp. 24--33, January 1994.

[Bal01] Balmelli, L., "Rate-Distortion Optimal Mesh Simplification for Communications", PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, 2001.

[CW00] Chen, M. and A. Willson, "Motion-Vector Optimization of Control Grid Interpolation and Overlapped Block Motion Compensation Using Iterated Dynamic Programming", IEEE Transactions on Image Processing 9(7):1145--1157, July 2000.

[For73] Forney, G., "The Viterbi Algorithm", Proceedings of the IEEE 61(3):268--278, March 1973.

[KO95] Katto, J. and M. Ohta, "An Analytical Framework for Overlapped Motion Compensation", Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'95) vol. 4, pp. 2189--2192, May 1995.

[KK97] Kuo, T. and C. Kuo, "A Hybrid BMC/OBMC Motion Compensation Scheme", Proc. of the International Conference on Image Processing (ICIP'97) vol. 2, pp. 795--798, October 1997.

[Lee95] Lee, J., "Optimal Quadtree for Variable Block Size Motion Estimation", Proc. of the IEEE International Conference on Image Processing vol. 3, pp. 480--483, October 1995.

[VG00] Velho, L. and J. Gomes, "Variable Resolution 4-k Meshes: Concepts and Applications", Computer Graphics Forum 19(4):195--212, December 2000.

[ZAS98] Zhang, J., Ahamd, M., and M. Swamy, "New Windowing Techinques for Variable-Size Block Motion Compensation", IEE Proceedings---Vision, Image, and Signal Processing 145(6):399--407, December 1998.

[ZSNKI02] Zhen, W., Shishikui, Y., Naemure, M., Kanatsugu, Y., and S. Itoh, "Analysis of Space-Dependent Characteristics of Motion-Compensated Frame Differences Based on a Statistical Motion Distribution Model", IEEE Transactions on Image Processing 11(4):377--386, April 2002.

[Daala-website] "Daala website", Xiph.Org Foundation , <https://xiph.org/daala/>.

4.2. URIs

[1] https://git.xiph.org/daala.git

Author's Address

Timothy B. Terriberry
Mozilla Corporation
331 E. Evelyn Avenue
Mountain View, CA 94041
USA

Phone: +1 650 903-0800
Email: tterribe@xiph.org