A ROBUST METHOD OF DETERMINING CONTEXT-CODER SOLUTIONS FOR ENCODING AFFINE MOTION VECTORS

Roman C. Kordasiewicz†, Michael D. Gallant‡, and Shahram Shirani†

ABSTRACT

The translational motion model can't effectively model complex motion such as scaling, shearing and rotation. That is why more complex motion models like the affine model have been proposed. However, affine motion vectors (AMV)s are more complicated to encode than the translational motion vectors. In this paper we propose several context-coder solutions based on our novel context type limited exhaustive search simulation. As a result the average compression gains of 7.9%, 5.9%, and 9.6%, for `Mobile`, `Cost Guard`, and `Modified Mobile` video sequences were respectively realized, with peek compression improvements of 13%. In addition, our simulation is described and successfully compared with [1].

Index Terms— context determination, affine motion vectors, CABAC, VLC, arithmetic coding,

1. INTRODUCTION

In the search for more efficient ways of encoding video sequences, the affine motion model has been used to very accurately model complex motion. There are however two problems with affine motion vectors (AMV)s. First, they are difficult to compute, however this is slowly offset by constantly increasing computational capabilities. Second, the AMVs are difficult to compress, taking up a significant portion of the compressed video bit rate. To address this issue, we propose in this paper a method for computing context-coder solutions for compressing AMVs.

Affine motion vectors are a subset of polynomial motion vectors [2]. Each motion vector consists of six parameters $\vec{v} = (v_1, v_2, v_3, v_4, v_5, v_6)$. It has been shown by [3] that orthogonalization makes the polynomial coefficients more robust to quantization. This was analyzed further in [4]. Orthogonalized affine motion vectors (AMV) have been also proven in [2, 5, 4] and in the proposed coder [6], and they are the focus of this paper. Typically, AMVs are coded as three symbols: the non-zero pattern, the amplitudes, and the signs. For example if $\vec{v} = (5, 0, 0, 7, 0, -1)$ then the non-zero pattern is $(1, 0, 0, 1, 0, 1)$, the amplitudes are $5, 7, 1$, and the signs are +,+,-. In MVC [6] and in related work, all of these symbols for all of the AMVs are then coded using variable length code (VLC) tables, without leveraging on the correlations between AMVs. One way of exploiting these correlations is to consider the contexts in which AMVs occur. Since context information is also available to the decoder no extra data should be transmitted (also called backward context-adaptive [7]). The context types used in this paper for AMVs are:

Motion Vector (MV) Type: There are two basic types of motion vectors. Type-I, are motion vectors which identify directly the motion between two frames. Type-II, are refinement motion vectors which identify the difference between the predicted motion vector and the current motion vector.

Quantization Step Size (QP): In this paper the most common uniform scalar quantization is considered with step sizes ranging from 2 to 8 as in [6].

Region Size: The orthogonalized AMV are orthogonalized with respect to the region size. Thus two region sizes will be considered 8×8 and 16×16.

Position Within the Vector: There are six coefficients in an AMV and potentially each of these could be coded with a customized coder (or coders).

Neighbour Information: Similarly to the reasoning in [8], the motion within a macroblock is highly correlated to motion in its neighbours.

Of these 5 context types, only "Neighbour Information" context type falls into the traditional definition of a context, as traditionally "The principle of context adaptive coding is to attempt to model the conditional probability of symbols based on their surrounding neighbourhood"[7]. However in this paper the concept of a context is extended to encompass all readily identifiable conditions on an AMV, based on the neighbourhood, the coder state, the motion vector type, and the coefficient type. This extension was deemed necessary such that as many as possible correlations within the AMV can be examined. Throughout this paper, backward context-adaptive methods are used since no additional transmission and coding overhead is desirable. The three symbol encoding approach (pattern, amplitude, and sign) was found to be particularly efficient way of encoding AMVs. We then extend this approach through the use of various context adaptive coders and other improvements, resulting in context-coder solutions.

†Roman C. Kordasiewicz and Dr. Shahram Shirani are with Department of Electrical and Computer Engineering, ITB A320, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S-4K1, their respective emails are kordasi@grads.ece.mcmaster.ca , shirani@mail.ece.mcmaster.ca

‡Dr. Michael D. Gallant is with LSI Logic Corporation, 97 Randall Drive, Waterloo, Ontario N2V-1C5, his email is mgallant@lsi.com

2. FINDING BEST CONTEXT-CODER COMBINATIONS

Since AMVs are coded as three different symbols, three different context-coder solutions will be required. In order to find the best solution we begin by analyzing the encoding of non zero amplitude symbols as an illustration of our approach.

To encode the AMV amplitudes using contexts, the context types defined in the previous section are used. However, "Neighbour Information" context type is represented by a neighbour context index which can be derived with the following algorithm:

1. For AMV i find AMVs from MBs above and to the left. Call this set S_i, and let the size of this set to be N. N is variable since some MBs may not have AMVs and some MBs may be split into smaller regions.

2. Find the summation of the AMVs in S_i. The result is a vector $\vec{u}_i = \sum_{\vec{v} \in S_i} \vec{v}$.

3. For j^{th} AMV coefficient of MB i denoted as $v_{(i,j)}$, find the neighbour context index denoted as $c_{(i,j)}$ according to:

$$c_{(i,j)} = \begin{cases} 0 & \text{if } u_{(i,j)} < 2N \text{ ;} \\ 1 & \text{else if } u_{(i,j)} < 8N \text{ ;} \\ 2 & \text{else if } u_{(i,j)} < 16N \text{ ;} \\ 3 & \text{otherwise;} \end{cases} \quad (1)$$

As a result, the amplitude neighbour context index is calculated by doing few simple additions and shift operations.

In addition to context types, it is possible to encode the various symbols with different entropy coding methods. As part of the study performed in this paper, several coders are considered to give an understanding between coder complexity and possible compression gain. Before any analysis is performed we introduce the types of coders considered in experiments, and these are:

(A) Original MVC: This is the original VLC coder used by [6]. This coder uses the same VLC table irrespective of the context.

(B) Modified MVC - This coder is similar to [6]. However, a context optimal VLC tree is first determined using Huffman codes, and then the symbols are encoded using the optimal VLC tree.

(C) Modified CABAC - This coder is similar to CABAC [8], with the exception that various possibilities for S and k are explored for a given context. S controls the width of the concatenated unary code, and k specifies the k-th order Exp-Golomb code, and both are used in the binarization process. The final value of S and k is chosen to maximize compression for a given context.

When one considers several coder types and the numerous possibilities for the different binarizations (S and k values), then the number of possible coders is very large. In addition, when one considers all of the possible context types, then the search space for the best coder and context type combination is very large. An additional consideration when encoding AMV besides compression efficiency, is coder complexity and the memory footprint. With an already complex and time consuming affine motion estimation, it is desirable to keep the AMV encoding as simple as possible. Thus, in the final solution a few context types should be chosen to keep the memory footprint small, and a relatively fast coder should be used.

To study the compression tradeoffs between various coder and context types a large sample of AMV amplitudes was generated along with side information for context determination. Almost 800000 amplitudes were gathered by encoding the popular QCIF video sequences (`Car-Phone`, `Foreman`, `News`, `Akiyo`, `Trevor`, `Mother and Daughter`, `Hall Objects`, `Clare`) with the MVC coder. Once the amplitudes were gathered, a limited-exhaustive search simulation was performed to find the best context type and coder combination. This search may be best described with the following pseudo-code:

`For i = 0 to 5;` (i represents the number of enabled context types)

- `For j = 1 to` C_5^i`;` (go through all of the possible context type combinations of length i, where C_5^i is the symbol for all combinations of length i out of 5)
  1. `For n = 1 to # of contexts;` (go through all of the contexts for all of the enabled context types at the j^{th} iteration)
     (a) Find the VLC table for Modified MVC coder for the n^{th} context
     (b) Encode the symbols using Modified MVC and using the new VLC table
     (c) `For S = 0 to 12`*
        i. `For k = 2 to 8`*
           A. Find CABAC parameters for n^{th} context and *S,k* binarization
           B. Modified CABAC with the new parameters was used to encode the symbols
           C. Record the best compression results
     (d) Record best results for all coders at n^{th} context
  2. Record the results for the best context type combination of length i

One modification to the above algorithm can be made, if one desired to test all *context-coder* combinations (fully-exhaustive), instead of testing all of the *context type-coder* combinations (limited-exhaustive). The modification would be to iterate through the number of enabled contexts in the outer loop (ie. i = 1 to 672 §for amplitude contexts, and thus j would go from 1 to C_{672}^i). However, this modification would

*The ranges for *S* and *k* were determined experimentally such that the best solutions never fall on these boundaries.

§672 = 2 MV types · 7 QP levels · 2 Region Sizes · 6 Positions within a vector · 4 Neighbour contexts

Table 1. Context types VS. coders for amplitude symbols

Coder	% bit rate reduction	Enabled Context Types				
		MV Type	QP	Region Size	Position	Neighbour
(B)	5.38	Off	Off	Off	On	Off
(C)	7.68	Off	Off	Off	On	Off
(B)	6.70	On	Off	Off	On	Off
(C)	10.38	On	Off	Off	On	Off
(B)	7.87	On	Off	Off	On	On
(C)	10.90	On	Off	On	On	Off
(B)	8.33	On	On	Off	On	On
(C)	11.00	On	Off	On	On	On

greatly increase the complexity of this algorithm without significantly better results. There are two advantages in using our limited search method. First, it is much quicker to iterate through context types, than contexts in the outer loop. Second, actual entropy coders are used in the inner loop to find their relative performance.

The results of the limited search simulation are shown in Table 1[¶]. In this table we compare the % bit rate reduction after encoding a large set of AMV amplitude symbols by two coders, Modified MVC (B) and Modified CABAC (C) relative to the Original MVC (A) coder. As one moves down the rows more context types are enabled. In the "Enabled Context Types" columns we show the contexts that yielded the best performance for a specific coder given a limited number of enabled contexts. For example, in the first two rows only one context type was allowed ($i = 1$), here we see that for both coders the most gain was obtained by using the "Position" context type. When # context types was increased to 3 the Modified CABAC gets 10.9% compression gain, compared to about 7.87% gain the Modified VLC gets. However, the 3 context types that Modified CABAC uses have only 24 contexts ("MV Type", "Region Size" and "Position"), whereas the best 3 context types that the Modified VLC uses have 48 contexts ("MV Type", "Position" and "Neighbour"). Thus, the memory requirements for Modified VLC would be significantly larger than the Modified CABAC, and it would still have worse performance.

In summary, for encoding amplitudes, using three context types may be sufficient, since this yields most of the compression gain. The choice of coder however, is more uncertain since it depends on the amount of hardware available, and on the compression algorithm (R-D optimized or not). Since Modified CABAC has the best performance and a smaller memory requirement than comparable Modified VLC coder, we can choose it and its three context types "MV Type", "Region Size" and "Position" as the context-coder combination for the amplitude symbols.

Similar simulation and reasoning can be used in determining context-coder combinations for the pattern and sign symbols. In the case of the pattern symbols which are highly random using many contexts and complex coders did not prove to greatly improve compression performance. A reasonable compromise for the pattern symbols is to use Modified VLC coder with two context types; "MV Type" and "Neighbour", which results in a fast implementation and a relatively small memory footprint. This combination yields a 2.3% compression gain for the pattern symbols over the Original MVC (A) coder. For the sign symbols, the original MVC coder used one bit per non-zero coefficient to indicate the sign (the simplest VLC coding). However, this type of encoding is extremely efficient since it requires just a few simple arithmetic operations. It is quite suitable for signs, since the probability of either - or + is almost equal. To achieve compression (less than 1 bit per sign) only arithmetic coders can be considered. We selected two context types "MV Type" and "Position" (12 contexts) and Modified CABAC to compress sign bits, which achieves a 3.83% compression gain. In the end we have three context-coder solutions for the three different AMV symbols.

3. RESULTS

In order to verify the three context-coder solutions, they were implemented in the MVC coder, and were tested on four different QCIF video sequences over a range of bit rates. These video sequences were; `Mobile`, `Cost Guard`, `Modified Mobile`. The `Modified Mobile` video sequence was generated from the original `Mobile` video sequence since no other suitable sequence was found that had visible but natural amounts of both scaling and rotation. Overall, our new coders had a very insignificant impact on the encoding time for these video sequences. For `Mobile`, `Cost Guard`, and `Modified Mobile` video sequences the encoding times grew by 2.8%, 1.8%, and 0.2% respectively on average. However, the AMV compression increased by 7.9%, 5.9%, and 9.6% respectively on average. In fact, with more complex video sequences, where more AMV parameters are sent, the compression gain increases. For example, the maximum compression gain of 11.3% and 13.3% was obtained for `Mobile` and `Modified Mobile` respectively.

4. COMPARISON

A significant contribution of this paper involves choosing contexts for AMV symbols. In most context-based coders, contexts are chosen based on the tradeoff between complexity (computation and memory) and coding gain. In this paper, contexts were chosen based on analyzing this tradeoff with the aid of the limited search simulation. There are other methods of partitioning the context space [9, 10, 1]. These methods are best described as context quantizer design. Using all five

[¶]The results when $i = 0$ (all AMVs treated as a single context) are omitted since they are trivial. The results for the case when all context types are enabled ($i = 5$) are also omitted from the table since the compression gain of all the coders either remains the same or it actually drops, as it is expected with this many contexts[1].

context types for encoding amplitude information results in 672 contexts (context space). An optimal split using a subset of the entire context space can be calculated using the algorithms described in [1]. The optimality of this solution is based on the distance measure between two histograms or two probability mass functions. This distance measure known as the Kullback Leibler distance, also known as Relative Entropy. In order to compare the partitioning method presented in [1] to our approach, the algorithm in [1] was implemented and used to quantize amplitude contexts. Once the best quantization was computed by using [1], coders for each quantized context were determined using a simulation. The total number of quantized context levels tried were 6 and 24, since this corresponds to using 1, and 3 context types in Table 1. The average number of bits required to encode an amplitude symbol using this new method for 6 contexts was 3% better then our context-coder result. When the number of contexts is increased to 24, the coding gain due to using quantized contexts drops to 0.1%.

Therefore, it is possible to achieve additional coding gain by using context quantization. However, this advantage is only significant when small number of quantized contexts is used (4 to 8 contexts). There are however three disadvantages of this method. First, a complicated context has to be computed, and this requires the use of neighbour information which is not always the case with our approach. Second, the potentially large context derived with [1] has to be quantized (mapped) to a small set of contexts. In practice when one uses [1] there is no simple mapping between the original and quantized contexts, and a lookup table is necessary. This lookup table can be quite large, for example in the case of amplitude symbols it is 672 deep, and it is 3 or 5 bits wide for 6 and 24 contexts respectively. Overall since 24 contexts are suggested as the final solution in Section 2, the advantage of deriving these 24 context partitions using [1] would yield only an additional 0.1% compression gain, which does not warrant the additional context computation followed by a large lookup table. There is a third disadvantage of using [1], and it is the distance measure used in the algorithm. This distance measure is used as a model and it does not always best describe real coders which may be limited by coder specific constraints. Thus the distance measure does not take into account: the complexity of context selection, the coder complexity, and memory overhead.

5. CONCLUSION

Affine motion vectors, and in particular their orthogonalized versions are an integral part of advanced coders which try to efficiently capture complex motion. This is mainly due to the fact that traditional translational motion vectors cannot effectively express scaling, shearing, and rotation. However AMVs are difficult to compress and can take up a significant portion of the overall bit rate, that is why their efficient coding needs to be explored. In this paper we propose context-coder combinations to best encode AMVs while taking into account the compression performance, coder complexity, and memory foot print. As a result AMV compression gains of up to 13% were easily realized. In addition, the context-coder solutions were developed using our proposed limited search simulation, which greatly simplifies the task of choosing the context-coder solutions. This approach is further discussed and effectively compared to work in [1].

6. REFERENCES

[1] J. Vaisey and Jin Tong, "An iterative algorithm for context selection in adaptive entropy coders," *International Conference on Image Processing*, vol. 3, June 2002.

[2] Marta Karczewicz, Jacek Nieweglowski, and Petri Haavisto, "Video coding using motion compensation with polynomial motion vector fields," *Signal Processing: Image Communiation*, vol. 10, pp. 63 – 91, 1997.

[3] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Norwell, MA, 1995.

[4] Roman C. Kordasiewicz, Michael D. Gallant, and Shahram Shirani, "Modelling quantization of affine motion vector coefficients," *Accepted for publication in the : IEEE Transactions on Circuits and Systems For Video Technology*, February 2006.

[5] Thomas Wiegand, Eckehard Steinbach, and Bernd Girod, "Affine multipicture motion-compensated prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 197 – 209, February 2005.

[6] Nokia Inc. - Nokia Research Center, "MVC coder/decoder submitted to ITU-T," 2000.

[7] Wenqing Jiang and A. Ortega, "Forward/backward adaptive context selection with applications to motion vector field encoding," *International Conference on Image Processing*, vol. 2, Octomber 1997.

[8] T. Wiegand and G. Sullivan, "Draft ITU-T recommendation and final draft international standard of joint video specification," 2003.

[9] S. Forchhammer, Wu Xiaolin, and J.D. Andersen, "Optimal context quantization in lossless compression of image data sequences," *Image Processing, IEEE Transactions on*, vol. 13, pp. 509 – 517, 2004.

[10] Xiaolin Wu, P.A. Chou, and Xiaohui Xue, "Minimum conditional entropy context quantization," *Information Theory, 2000. Proceedings. IEEE International Symposium on*, pp. 43–, 2000.