Fast subpixel motion estimation based on the interpolation effect on different block sizes for H264/AVC

Abdelrahman Abdelazim, MEMBER SPIE, **Mingyuan Yang, and Christos Grecos**, SENIOR MEMBER SPIE
University of Central Lancashire, School of Computing, Engineering and Physical Sciences, ADSIP Research Centre, Preston PR1 2HE United Kingdom
E-mail: CGrecos@uclan.ac.uk

Abstract. We propose a fast subpixel motion estimation algorithm for the H.264 advanced video coding (AVC) standard. The algorithm utilizes the correlation of the spatial interpolation effect on the full-pixel motion estimation best matches between different block sizes in order to reduce the computational cost of the overall motion-estimation process. Experimental results show that the proposed algorithm significantly reduces the CPU cycles in the various motion estimation schemes by up to 16% with similar rate-distortion performance when weighed up against the H.264/AVC standard.
[DOI: 10.1117/1.3095795]

Subject terms: H.264; fast motion estimation; block matching; interpolation.

Paper 080867LR received Nov. 7, 2008; revised manuscript received Dec. 23, 2008; accepted for publication Jan. 13, 2009; published online Mar. 12, 2009.

1 Introduction

The H.264 advanced video coding (AVC) standard[1] is the newest standard from the ITU-T Video Coding Experts Group and the ISO/IEC Moving Pictures Experts Group. Its main advantages are the great variety of applications in which it can be used and its versatile design. This standard has shown significant rate-distortion (RD) improvements as compared to other standards for video compression.

The standard provides great flexibility in the selection of block sizes for motion estimation/compensation, with a minimum luma block size as small as 4×4. Although most prior standards enable half-pixel motion vector accuracy at most, the H264/AVC further allows quarter-pixel motion vector accuracy for improved performance. Although the standard has shown significant RD improvements, it has also increased the overall encoding complexity due to the very refined motion-estimation (ME) process. The ME process consists of two stages: integer-pixel motion search and fractional-pixel motion search. Because the complexity of integer-pixel ME has been greatly reduced by numerous fast ME algorithms,[2,3] the computation overhead required by fractional-pixel ME has become relatively significant.

Different fast fractional-pixel ME algorithms[3–6] have been proposed, and some of them are used by the JM reference software.[7] Their common idea is to simplify the search pattern by applying very refined prediction algorithms and improved adaptive threshold schemes to terminate unnecessary search positions.

In this paper, we focus on decreasing the complexity of fractional-pixel ME by effectively applying a two-step algorithm. First, we examine the 16×16 macroblock fractional-pixel ME best match, derived from the outcome we eliminate the fractional-pixel motion search for 16×8 and 8×16 macroblock partitions. Likewise, in the second step we examine the 8×8 macroblock partitions fractional-pixel ME best matches and, derived from the outcome, we eliminate the fractional-pixel motion search for 8×4, 4×8, and 4×4 macroblock partitions.

Our algorithm differs from the previous methods in two aspects: (i) It uses the similarities between the interpolation effect on the macroblock and its partitions to completely eliminate the fractional-pixel ME. (ii) The proposed algorithm is adaptive and can be applied to any combination of integer and fractional-pixel ME schemes.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the ME algorithms proposed in the H.264/AVC. Section 3 describes the proposed ME algorithm. Section 4 contains a comprehensive list of experiments and a discussion. Section 5 concludes the letter.

2 ME in the H.264/AVC

In the first stage of ME, integer-pixel motion search is performed for each square block of the slice to be encoded in order to find one (or more) displacement vector(s) within a search range. The best match is the position that minimizes the Lagrangian cost function J_{motion}

$$J_{\text{motion}} = D_{\text{motion}} + \lambda_{\text{motion}} R_{\text{motion}} \tag{1}$$

where λ_{motion} is the Lagrangian multiplier, D_{motion} is an error measure between the candidate macroblock taken from the reference frame(s) and the current macroblock, and R_{motion} is the number of bits required to encode the difference between the motion vector(s) and its prediction from the neighboring macroblocks (differential coding). A similar functional to Eq. (1) is used to decide the optimal block size for ME. The most common error measures are the sum of absolute difference (SAD) and the sum of absolute transformed differences (SATD).

After the integer-pixel motion search finds the best match, the values at half-pixel positions around the best match are interpolated by applying a one-dimensional six-tap finite impulse response (FIR) filter horizontally and vertically. Then the values of the quarter-pixel positions are generated by averaging pixels at integer and half-pixel positions. Figure 1 illustrates the interpolated fractional pixel positions. Uppercase letters indicate pixels on the full-pixel grid, while numeric values indicate elements at half-pixel positions and lowercase letters indicate pixels in-between, at quarter-pixel positions.

For example, in Fig. 1, if the integer best match is position E, the half-pixel positions 1–8 are searched using Eq. (1). Suppose position 7 is the best match of the half pixel search. Then the quarter-pixel positions a–h are searched, again using Eq. (1).

0091-3286/2009/$25.00 © 2009 SPIE

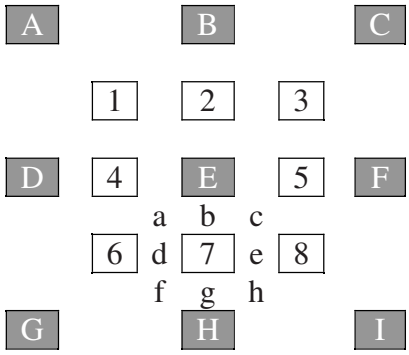

Fig. 1 Fractional pixel search positions.

3 Proposed Scheme

In slow-motion video sequences or in the slow motion segments of fast video sequences, the ME process might find a best-match position during the integer-pixel motion search, which does not change after the subsequent fractional-pixel motion search. Furthermore, if the integer-pixel ME best match for a bigger block size does not change during fractional-pixel motion search, it is "highly likely" that this blocks' partitions integer-pixel ME result will also not change during the fractional-pixel motion search. How likely, depends on the difference in block sizes as demonstrated below. The above observations are shown in Table 1.

Table 1 is divided into three rows. The first row shows the probability that the 16×8 and 8×16 macroblock partitions have the same best match in integer- and fractional-pixel ME, given that the 16×16 macroblock has the same best match in integer- and fractional-pixel ME. We call this probability PROB(1). Similarly, the second row shows the probability of 8×8, 8×4, 4×8, and 4×4 blocks having the same best match in integer and fractional-pixel-ME, given that the 16×16 macroblock has the same best match in integer- and fractional-pixel ME. We call this probability PROB(2). The third row shows the probability of 8×4, 4×8, 8×4, and 4×4 blocks partitions having the same best match in integer- and fractional-pixel ME, given that the 8×8 blocks have the same best match in integer- and fractional-pixel ME. We call this probability PROB(3).These probabilities are averaged across sequences with different motion characteristics and are shown in the second column of Table 1.

From Table 1, it can be seen that the conditional probabilities are reasonably high ($\leqslant70\%$) only when the macroblock/block and their partitions do not differ much in terms of size. For example, we cannot safely say that the 8×4, 4×8, and 4×4 partitions would find the same best match in the integer- and fractional-pixel motion search, given that enclosing 16×16 macroblock does so. In this case, the difference in size is big, because the 16×16 macroblock is 8, 8, and 16 times bigger with respect to the aforementioned block sizes. Using the above insights, we have developed the following scheme:

If the 16×16 macroblock finds the same best match in the integer- and fractional-pixel motion searches, then we disable the fractional-pixel motion search for all the enclosed 16×8 and 8×16 blocks. Thus, we can save all the fractional-pixel search, SAD, and Hadamard transform calculations for these blocks, Otherwise, the fractional-pixel motion search is performed.

Similarly, if the 8×8 block partitions of the 16×16 macroblock find the same best match in the integer- and fractional-pixel motion searches, we disable the fractional-pixel motion search for all the enclosed 8×4, 4×8, and 4×4 blocks. Otherwise, the fractional-pixel motion search is performed.

Table 1 Evaluation of the conditional probabilities.

Probabilities	Average
PROB(1)	70%
PROB(2)	59%
PROB(3)	70%

Table 2 Encoder experiment conditions

Parameter	Value	Parameter	Value
Profile	100 (Main)	YUV format	YUV 4:2:0
Level IDC[7]	40	B-Frame	Not used
Entropy coding	CABAC	Frame skip	0
References	5	Search range	32
ME metric level 0	SAD	ME metric levels 1&2	Hadamard SAD

4 Experiments

To assess the proposed algorithm, a comprehensive set of experiments for a variety of video sequences with different motion characteristics was performed. In this experiment, the source code for the H.264 Reference Software Version JM12.2[7] was used in a Pentium-4 PC running at 2.8 GHz with 1.0 GB RAM. Table 2 illustrates the conditions of the experiments.

Table 3 shows the percentage cycle savings, the Bjontegaard Delta bit rate (BDBR) percentage differences, and the Bjontegaard Delta Peak signal-to-noise ratio (BDPSNR) differences (in decibels)[8] between the H264/AVC and the algorithm we propose when full search (FS), enhanced predictive zonal search (EPZS),[2] and unsymmetrical-cross multi-hexagon-grid search (UMHEXS)[3] are used as full and fractional-pixel ME schemes.

The Intel VTune performance analyzer was used to measure the number of machine cycles differences. Table 3 shows that the BDBR percentage differences are in the range of $[-0.5,1.2]$, while the BDPSNR differences are in the range of $[-0.04,0.02]$. The minus signs denote PSNR degradation and bit-rate savings, respectively.

Table 3 Experimental results.

		Full pixel ME		Sub pixel ME	Full pixel ME		Sub pixel ME	Full pixel ME		Sub pixel ME	Full pixel ME		Sub pixel ME	Full pixel ME		Sub pixel ME
		FFS		FS	UM HEX		UM HEX	UM HEX		FS	EPZS		EPZS	EPZS		FS
Sequence	Size	BDPSNR *(db)*	BDBR *(%)*	Cycles *(%)*	BDPSNR *(db)*	BDBR *(%)*	Cycles *(%)*	BDPSNR *(db)*	BDBR *(%)*	Cycles *(%)*	BDPSNR *(db)*	BDBR *(%)*	Cycles *(%)*	BDPSNR *(db)*	BDBR *(%)*	Cycles *(%)*
Akiyo	QCIF	+ 0.02	− 0.45	6.5	− 0.01	+ 0.34	14.32	0.0	0.0	14.85	− 0.01	+ 0.24	10.9	− 0.01	+ 0.11	14.9
	CIF	− 0.04	− 0.12	8.2	− 0.01	+ 0.43	16.06	− 0.02	+ 0.49	15.89	− 0.01	+ 0.25	11.04	− 0.02	+ 0.49	15.14
Foreman	QCIF	− 0.05	+ 1.2	2.45	− 0.02	+ 0.49	2.94	− 0.02	+ 0.67	3	− 0.03	+ 0.84	3.61	− 0.01	+ 0.24	6.7
	CIF	− 0.01	+ 0.28	2.77	− 0.03	+ 0.72	3.17	− 0.02	+ 0.6	3.23	− 0.04	+ 0.81	3.23	− 0.01	+ 0.29	3.35
Mobile	QCIF	− 0.03	+ 0.43	1.98	− 0.05	+ 0.51	2.09	− 0.04	+ 0.35	2	− 0.03	+ 0.39	2.03	− 0.03	+ 0.28	1.3
	CIF	− 0.01	+ 0.12	1.68	− 0.02	+ 0.31	1.08	− 0.01	+ 0.17	1.21	− 0.01	+ 0.16	1.36	− 0.02	+ 0.38	1.29
Stefan	QCIF	− 0.03	+ 0.51	1.88	− 0.03	+ 0.52	2.2	− 0.04	+ 0.01	2.1	− 0.13	+ 0.2	2.07	− 0.02	+ 0.4	2.4
	CIF	− 0.03	+ 0.51	1.87	− 0.01	+ 0.12	1.7	− 0,01	+ 0.23	2	− 0.01	+ 0.2	1.89	− 0.01	+ 0.24	2.2
Silent	QCIF	− 0.02	+ 0.48	6.2	− 0.01	+ 0.1	13.16	− 0.03	+ 0.48	11.6	− 0.02	+ 0.43	8.92	+ 0.01	− 0.14	11.71
	CIF	− 0.02	+ 0.55	6.01	− 0.01	+ 0.37	11.79	− 0.02	+ 0.47	12.65	− 0.02	+ 0.46	9.37	− 0.02	+ 0.43	12.6
Tempete	QCIF	− 0.01	+ 0.2	1.9	− 0.02	+ 0.25	4.5	− 0.03	+ 0.33	1.92	− 0.03	+ 0.37	2.14	− 0.01	+ 0.29	1.44
	CIF	− 0.01	− 0.01	1.15	− 0.01	+ 0.16	1.08	0.0	+ 0.06	1.4	− 0.01	+ 0.02	1.34	− 0.01	+ 0.18	3.05
Opening ceremony	720× 480	− 0.01	+ 0.2	2.89	− 0.01	+ 0.22	3.67	− 0.01	+ 0.17	4.1	− 0.01	+ 0.22	2.56	− 0.01	+ 0.15	3.2
Driving	720× 480	− 0.01	+ 0.16	1.3	− 0.01	+ 0.12	1.45	0	+ 0.05	1.9	0	+ 0.05	1.80	− 0.02	+ 0.08	1.1

This clearly shows that the proposed algorithm has very similar RD performance to the H.264/AVC. Furthermore, percentage cycle savings up to 16% are observed. It also can be seen that the reduction in the CPU cycles depends on the characteristics of the image sequences. For a slow image sequence with a simple background, the reduction is much more significant than for a fast image sequence or sequences with a more complex background.

5 Conclusion

In conclusion, we proposed a fast Subpixel ME based on the interpolation effect on different block sizes for H264/AVC standard. For RD performance very similar to the standard, the proposed technique can reduce up to 16% of the CPU cycles required for different ME schemes. Our scheme is very relevant to low-complexity video-coding systems.

References

1. T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.* **13**(7), 560–577 (2003).
2. A. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," In *Proc. of VCIP 2002*, pp.1069–1079, SPIE, Jan. 2002.
3. Z. B. Chen, P. Zhou, and Y. He, "Fast integer pel and fractional pel motion estimation for JVT," JVT-F017, 6th meeting, Awaji, Japan, Dec. 5–13 2002.
4. P. Yin, H. Y. C. Tourapis, A. M. Tourapis, and J. Boyce, "Fast mode decision and motion estimation for JVT/H.264," in *Proc. of ICIP 2003*, pp. 853–856, IEEE, Sep. 2003
5. Z. B. Chen, C. Du, J. H. Wang, and Y. He, "PPFPS—a paraboloid prediction based fractional pixel search strategy for H.26L," In *Proc. of ISCAS 2002*, pp. 9–12, IEEE, May 2002.
6. Z. B. Chen and Y. He, "Prediction based directional refinement (PDR) algorithm for fractional pixel motion search strategy," JVT-D069, 4th meeting, Klagenfurt, Austria, July 22–26, 2002
7. K. Sühring, H.264/AVC Reference Software Version JM12.2, http://iphome.hhi.de/suehring/tml/download/, Joint Video Team (2003).
8. G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," Doc. VCEG-M33, Apr. 2001.