SPEEDING UP MOTION ESTIMATION IN MODERN VIDEO ENCODERS USING APPROXIMATE METRICS AND SIMD PROCESSORS

Steven Pigeon and Stéphane Coulombe

Department of Software and
Information Technology Engineering
École de Technologie Supérieure
1100 Notre-Dame Ouest, Montréal, Qc, H3C 1K3

ABSTRACT

In the past, efforts have been devoted to the amelioration of motion estimation algorithms to speed up motion compensated video coding. Now, efforts are increasingly being directed at exploiting the underlying architecture, in particular, single instruction, multiple data (SIMD) instruction sets. The resilience of motion estimation algorithms to various error metrics allows us to propose new high performance approximate metrics based on the sum of absolute differences (SAD). These new approximate metrics are amenable to efficient branch-free SIMD implementations which yield impressive speed-ups, up to 11:1 in some cases, while sacrificing image quality for less than 0.1 dB on average.

***Index Terms*—** Video coding, SIMD, motion estimation, motion compensation, error metric, approximate metric, SSE, SSE2

1. INTRODUCTION

Modern video codecs, such as MPEG-2, MPEG-4, and H.264, rely on motion compensated predictive coding to achieve high compression ratios, which, in turn, implies the use of motion estimation, a process that is quite computationally intensive, even with the best algorithms. The high computational cost of motion estimation makes it an obvious target for optimization. While increasingly clever and efficient motion estimation algorithms have been devised [1–17] and contribute algorithmic speed-ups, implementation-specific speed-ups must rely on the astute use of the underlying machine's instruction set architecture (ISA). Today, this means exploiting data parallelism through single instruction, multiple data (SIMD) ISA extensions.

While efforts have been devoted to the study of fast motion estimation algorithms—finding algorithms that could locate the best matching block faster, principally through a gradient descent type search guided by a predictive algorithm that estimates the general vicinity of the optimal solution—the effects of the metrics used for error estimation have received less attention. The merics available in most codec implementations are limited to the mean squared error (MSE) and the sum of absolute differences (SAD). The MSE metric assesses the goodness of match between two image blocks by computing the mean squared error of corresponding pixels. However, doing so involves the use of multiply instructions which are generally considered to be computationally expensive. For this reason, it was proposed that the SAD be used instead, as it uses the supposedly less expensive sum of absolute differences to assess goodness of match [10]. However, this hypothesis only holds if the processor has instructions to compute the absolute values efficiently; if it does not, it is likely that a single multiply operation compares well to the series of instructions needed to compute the absolute value.

To speed up the computation of the metrics, whether using the MSE or the SAD, itechniques such as early termination, progressive or hierarchical sampling have been proposed, but without really considering the effects of the proposed solution on the underlying machine [6, 18–20]. In many cases, the proposed solutions result in highly branching code which mitigates speed-ups as they interfere greatly with the processor's branching and memory prefetching prediction units, yielding suboptimal results.

In this paper, we propose ways to estimate the metric that yield a good approximation of the SAD while being amenable to branch-free, high-performance SIMD implementations. The paper is organized as follows. In section 2, we outline the proposed solution and the assumptions on which it is based. In section 3, we describe our implementation and simulations. Quality and speed-up results are presented in section 4 and discussed in section 5. We close with concluding remarks.

This work was funded by Vantrix Corporation and by the Natural Sciences and Engineering Research Council of Canada under the Collaborative Research and Development Program (NSERC-CRD 326637-05). E-mails: {steven.pigeon,stephane.coulombe}@etsmtl.ca)

2. PROPOSED APPROXIMATE METRICS

Fast motion estimation algorithms are based on the assumption that the block matching error function is approximately concave with its minimum located at the position of the best match. Motion estimation algorithms are therefore gradient descent-type heuristics. However, no special assumptions are made on the nature of the error function, except that it is approximately concave and the algorithms merely seek to minimize the error by exploring the error surface. If it can be shown that the best motion estimation algorithms are capable of finding good minima regardless of the specific metric, it would imply that it is possible to use this resilience to modify the way the error is computed. One possible way to reduce the cost of computing the error function is to use sampling.

The computational cost of estimating the error metric by sampling depends on several factors. One is the number and arrangement of points sampled. Another is the mathematical operations needed to formulate the estimate from the sampled absolute differences or squared error. Finally, the underlying machine's ISA plays a major role in the efficient implementation of any such method. We consider safe to assume that the target processor sports efficient SIMD instructions and, accordingly, machine-specific optimizations *must* be considered [21–24].

The SAD computed on 16×16 pixels serves as the reference against which we will test the proposed metrics. The SAD between two image patches I and J is given by:

$$\mathrm{SAD}(I, J) = \sum_{x=1}^{16} \sum_{y=1}^{16} \left| I_{x,y} - J_{x,y} \right| \qquad (1)$$

The SAD given by eq. (1) can be transformed to take into account a 16×16 binary matrix M used as a mask that conditionally enables or disables pixels considered in the sum. Transforming eq. (1) to include such a matrix M yields

$$\mathrm{SAD}_M(I, J) = \sum_{x=1}^{16} \sum_{y=1}^{16} M_{x,y} \left| I_{x,y} - J_{x,y} \right| \qquad (2)$$

which may be further scaled by $256 / \sum_{i=1,j=1}^{16,16} M_{ij}$ should it be necessary to obtain a value of the same order of magnitude as the full SAD.

The matrix M allows us to construct arbitrary arrangements. The first sampling arrangement considered, hereafter referred to as the sparse metric, is shown in Fig. 1(a). This metric uses only 64 pixels to formulate an estimate of eq. (1). The second approximate metric considered arranges the points in a quincunx pattern and is referred to as the quincunx, shown in Fig. 1(b). This second pattern is much denser and evaluates 128 points. The proposed metrics, the "deinterlaced" SAD, are shown in Fig. 1(c) and (d). In (d), the metric is sampled in rows of 8 contiguous pixels, the rows being evenly spread over the macro-block. In (c), we propose a similar arrangement where the numbers of rows examined is only 14, but now the sampled lines are spread as evenly as possible over the macro-block. In (d), 50% of the pixels are used while the variant in (c) uses slightly fewer, at 44%. The fifth variant, shown in Fig. 1(e) shows the interlaced pattern that considers only even offset rows (with the first row numbered 0). Finally, in (f), we have the full SAD where all the points are considered.

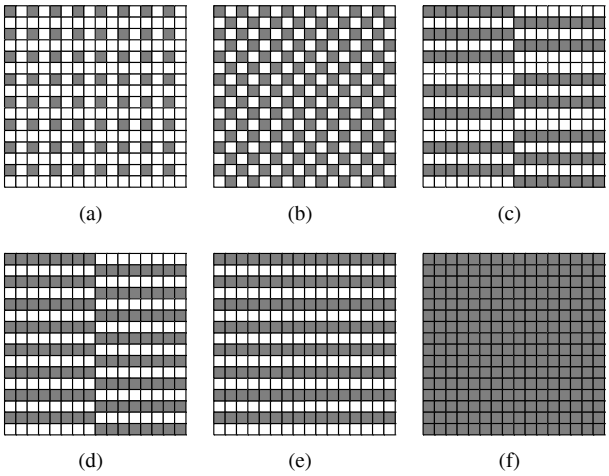

Fig. 1. Approximate metrics. (a) Sparse; (b) Quincunx; (c) Subsampled deinterlaced; (d) Deinterlaced; (e) Interlaced; (f) Full SAD. Shaded squares represent non zero elements in matrix M.

The proposed approximate metrics, the deinterlaced SADs, are structured to sample the macro-block efficiently, prioritizing horizontal movement, which we know is the dominant motion in most scenes. They are also structured to take advantage of fast, machine-specific implementations using SIMD instructions—-provided that the ISA considered sports efficient instructions to compute the SAD on a series of eight contiguous pixels.

3. SIMULATIONS

To confirm the hypothesis that motion estimation algorithms are resilient to approximate metrics, we proceeded in two steps. The first step was to gather standard CIF and QCIF video sequences such as Akiyo, Bus, Foreman, etc. The second step was to implement motion estimation algorithms that allow different metrics to be plugged in. While there is a large number of motion estimation algorithms, only a few are still of interest. Of those, EPZS [10], PMVFAST [14], UMHexS [11] and the Full Search are amongst the most efficient and most commonly implemented methods.

The quality assessment experiments consisted of performing motion estimation and compensation at the macro-block level on the chosen standard video test sequences, using the selected algorithms and approximate metrics, and computing the resulting image quality. In our experiments, we used H.263/MPEG-4 half pixel interpolation with a H.263 search window of -31 to +31.5 pixels in both directions, centered on the macro-block.

The second part of the experiment consisted of measuring the speed-ups in metric evaluation obtained from the various approximate metrics and their efficient implementations, independently of specific motion-estimation algorithms or codecs.

Amongst the many SIMD-capable processors, the x86 ISA processors are by far the most common, dominating both the workstation and server spaces. The x86 ISA sports a series of SIMD extensions, of which MMX, SSE, and SSE2 are the most widely available. We considered it realistic to focus our implementations efforts on the x86 ISA with the SSE and SSE2 SIMD extensions.

To assess speed-ups, we compared the implementations of the approximate metrics, ranging from standard, non vectorized C to the constant-propagated SSE2 assembly code. To ensure a fair comparison, all implementations were optimized equally carefully, within the constraints imposed by the type of implementation. We included results from a third-party library, namely the Intel Integrated Performance Primitives v 6.0 (IPP), to allow the reader to compare our results with what can be expected from an otherwise efficient implementation [25].

The test processor was an Intel Core Duo at 2.0 GHz, a fairly typical processor. The C compiler used was Intel's C Compiler, version 11.0, under Linux Ubuntu 8.04 LTS. Timings were obtained by a system-specific high resolution timer accurate to the μs.

4. RESULTS

Image quality results from the first experiment—the combination of the motion estimation algorithms and the approximate metrics—are summarized in Tables 1 to 6. Figs. 2 to 4 compare the approximate metrics given a motion estimation method on the Foreman CIF sequence. The PSNR for a sequence is the average PSNR between frames and the motion-compensated prediction for those frames. The results are sorted from best to worst, the first number being the PSNR in dB obtained from the full SAD. Other results are shown relative to the full SAD and are expressed as dB loss. The results are shown for CIF and QCIF sequences, using the Full Search, EPZS and PMVFAST motion estimation algorithms.

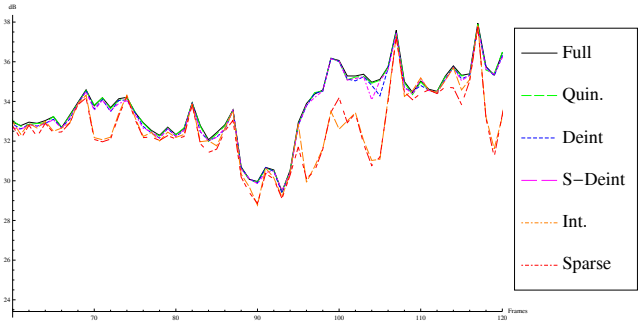

Fig. 2. Approximate metrics results for the Foreman CIF sequence using Full Search.

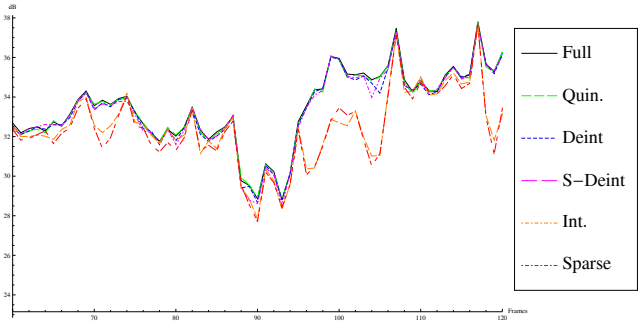

Fig. 3. Approximate metrics results for the Foreman CIF sequence using EPZS.

Tables 7 and 8 show the timing results obtained by the efficient implementations of the approximate metrics during the second phase of the experiments. The timing results are presented in number of completed calls to the SAD per μs, indicative of the absolute speeds of each implementation. The number of processed pixels per μs is also given, so that the raw processing speed versus the cost of complexification for alternate implementations can be assessed. In all cases, the full non-vectorized C SAD implementation (using the basic x86 ISA integer instructions only) is the standard against which speed-ups were compared. The qualifier 'vect.' attached to the implementation names indicates that the compiler auto-vectorization optimizations were enabled. 'SSE2' indicates that the implementations used manually generated SSE2 assembly language, which implies careful vectorization.

5. DISCUSSION

The speed-ups obtained from the proposed approximate metrics, ranging from 4.3 to 11.4:1, are all very interesting. The deinterlaced and sampled deinterlaced metrics yield 9.9 and 11.4:1 speed-ups for the QCIF sequences, and 6.3 and 7.4:1 speed-ups for the CIF sequences.

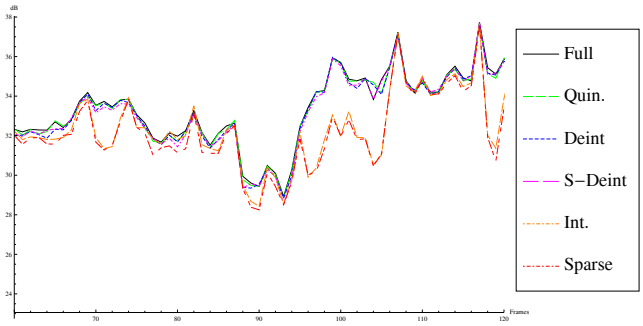

Fig. 4. Approximate metrics results for the Foreman CIF sequence using PMVFAST.

QCIF	Full	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	43.3	-0.01	-0.03	-0.03	-0.05	-0.07
Bus	24.1	-0.03	-0.07	-0.10	-0.11	-0.28
Foreman	31.5	-0.05	-0.06	-0.09	-0.13	-0.18
News	35.9	-0.03	-0.06	-0.07	-0.06	-0.19
Mobile	25.4	-0.04	-0.03	-0.04	-0.04	-0.16
Stefan	25.1	-0.04	-0.07	-0.11	-0.09	-0.22
Tempete	27.2	-0.02	-0.03	-0.03	-0.03	-0.07

Table 1. PSNR (dB) for selected QCIF sequences using full search. SAD, Deint., S-Deint, Int., Quin. stand for the full SAD, the deinterlaced, the subsampled deinterlaced, interlaced, and quincunx patterns, respectively.

From Tables 7 and 8 we see a non negligible difference in timings between the QCIF and CIF results. The QCIF results are, on average, considerably faster—by about 20%—than the CIF results. This can be due to a number of factors, including locality of reference, address generation, and cache replenishment policy: all of which are processor-specific artifacts. Address generation, in our implementation, was all but eliminated by systematic constant propagation, all addresses being expressed relatively to the upper left corners of the macro-blocks for comparison. It may be that an early termination computation of addresses is responsible for the differences as the offsets generated for QCIF are, on average, one bit shorter than for CIF. The effects of CPU-specific

QCIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	43.3	-0.01	-0.03	-0.02	-0.05	-0.07
Bus	23.5	-0.01	-0.02	-0.06	-0.09	-0.20
Foreman	31.3	-0.08	-0.08	-0.11	-0.17	-0.28
News	35.9	-0.04	-0.05	-0.07	-0.05	-0.22
Mobile	25.4	-0.03	-0.02	-0.04	-0.04	-0.15
Stefan	24.8	-0.02	-0.02	-0.03	-0.04	-0.11
Tempete	27.0	-0.03	-0.04	-0.05	-0.03	-0.07

Table 2. PSNR (dB) for selected QCIF sequences using EPZS. See Table 1 for legend.

QCIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	43.2	-0.00	-0.03	-0.03	-0.05	-0.07
Bus	23.5	-0.02	-0.03	-0.08	-0.12	-0.24
Foreman	31.1	-0.07	-0.09	-0.11	-0.20	-0.29
News	35.8	-0.06	-0.07	-0.08	-0.07	-0.22
Mobile	25.3	-0.04	-0.02	-0.04	-0.03	-0.14
Stefan	24.8	-0.03	-0.03	-0.05	-0.05	-0.15
Tempete	27.0	-0.02	-0.05	-0.05	-0.02	-0.09

Table 3. PSNR (dB) for selected QCIF sequences using PMVFAST. See Table 1 for legend.

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	42.8	-0.02	-0.05	-0.07	-0.05	-0.11
Bus	25.1	-0.01	-0.06	-0.10	-0.13	-0.34
Foreman	32.2	-0.03	-0.10	-0.11	-0.76	-0.86
News	36.5	-0.03	-0.06	-0.09	-0.10	-0.23
Mobile	25.2	-0.04	-0.07	-0.08	-0.09	-0.26
Stefan	26.0	-0.01	-0.08	-0.10	-0.12	-0.25
Tempete	27.0	-0.01	-0.04	-0.06	-0.05	-0.12

Table 4. PSNR (dB) for selected CIF sequences using full search. See Table 1 for legend.

behavior will be investigated in future work.

For the QCIF sequences, all metrics, save the sparse variant, perform essentially equally well, leading to maximum losses of the order of 0.1 dB, which is, for all intents and purposes, negligible. The interlaced approximate metric, shown in Fig.1(e), appears to be sensitive to sequences with vertical motion components, such as Foreman, and accordingly performs considerably worse than the other approximate metrics. The loss is relative, however. While it may seem about twice as large as the loss assiociated with other metrics, the difference is only 0.17 dB, which is still quite small in absolute terms.

The same observation holds for the CIF sequences, except that the maximum error produced from the interlaced approximate metric is now much larger; from a mere 0.17 dB loss, it jumps to 0.76 dB, which is now considerable. If the objective is to constrain the maximum loss, the interlaced

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	42.7	-0.02	-0.05	-0.07	-0.06	-0.11
Bus	24.3	-0.00	-0.05	-0.05	-0.17	-0.34
Foreman	31.9	-0.04	-0.11	-0.11	-0.73	-0.83
News	36.2	-0.03	-0.08	-0.12	-0.08	-0.24
Mobile	25.1	-0.03	-0.05	-0.06	-0.07	-0.23
Stefan	25.7	-0.01	-0.09	-0.09	-0.11	-0.22
Tempete	26.5	-0.02	-0.05	-0.07	-0.06	-0.13

Table 5. PSNR (dB) for selected CIF sequences using EPZS. See Table 1 for legend.

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	42.6	-0.02	-0.05	-0.07	-0.06	-0.10
Bus	24.0	+0.03	+0.02	-0.03	-0.14	-0.34
Foreman	31.7	-0.07	-0.14	-0.16	-0.74	-0.85
News	36.1	-0.06	-0.09	-0.12	-0.15	-0.34
Mobile	24.9	-0.04	-0.06	-0.07	-0.08	-0.25
Stefan	25.6	-0.02	-0.08	-0.10	-0.09	-0.23
Tempete	26.4	-0.02	-0.04	-0.08	-0.05	-0.14

Table 6. PSNR (dB) for selected CIF sequences using PMV-FAST. See Table 1 for legend.

	QCIF			
Implementation	pixels	Calls/μs	Pixels/μs	Speed-up
SAD, C	100%	1.40	358.4	1:1
SAD, IPP	100%	7.14	1827.8	5.1:1
SAD, C, Vect.	100%	**7.53**	**1927.7**	**5.4:1**
MSE, C	100%	1.45	371.2	1:1
MSE, C, Vect.	100%	4.45	1139.2	3.2:1
Sparse, C, Vect.	25%	**5.53**	353.9	**4:1**
S-Deint, C, Vect.	44%	3.67	411.0	2.6:1
Quin., C, Vect.	50%	2.67	341.8	1.9:1
Int., C, Vect.	50%	3.46	**442.9**	2.5:1
Deint, C, Vect.	50%	3.20	409.6	2.3:1
SAD, SSE2	100%	8.27	**2117.1**	5.9:1
Sparse, SSE2	25%	13.13	850.3	9.4:1
S-Deint, SSE2	44%	**15.98**	1789.8	**11.4:1**
Quin., SSE2	50%	7.39	945.9	5.3:1
Int., SSE2	50%	14.50	1856.0	10.4:1
Deint, SSE2	50%	13.88	1776.6	9.9:1

Table 7. Timing results (accuracy within $\pm 1\%$) for the various implementations on QCIF images.

	CIF			
Implementation	pixels	Calls/μs	Pixels/μs	Speed-up
SAD, C	100%	1.30	332.8	1:1
SAD, IPP	100%	5.36	1372.2	4.1:1
SAD, C, Vect.	100%	**5.71**	**1461.8**	**4.4:1**
MSE, C	100%	1.41	361.0	1.1:1
MSE, C, Vect.	100%	3.93	1006.1	3.0:1
Sparse, C, Vect.	25%	**4.87**	311.7	**3.7:1**
S-Deint, C, Vect.	44%	3.40	380.8	2.6:1
Quin., C, Vect.	50%	2.42	309.8	1.9:1
Int., C, Vect.	50%	3.33	**426.2**	2.6:1
Deint, C, Vect.	50%	2.97	380.2	2.3:1
SAD, SSE2	100%	5.94	**1520.6**	4.6:1
Sparse, SSE2	25%	9.95	636.8	7.7:1
S-Deint, SSE2	44%	9.60	1075.2	7.4:1
Quin., SSE2	50%	5.63	720.6	4.3:1
Int., SSE2	50%	**10.48**	1341.4	**8.1:1**
Deint, SSE2	50%	8.14	1041.9	6.3:1

Table 8. Timing results (accuracy within $\pm 1\%$) for the various implementations on CIF images.

metric should be avoided, despite offering one of the highest speed-ups. The deinterlaced and sampled deinterlaced metrics also offer high speed-ups, but without the sensitivity of the interlaced metrics to scenes with high vertical motion components; which makes them a very attractive way of balancing speed and quality.

We can see from the timing results that the auto-vectorizing compiler does not always recognize the possible vectorization from the C code, despite being carefully written to help the compiler as much as possible. For example, the C quincunx metric yields a meager 1.9:1 speed-up, while a carefully written SSE2 version yields a much more respectable 5.3:1 for QCIF. It is therefore not sufficient to delegate the generation of efficient SIMD code to the compiler hoping that *efficient* auto-vectorization will occur. Our experiment showed that it is, in fact, fairly difficult to get the compiler to generate efficient SIMD code even given quite carefully crafted C code using specific patterns that should help the compiler to generate vectorized code.

We also note from the tables that even the proposed full SAD implementation beats the IPP v6.0 implementation to some degree. While the IPP implementation is no doubt quite efficient, it is also generic, taking multiple factors into account but without taking full advantage of the image geometry. This allows us to beat the IPP implementation of the full SAD by some 5% to 15%. The speed-up using the approximated metrics are even more interesting, yielding speed-ups of 1.7 to 2.2:1 relative to the IPP implementation, again, with a minimal loss in resulting image quality.

6. CONCLUSION

In this short paper, we have shown that the proposed deinterlaced and sampled deinterlaced approximate metrics yield good image quality, showing a loss of less than 0.1 dB on average compared to the exact SAD when used with the selected motion estimation algorithms. We have also shown that their efficient SIMD implementation yields very high speed-ups—up to 11.4:1—compared to the non vectorized C version of the full SAD. The SIMD-friendly structure of both the deinterlaced and sampled deinterlaced metrics makes them significantly faster than the quincunx metric by a factor of as much as 2:1, despite testing the same number of points, or slightly fewer in the case of the sampled deinterlaced approximate metric. Future work will include characterization of the behavior of approximated metrics in codecs such as MPEG4 and H.264.

7. REFERENCES

[1] J. C. Candy, M. A. Franke, Barry G. Haskell, and F. W. Mounts, "Transmitting television as clusters of frame-

to-frame differences," *Bell Systems Technical Journal*, vol. 50, pp. 1889–1917, Aug. 1971.

[2] Sergio Brofferio and Fabio Rocca, "Interframe redundancy rediction of video signals generated by translating objects," *IEEE Trans. Comm.*, pp. 448–455, Apr. 1977.

[3] Jaswant R. Jain and Anil K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Comm.*, vol. 29, no. 12, pp. 1799–1808, Dec. 1981.

[4] T. Koga, K. Iinuma, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *IEEE NTC*, 1981, pp. G5.3.1–G5.3.5.

[5] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Comm.*, vol. 38, no. 7, pp. 950–953, July 1990.

[6] Bede Liu and André Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits and Systems For Video Technology*, vol. 3, no. 2, pp. 148–157, Apr. 1993.

[7] Renxiang Li, Bing Zeng, and Ming Liou, "A new three-step search algorithm for block-motion estimation," *IEEE Trans. Circuits and Systems For Video Technology*, vol. 4, no. 4, pp. 438–442, Aug. 1994.

[8] Lai-Man Po and Wing-Chung Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits and Systems For Video Technology*, vol. 6, no. 3, pp. 313–317, June 1996.

[9] Ce Zhu, Xiao Lin, and Lap-Pui Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuits and Systems For Video Technology*, vol. 12, no. 5, pp. 349–355, May 2002.

[10] Alexis Michael Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Visual Communications and Image Processing*, Jan. 2002, pp. 1069–1079.

[11] Zhibo Chen, Peng Zhou, and Yun He, "Fast integer and fractional pel motion estimation for JVT," Tech. Rep. JVT-F017, Dec. 2002.

[12] Xuan-Quang Banh and Yap-Peng Tan, "Efficient video motion estimation using dual-cross search algorithms," in *Int. Symposium on Circuits and Systems*, May 2005, pp. 5485–5488.

[13] Chorng-Yann Su, Yi-Pin Hsu, and Cheng-Tao Chang, "Efficient hexagonal inner search for fast motion estimation," in *Int. Conference on Image Processing (ICIP)*, Sept. 2005, pp. 1093–1096.

[14] Alexis Michael Tourapis, Oscar C. Au, and Ming Liou, "Predictive motion vector field adaptive search technique (PMVFAST) - enhancing block based motion estimation," in *Int. Conference on Image Processing (ICIP)*, Jan. 2001.

[15] Hoi-Ming Wong, Oscar C. Au, Chi-Wang Ho, and Shu-Kei Yip, "Enhanced predictive motion vector field adaptive search technique (E-PMVFAST) based on future mv prediction," in *IEEE Int. Conf. Multimedia and Expo.*, July 2005.

[16] Han-Ting Lin, Chih-Yueh Chang, and Jen-Shiun Chiang, "Hierarchical predictable hexagon search algorithm for MPEG4-AVC/H.264 coding," in *IEEE North-East Workshop on Circuits and Systems*, June 2006, pp. 153–156.

[17] Svetislav Momcilovic, Nuno Roma, and Leonel Sousa, "Adaptive motion estimation algorithm for H.264/AVC," in *15th Int. Conf. on Digital Signal Processing*, July 2007.

[18] Federico Tombari and Stefano Mattoccia, "Template matching based on the l_p norm using sufficient conditions with incremental approximations," in *Procs. IEEE int. Conf. on Advanced Video and Signal-Based Surveillance*, Nov. 2006, pp. 20–26.

[19] Chok-Kwan Cheung and Lai man Po, "A hierarchical block motion estimation algorithm using partial distortion measures," *Int. Conference on Image Processing (ICIP)*, vol. 3, pp. 606–609, 1997.

[20] Yui-Lam Chan and Wan-Chi Siu, "New adaptive pixel decimation for block motion vector estimation," *IEEE Trans. Circuits and Systems For Video Technology*, vol. 6, no. 1, pp. 113–118, Jan. 1996.

[21] *Power ISA Version 2.05*, International Business Machines, Inc., Oct. 2007.

[22] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M*, Intel Corporation, Nov. 2008.

[23] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z*, Intel Corporation, Nov. 2008.

[24] "The ARM Cortex-A9 processors," Tech. Rep., ARM Limited, Sept. 2007.

[25] *Intel Integrated Performance Primitives for Intel Architecture Volume 2: Image and Video Processing*, Intel, Sept. 2007.