that our wavelet packet SFQ coder outperforms the state-of-the-art SPIHT coder of [12] by 1.5–2.1 dB. 0.3–0.5 dB of this gain is from the filters (7/9 versus 10/18), 0.8–1.3 dB is from the transform structure (wavelet versus wavelet packet), and 0.5–0.7 dB is due to the quantizer (SFQ versus SPIHT).

We also compare the performance of the sequential wavelet packet SFQ design with the optimal wavelet packet SFQ design for the Barbara image. Numerical PSNR values obtained from the sequential wavelet packet SFQ design are tabulated in Table II. The original Barbara image together with the wavelet packet decomposed and decoded Barbara images at 0.5 b/pixel are shown in Fig. 3. By comparing the coding results in Tables I and II, we see that the performance difference between the joint optimal wavelet packet SFQ design and the suboptimal heuristic is at most 0.1 dB at comparable bit rates. The computational savings of the suboptimal heuristic, however, is quite impressive. In our experiments, we only need to run the SFQ algorithm once in the practical coder, but 85 times in the optimal coder! Based on the near-optimal performance of the practical coder, we henceforth use the practical coder exclusively in our experiments for other images. Results of the wavelet packet SFQ coder for the 512×512 boat and Lena are also tabulated in Table II. The original boat image together with the wavelet packet decomposed and decoded boat images at 0.3 b/pixel are shown in Fig. 4.

To demonstrate the versatility of our wavelet packet SFQ coder, we benchmark against a 768×768 (8 b resolution) fingerprint image using the FBI wavelet scalar quantization (WSQ) standard [15]. Brislawn *et al.* reported a PSNR of 36.05 dB when coding the fingerprint image using the WSQ standard at 0.5882 b/pixel (43366 bytes) [15]. Using the wavelet packet SFQ algorithm, we get a PSNR of 37.30 dB at the same rate. The original fingerprint together with the wavelet packet decomposed and decoded fingerprint images at 0.5882 b/pixel are shown in Fig. 5.

VI. Conclusion

A new wavelet packet SFQ coder is presented in this correspondence. It allows joint transform and quantizer design within the framework of wavelet packet transform and SFQ. A practical wavelet packet SFQ coder is also described in a suboptimal heuristic with sequential optimizations of transform and quantizer. The practical coder achieves significant computational savings, while providing near-optimal coding performance. Our wavelet packet SFQ coder can be considered as a superset of the wavelet-based SFQ coder with coding results no worse than those obtained from the wavelet-based SFQ coder. For some image classes, the wavelet packet SFQ coder produces considerable gain over the wavelet-based SFQ coder. The main advantage of the wavelet packet SFQ coder is its versatility in adapting the wavelet packet transformation to diverse classes of images.

References

[1] Z. Xiong, K. Ramchandran, and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. Image Processing*, vol. 6, pp. 677–693, May 1997.
[2] R. Coifman and M. V. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Trans. Inform. Theory*, vol. 38, pp. 713–718, Mar. 1992.
[3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205–221, Apr. 1992.
[4] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[5] *Subband Image Coding*, J. W. Woods, Ed.. Boston, MA: Kluwer, 1991.
[6] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3445–3463, Dec. 1993.
[7] K. Ramchandran and M. Vetterli, "Best wavelet packet bases in a rate-distortion sense," *IEEE Trans. Image Processing*, vol. 2, pp. 160–176, Apr. 1993.
[8] K. Ramchandran, Z. Xiong, K. Asai, and M. Vetterli, "Adaptive transforms for image coding using spatially-varying wavelet packets," *IEEE Trans. Image Processing*, vol. 5, pp. 1197–1204, July 1996.
[9] Z. Xiong, K. Ramchandran, C. Herley, and M. T. Orchard, "Flexible tree-structured signal expansions using time-varying wavelet packets," *IEEE Trans. Signal Processing*, vol. 45, pp. 333–345, Feb. 1997.
[10] C. Herley, Z. Xiong, K. Ramchandran, and M. T. Orchard, "Joint space-frequency segmentation using balanced wavelet packets trees for least-cost image representation," *IEEE Trans. Image Processing*, vol. 6, pp. 1213–1230, Sept. 1997.
[11] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1445–1453, Sept. 1988.
[12] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.
[13] M. Tsai, J. Villasenor, and F. Chen, "Stack-run image coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 519–521, Oct. 1996.
[14] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, 1987.
[15] J. N. Bradley, C. M. Brislawn, and T. Hopper, "The FBI wavelet/scalar quantization standard for gray-scale fingerprint image compression," in *Proc. VCIP'93*, Orlando, FL, Apr. 1993.
[16] Z. Xiong, K. Ramchandran, M. T. Orchard, and K. Asai, "Wavelet packets-based image coding using joint space-frequency quantization," in *Proc. ICIP'94*, Austin, TX, Nov. 1994, vol. 3, pp. 324–328.

Fast Computation of the Discrete Walsh and Hadamard Transforms

Duraisamy Sundararajan and M. Omair Ahmad

***Abstract*—The discrete Walsh and Hadamard transforms are often used in image processing tasks such as image coding, pattern recognition, and sequency filtering. In this correspondence, a new discrete Walsh transform (DWT) algorithm is derived in which a modified form of the DWT relation is decomposed into smaller-sized transforms using vectorized quantities. A new sequency-ordered discrete Hadamard transform (DHAT) algorithm is also presented. The proposed approach results in more regular algorithms requiring no independent data swapping and fewer array-index updating and bit-reversal operations. An analysis of the computational complexity and the execution time performance are provided. The results are compared with those of the existing algorithms.**

***Index Terms*— Fast algorithms, fast Fourier transform, Walsh and Hadamard transforms.**

I. Introduction

The discrete Walsh and Hadamard transforms are found to be useful in image processing tasks such as image coding, pattern

Manuscript received January 21, 1996; revised October 17, 1996. This work was supported by the Natural Sciences and Engineering Research Council of Canada and by the MICRONET National Network of Centres of Excellence. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ping Wah Wong.

The authors are with the Centre for Signal Processing and Communications, Department of Electrical and Computer Engineering, Concordia University, Montreal, P.Q., Canada H3G 1M8 (e-mail: omair@ece.concordia.ca).

Publisher Item Identifier S 1057-7149(98)03992-X.

1057–7149/98$10.00 © 1998 IEEE

recognition, and sequency filtering, and in design and analysis of digital logic circuits. The discrete Walsh transform (DWT) and the discrete Hadamard transform (DHAT) can be computed using algorithms similar to the discrete Fourier transform (DFT) algorithms. A description of the existing DWT and DHAT algorithms can be found in [2]–[11]. Since the arithmetic complexity ($N \log_2 N$ addition operations) is fixed for these algorithms, i.e., it cannot be further reduced, it is imperative that the focus of the algorithm development be on reducing the structural complexity. The drawbacks of the existing algorithms are that they require overhead operations such as bit-reversals, data-swapping, array-index updating, etc. The execution of these overhead operations contributes considerably to the run time. Recently, a new family of DFT algorithms using an approach that employs vectorized data has been reported [1]. In the present work, this approach is used to derive new DWT and DHAT algorithms, designated as the PM DWT and PM DHAT algorithms[1], that provide enhanced performance in terms of the overhead operations of bit-reversal, data swapping, and array-index updating. As confirmed experimentally, this reduction in the overhead operations, without increasing any other operations, significantly contributes to reducing the execution time, as compared with the existing algorithms [2]–[7].

In Section II, the PM DWT algorithm is derived. In Section III, the sequency-ordered PM DHAT algorithm is developed. The computational complexity and the performance of the software implementation of the proposed algorithms are presented in Section IV.

II. Derivation of the PM DWT Algorithm

The development of the PM DWT algorithm consists of the following three steps:

1) representing the time-domain and the sequency-domain samples in vector forms;
2) expressing the DWT relation in an equivalent form in which the input and output quantities are represented as vectors;
3) using the classical divide-and-conquer strategy in decomposing the vectorized form of the DWT computation.

The DWT of a data sequence $\{x(n), n = 0, 1, \cdots, N-1\}$ is defined[2] [2] as

$$X(k) = \sum_{n=0}^{N-1} x(n) \prod_{i=0}^{M-1} (-1)^{b_i(n) b_{M-1-i}(k)}, \quad k = 0, 1, \cdots, N-1 \tag{1}$$

where $b_i(n)$ is the ith bit in the binary representation of n, the least significant bit having a subscript of zero. Equation (1) can be rewritten as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left[\sum_{s=0}^{1} x\left(n + s\frac{N}{2}\right) (-1)^{b_{M-1}(n+s\frac{N}{2}) b_0(k)} \right] \times \prod_{i=0}^{M-2} (-1)^{b_i(n) b_{M-1-i}(k)}. \tag{2}$$

Note that, for each value of n, the inner summation is a two-point DWT which, when evaluated, will yield only two distinct values for all values of k. Let $\mathbf{a}(n)$ denote the nth input vector[3] as defined below.

$$\mathbf{a}(n) = \{a_0(n), a_1(n)\} = \left\{ \sum_{s=0}^{1} x\left(n + s\frac{N}{2}\right)(-1)^{sj}, j = 0, 1 \right\}, \quad n = 0, 1, \cdots, \tfrac{N}{2} - 1. \tag{3}$$

With the vectorized input quantities, (2) can be rewritten as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} a_q(n) \prod_{i=0}^{M-2} (-1)^{b_i(n) b_{M-1-i}(k)}. \tag{4}$$

Since the values of the two-point transform $a_q(n)$ are periodic with a period of two, for k equal to or greater than two in (4), the values of $a_q(n)$ repeat. Therefore

$$q = k \bmod 2. \tag{5}$$

Replacing the argument k by $k + p\frac{N}{2}$ in (4) and (5), yields

$$X\left(k + p\frac{N}{2}\right) = \sum_{n=0}^{\frac{N}{2}-1} a_q(n) \prod_{i=0}^{M-2} (-1)^{b_i(n) b_{M-1-i}(k + p\frac{N}{2})}, \quad k = 0, 1, \cdots, \tfrac{N}{2} - 1,\ p = 0, 1 \tag{6}$$

$$q = \left(k + p\frac{N}{2}\right) \bmod 2. \tag{7}$$

Let us now vectorize the output transform values $X(k)$ as

$$\mathbf{A}(k) = \{A_0(k), A_1(k)\} = \left\{ X\left(k + p\frac{N}{2}\right), p = 0, 1 \right\}, \quad k = 0, 1, \cdots, \tfrac{N}{2} - 1. \tag{8}$$

With the vectorization of the input quantities $x(n)$ as carried out in (3) and that of the output quantities $X(k)$ as carried out in (8) and from (6), the DWT as defined by (1) can now be equivalently written as

$$A_p(k) = \sum_{n=0}^{\frac{N}{2}-1} (-1)^{n p} a_q(n) \prod_{i=0}^{M-3} (-1)^{b_{i+1}(n) b_{M-2-i}(k)}, \quad k = 0, 1, \cdots, \tfrac{N}{2} - 1, \tag{9}$$

where $q = (k + p\frac{N}{2}) \bmod 2$ and $p = 0, 1$. Thus, the incorporation of the vectorization of the input and output quantities changes the form of the basic definition of the DWT from (1) to (9), which requires the computation of 2-point transform of each of the $\frac{N}{2}$ ordered sets of two values of the given N-point input sequence that are $\frac{N}{2}$ samples apart.

The inverse DWT is given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \prod_{i=0}^{M-1} (-1)^{b_i(n) b_{M-1-i}(k)}, \quad n = 0, 1, \cdots, N-1. \tag{10}$$

As the inverse transform definition is similar to the forward transform except for a constant divisor, the algorithm for computing the forward transform can be used directly for computing the inverse transform. Therefore, computation of the inverse DWT is not described.

We now describe the procedure for the decomposition of (9) to derive the PM DWT algorithm. The DWT algorithm based on a divide-and-conquer approach can be considered as a process of breaking down a one-dimensional (1-D) sequence recursively into smaller two-dimensional (2-D) sequences until the evaluation becomes trivial. The problem of computing a 1-D DWT using vectors is transformed into a problem of computing a 2-D DWT

[1] The computation of the two-point transform of data values a and b is just a *plus* b and a *minus* b This is the fundamental operation in the algorithm implementation. Hence, the derived algorithms are designated as plus-minus (PM) algorithms.

[2] In this paper, it is assumed that N, the number of samples in the data set, is equal to an integral power of two i.e., $M = \log_2 N$ is a positive integer.

[3] Vectors will be denoted by boldface letters.

by substituting

$$n = 2n_1 + n_2, \quad n_1 = 0, 1, \ldots, \frac{N}{4} - 1, \quad n_2 = 0, 1$$
$$k = k_1 + \frac{N}{4}k_2, \quad k_1 = 0, 1, \cdots, \frac{N}{4} - 1 \quad k_2 = 0, 1,$$

in (9), yielding

$$A_p\left(k_1 + \frac{N}{4}k_2\right) = \sum_{n_1=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} (-1)^{pn_2} a_q(2n_1 + n_2) \times \prod_{i=0}^{M-3} (-1)^{b_{i+1}(2n_1+n_2)b_{M-2-i}(k_1+\frac{N}{4}k_2)}, \quad k_1 = 0, 1, \ldots, \tfrac{N}{4} - 1,\ k_2 = 0, 1 \quad (11)$$

where $q = (k_1 + \frac{N}{4}k_2) \bmod 2$. With this mapping, the input vectors are placed in 2-D space with

$$a_q(n_1, n_2), \quad n_1 = 0, 1, \cdots, \frac{N}{4} - 1, \quad n_2 = 0, 1.$$

The input vectors $a_q(n)$ are placed row-by-row to form the 2-D array (The even-indexed and odd-indexed input vectors are placed, respectively, in the first and second columns). Equation (11) can be rewritten as

$$A_p\left(k_1 + \frac{N}{4}k_2\right) = \sum_{n_2=0}^{1} (-1)^{pn_2} \left\{ \sum_{n_1=0}^{\frac{N}{4}-1} (-1)^{n_1 k_2} a_q(2n_1 + n_2) \times \prod_{i=0}^{M-4} (-1)^{b_{i+1}(n_1)b_{M-3-i}(k_1)} \right\}. \quad (12)$$

The inner summation in (12) represents 2 independent column DWT's (as n_2 varies from 0 to 1) each of $\frac{N}{4}$-vector length. Let the column DWT's be denoted as

$$A_{k_2}^{(\mathrm{col})}(k_1, n_2) = \sum_{n_1=0}^{\frac{N}{4}-1} (-1)^{n_1 k_2} a_q(2n_1 + n_2) \times \prod_{i=0}^{M-4} (-1)^{b_{i+1}(n_1)b_{M-3-i}(k_1)} \quad (13)$$

where $q = (k_1 + k_2\frac{N}{4}) \bmod 2$. Now, (12) can be rewritten as

$$A_p\left(k_1 + \frac{N}{4}k_2\right) = \sum_{n_2=0}^{1} (-1)^{pn_2} \left\{ A_{k_2}^{(\mathrm{col})}(k_1, n_2) \right\}. \quad (14)$$

As k_1 varies from 0 to $\frac{N}{4} - 1$, (14) represents $\frac{N}{4}$ independent row DWT's each of two-vector length.

The transform values of the elements of the $\frac{N}{4}$ even-indexed input vectors, $\mathbf{a}(n)(n = 0, 2, \cdots, \frac{N}{2} - 2)$ in (13), is given by $A_0^{(\mathrm{col})}(k_1, 0)$ and $A_1^{(\mathrm{col})}(k_1, 0)$, and the transform values of the elements of the $\frac{N}{4}$ odd-indexed input vectors, $\mathbf{a}(n)(n = 1, 3, \cdots, \frac{N}{2} - 1)$, is given by $A_0^{(\mathrm{col})}(k_1, 1)$ and $A_1^{(\mathrm{col})}(k_1, 1)$. By putting $k_2 = 0$ in (14) and evaluating the summation, we get

$$A_p(k_1) = A_0^{(\mathrm{col})}(k_1, 0) + (-1)^p A_0^{(\mathrm{col})}(k_1, 1) \quad (15)$$

and by substituting $k_2 = 1$ in (14) and evaluating the summation, we get

$$A_p\left(k_1 + \frac{N}{4}\right) = A_1^{(\mathrm{col})}(k_1, 0) + (-1)^p A_1^{(\mathrm{col})}(k_1, 1) \quad (16)$$

where $p = 0$ or 1, and $k_1 = 0, 1, \cdots, \frac{N}{4} - 1$. From (15) and (16), it is seen that in order to compute the transform values $A_p(k)$, the precomputation of $A_{k_2}^{(\mathrm{col})}(k_1, 0)$ and $A_{k_2}^{(\mathrm{col})}(k_1, 1)$ is required. Thus, the problem of computing an $\frac{N}{2}$-vector DWT has been decomposed into a problem of computing two $\frac{N}{4}$-vector DWT's. This process of decomposition can be continued until the problem is reduced to two-vector DWT's.

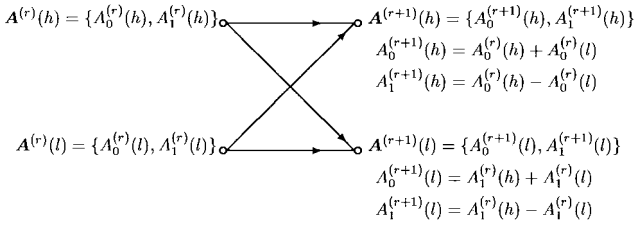

Fig. 1. Butterfly of the PM DWT algorithm.

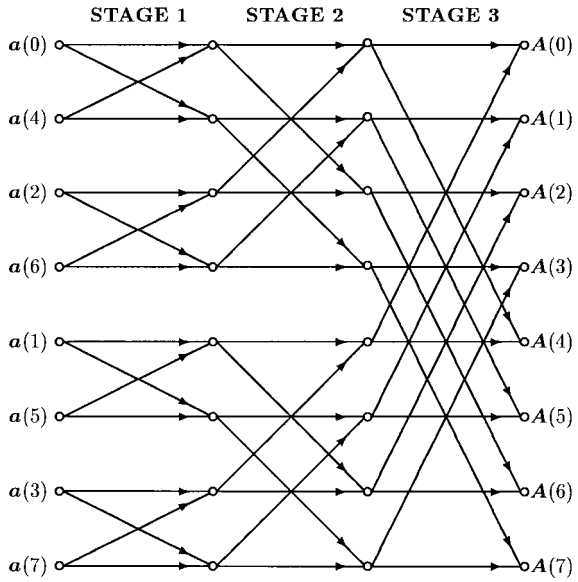

Fig. 2. Signal flow graph of the PM DWT algorithm with $N = 16$.

Even though a 1-D sequence of vectors has been mapped into a 2-D sequence for deriving the algorithm, the data vectors are manipulated in a 1-D array. Assuming that the transform values of the odd-indexed input vectors follow the transform values of the even-indexed input vectors in a 1-D array of vectors, the relation governing the basic computation at the rth stage can be deduced from (15) and (16) as

$$A_0^{(r+1)}(h) = A_0^{(r)}(h) + A_0^{(r)}(l) \quad (17a)$$
$$A_1^{(r+1)}(h) = A_0^{(r)}(h) - A_0^{(r)}(l) \quad (17b)$$
$$A_0^{(r+1)}(l) = A_1^{(r)}(h) + A_1^{(r)}(l) \quad (17c)$$
$$A_1^{(r+1)}(l) = A_1^{(r)}(h) - A_1^{(r)}(l). \quad (17d)$$

These equations characterize the PM DWT butterfly shown in Fig. 1. This butterfly computes two two-point transforms. For a specific value of N, repeated use of the butterflies yields the flow graph of the algorithm. The number of butterflies in each of the m stages is $\frac{N}{4}$, where $m = \log_2 \frac{N}{2}$. For example, with $N = 16$ there are three stages specified as, $r = 1, 2$ and 3, and four butterflies make up a stage. Indices h and l, for each group of butterflies are given, respectively, by $h = i \bmod 2^{r-1} (i = 0, 1, \cdots, 3)$ and $l = h + 2^{r-1}$. The flow graph of the PM DWT algorithm with $N = 16$ is shown in Fig. 2.

The 2-D DWT of an $N_1 \times N_2$ image[4] $\{x(n_1, n_2), n_1 = 0, 1, \cdots, N_1 - 1, n_2 = 0, 1, \cdots, N_2 - 1\}$ is defined as

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \prod_{i=0}^{M_1-1} (-1)^{b_i(n_1) b_{M_1-1-i}(k_1)} \times \prod_{i=0}^{M_2-1} (-1)^{b_i(n_2) b_{M_2-1-i}(k_2)}, \quad k_1 = 0, 1, \cdots, N_1 - 1,\ k_2 = 0, 1, \cdots, N_2 - 1. \tag{18}$$

It can be shown that the computation is the same as that of (1). As such, the flow graph remains the same except that $N = N_1 \times N_2$. It should be noted that if data is read row-by-row from the input file, the output would be written column-by-column or vice versa.

III. Derivation of the Sequency-Ordered PM DHAT Algorithm

The sequency-ordered DHAT of a data sequence $\{x(n), n = 0, 1, \cdots, N - 1\}$ is defined as [2]

$$X(k) = \sum_{n=0}^{N-1} x(n)(-1)^{\sum_{i=0}^{M-1} b_i(n) p_i(k)}, \quad k = 0, 1, \cdots, N - 1 \tag{19}$$

where $b_i(n)$ is the ith bit in the binary representation of n, the least significant bit having a subscript of zero and

$$\begin{aligned} p_0(k) &= b_{M-1}(k) \\ p_1(k) &= b_{M-1}(k) + b_{M-2}(k) \\ p_2(k) &= b_{M-2}(k) + b_{M-3}(k) \\ &\vdots \\ p_{M-1}(k) &= b_1(k) + b_0(k). \end{aligned}$$

Equation (19) can be rewritten as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} (-1)^{\sum_{i=0}^{M-2} b_{i+1}(2n) p_{i+1}(k)} \times \sum_{s=0}^{1} x(2n + s)(-1)^{b_0(2n+s) b_{M-1}(k)}. \tag{20}$$

Note that, for each value of n, the inner summation is a two-point DHAT which, when evaluated, will yield only two distinct values for all values of k. Let $\mathbf{a}'(n)$ denote the nth input vector defined as

$$\begin{aligned} \mathbf{a}'(n) &= \{a'_0(n), a'_1(n)\} \\ &= \left\{ \sum_{s=0}^{1} x(2n + s)(-1)^{sj}, j = 0, 1 \right\}, \\ & \qquad n = 0, 1, \cdots, \tfrac{N}{2} - 1. \end{aligned} \tag{21}$$

With the vectorized input quantities, (20) can be rewritten as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} a'_q(n)(-1)^{\sum_{i=0}^{M-2} b_{i+1}(2n) p_{i+1}(k)}. \tag{22}$$

Since $b_{M-1}(k)$ is zero for $k < \frac{N}{2}$ and 1 for $k \geq \frac{N}{2}$, the values of $a'_q(n)$ repeat. Therefore

$$q = \left\lfloor \frac{k}{N/2} \right\rfloor. \tag{23}$$

Let $a_q(n) = a'_q(\mathrm{br}(n))$, $n = 0, 1, \cdots, \frac{N}{2} - 1$, where $\mathrm{br}(n)$ denotes the bit-reversal operation on the index n represented with $\log_2 \frac{N}{2}$ bits. Then, (22) can be rewritten as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} a_q(n)(-1)^{\sum_{i=0}^{M-2} \mathrm{br}_i(2n) p_{i+1}(k)}. \tag{24}$$

Replacing the argument k by $2k + p$, (24) and (23) become

$$X(2k + p) = \sum_{n=0}^{\frac{N}{2}-1} a_q(n)(-1)^{\sum_{i=0}^{M-2} \mathrm{br}_i(2n) p_{i+1}(2k+p)}, \quad k = 0, 1, \cdots, \frac{N}{2} - 1,\ p = 0, 1 \tag{25}$$

$$q = \left\lfloor \frac{2k + p}{N/2} \right\rfloor. \tag{26}$$

Let us vectorize the output transform values $X(k)$ as

$$\mathbf{A}(k) = \{A_0(k), A_1(k)\} = \{X(2k + p), p = 0, 1\}, \quad k = 0, 1, \cdots, \tfrac{N}{2} - 1. \tag{27}$$

With the vectorization of the input quantities $x(n)$ as carried out in (21) and that of the output quantities $X(k)$ as carried out in (27) and from (25), the DHAT as defined by (19) can now be equivalently written as

$$A_p(k) = \sum_{n=0}^{\frac{N}{2}-1} (-1)^{np} a_q(n)(-1)^{\sum_{i=0}^{M-2} \mathrm{br}_i(2n) p_{i+1}(2k)}, \quad k = 0, 1, \cdots, \tfrac{N}{2} - 1 \tag{28}$$

where $q = \lfloor \frac{2k+p}{\frac{N}{2}} \rfloor$ and $p = 0, 1$. Thus, the incorporation of the vectorization of the input and output quantities changes the form of the basic definition of the DHAT from (19) to (28) which requires the computation of two-point transform of each of the $\frac{N}{2}$ ordered sets of two values of the given N-point input sequence that are one sample apart.

The inverse DHAT is given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)(-1)^{\sum_{i=0}^{M-1} b_i(n) p_i(k)}, \quad n = 0, 1, \cdots, N - 1. \tag{29}$$

As the inverse transform definition is similar to the forward transform except for a constant divisor, the algorithm for computing forward transform can be used directly for computing inverse transform. Therefore, computation of the inverse DHAT is not described.

We now describe the procedure for the decomposition of (28) to derive the PM DHAT algorithm. The problem of computing 1-D sequency-ordered DHAT using vectors is transformed into a problem of computing 2-D DHAT by substituting

$$\begin{aligned} n &= n_1 + \frac{N}{4} n_2, \quad n_1 = 0, 1, \ldots, \frac{N}{4} - 1,\ n_2 = 0, 1, \\ k &= k_1 + \frac{N}{4} k_2, \quad k_1 = 0, 1, \cdots, \frac{N}{4} - 1,\ k_2 = 0, 1 \end{aligned}$$

in (28), yielding

$$\begin{aligned} A_p\left(k_1 + \frac{N}{4} k_2\right) &= \sum_{n_1=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} (-1)^{(n_1 + \frac{N}{4} n_2) p} a_q\left(n_1 + \frac{N}{4} n_2\right) \\ &\quad \times (-1)^{\sum_{i=0}^{M-2} \mathrm{br}_i(2(n_1 + \frac{N}{4} n_2)) p_{i+1}(2(k_1 + \frac{N}{4} k_2))}, \\ &\qquad k_1 = 0, 1, \ldots, \tfrac{N}{4} - 1,\ k_2 = 0, 1 \end{aligned} \tag{30}$$

[4] In this work, it is assumed that N_1 and N_2 are integral powers of two.

where $q = \lfloor \frac{2(k_1+\frac{N}{4}k_2)+p}{\frac{N}{2}} \rfloor$. With this mapping, the first $\frac{N}{4}$ input vectors are placed in the first column and the second $\frac{N}{4}$ input vectors are placed in the second column of a 2-D array (column-by-column placement of the input vectors $a_q(n)$). Equation (30) can be rewritten as

$$A_p\left(k_1+\frac{N}{4}k_2\right) = \sum_{n_1=0}^{\frac{N}{4}-1}(-1)^{pn_1}\left\{\sum_{n_2=0}^{1}(-1)^{n_2(\lfloor\frac{2k_1}{\frac{N}{4}}\rfloor+\frac{N}{4}p)}a_q\left(n_1+\frac{N}{4}n_2\right)\times(-1)^{\mathrm{br}_0(2n_2)p_1(2k_2)}\right\}(-1)^{\sum_{i=0}^{M-3}\mathrm{br}_i(2n_1)p_{i+1}(2k_1)}. \quad (31)$$

As n_1 varies from zero to $\frac{N}{4}-1$, the inner summation in (31) represents $\frac{N}{4}$ row DHAT's each of two-vector length. Let the row transform values be denoted by

$$a_{q'}^{(\mathrm{row})}(n_1,k_2) = \sum_{n_2=0}^{1}(-1)^{n_2q'}a_q\left(n_1+\frac{N}{4}n_2\right)(-1)^{\mathrm{br}_0(2n_2)p_1(2k_2)},$$
$$q=k_2,\ q'=\left\lfloor\frac{2k_1}{N/4}\right\rfloor+\frac{N}{4}p=\left\lfloor\frac{2k_1+p}{N/4}\right\rfloor=0,1. \quad (32)$$

Using (32), (31) can be rewritten as

$$A_p\left(k_1+\frac{N}{4}k_2\right) = \sum_{n_1=0}^{\frac{N}{4}-1}(-1)^{pn_1}a_{q'}^{(\mathrm{row})}(n_1,k_2)(-1)^{\sum_{i=1}^{M-3}\mathrm{br}_i(2n_1)p_i(2k_1)}. \quad (33)$$

For $k_2=0$ and 1, (32) represents two independent column DHAT's each of $\frac{N}{4}$-vector length. Putting $k_2=0$ in (32) and evaluating the summation yields

$$a_{q'}^{(\mathrm{row})}(n_1,0) = a_0(n_1)+(-1)^{q'}a_0\left(n_1+\frac{N}{4}\right), \quad n_1=0,1,\cdots,\tfrac{N}{4}-1,\ q'=0,1. \quad (34)$$

Putting $k_2=1$ in (32) and evaluating the summation, we get

$$a_{q'}^{(\mathrm{row})}(n_1,1) = a_1(n_1)-(-1)^{q'}a_1\left(n_1+\frac{N}{4}\right), \quad n_1=0,1,\cdots,\tfrac{N}{4}-1,\ q'=0,1. \quad (35)$$

From (34) and (35), it is seen that the input values $a_q(n_1)$ and $a_q(n_1+\frac{N}{4})$ are combined to reduce the problem size. The problem of computing an $\frac{N}{2}$-vector DHAT, therefore, has been decomposed into a problem of computing two $\frac{N}{4}$-vector DHAT's. This process of decomposition can be continued until the problem is reduced to two-vector DHAT's.

Assuming that the input vectors for the first-half of the transform values follow the input vectors for the second half of the transform values in a 1-D array of vectors, the relation governing the basic computation at the rth stage can be deduced from (34) and (35) as

$$a_0^{(r+1)}(h) = a_0^{(r)}(h)+a_0^{(r)}(l) \quad (36a)$$
$$a_1^{(r+1)}(h) = a_0^{(r)}(h)-a_0^{(r)}(l) \quad (36b)$$
$$a_0^{(r+1)}(l) = a_1^{(r)}(h)-a_1^{(r)}(l) \quad (36c)$$
$$a_1^{(r+1)}(l) = a_1^{(r)}(h)+a_1^{(r)}(l). \quad (36d)$$

These equations characterize the sequency-ordered PM DHAT butterfly shown in Fig. 3. This butterfly computes two two-point transforms. For a specific value of N, repeated use of the butterfly yields the flow graph of the algorithm. The number of butterflies in each of the m stages is $\frac{N}{4}$, where $m=\log_2\frac{N}{2}$. For example, with $N=16$ there are three stages specified as, $r=1,2$ and 3, and four butterflies make up a stage. Indices h and l, for each group of butterflies are given, respectively, by $h = i \bmod 2^{M-r-1}$ $(i=0,1,\cdots,3)$ and $l=h+2^{M-r-1}$. The flow graph of the sequency-ordered PM DHAT algorithm with $N=16$ is shown in Fig. 4. Algorithm for the computation of unordered DHAT can be easily derived. In that algorithm, the input vector formation and the butterfly are similar to the DWT algorithm. The flowgraph and the output vector formation is similar to the sequency-ordered DHAT algorithm. The input and output vectors appear in natural order.

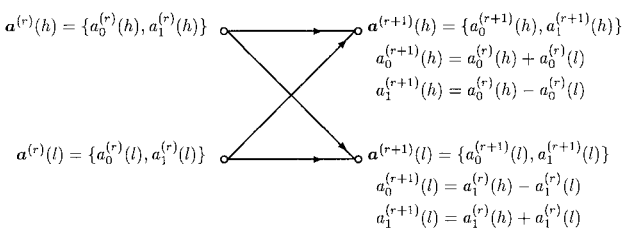

Fig. 3. Butterfly of the sequency-ordered PM DHAT algorithm.

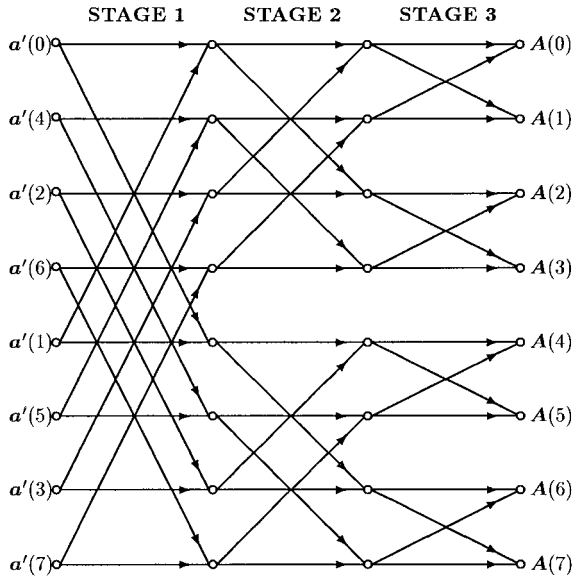

Fig. 4. Signal flow graph of the sequency-ordered PM DHAT algorithm with $N=16$.

The 2-D DHAT of an $N_1 \times N_2$ image $\{x(n_1,n_2),\ n_1=0,1,\cdots,N_1-1,\ n_2=0,1,\cdots,N_2-1\}$ is defined as (37), shown at the bottom of the next page. It can be shown that the computation of (37) is almost the same as that of (19). As such the flowgraph remains the same except that $N=N_1\times N_2$ and with a minor change as explained now. In the vector formation stage, the sum is stored as the first element of the vector and the difference as the second element. At the bottom node of the butterfly, the difference is stored in the first element of the vector. In the 2-D case, two sets of 1-D transforms have to be carried out. For the row (column) transform, the vector formation is carried out in the required manner. However, for the column (row) transforms, the vector formation stage occurs at a point where the row (column) transforms are completed. In this intermediate vector formation stage, the butterfly of Fig. 3 has to be used repeatedly with the difference that the sum and difference are stored at the bottom node in the same order as that for the top node. It should be noted that if data is read row-by-row from the input file, the output would be written column-by-column or vice versa.

TABLE I
FIGURES OF COMPUTATIONAL COMPLEXITY OF THE PM AND THE CONVENTIONAL (CON) ALGORITHMS [6], [7]

	Number of						
	Additions	Bit-Reversals		Array-Index Updatings		Independent Data Swappings	
N	PM & CON	PM	CON	PM	CON	PM	CON
16	64	8	16	24	64	0	6
32	160	16	32	64	160	0	12
64	384	32	64	160	384	0	28
128	896	64	128	384	896	0	56
256	2048	128	256	896	2048	0	120
512	4608	256	512	2048	4608	0	240
1024	10240	512	1024	4608	10240	0	496
2048	22528	1024	2048	10240	22528	0	992
4096	49152	2048	4096	22528	49152	0	2016

IV. Computational Complexity of the PM DWT and PM DHAT Algorithms

It can be easily deduced that the PM algorithms, as in the case of the other existing algorithms, require NM real additions, where $M = \log_2 N$. In terms of the structural complexity, the PM algorithms provide the following advantages compared with the existing algorithms.

- Irrespective of the scheme used for the bit-reversal process, the number of bit reversals in the PM DWT and the sequency-ordered PM DHAT algorithms is one-half of that required in the existing algorithms, since this process is carried out on the indices of an $\frac{N}{2}$-element array of vectors.
- The number of array-index updatings required in both the PM algorithms is $\frac{N}{2}\log_2\frac{N}{2}$ as compared with $N\log_2 N$ in the existing algorithms.
- In the PM DWT algorithm, independent data swapping operations can be easily eliminated. The first half and the second half of the input data values are read, respectively, into the storage locations assigned for the first and the second element of the input vectors. The elements of a pair of vectors are swapped at the time of vector formation. For the Shanks algorithm [6], $(N-\sqrt{N})/2$ swaps when M is even and $(N/2)-\sqrt{(N/2)}$ swaps when M is odd, are required. Similar savings in sequency-ordered PM DHAT algorithm is also obtained.
- The characteristic of the PM algorithm requiring only one type of butterfly to implement the sequency-ordered DHAT algorithm improves regularity and provides efficiency and ease of implementation. The Manz algorithm [7] requires two types of butterflies at the expense of increased implementation complexity and execution time.
- It is possible to design a three-address primitive instruction to compute a two-point transform for both the PM algorithms, since the results of that operation can be stored in consecutive memory locations even for an in-place algorithm. This instruction would reduce the number of additions by one-half. The existing algorithms would require a four-address instruction for the same reduction.

The figures of computational complexity for the PM and the conventional algorithms [2]–[7], for various values of N, are shown in Table I. The software implementations of the proposed PM algorithms are available on request.

TABLE II
COMPARISON OF EXECUTION TIMES (ON A SUN SPARC 2 WORKSTATION) OF THE IMPLEMENTATION OF THE PM DWT ALGORITHM WITH THAT OF THE SHANKS DWT ALGORITHM [6] FOR TRANSFORM LENGTHS THAT ARE EVEN POWERS OF TWO

N	Time in Milliseconds		
	PM Algorithm	Shanks Algorithm	Savings
256	0.59	0.65	9.2%
1024	2.91	3.16	7.9%
4096	13.67	15.05	9.2%

TABLE III
COMPARISON OF EXECUTION TIMES (ON A SUN SPARC 2 WORKSTATION) OF THE IMPLEMENTATION OF THE PM DWT ALGORITHM WITH THAT OF THE SHANKS DWT ALGORITHM [6] FOR TRANSFORM LENGTHS THAT ARE ODD POWERS OF TWO

N	Time in Milliseconds		
	PM Algorithm	Shanks Algorithm	Savings
512	1.23	1.45	15.2%
2048	5.97	7.00	14.7%
8192	28.59	33.96	15.8%

Both the proposed algorithms were tested and they run faster than the existing algorithms. In order to compare the efficiency of the PM DWT algorithm with that of the Shanks DWT algorithm [6], efficient two-stage-at-a-time implementations of both the algorithms were carried out. The execution times of the two algorithms were recorded over 100 runs for several sets of input data. The software implementation of the PM DWT algorithm was found to be faster, as shown in Table II, for transform lengths that are even powers of two. For transform lengths that are odd powers of two, the proposed algorithm requires $2N$ less data transfer operations. Therefore, for these transform lengths, the execution time savings is higher as shown in Table III. Additional savings would be possible with the availability of a three-address instruction for computing the two-point transform.

V. Conclusion

In this work, it has been shown that algorithms for computing the discrete Walsh and Hadamard transforms provide a much reduced structural complexity if the classical divide-and-conquer strategy is applied to a form of the relation defining the DWT or DHAT computation in which the input and output quantities are treated as vectors with two elements. The formation of vectors from the raw input data allows the computation of two two-point transforms by a single butterfly. The PM algorithm essentially consists of two parts. In the first part, $\frac{N}{2}$ two-element vectors are formed by computing two-point transforms of ordered sets of two values of the N input samples that are certain number of samples apart. In the second part, each butterfly of an interconnected network of butterflies operating on two two-element input vectors produces two two-element output vectors. The PM algorithms provide the advantages of reduced number of bit-reversals and array-index updating operations and do not require independent data swapping operations. The reduction in these operations readily translates into a reduction in the execution time.

References

[1] D. Sundararajan, M. O. Ahmad, and M. N. S. Swamy, "Computational structures for fast Fourier transform analyzers," U.S. Patent 5 371 696, Dec. 6, 1994.

$$X(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}x(n_1,n_2)(-1)^{\sum_{i=0}^{M_1-1}b_i(n_1)p_i(k_1)+\sum_{i=0}^{M_2-1}b_i(n_2)p_i(k_2)},\ k_1=0,1,\cdots,N_1-1,\ k_2=0,1,\cdots,N_2-1. \quad (37)$$

[2] R. C. Gonzalez and P. Wintz, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1987.
[3] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Image Processing*. Berlin, Germany: Springer-Verlag, 1975.
[4] K. G. Beauchamp, *Applications of Walsh and Related Functions*. New York: Academic, 1984.
[5] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*. New York: Academic, 1985.
[6] J. L. Shanks, "Computation of the fast Walsh–Fourier transform," *IEEE Trans. Comput.*, vol. C-18, pp. 457–459, 1969.
[7] J. W. Manz, "A sequency-ordered fast Walsh transform," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 204–205, 1972.
[8] S. Boussakta and A. G. J. Holt, "Fast algorithm for calculation of both the Walsh–Hadamard and Fourier transforms (FWFTS)," *Electron. Lett.*, vol. 25, pp. 1352–1353, Sept. 28, 1989.
[9] S. C. Noble, "A comparison of hardware implementation of the Hadamard transform for real time image coding," in *Proc. Soc. Photo-Optical Instrumentation Engineers*, 1975, pp. 207–211.
[10] P. C. Ching and C. C. Goodyear, "Walsh-transform coding of the speech residual in RELP coders," *Proc. Inst. Elect. Eng. G*, vol. 131, no. 1, pp. 29–34, Feb. 1984. .
[11] Y. Tadokoro and T. Higuchi, "Conversion factors from Walsh coefficients to Fourier coefficients," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 231–232, Feb. 1983.

Estimation of Depth Fields Suitable for Video Compression Based on 3-D Structure and Motion of Objects

A. Aydın Alatan and Levent Onural

***Abstract*—Intensity prediction along motion trajectories removes temporal redundancy considerably in video compression algorithms. In three-dimensional (3-D) object-based video coding, both 3-D motion and depth values are required for temporal prediction. The required 3-D motion parameters for each object are found by the correspondence-based E-matrix method. The estimation of the correspondences—two-dimensional (2-D) motion field—between the frames and segmentation of the scene into objects are achieved simultaneously by minimizing a Gibbs energy. The depth field is estimated by jointly minimizing a defined distortion and bit-rate criterion using the 3-D motion parameters. The resulting depth field is efficient in the rate-distortion sense. Bit-rate values corresponding to the lossless encoding of the resultant depth fields are obtained using predictive coding; prediction errors are encoded by a Lempel–Ziv algorithm. The results are satisfactory for real-life video scenes.**

***Index Terms*—Dense depth estimation, depth encoding, motion analysis, object-based video coding, rate-distortion theory, 3-D motion, 3-D structure.**

I. Introduction

In very low bit-rate coding applications, the current trend is shifting from motion compensated discrete cosine transform (DCT) type algorithms, like MPEG-X, H.26X, to object-based methods [1]. In most of the current object-based algorithms, two-dimensional (2-D) motion models are used, although such motion models have limited performance due to lack of representation of three-dimensional (3-D) world dynamics. Currently, 3-D motion models are rarely used in video compression systems [1]–[5], and these approaches are usually far from representing general solutions. However, in such algorithms compression is still possible after removal of the temporal redundancy by predicting intensities along motion trajectories. Both 3-D motion and depth information are necessary to achieve this goal.

A 3-D motion model is the "simplest" way to describe any physical motion, especially when the moving object is rigid, because any rigid 3-D motion is represented by only six degrees of freedom, i.e., six parameters. Estimation of the 3-D motion parameters for a rigid body observed through two consecutive 2-D frames has well-developed solutions [6], [7] and, hence, this estimation problem can be easily overcome. Although depth estimation using these methods can be achieved, the obtained depth fields are usually sparse, whereas for coding purposes it is preferable to have a dense depth field in order to predict the intensities by motion compensation at each pixel. Given two 2-D consecutive video frames, one or more 3-D structures may give perfect intensity match by 3-D motion compensation. A structure that results in perfect intensity match, if it exists, may not be suitable for efficient encoding. Furthermore, one can find structures that are easier to code by allowing some intensity mismatch during the motion compensation. Estimating a dense depth field (structure) suitable for very low bit-rate video compression is the primary issue in this paper.

None of the current video coding methods with 3-D motion models propose a method for estimating a depth field that is suitable for encoding. Some depth encoding algorithms exist for stereo video coding applications [8] in which the depth field is simply obtained by using the disparity information between stereo frames. In these methods the obtained depth map is either DPCM-coded after quantization or fitted onto a wireframe [8]. However, such methods do not take distortion and bit rate into account simultaneously while estimating the depth field.

It should be noted that if the number of bits to encode the depth field is reduced to reach a target rate, some distortion in the depth field, compared to the one which yields perfect intensity matches, may be inevitable. Rate-distortion theory [9] gives a relationship between the minimum number of bits to encode a distorted symbol sequence from a source and the distortion between the true and encoded versions of that sequence. Using similar ideas, a lossy version of the depth field can be found by jointly minimizing the required number of bits and a distortion measure. Such approaches are also used to estimate 2-D motion vectors between video frames [10].

The main focus of this paper is to formulate a novel method for estimating (and thus generating) a depth field that is convenient for encoding. In order to estimate the desired depth field, the frames should be segmented into a number of moving objects and the 3-D motion parameters of the objects should be found. Dense 2-D motion vectors are needed for both object segmentation and correspondence-based 3-D motion estimation. In order to carry out simulations, a simultaneous 2-D motion estimation and segmentation algorithm, and a 3-D motion estimation algorithm, are proposed in Sections II-A and II-B, respectively. Moreover, in order to give an idea about the actual bit requirements associated with the coding of the estimated depth fields, a lossless encoder is utilized in Section IV. Algorithms in Sections II and IV are not the main concern of the paper; they cannot be claimed to have the best performance. However, they do give satisfactory results.

Manuscript received January 14, 1996; revised March 4, 1997. This work was supported by TÜBİTAK of Turkey under the COST 211 Project. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Janusz Konrad.

The authors are with the Department of Electrical and Electronics Engineering, Bilkent University, TR-06533 Ankara, Turkey (e-mail: alatan@ee.bilkent.edu.tr; onural@ee.bilkent.edu.tr).

Publisher Item Identifier S 1057-7149(98)03997-9.

1057–7149/98$10.00 © 1998 IEEE