

A Truncation Method for Computing Slant Transforms with Applications to Image Processing

Maurence M. Anguh and Ralph R. Martin

Abstract—A Truncation method for computing the Slant Transform is presented. The Slant Transform Truncation (STT) algorithm uses the divide and conquer principle of hierarchical data structures to factorize coherent image data into sparse subregions. In one dimension with a data array of size $N = 2^n$, the Truncation method takes time between $O(N)$ and $O(N \log_2 N)$, degenerating to the performance of the Fast Slant Transform (FST) method in its worst case. In two dimensions, for a data array of size $N \times N$, the one-dimensional Truncation method is applied to each row, then to each column of the array, to compute the transform in time between $O(N^2)$ and $O(N^2 \log_2 N)$. Coherence is a fundamental characteristic of digital images and so the Truncation method is superior to the FST method when computing Slant Transforms of digital images. Experimental results are presented to justify this assertion.

I. INTRODUCTION

A PHENOMENON characteristic of digital images is the presence of (approximately) constant or uniformly changing gray level (coherence) over a considerable distance or area. The Slant Transform [4], [10], [11] is specifically defined for an efficient representation of such images. The basis vectors used are discrete sawtooth waveforms changing uniformly with distance, representing gradual increase of brightness. The discrete unitary Slant Transform of order N is defined using a unitary matrix S_N such that $S_N S_N^T = I_N$, where T denotes the transpose and I_N is the $N \times N$ identity matrix. The Slant Transform $S[u]$ of an array $s[x]$, where $u, x = 0, \dots, N-1$ is defined as

$$S^T[u] = S_N s^T[x]. \quad (1)$$

A brute force computation of $S[u]$ by multiplying $s[x]$ and S_N requires $O(N^2)$ operations. The Fast Slant Transform method [5], [10] for computing $S[u]$ is based on the factorization of S_N into a set of largely sparse matrices, each expressing a stage of the computation. This requires $O(N \log_2 N)$ elementary operations.

Our earlier work [7] established relationships between Walsh-Hadamard transforms and hierarchical coding, particularly noting that both methods use the same basis to represent information present in the data. Furthermore, computing the

Paper approved by A. N. Netravali, the Editor for Image Processing of the IEEE Communications Society. Manuscript received February 22, 1993; revised October 13, 1993. This work was supported by the Cameroon Government.

M. M. Anguh is with the Department of Computer Science, Universidade Federal do Maranhao, Sao Luis, Maranhao, Brazil.

R. R. Martin is with the Department of Computing Mathematics, University of Wales College of Cardiff, Cardiff CF2 4YN, U.K.

IEEE Log Number 9410446.

Walsh-Hadamard transform of an image directly in terms of its pixel values as done by the Fast Walsh Transform method is inefficient because coherence is not used to advantage. The Truncation method [1]–[3] for computing one- and two-dimensional Walsh-Hadamard transforms of images utilizes both the factorization of the Walsh unitary matrix and partitioning of the data using hierarchical data structures.

This paper presents a similar idea for efficient computation of Slant Transforms of coherent data such as that of digital images. Our method is faster than the FST method. We call our method the Slant Transform Truncation (STT) method. The STT method simultaneously factorizes S_N into sparse matrices and the data $s[x]$ into sparse subregions using binary tree segmentation and aggregation. In particular, we prove that if the corresponding binary tree representation of $s[x]$ has maximum depth d , then the only nonzero Slant Transform coefficients are $S[i2^{n-d}]$ and $S[(1+4\beta)2^{n-d-1}]$ for $i = 0, \dots, 2^d - 1$ and $\beta = 0, \dots, 2^{d-1} - 1$. This result readily generalizes to two dimensions as will be seen. The observation that in the presence of coherence in the image, some intermediate results and transform coefficients are known to be zero in advance accounts for the reason for calling this approach the Truncation method.

Analysis of time complexity in one and two dimensions shows that the STT method takes time between $O(N)$ and $O(N \log_2 N)$ in one dimension, and time between $O(N^2)$ and $O(N^2 \log_2 N)$ in two dimensions, in particular degenerating to the performance of the FST in the worst case. A set of experiments conducted on digital images using the FST and STT methods is presented whose results demonstrate overall actual time savings by the STT method.

II. HIERARCHICAL STRUCTURES

Hierarchical data structures are coherence exploiting tools. The basic idea is to divide an array of data until uniform regions are obtained. The process commences with the entire array set to be the region of interest. If the region of interest is not uniform, it is subdivided and the process is repeated in turn on each of its subregions. The process terminates when an atomic region (pixel) or a uniform region is encountered. The manner in which the region of interest is subdivided at each stage determines the resulting structure. When an array of $N = 2^n$ data values is recursively subdivided into left and right halves, the resulting structure is a binary tree. Binary tree construction requires $O(N)$ logical operations. Other hierarchical structures include the quadtree and bintree used to represent two-dimensional arrays. Many details of

hierarchical data structures and their uses can be found in the pair of books by Samet [8], [9].

III. SLANT TRANSFORM

An image is characterized by often having (approximately) constant or slowly varying gray level over a considerable area. A major aim of an image transform is to compact the image energy into as few of the transform components as possible. With this background, the Slant Transform is defined to have the following properties: 1) An orthonormal set of basis vectors, 2) one constant basis vector, 3) one slant basis vector, 4) the sequency property, 5) variable size transformation, 6) a fast computational algorithm, and 7) high energy compaction. The Slant Transform definition and its properties are given in [10], [11].

A. The Slant Transform Matrix

The Slant Transform matrix is orthogonal with a constant function for the first row. The elements in other rows are defined by linear functions of the column index. The matrix is formed by an iterative construction of products of sparse matrices with a factorization which leads to a fast construction algorithm. The Slant Transform matrix of order two is defined by

$$S_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2)$$

while the Slant Transform matrix of order four is defined by

$$S_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix} \quad (3)$$

The Slant Transform matrix S_N of order N is defined by (4) where $I_{(N/2)-2}$ is the identity matrix of size $(N/2) - 2$ and $S_{N/2}$ is the Slant matrix of order $N/2$ shown in (4) at the bottom of the page. The constants a_N and b_N can be computed either by the recursive relations $a_2 = 1$, $b_2 = [1 + 4(a_{N/2})^2]^{-1/2}$, $a_N = 2b_N a_{N/2}$, or by $a_{2N} = \left[\frac{3N^2}{4(N^2-1)} \right]^{1/2}$, $b_{2N} = \left[\frac{N^2-1}{4(N^2-1)} \right]^{1/2}$.

B. The Fast Slant Transform (FST) Algorithm

The Fast Slant Transform algorithm is based on the Slant matrix factorization. The structural decomposition of the Slant matrix of order 8 in natural (normal) order is as shown in (5) at the bottom of the page. In general, the Slant matrix S_{2^n} of order 2^n in natural order is computed from the Slant matrix $S_{2^{n-1}}$ of order 2^{n-1} in natural order by first computing

$$S_2 \otimes S_{2^{n-1}} \quad (6)$$

$$S_N = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ a_N & b_N & 0 & 0 & -a_N & b_N & 0 & 0 \\ - & - & - & - & - & - & - & - \\ & 0 & & I_{(N/2)-2} & & 0 & & I_{(N/2)-2} \\ - & - & - & - & - & - & - & - \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ -b_N & a_N & 0 & 0 & b_N & a_N & 0 & 0 \\ - & - & - & - & - & - & - & - \\ & 0 & & I_{(N/2)-2} & & 0 & & -I_{(N/2)-2} \end{bmatrix} \begin{bmatrix} S_{N/2} & 0 \\ - & - \\ 0 & S_{N/2} \end{bmatrix} \quad (4)$$

$$S_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -3 & 3 & -1 & 1 & -3 & 3 & -1 \times \frac{1}{\sqrt{5}} \\ 7 & -1 & -9 & -17 & 17 & 9 & 1 & -7 \times \frac{1}{\sqrt{5 \times 21}} \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 7 & 5 & 3 & 1 & -1 & -3 & -5 & -7 \times \frac{1}{\sqrt{21}} \\ 1 & -3 & 3 & -1 & -1 & 3 & -3 & 1 \times \frac{1}{\sqrt{5}} \\ 3 & 1 & -1 & -3 & -3 & -1 & 1 & 3 \times \frac{1}{\sqrt{5}} \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad (5)$$

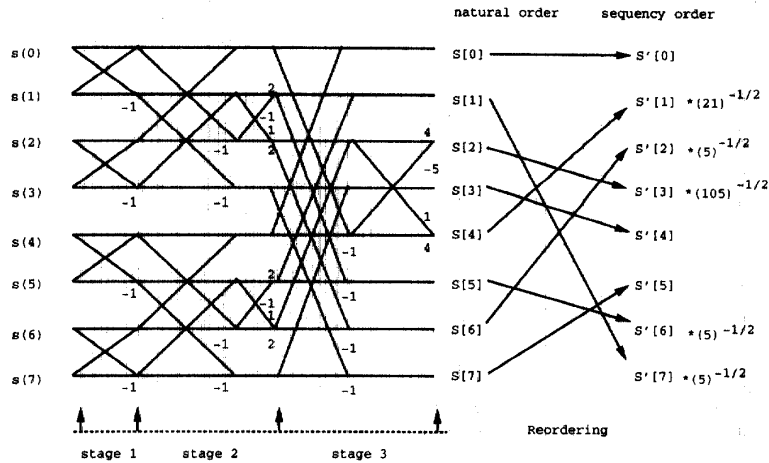


Fig. 1. The FST framework.

(\otimes denotes the Kronecker product) followed by a rotation of rows 2^{n-2} and 2^{n-1} by

$$\begin{bmatrix} \sin \theta_n & \cos \theta_n \\ \cos \theta_n & -\sin \theta_n \end{bmatrix} \quad (7)$$

where

$$\sin \theta_n = \sqrt{\frac{2^{2n-2} - 1}{2^{2n} - 1}}, \quad 0 < \theta < \frac{\pi}{2}. \quad (8)$$

The Slant matrix decomposition in natural order establishes the fast algorithm framework in Fig. 1 which requires $N(n+1) - 2$ additions, $N - 2$ shifts, $N/4 - 1$ multiplications, and finally $3N/4 - 1$ normalizations at the last stage of computation. Note that after the computation at each stage $k > 1$, partial normalization is performed by postmultiplying the matrix in (9) by rows $(4i+1)2^{k-2}$ and $(2i+1)2^{k-1}$ for $i = 0, \dots, n-k$ as shown in Fig. 1. This requires 2 shifts, 2 additions and 1 multiplication for each pair of rows.

$$\begin{bmatrix} 2^{k-1} & -(2^{2k-2} - 1) \\ 1 & 2^{k-1} \end{bmatrix}. \quad (9)$$

A final normalization step is also required at the end to obtain a unitary transform with modulus 1. A detailed mathematical analysis is given in [5]. The sequency order representation can be recovered from the natural order representation using a perfect shuffle as described in [6]. Another Slant matrix decomposition by Pratt *et al.* [10], [11] exists which computes the Slant Transform in sequency order with a fast algorithm which trades 2^{n-1} additions and 2^{n-1} shifts for 2^{n-1} multiplications in the above algorithm.

C. The Slant Transform Truncation (STT) Method

1) Theory: The mathematical formulation of the STT method is based on the FST algorithm, but takes advantage of coherence to provide a faster algorithm. Consider the binary tree representation of the data in Fig. 2(a) as shown in Fig. 2(b), where terminal nodes store data values and non

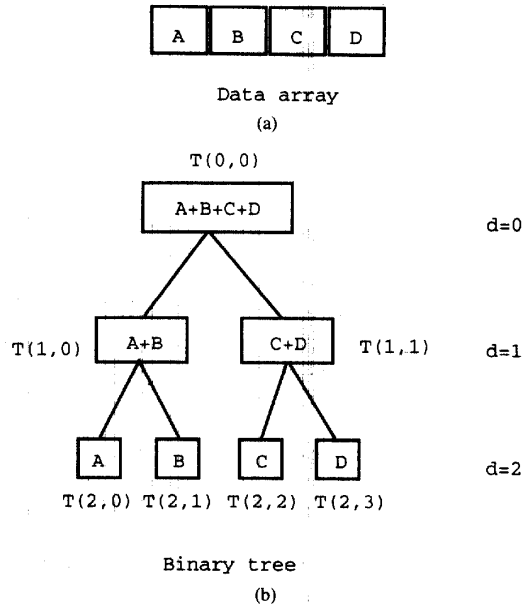


Fig. 2. An array and its binary tree.

terminal nodes store the sum of their children's node values. Furthermore, let terminal nodes which are not at the lowest level have a value $P2^m$ if they represent 2^m pixels all having color P .

Let us denote the nodes starting with the root node at depth $d = 0$ as $T(0,0)$, then at depth $d = 1$ starting with the leftmost node as $T(1,0)$ as shown in Fig. 2(b) and in general, let a node at depth $d = \alpha$ which is at position β from the leftmost node at that level be denoted by $T(\alpha, \beta)$.

Using the data in Fig. 2(b) at depth $d = 2$ and the FST framework, the Slant Transform of a data array with 4 elements is computed as below (ignoring normalization and reordering

to sequency order). First we compute

$$\begin{aligned}
S'[0] &= A + B + C + D \\
&= T(2, 0) + T(2, 1) + T(2, 2) + T(2, 3) \\
S'[1] &= A - B + C - D \\
&= T(2, 0) - T(2, 1) + T(2, 2) - T(2, 3) \\
S'[2] &= A + B - C - D \\
&= T(2, 0) + T(2, 1) - T(2, 2) - T(2, 3) \\
S'[3] &= A - B - C + D \\
&= T(2, 0) - T(2, 1) - T(2, 2) + T(2, 3)
\end{aligned}$$

Finally, rotation gives the Slant Transform $S[0] = S'[0], S[1] = 2S'[1] - S'[2], S[2] = S'[1] + 2S'[2]$ and $S[3] = S'[3]$. We note that the Slant Transform is computed from the linear combinations of the data at depth $d = 2$.

Suppose $A = B$ and $C = D$. In this case, the binary tree has depth $d = 1$ and the two terminal nodes have values $2A$ and $2C$. Again using the data in Fig. 2(b) at depth $d = 1$ and the FST framework, we have

$$\begin{aligned}
S'[0] &= 2A + 2C = T(1, 0) + T(1, 1) \\
S'[1] &= 0 \\
S'[2] &= 2A - 2C = T(1, 0) - T(1, 1) \\
S'[3] &= 0
\end{aligned}$$

and rotation gives the Slant Transform $S[0] = S'[0], S[1] = -S'[2], S[2] = 2S'[2]$ and $S[3] = 0$. Note in this case that the Slant Transform can be computed solely from the data stored at depth $d = 1$ in the tree. Similarly, if $A = B = C = D$, the Slant Transform is $S[0] = 4A = T(0, 0)$ and $S[i] = 0$ for $i = 1, 2, 3$. This approach using binary trees and the FST framework can be generalized to a theorem for any array of $N = 2^n$ data values.

Theorem 1: If the corresponding binary tree representation of an array of size $N = 2^n$ has maximum depth d , then the nonzero elements after $n - d$ stages of the FST method excluding the rotation at stage $n - d$ are $S_{n-d}[i2^{n-d}]$, and the nonzero elements introduced by the rotation at stage $n - d$ are $S_{n-d}[(4\beta + 1)2^{n-d-1}]$ for $i = 0, \dots, 2^d - 1$ and $\beta = 0, \dots, 2^{d-1} - 1$.

Proof of Theorem 1: Starting with the data array $s[i]$ as $S_0[i]$, stage $k = 1, \dots, n$ of the FST algorithm in Fig. 1 is expressed mathematically in terms of stage $k - 1$ by

$$S_k[i + j\delta] = S_{k-1}[i + j\delta] + S_{k-1}[i + (j + \frac{1}{2})\delta] \quad (10)$$

$$S_k[i + (j + \frac{1}{2})\delta] = S_{k-1}[i + j\delta] - S_{k-1}[i + (j + \frac{1}{2})\delta] \quad (11)$$

and the rotation for $k > 1$ by

$$t = S_k[(j + \frac{1}{4})\delta] \quad (12)$$

$$S_k[(j + \frac{1}{4})\delta] = \phi_1 t - \phi_2 S_k[(j + \frac{1}{2})\delta] \quad (12)$$

$$S_k[(j + \frac{1}{2})\delta] = t + \phi_1 S_k[(j + \frac{1}{2})\delta] \quad (13)$$

where $\phi_1 = 2^{k-1}, \phi_2 = (2^{2k-2} - 1)/3, \delta = 2^k, j = 0, \dots, 2^{n-k} - 1$ and $i = 0, \dots, 2^{k-1} - 1$. We now use induction on $n - d$ to carry out the proof.

- 1) When $n - d = 0$, the FST method gives $S_1[2j] = s[2j] + s[2j + 1], S_1[2j + 1] = s[2j] - s[2j + 1]$ for $j = 0, \dots, 2^{n-1} - 1$. Since $s[2j] \neq s[2j + 1]$ for all j , $S_1[\alpha]$ is an array of nonzero elements which proves the theorem for $n - d = 0$. No rotation is required for $k = 1$.
- 2) When $n - d = 1$, adjacent array elements $s[2j] = s[2j + 1]$. The FST method gives $S_1[2j] = s[2j] + s[2j + 1], S_1[2j + 1] = 0$ and no rotation is required. Stage $k = 2$ gives $S_2[4j] = S_1[4j] + S_1[4j + 2], S_2[4j + 1] = 0, S_2[4j + 2] = S_1[4j] - S_1[4j + 2]$ and $S_2[4j + 3] = 0$. Finally, rotation gives $S_2[4j + 1] = -\phi_2 S_2[4j + 2]$ and $S_2[4j + 2] = \phi_1 S_2[4j + 2]$. Therefore $S_2[4j + 3] = 0$ which proves the theorem for $n - d = 1$ as required.
- 3) By repeated application of 2) for all $r < (n - d)$, $S_{r-1}[i + (j + 1/2)\delta_1] = S_{r-1}[i + j\delta_1]$, and thus the nonzero elements are $S_{r-1}[j\delta_1]$ where $\delta_1 = 2^{r-1}$. The FST method at stage r gives $S_r[j\delta] = S_{r-1}[j\delta_1] + S_{r-1}[(j + 1/2)\delta_1]$ and $S_r[(j + 1/2)\delta] = 0$ where $\delta = 2^r$. Therefore, $S_r[i + j\delta] = 0$ for $i = 1, \dots, \delta - 1$. It therefore follows that since both $S_r[(j + 1/2)\delta] = 0$ and $S_r[(j + 1/4)\delta] = 0$, rotation does not affect the final result giving $S_r[j\delta] = 2S_{r-1}[j\delta_1]$ and $S_r[i + j\delta] = 0$ for $i = 1, \dots, \delta - 1$. At stage $p = (n - d)$, $S_r[(j + 1/2)\delta] \neq S_r[j\delta]$, therefore $S_p[j\delta_2] = S_r[j\delta] + S_r[(j + 1/2)\delta]$ and $S_p[(j + 1/2)\delta_2] = S_r[j\delta] - S_r[(j + 1/2)\delta]$ where $\delta_2 = 2^p$. Since $S_p[(j + 1/4)\delta_2] = 0$ rotation introduces additional nonzero elements $S_p[(j + 1/4)\delta_2] = -\phi_2 S_p[(j + 1/2)\delta_2]$ which completes the proof of the theorem.

Theorem 2: If the corresponding binary tree representation of an array of size $N = 2^n$ has maximum depth d , then the nonzero coefficients of the Slant Transform are $S[i2^{n-d}]$, and $S[(4\beta + 1)2^{n-d-1}]$ for $i = 0, \dots, 2^d - 1$ and $\beta = 0, \dots, 2^{d-1} - 1$.

Proof of Theorem 2: The proof of this theorem follows directly from Theorem 1. We only need to prove that for stages $p > (n - d)$, no further nonzero elements are created. At stages $p > (n - d)$, the FST method performs additions and subtractions on elements at a distance 2^p apart. From Theorem 1, these elements are nonzero at stage $n - d$ because 2^{n-d} is a factor of 2^p for all $p > (n - d)$, and therefore no extra nonzero elements are introduced. Rotation is performed on elements at positions $(j + 1/2)2^p$ and $(j + 1/4)2^p$. From Theorem 1, $S_{n-d}[(j + 1/2)2^{n-d}]$ and $S_{n-d}[(j + 1/4)2^{n-d}]$ are nonzero. Since 2^{n-d} is a factor of 2^p , the elements at positions $(j + 1/2)2^p$ and $(j + 1/4)2^p$ are nonzero at stage $n - d$. Therefore, nonzero elements are not introduced at stages $p > (n - d)$, which concludes the proof.

Note that Theorem 2 can be restated as follows:

Theorem 3: If the corresponding binary tree representation of an array of size $N = 2^n$ has maximum depth d , then the Slant Transform coefficients depend solely on the data in the tree at depth d .

2) *The Slant Transform Truncation Algorithm:* The Slant Transform Truncation Algorithm uses a binary tree to segment and aggregate the data $s[x]$ when computing the Slant Transform of an array. The fundamental idea is to identify the zeros at each stage of the computation using logical operations in order to avoid performing shifts, additions, subtractions and multiplications involving zeros. The algorithm commences with the computation of Slant coefficients of subarrays of size 2 and proceeds until the Slant Transform of the entire input array is computed. The Slant coefficients of a subarray of size 2^k are computed in place from those of its subarrays of size 2^{k-1} .

We define the *sparseness degree* ξ of a subarray to be the maximum distance between any two nonzero elements in the array. Therefore a subarray of size 2^v with nonzero Slant coefficients $s[i2^r]$ and $s[(4\beta+1)2^{r-1}]$ for $i = 0, \dots, 2^{v-r} - 1$ and $\beta = 0, \dots, 2^{v-r-1} - 1$ has a sparseness degree $\xi = 2^r$. When computing the Slant coefficients of a subarray of size 2^k from its child subarrays of sizes 2^{k-1} there are three cases to consider depending on the sparseness degrees ξ_1 and ξ_2 of its left and right child subarrays.

- I) $\xi_1 = \xi_2$. If $\xi_1 > 1$, the nonzero elements are at a distance ξ_1 apart and the nonzeros introduced by rotations are at a distance $2\xi_1$ apart. Therefore, (10) and (11) become

$$S_k[i\xi_1 + j\delta] = S_{k-1}[i\xi_1 + j\delta] + S_{k-1}[i\xi_1 + (j + \frac{1}{2})\delta] \quad (14)$$

$$S_k[i\xi_1 + (j + \frac{1}{2})\delta] = S_{k-1}[i\xi_1 + j\delta] - S_{k-1}[i\xi_1 + (j + \frac{1}{2})\delta] \quad (15)$$

$$S_k[2\beta\xi_1 + (j + \frac{1}{4})\delta] = S_{k-1}[2\beta\xi_1 + (j + \frac{1}{4})\delta] + S_{k-1}[2\beta\xi_1 + (j + \frac{1}{2})\delta] \quad (16)$$

$$S_k[2\beta\xi_1 + (j + \frac{1}{2})\delta] = S_{k-1}[2\beta\xi_1 + (j + \frac{1}{4})\delta] - S_{k-1}[2\beta\xi_1 + (j + \frac{1}{2})\delta] \quad (17)$$

for $i = 0, \dots, 2^{k-1}/\xi_1 - 1$ and $\beta = 0, \dots, 2^{k-2}/\xi_1 - 1$. Finally, rotation is applied to the result using (12) and (13). Computational savings are achieved when $\xi_1 > 1$ as redundant computations involving shifts, additions, subtractions and multiplications of zeros are avoided. Such redundant computations are performed by the FST method. Fewer additions, subtractions, shifts and multiplications are performed by the STT algorithm in this case. After the computation, the sparseness degree of the resulting region is ξ_1 . If $\xi = 1$, the STT algorithm collapses to the FST algorithm and the Slant coefficients are computed by (10)–(13).

- II) $\xi_1 > \xi_2$. Corresponding nonzero coefficients in the right and left subarrays are computed using (14)–(17). The remaining nonzeros in the right subarray have zeros as corresponding elements in the left subarray; therefore these nonzeros are simply copied to the corresponding positions in the left subarray and themselves negated. Finally, rotation is performed using (12) and (13). Again computational savings are achieved as fewer additions and subtractions are performed. Second, some additions and subtractions are replaced by negation and copy

operations which are faster to perform. The sparseness degree of the resulting region becomes ξ_2 .

- III) $\xi_1 < \xi_2$. Equations (14)–(17) are applied with ξ_1 replaced by ξ_2 . The remaining nonzero elements in the left subarray are copied to their corresponding positions in the right subarray. Computational savings are achieved as explained in I) and II). In addition, some additions are replaced by copy operations which are faster to perform. The sparseness degree of the resulting region becomes ξ_1 .

Note that in I) to III), if a subarray of size 2^k is uniform with value λ , the first Slant Transform coefficient is $2^k\lambda$ and the other coefficients are zeros. In the implementation, the first coefficient is obtained by simply shifting the binary representation of λ by k places to the left and zeros are implicitly stored in the remaining locations. This is to avoid setting coefficients to zero which may be over-written by other coefficients in later stages. Therefore, the method only computes coefficients when the subarrays are nonuniform. This will become clear in the implementation given in Algorithm 1.

3) *Computational Complexity:* Preprocessing of the data into a binary tree requires $O(N)$ logical operations. If the data is uniform, $S[0] = 2^v s[0]$ and $S[i]$ is set to zero for $i = 1, \dots, N - 1$. In this case, the STT method requires one shift, $N - 1$ zero operations and the tree preprocessing, and therefore the performance is $O(N)$. In the worst case when there is no coherence in the data, the STT algorithm degenerates to the FST algorithm which takes time $O(N \log_2 N)$, with an additional $O(N)$ logical operations required to preprocess the data. Since coherence is a phenomenon characteristic of digital images, the STT algorithm is expected to be superior to the FST algorithm for such data. The superiority of the method arises because we avoid computing elements which can be deduced to be zero by simply using logical operations.

Standard transform theory [4] tells us that two-dimensional transforms can be computed using one-dimensional transform methods on each row of the $N \times N$ array, then on each column of the semitransformed array. Using this approach the one-dimensional STT algorithm can be extended to compute the Slant Transform of $N \times N$ arrays. Following a similar analysis, the row/column STT method takes time between $O(N^2)$ and $O(N^2 \log_2 N)$, at worst degenerating to the performance of the row/column FST method apart from the preprocessing step which is now $O(N^2)$ logical operations.

4) *Implementations:* One method of implementing the STT algorithm is to use an explicit binary tree structure to exploit coherence. Data is stored in terminal nodes and nonterminal nodes are used to store intermediate Slant coefficients. A detailed explanation of such implementations in the context of Walsh-Hadamard Transform Truncation algorithms is given in [2]. In this case the only drawback is the memory required by the binary tree.

A second approach is to perform a recursive computation. The binary tree is simulated on the data array and a stack is used to store the sparseness degrees of the subarrays. Since the simulated tree has depth n , a stack of n locations is required. Similar approaches have been used to develop one

and two-dimensional algorithms [1], [3] for faster computation of Walsh-Hadamard transforms.

Third, an iterative method can be used which requires $N/4$ locations to store the sparseness degrees. An in-place binary tree is constructed on the data array and the $N/4$ locations are used to store the sparseness degrees of each subarray. This approach is given in Algorithm 1 below. For two-dimensional array of size $N \times N$, Algorithm 1 is applied to each row to compute row-transforms, then to each column of the row-transforms to compute the Slant transform. This is the approach used to obtain the experimental results given in Section III-D.

In Algorithm 1, $SP[i]$ is an array which stores the sparseness degrees. The expression $a \ll b$ means shift the binary representation of a by b places to the left. The first and second stages of this algorithm are treated as special cases for extra efficiency to avoid shifts by zero and multiplications by 1 and -1 , so computation commences with subarrays of size 4 as shown below.

If the output from Algorithm 1 has sparseness degree $\xi > 1$, then the nonzero coefficients are $S[i\xi]$ and $S[(4\beta + 1)\xi/2]$ for $i = 0, \dots, 2^{k-1}/\xi - 1$ and $\beta = 0, \dots, 2^{k-2}/\xi - 1$ as shown in Theorem 2. Therefore the remaining implicitly zero coefficients must be set to zero. Conversion to sequency order is done using the perfect shuffle and the final result is normalized as explained in [6].

D Experimental Results

Experimental results are presented for three digital images of sizes 512×512 shown in Figs. 3–5 with 256 gray levels. Algorithm 1 is applied to each row of the image data to compute row-transforms, then to each column of the row-transforms to compute the two-dimensional Slant transform, without using any extra technique to reduce the number of terminal nodes. The images in Figs. 3 and 4 were deliberately chosen with relatively little coherence so as not to exaggerate the claims we are making. The image in Fig. 5 is a typical digital image characterized by coherent subregions. The results are given for the Slant Transform Truncation method and the FST method for comparison in Table I. (Note that the values for the FST method are independent of the input data and thus apply to all images).

The entries in the table are as follows: STT is the Slant Transform Truncation method and FST is the Fast Slant Transform method. %C is the percentage of image coherence exploited by the Slant Transform Truncation method. $\%C = 100(1 - \text{term}/(2 \times 512^2))$ where *term* is the total number of terminal nodes in the rows and columns. Note that the maximum possible number of terminal nodes when computing row-transforms is 512^2 , and the same number is possible when computing column-transforms of the semi-transformed data, justifying the reason for dividing *term* by 2. S, A, N, Z, AS and M are the number of shifts, assignments (copy), negations, set-to-zeros, additions/subtractions and multiplications by each method respectively. The overall time in the table includes both the time required for factorising the input data using logical operations and the time required for computing the



Fig. 3. Test image showing M. M. Anguh.



Fig. 4. Test image showing R. R. Martin.

transform. It can be seen for these images that the STT method requires less total operations than the FST method even allowing for tree construction overheads, and even more importantly has smaller *overall* times of execution. Note that the performance of the STT method increases with coherence. The total number of operations is not actually proportional to the running time because some operations like copy and negation are faster to perform than addition/subtraction.

IV. GENERAL REMARKS

The computation of two-dimensional transforms using the row/column Truncation method does not maximize the exploitation of image coherence. This is because area (two

Algorithm 1

(Do first and second stages)

```

i = 1;
For (I = 0; I < N; I = I + 4)
{
    I1 = I + 1; I2 = I + 2; I3 = I + 3;
    If (s [I] = s[I1])
    {
        If (s [I2] = s[I3])
        {
            If (s [I] = s[I2]){SP[i] = 4; } else
            {
                t = s[I]; s[I] = (t + s[I2]) ≪ 1; s[I1] = (s[I2] - t) ≪ 1;
                s[I2] = (t - s[I2]) ≪ 2; SP[i] = 2;
            }
        } else
        {
            t = s[I2]; s[I2] = t + s[I3]; s[I3] = t - s[I3]; t = s[I] ≪ 1;
            s[I] = t + s[I2]; s[I2] = t - s[I2]; s[I1] = (s[I3] ≪ 1) - s[I2];
            s[I2] = (s[I2] ≪ 1) + s[I3]; s[I3] = -s[I3], SP[i] = 1;
        }
    } else
    {
        If (s[I2] = s[I3])
        {
            t = s[I]; s[I] = t + s[I1]; s[I1] = t - s[I1]; t = s[I2] ≪ 1;
            s[I2] = s[I] - t; s[I] = s[I] + t; s[I3] = s[I1]; SP[i] = 1;
            s[I1] = (s[I1] ≪ 1) - s[I2]; s[I2] = (s[I2] ≪ 1) + s[I3];
        } else
        {
            t = s[I]; s[I] = t + s[I1]; s[I1] = t - s[I1]; t = s[I2]; SP[i] = 1;
            s[I2] = t + s[I3]; s[I3] = t - s[I3]; t = s[I]; s[I] = t + s[I2];
            s[I2] = t - s[I2]; t = s[I1]; s[I1] = t + s[I3]; s[I3] = t - s[I3];
            t = s[I1]; s[I1] = (t ≪ 1) - s[I2]; s[I2] = t + (s[I2] ≪ 1);
        }
    }
}
i = i + 1

```

(Do remaining stages)

```

P = 4; P1 = 8;
For (r = 3; r ≤ n; r = r + 1)
{
    i = 1; i1 = 1; i2 = 2
    For (I = 0; I < N; I = I + P1)
    {
        If (SP[i1] = SP[i2])
        {
            If (SP[i1] = P)
            {
                If (s[I] = s[P + I])SP[i] = P1 else Section III-C2)I)
            } else Section III-C2)II)
        } else If (SP[i1] > SP[i2]) Section III-C2)II) else Section III-C2)III)
        i = i + 1; i1 = i1 + 2; i2 = i2 + 2;
    }
}
P = P ≪ 1; P1 = P1 ≪ 1;
}

```

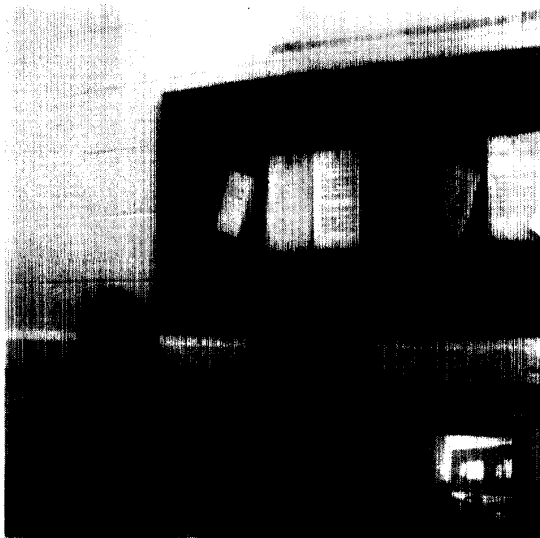


Fig. 5. Another test image showing M. M. Anguh.

TABLE I
EXPERIMENTAL RESULTS

	%C	S	A	N	Z	AS	M
STT							
M.M.Anguh	5.855751	474084	24010	16905	0	4926768	129625
	Total operations = 5571392			Overall time = 17s			
R.R.Martin	6.642914	469209	30598	20914	0	4891440	129376
	Total operations = 5541537			Overall time = 16s			
At work	25.616074	485102	89240	39465	0	433741	128970
	Total operations = 5080193			Overall time = 13s			
FST							
All images	0	522240	0	0	0	5240832	130048
	Total operations = 5893120			Overall time = 19s			

dimensional) coherence is not fully exploited. When the one-dimensional Truncation method is initially applied to the rows, some coherence along the columns is lost (or vice versa if columns are transformed before rows). For the Slant Transform, rotation introduces extra nonzero elements when preprocessing along the columns and these nonzero elements reduce the amount of coherence along the columns. In our earlier work [2], [3] on Walsh transforms, we presented direct two-dimensional methods for computing Walsh transforms using a quadtree and an adaptive quadtree. We noted that preprocessing the data using a quadtree increased the exploitation of coherence; an adaptive quadtree did so to an even greater degree. Furthermore, we also extended the direct methods to radix 4 computation. Analogous (adaptive) quadtree preprocessing of the two-dimensional data can be used to develop direct two-dimensional methods for computing the Slant Transform. Note that the ability to exploit coherence depends on the transform.

V. CONCLUSIONS

The Truncation algorithm for computing Slant Transforms of one- and two-dimensional arrays of data are presented. The method uses hierarchical data structures to exploit coherence by factorizing the data into sparse subregions. The method takes time between $O(N)$ and $O(N \log_2 N)$ in one dimension and time between $O(N^2)$ and $O(N^2 \log_2 N)$ in two dimensions, degenerating to the performance of the FST algorithm in the absence of image coherence. In conclusion, the Truncation algorithm is shown to be superior to the FST algorithm on digital images both in theory and practice due to the presence of coherence.

REFERENCES

- [1] M. M. Anguh and R. R. Martin, "An in-place Truncation Walsh transform for image processing," *Actes du 1er Colloque Africain sur la Recherche en Informatique*, Cameroon, vol. 1, pp. 457-468, Oct. 1992.
- [2] ———, "A Truncation method for computing Walsh transforms with applications to image processing," in *Comput. Vision, Graph. & Image Processing: Graphical Models & Image Processing*, vol. 55, no. 6, pp. 482-493, 1993.
- [3] ———, "A two dimensional in-place Truncation Walsh transform method," *J. Visual Commun. Image Represent.*, to be published.
- [4] D. F. Elliot and K. R. Rao, *Fast Transforms Algorithms, Analysis and Applications*. New York: Academic, 1982.
- [5] B. J. Fino and V. R. Algazi, "A unified treatment of discrete fast unitary transforms," *SIAM J. Comput.*, vol. 6, no. 3, Dec. 1977.
- [6] ———, "A unified matrix approach to Walsh Hadamard transforms," *IEEE Trans. Comput.*, vol. C-25, pp. 1142-1145, 1976.
- [7] R. R. Martin and M. M. Anguh, "Transforms and image coding," *Comput. Graph. Forum*, vol. 10, pp. 91-96, 1991.
- [8] H. Samet, *Applications of Spatial Data Structures*. Reading, PA: Addison-Wesley, 1990.
- [9] ———, *The Design and Analysis of Spatial Data Structures*. Reading, PA: Addison-Wesley, 1990.
- [10] W. K. Pratt, W. H. Cheng, and L. R. Welch, "Slant transform image coding," *IEEE Trans. Commun.*, vol. COM-22, pp. 1075-1093, Aug. 1974.
- [11] ———, "Slant transform for image coding," in *Proc. 1972 Symp. Application of Walsh Functions*, vol. AD-744650, 1972, pp. 229-234.



Maurene M. Anguh was born October 15, 1967, in Cameroon. He received the undergraduate degree from the University of Wales College of Cardiff from 1987-1990. He received the Ph.D. degree in computer science from the same university from 1990 to 1993.

He was awarded a scholarship to study in the U.K. by the Cameroon Government in 1987. He is presently working as a Professor in the Department of Electrical Engineering and Computer Science in the Federal University of Maranhao, Brazil.



Ralph R. Martin received the Ph.D. degree in 1983 from Cambridge University for a dissertation on "Principal Patches".

He has been working in the field of CAD/CAM since 1979. He has also been a Lecturer at the University of Wales College of Cardiff. He has published over 40 papers and 4 books covering the area of geometric computing.

Dr. Martin is a Fellow of the Institute of Mathematics and its Applications, and a Member of the British Computer Society. He has held two SERC ACME grants.