Faster Methods for Computing Slant Haar Transforms of Digital Images

M. M. ANGUH[1]
&
R. R. MARTIN [2]

[1]Department of Electrical Engineering,
Centro Tecnológia, Bacanga
Federal University of Maranhão
CEP: 65085-580, São Luis, MA. Brazil
mma@fapema.br

[2]Dept. of Computing Mathematics, UWCC,
P.O. Box 916, Cardiff, CF2 4YN, UK.
ralph@cm.cf.ac.uk

Abstract. A phenomenon characteristic of digital images is the presence of approximately constant or uniformly changing gray levels over a considerable distance or area. A novel method, the *Truncation Slant Haar Transform (TSHT)* method for computing the Slant Haar Transform (SHT) of digital images is presented which exploits this phenomenon to advantage. The TSHT method utilises a hierarchical tree to segment and aggregate uniform image data and a matrix factorisation to eliminate transform matrix redundancies. This simultaneous exploitation of inter-pixel relationships and the elimination of transform matrix redundancies produces an efficient method for computing the SHT of digital images. In one dimension with an array of $N = 2^n$ data values, the TSHT method takes time between $O(N)$ and $O(N \log_2 N)$, thus degenerating to the performance of the standard Fast Slant Haar Transform (FSHT) method. In two dimensions with an array of size $N \times N$, the performance of the TSHT method is between $O(N^2)$ and $O(N^2 \log_2 N)$, again degenerating to that of the FSHT method in its worst case. Since coherence is a fundamental characteristic of digital images, the TSHT method is therefore superior to the FSHT method when used to compute the SHT of coherent digital images. Experimental results are presented to justify this assertion.

1 Introduction

Digital images are characterised by the presence of constant or uniformly changing gray levels over a considerable distance or area. The Slant Haar Transform (SHT) [4, 5, 10, 11] is defined with a discrete sawtooth basis vector which changes uniformly with distance for representing gradual increase in brightness.

The discrete SHT matrix S_N of order $N = 2^n$ is defined such that

$$\frac{1}{N} S_N S_N^{-1} = I_N \tag{1}$$

where S_N^{-1} is the inverse of S_N and I_N is the $N \times N$ identity matrix. The Slant Haar Transform F of an input array f both of size N is defined as

$$F = \frac{1}{N} f S_N \tag{2}$$

A brute force computation of F by multiplying f and S_N requires $O(N^2)$ operations. The Fast Slant Haar Transform (FSHT) method [4, 5] for computing F is based on the factorisation of S_N into a set of n largely sparse matrices, each expressing a stage of the computation. This requires $O(N \log_2 N)$ elementary operations.

Our earlier work [1, 2, 3, 8] established relationships between Walsh-Hadamard, Slant Transforms and Hierarchical encodings, particularly noting that both methods of encoding use the same basis to represent information present in the data. Furthermore, computing orthogonal transforms of digital images directly in terms of pixel values as done by fast transform methods [6] is somewhat inefficient because interpixel relationships (coherence) are not used to advantage. An efficient method for computing orthogonal transforms of digital images should therefore use data coherence to an advantage. Truncation methods [1, 2, 3, 8] for computing Walsh-Hadamard and Slant Transforms of images have been developed which utilise both the factorisation of the transform matrix and the exploitation of data coherence using hierarchical data structures.

This paper presents an analogous idea for computing the SHT of coherent data such as that representing digital images. This method is faster than the corresponding FSHT method as it simultaneously factorises S_N into sparse matrices and the data f into sparse sub-regions using a hierarchical structure. In particular, we prove that if the corresponding binary tree representation of f has maximum depth d, then the SHT coefficients

$$F[u] = \begin{cases} 0 & for\ \ u \neq i2^{n-d} \\ & (0 \leq i \leq 2^d - 1) \\ 0 & for\ \ u \neq (2i+1)2^{n-d-1} \\ & (0 \leq i \leq 2^{d-1} - 1)\ \ (0 < d < n) \end{cases} \tag{3}$$

These two results readily generalise to two and higher dimensions as will been seen.

The TSHT method utilises a hierarchical data structure to identify and avoid the computation of the zeros in Equation (3) which are normally computed by the FSHT method. This makes the TSHT method superior to the FSHT method for computing the SHT of coherent digital images.

Analysis of time complexity in one and two dimensions shows that the TSHT 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 FSHT method in the worst case. A set of experiments conducted on digital images using the FSHT and TSHT methods is presented to demonstrate overall actual time savings.

2 Hierarchical structures

Hierarchical data structures are compression tools which exploit inter-pixel relationships to aggregate data into sparse sub-regions.. 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 sub-regions until 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 f is recursively subdivided into left and right halves, the resulting structure is a binary tree. The binary tree construction requires $O(N)$ logical operations. Other hierarchical structures include the quadtree and bintree used to represent two dimensional arrays. Hierarchical data structures have been used for compression [12], image segmentation [7] amongst many others [9, 13, 14].

3 The Slant Haar Transform (SHT)

The Slant Haar Transform matrix [4, 5] S_N of order $N = 2^n$ is computed from the SHT matrix $S_{\frac{N}{2}}$ of order $N = 2^{n-1}$ by first computing

$$S_2 \otimes S_{2^{n-1}} \tag{4}$$

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

$$\begin{bmatrix} \sin\theta_n & \cos\theta_n \\ \cos\theta_n & -\sin\theta_n \end{bmatrix} \tag{5}$$

where

$$\cos\theta_n = \frac{2^{2n-1}}{\sqrt{\frac{2^{2n}-1}{3}}}, \qquad 0 < \theta < \frac{\pi}{2} \tag{6}$$

which introduces the slant properties of linearly decreasing components. Factorisation of the SHT matrix S_8 gives the FSHT framework in Figure 1. The

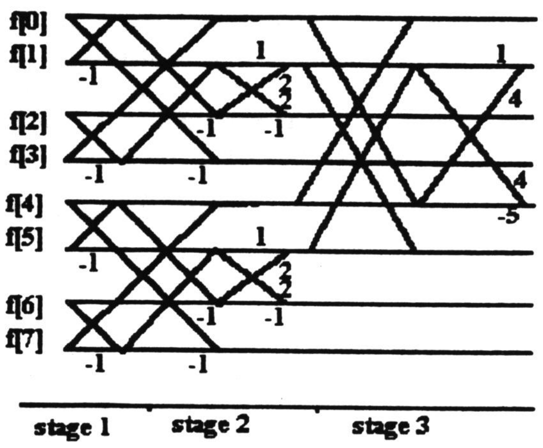

Figure 1: The FSHT framework

Fast Slant Haar Transform (FSHT) method requires $4N - 6$ additions, $N/4 - 1$ multiplications, $N - 2$ shifts and $3N/4$ normalisations at the last stage of the computation. A partial step by step normalisation is performed at each stage $k > 1$ by post-multiplying rows $(1 + 4i)2^{k-2}$ and $(1 + 2i)2^{k-1}$ for $i = 0, \ldots, n - k$ by

$$\begin{bmatrix} 1 & 2^{k-1} \\ 2^{k-1} & \frac{2^{2k-2}-1}{3} \end{bmatrix} \tag{7}$$

as shown in Figure 1. This requires 2 shifts, 2 additions and 1 multiplication for each pair of rows.

4 The TSHT method

This section presents the mathematical foundation of the Truncation Slant Haar Transform (TSHT) method. The theory is used to develop the TSHT algorithm. The algorithm is developed in one dimension and generalised to higher dimensions. The complexity of the TSHT method is analysed in comparison with that of the FSHT method. Finally, experimental results are presented to demonstrate the superiority of the TSHT method over the FSHT method.

4.1 Theory of the TSHT method

The mathematical formulation of the TSHT method is based on the FSHT framework shown in Figure 1, but takes advantage of coherence to provide a faster algorithm. Consider the binary tree representation of the data in Figure 2(a) as shown in Figure 2(b), where terminal nodes store data values and non 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 $P2^d$ if they represent 2^d pixels all having colour 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 Figure 2(b) and in general, let a node at depth $d = \gamma$ which is at position ω from the leftmost node at that level be denoted by $T[\gamma, \omega]$.

Using the data in Figure 2(b) at depth $d = 2$

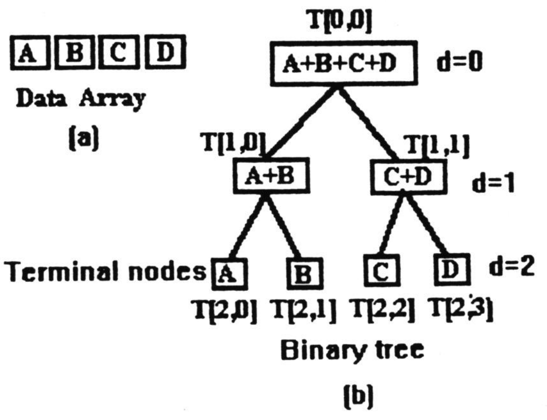

Figure 2: An array and its binary tree

and the FSHT framework, the SHT of a data array with 4 elements is computed as below (ignoring normalisation and reordering to sequency order). First we compute

$$F_1[0] = A + B + C + D = T[2,0] + T[2,1] + T[2,2] + T[2,3] \quad (8)$$

$$F_1[1] = A - B + C - D = T[2,0] - T[2,1] + T[2,2] - T[2,3] \quad (9)$$

$$F_1[2] = A + B - C - D = T[2,0] + T[2,1] - T[2,2] - T[2,3] \quad (10)$$

$$F_1[3] = A - B - C + D = T[2,0] - T[2,1] + T[2,2] - T[2,3] \quad (11)$$

Finally, rotation gives the SHT $F[0] = F_1[0]$, $F[1] = F_1[1] + 2F_1[2]$, $F[2] = 2F_1[1] - 2F_1[2]$ and $F[3] = F_1[3]$. We note that the SHT 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 Figure 2(b) at depth $d = 1$ and the FSHT framework, we have

$$\begin{aligned} F_1[0] &= 2A + 2C = T[1,0] + T[1,1] \\ F_1[1] &= 0 \\ F_1[2] &= 2A - 2C = T[1,0] - T[1,1] \\ F_1[3] &= 0 \end{aligned}$$

and rotation gives the SHT $F[0] = F_1[0]$, $F[1] = 2F_1[2]$, $F[2] = -F_1[2]$ and $F[3] = 0$. Note in this case that the SHT can be computed solely from the data stored at depth $d = 1$ in the tree. Similarly, if $A = B = C = D$, the SHT is $F[0] = 4A = T[0,0]$ and $F[i] = 0$ for $i = 1,2,3$. This approach using binary trees and the FSHT framework can be generalised to a theorem for any array of $N = 2^n$ data values.

Theorem 1 *If the corresponding binary tree representation of an array f_N has maximum depth d, then the SHT coefficients $F[\alpha] = 0$ for $\alpha = j2^{n-d} + 2, \ldots, (j+1)2^{n-d} - 1$.*

Proof of Theorem 1:
Firstly, we note that stage $k = 1, \ldots, n$ of the FSHT method can be expressed in terms of stage $k - 1$ as follows:

$$F_k[i + j\delta] = F_{k-1}[i + j\delta] + F_{k-1}[i + (j + \frac{1}{2})\delta] \quad (12)$$

$$F_k[i + (j + \frac{1}{2})\delta] = F_{k-1}[i + j\delta] - F_{k-1}[i + (j + \frac{1}{2})\delta] \quad (13)$$

and the rotation for $k > 1$ as

$$\begin{aligned} \Phi &= F_k[1+j\delta] \\ F_k[1+j\delta] &= \Phi + \phi_1 F_k[(j+\frac{1}{2})\delta] \qquad (14) \\ F_k[(j+\frac{1}{2})\delta] &= \phi_1\Phi - \phi_2 F_k[(j+\frac{1}{2})\delta] \qquad (15) \end{aligned}$$

for $i = 0$ when $k = 1$, and $i = 0, 1$ when $k > 1$, $j = 0\ldots, 2^{n-k} - 1$, $\delta = 2^k$, $\phi_1 = 2^{k-1}$ and $\phi_2 = (2^{2k-2} - 1)/3$.
We proceed with this proof by induction on $n - d$.

(i) For $n - d = 0$, the FSHT method gives $F_1[2j] = f[2j] + f[2j+1]$, $F_1[2j+1] = f[2j] - f[2j+1]$ for $j = 0\ldots, 2^{n-1} - 1$. Since $f[2j] \neq f[2j+1]$, F_1 is an array of non-zeros which completes the proof for $n - d = 0$.

(ii) For $n - d = 1$, adjacent array elements have $f[2j] = f[2j+1]$. Stage $k = 1$ of the FSHT method gives $F_1[2j] = f[2j] + f[2j+1]$ and $F_1[2j+1] = 0$ for $j = 0, \ldots, 2^{n-1} - 1$. No rotation is required at stage 1. Stage $k = 2$ gives $F_2[4j] = F_1[4j] + F_1[4j+2]$, $F_2[4j+1] = 0$, $F_2[4j+2] = F_1[4j] - F_1[4j+2]$ and $F_2[4j+3] = 0$ for $j = 0, \ldots, 2^{n-2} - 1$. Rotation gives $F_2[4j+1] = 2F_2[4j+2]$ and $F_2[4j+2] = -F_2[4j+2]$. Thus $F_2[4j+3] = 0$ which is consistent with the theorem for $n - d = 1$.

(iii) By repeated use of (ii) we establish the inductive hypothesis for all $r \leq n - d$. For $p > n - d$ we observe from Equations (12) to (15) that the coefficients $F_{k-1}[\alpha]$ for $\alpha = j2^{k-1} + 2, \ldots, (j+1)2^{k-1} - 1$ are *not* involved in the computation at stage k in the FSHT method. Therefore the zeros at step $n - d$ are not changed by the computations at steps $p > n - d$. This concludes that $F[\alpha] = 0$ for $\alpha = j2^{n-k} + 2, \ldots, (j+1)2^{n-k} - 1$ which proves the theorem □

A special case of this theorem occurs when the whole array is uniform, and the SHT coefficients $F[0] = 2^n f[0]$ and $F[\alpha] = 0$ for $\alpha = 1, \ldots, N - 1$. Note that Theorem 1 can be restated as follows:

Theorem 2 *If the corresponding binary tree representation of an array of $N = 2^n$ elements has maximum depth d, then the SHT coefficients depend solely on the data at depth d. In particular, the zero coefficients $F[u]$ are*

$$F[u] = \begin{cases} 0 & for \;\; u \neq i2^{n-d} \\ & (0 \leq i \leq 2^d - 1) \\ 0 & for \;\; u \neq (2i+1)2^{n-d-1} \\ & (0 \leq i \leq 2^{d-1} - 1) \;\; (0 < d < n) \end{cases} \qquad (16)$$

4.2 Implementation

The TSHT method uses a binary tree to segment and aggregate the data f_N when computing the SHT of an array. The fundamental idea is to unify Theorem 2 with the FSHT framework. This enables the identification of zero coefficients at each stage of the computation using only logical operations. Thus, the TSHT method avoids performing shifts, additions, multiplications and subtractions involving zero coefficients. Such computations involving zeros are normally done by the FSHT method, and hence the TSHT method is therefore superior to the FSHT method.

Define the *sparseness degree* ξ of a sub-array to be the maximum distance between any two non-zero elements in the array. When computing the SHT coefficients of a sub-array of size 2^k from its child sub-arrays 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 sub-arrays.

(I) $\xi_1 = \xi_2$. If $\xi_1 > 1$, Equations (12) to (15) are applied only to non-zero coefficients identified by the hierarchical tree. Computational savings are achieved when $\xi_1 > 1$ as redundant computations involving shifts, additions, subtractions and multiplications of zeros are avoided. Fewer additions, subtractions, shifts and multiplications are performed by the TSHT algorithm in this case. After the computation, the sparseness degree of the resulting region is ξ_1. If $\xi_1 = 1$, the TSHT algorithm collapses to the FSHT algorithm.

(II) $\xi_1 > \xi_2$. Corresponding non-zero coefficients in the right and left sub-arrays are computed using Equations (12) to (15). The remaining non-zeros in the right sub-array have zeros as corresponding elements in the left sub-array; therefore these non-zeros are simply copied to the corresponding positions in the left sub-array and themselves negated. Again computational savings are achieved as fewer additions and subtractions are performed. Secondly, 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 (12) to (15) are applied with ξ_1 replaced by ξ_2. The remaining non-zero elements in the left sub-array are copied to their corresponding positions in the right sub-array. 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 sub-array of size 2^k is uniform with value λ, the first Slant transform co-

efficient 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 sub-arrays are non-uniform.

Algorithm 1 is the TSHT algorithm for computing the SHT of an array f of N data values. The first and second stages of this algorithm are treated separately for extra efficiency to avoid shifts by zero and multiplications by 1 and -1, so computation commences with sub-arrays of size 4 as shown below. The array SP of size $N/4$ stores the sparsity values of each sub-array. For a uniform sub-array 0 is stored and 1 for a non-uniform sub-array.

Algorithm 1

```
(Do first and second stages)
$i = 1;$
For $(I = 0; I < N; I = I + 4)$
{
  $I_1 = I + 1;\ I_2 = I + 2;\ I_3 = I + 3;$
  If $(f[I] = f[I_1])$
  {
    If $(f[I_2] = f[I_3])$
    {
      If $(f[I] = f[I_2])$ { $SP[i] = 0;$ } else
      {
        $t = f[I];\ f[I] = (t + f[I_2]) << 1;$
        $f[I_1] = (t - f[I_2]) << 2;$
        $f[I_2] = (f[I_2] - t) << 1;$
        $f[I_3] = 0;\ SP[i] = 1;$
      }
    } else
    {
      $t = f[I_2];\ f[I_2] = t + f[I_3];$
      $f[I_3] = t - f[I_3];\ t = f[I] << 1;$
      $f[I] = t + f[I_2];\ f[I_2] = t - f[I_2];$
      $f[I_1] = f[I_3] + (f[I_2] << 1);$
      $f[I_2] = (f[I_3] << 1) - f[I_2];$
      $f[I_3] = -f[I_3],\ SP[i] = 1;$
    }
  } else
  {
    If $(f[I_2] = f[I_3])$
    {
      $t = f[I];\ f[I] = t + f[I_1];$
      $f[I_1] = t - f[I_1];\ t = f[I_2] << 1;$
      $f[I_2] = f[I] - t;$
      $f[I] = f[I] + t;\ f[I_3] = f[I_1];$
      $f[I_1] = f[I_1] + (f[I_2] << 1);$
      $f[I_2] = (f[I_3] << 1) - f[I_2];$
      $SP[i] = 1;$
    } else
    {
      $t = f[I];\ f[I] = t + f[I_1];$
      $f[I_1] = t - f[I_1];$
      $t = f[I_2];\ f[I_2] = t + f[I_3];$
      $f[I_3] = t - f[I_3];$
      $t = f[I];\ f[I] = t + f[I_2];\ f[I_2] = t - f[I_2]$
      $t = f[I_1];\ f[I_1] = t + f[I_3];$
      $f[I_3] = t - f[I_3];\ t = f[I_1];$
      $f[I_1] = t + (f[I_2] << 1);$
      $f[I_2] = (t << 1) - f[I_2];\ SP[i] = 1;$
    }
  }
  $i = i + 1$
}
```

```
(Do remaining stages)
$P = 4;\ P_1 = 8;\ SS=5;$
For $(r = 3; r \leq n; r = r + 1)$
{
  $i = 1;\ i_1 = 1;\ i_2 = 2$
  For $(I = 0; I < N; I = I + P_1)$
  {
    $P_2 = P + I;\ P_3 = P_2 + P;$
    If $(SP[i_1] = 0$
    {
      If $(SP[i_2] = 0)$
      {
        If $(f[I] = f[P_2])\ SP[i]=0;$ else
        {
          $t = f[I];\ f[I] = (t + f[P_2]) << r;$
          $f[P_2] = (t - f[P_2]) << r;$
          $f[I + 1] = f[P_2] << r;$
          $f[P_2] = -SS * f[P_2];\ SP[i] = 1;$
          for $(j = I + 2; j < P_2; j = j + 1)$
          $f[j] = 0;$
          for $(j = P_2 + 1; j < P_3; j = j + 1)$
          $f[j] = 0;$
        }
      } else
      {
        $k_1 = P_2 + 1;\ k_2 = I + 1;$
        $t = f[I] << r;\ f[I] = t + f[P_2];$
        $f[P_2] = t - f[P_2];$
        $f[k_2] = f[k_1] + (f[P_2] << r);$
        $f[P_2] = (f[k_1] << r) - SS * f[P_2];$
        $f[k] = -f[k];$
        for $(j = I + 2; j < P_2; j = j + 1)$
        $f[j] = 0;\ SP[i] = 1;$
      }
    } else
    {
        $k_1 = P_2 + 1;\ k_2 = I + 1;$
        $If\ (SP[i_2] = 0)$
        {
          $t = f[P_2] << r;\ f[P_2] = f[I] - t;$
          $f[I] = f[I] + t;\ f[k_1] = f[k_2];$
          $f[k_2] = f[k_1] + (f[P_2] << r);$
          $f[P_2] = (f[k_1] << r) - SS * f[P_2];$
          $for\ (j = P_2 + 2; j < P_3; j = j + 1)$
          $f[j] = 0;\ SP[i] = 1;$
        } else
        {
          $t = f[I];\ f[I] = t + f[P_2];$
          $f[P_2] = t - f[P_2];\ t = f[k_2];$
          $f[k_2] = t + f[k_1];\ f[k_1] = t - f[k_1];$
          $t = f[k_2];\ f[k_2] = t + (f[P_2] << r);$
          $f[P_2] = (t << r) - SS * f[P_2];\ SP[i] = 1;$
        }
      }
      $i = i + 1;\ i_1 = i_1 + 2;\ i_2 = i_2 + 2;$
    }
    $P = P << 1;\ P_1 = P_1 << 1;\ SS = 1 + (SS << 2);$
}
```

4.2.1 Computational Complexity

Data preprocessing using a binary tree requires $O(N)$ logical operations. If the data is uniform, $F[0] = 2^n f[0]$ and $F[i]$ is set to zero for $i = 1, \ldots, N-1$. In this case, the TSHT 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 TSHT algorithm degenerates to the FSHT 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 TSHT algorithm is expected to be superior to the FSHT 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 by applying a one dimensional FSHT method on each row of the $N \times N$ array, then on each column of the semi-transformed array. This approach can be used to compute transforms of higher dimensions. Using an analogous approach, the one dimensional TSHT algorithm can be extended to compute the SHT of $N \times N$ data. Following a similar analysis for the case of two dimensional data, the row and column TSHT method takes time between $O(N^2)$ and $O(N^2 \log_2 N)$, at worst degenerating to the performance of the row and column FSHT method apart from the preprocessing step which is now $O(N^2)$ logical operations.

5 Experimental results

Experimental results are given for the three digital images of size 512×512 shown in Figures 3, 4 and 5.

Results are given for the TSHTand FSHT algorithms (for comparison) in Table 1. The entries in

Figure 3: Anguh

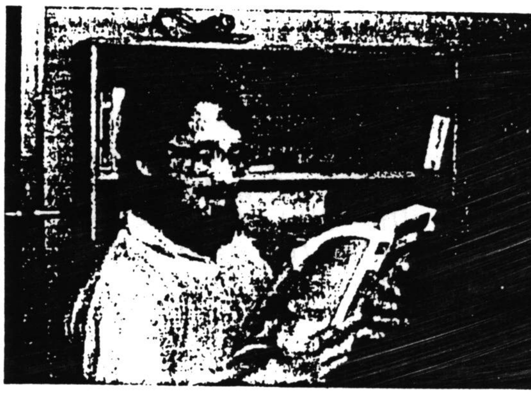
Figure 4: Martin

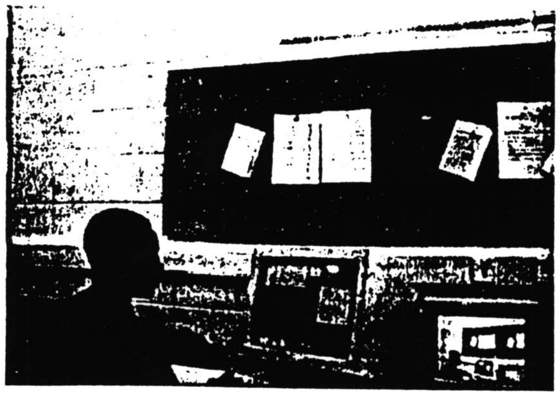

Figure 5: At work

the tables are as follows: %C is the percentage of coherence exploited by the TSHT algorithm. S, A, N, Z, AS and M are the number of shifts, assignments (copy), negations, zeros, additions/subtractions and

	Anguh	Martin	At work	
	TSHT			FSHT
%	5.855751	6.642914	45.110512	0
S	300727	307233	495699	260096
A	6262	7638	85582	0
N	12932	14937	85178	0
Z	5260	6186	113572	0
AS	2202034	2181602	1203634	2352152
M	129497	129566	123120	65024
T	2656712	2647162	2106785	2678272
t/s	6s	5s	3s	8s

Table 1: Experimental results

multiplications required by each method. The overall running time of the TSHT method includes the time required for performing logical operations and performing the transform. Note that the performance of the TSHT method is better than that of the FSH

6 Conclusions

The Truncation Slant Haar Transform (TSHT) method for computing the Slant Haar transform is presented. The method incorporates a hierarchical data structures with the Fast Slant Haar Transform (FSHT) method to accelerate the computation of the Slant Haar transform of digital images. The TSHT 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 FSHT method in the absence of image coherence. In conclusion, the TSHT method is shown to be superior to the FSHT method on digital images both in theory and practice due to the presence of coherence.

7 Acknowledgement

We wish to thank FAPEMA for supporting this research.

References

[1] M. M. Anguh & R. R. Martin, An In-place Truncation Walsh Transform for Image Processing, *Actes du 1er Colloque Africaine sur la Recherche en Informatique*, Volume I, 457-468, Cameroon, October 1992.

[2] M. M. Anguh & R.R. Martin, A Truncation Method For Computing Walsh Transforms With Applications To Image Processing, *Computer Vision, Graphics & Image Processing (CVGIP): Graphical Models & Image Processing*, **55**(6), 482-493, 1993.

[3] M. M. Anguh & R. R. Martin, A Truncation method for Computing Slant Transforms with Applications in Image Processing, In print, IEEE Trans. on Communications, 1994.

[4] D. F. Elliot & K. R. Roa, *Fast Transforms Algorithms, Analysis and Applications*, Academic Press, 1982.

[5] B. J. Fino & V. R. Algazi, Slant Haar Transforms, *IEEE Proceedings*, 653-654, 1974.

[6] B. J. Fino & V. R. Algazi, A unified Treatment of Discrete Fast Unitary Transforms, *Siam J. Comput.*, **6**(4), 700-717, 1977.

[7] R. C. Gonzalez & R. E. Woods, *Digital Image Processing*, Addison-Weslsy, 1991.

[8] R. R. Martin & M. M. Anguh, Transforms and Image Coding, *Computer Graphics Forum*, **10**, 91-96, 1991.

[9] M. A. Oliver & N. E. Wiseman, Operations on Quadtree Encoded Images, *The Computer Journal*, **26**(1), 83-91,1983.

[10] W. K. Pratt, W. H. Chang & L. R. Welch, Slant Transforms for Image Coding, *Proc. 1972 Symp. on Applications of Walsh Functions*, AD-744650, 229-234, 1972.

[11] W. K. Pratt, W. H. Cheng & L. R. Welch, Slant Transform Image Coding, *IEEE Trans. on Comm*, **COM-22**(8), 1075-1093, 1974.

[12] H. Samet, Data Structures for Quadtree Approximation and Compression *Communication of the ACM*, **28**(9) 973-993, 1975.

[13] H. Samet, *Applications of Spatial Data Structures*, Addison Wesley, 1990.

[14] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley, 1990.