The Determination of 3D Multiwavelet Transform

W. A. Mahmoud* **M. S. AbdulWahab**** **H. N. AL-Taai****

Received on :23/2/2005
Accepted on :26/5/2005

Abstract

The relatively new field of multiwavelets shows promise in removing some of the limitations of wavelets. Multiwavelets offer more design options and hence can combine all desirable transform features. In this paper several new algorithms for computing advance transforms are proposed. Firstly a fast procedure for computing of 1-D and 2-D multiwavelet transforms is introduced. Secondly, for the first time, a complete new procedure for computing of 3-D multiwavelet transforms is given. Thirdly, the inverse procedures of all the above transform for multi-dimensional cases are verified. In addition to the mathematical prove, all these new algorithms were verified also using illustrated example.

الخلاصة

يقدم متعدد المويجات (multiwavelets) امكانية اداء فائق لتطبيقات معالجة الصور وذلك قياسا بالمويجات القياسية، ومع ذلك تبقى هنالك مساحات لبحوث متعدد المويجات التي تتطلب دراسات اخرى لغرض البلوغ الى تطبيقات افضل، ولعل من تلك المساحات البحثية محاولة ايجاد نظام حل مبسط الذي من شأنه تسهيل حساب معاملات متعدد المويجات المنفصلة. لذا فان هذا البحث يقدم اولا، طريقة طريقة سريعة في حساب تحويل متعدد المويجات المنفصلة بالنسبة للاشارات ذات البعد الواحد والبعدين. ثانيا، ولاول مرة، يتم تقديم طريقة جديدة لحساب تحويل متعدد المويجات المنفصلة بالنسبة للاشارات ثلاثية الابعاد. ثالثا، الاجراءات المعكوسة (اعادة بناء) لجميع التحويلات اعلاه تم تحقيقها. بالاضافة الى برهنتها رياضيا، فان جميع هذه الخوارزميات الجديدة حققت باستخدام امثلة. تكمن اهمية الطرق المقترحة في تبسيطها عملية حساب معاملات متعدد المويجات لعمليتي التحويل واعادة البناء وبالاضافة الى استرجاع الصورة المثالي بعد عملية اعادة البناء والذي يتم توضيحه من خلال الفحوصات على احدى الصور كمثال تطبيقي على الطرق.

Keywords

3-D Discrete Multiwavelet Transform (DMWT), 3-D Inverse Discrete Multiwavelet Transform (IDMWT), Oversampled Scheme of Preprocessing.

1-Introduction

The fundamental idea behind wavelets is to analyze the signal at different scales or resolutions, which is called multiresolution analysis. Wavelets are a class of functions used to localize a given signal in both space and scaling domains. A family of wavelets can be constructed from a mother wavelet. Compared to Windowed analysis, a mother wavelet is stretched or compressed to change the size of the windows. The wavelet transform is suited for nonstationary signals, such as very brief signals and signals with interesting components at different scales.

For best performance in some applications, wavelet transform require filters that combine a number of desirable properties, such as orthogonality and symmetry. The relatively new field of multiwavelets shows promise in removing some of the

* Dept.of Electrical Eng., University of Baghdad, Baghdad-IRAQ.
** Dept.of Electrical Eng., University of Technology, Baghdad-IRAQ.

limitations of wavelets. Multiwavelets offer more design options and hence can combine all desirable transform features.

Over the past decade, the success of wavelets in solving many difficult problems has contributed to its unprecedented popularity among many research and development communities, ranging from mathematics and computer science to physics and engineering. Multiwavelets, as an extension to wavelets with only one basis, have also generated significant interest as they promise the potential to construct better multifiliters with desirable properties and lower computation complexity. The challenge still remains as what constitutes good multiwavelets and how they can be constructed and applied easily.

Multiwavelets are new addition to the body of wavelet theory. The study of multiwavelets was initiated by Goodman, Lee and Tang in [1]. Then Goodman and Lee in [2] discovered the characterization of scaling functions wavelets. In [3], Jia constructed a class of continuous orthogonal double wavelets with symmetry, short support, and orthogonality. The special case of [3] with multiplicity 2 and support [0,2], was studied by Chui and Lian [4]. Generally, after the presentation of prefilter technique, multiwavelets with multiplicity 2 can be applied in image compression application successfully [5-8].

2-Preliminaries of Multiwavelets

The wavelet transform is a type of signal transform that is commonly used in image compression. A newer alternative to the wavelet transform is the multiwavelet transform. Multiwavelets are very similar to wavelets but have some important differences. In particular, whereas wavelets have an associated scaling function $\phi(t)$ and wavelet function $\psi(t)$, multiwavelets have two or more scaling and wavelet functions. For notational convenience, the set of scaling functions can be written using the vector notation $\Phi(t) = [\phi_1(t) \;\; \phi_2(t) \;\; \cdots \;\; \phi_r(t)]^T$, where $\Phi(t)$ is called the multiscaling function. Likewise, the multiwavelet function is defined from the set of wavelet functions as $\Psi(t) = [\psi_1(t) \;\; \psi_2(t) \;\; \cdots \;\; \psi_r(t)]^T$. When n = 1, $\Psi(t)$ is called a scalar wavelet, or simply wavelet. While in principle n can be arbitrarily large, the multiwavelets studied to date are primarily for n=2.
The multiwavelet two-scale equations resemble those for scalar wavelets [9]:

$$\Phi(t) = \sqrt{2} \sum_{k=-\infty}^{\infty} H_k \Phi(2t-k) \quad \ldots (1)$$

$$\Psi(t) = \sqrt{2} \sum_{k=-\infty}^{\infty} G_k \Phi(2t-k) \quad \ldots (2)$$

However, that H_k and G_k are *matrix* filters, i.e H_k and G_k are nxn matrices instead of scalars. The matrix elements in these filters provide more degrees of freedom than a traditional scalar wavelet. These extra degrees of freedom can be used to incorporate useful properties into the multiwavelet filters, such as orthogonality, symmetry, and high order of approximation. The key, then, is to figure out how to make the best use of these extra degrees of freedom. Multifilter construction methods are already being developed to exploit them. However, the multi-channel nature of multiwavelets also means that the subband structure resulting from passing a signal through a multifilter bank is different. Sufficiently different, in fact, so that established quantization methods do not perform as

well with multiwavelets as they do with wavelets [10]. A very important multiwavelet filter is the GHM filter proposed by Geronimo, Hardian, and Massopust [11]. The GHM basis offers a combination of orthogonality, symmetry, and compact support, which cannot be achieved by any scalar wavelet basis. Where H_k for GHM system are four scaling matrices H_0, H_1, H_2, and H_3 [12]:

$$H_0=\begin{bmatrix}\frac{3}{5\sqrt{2}} & \frac{4}{5}\\ -\frac{1}{20} & -\frac{3}{10\sqrt{2}}\end{bmatrix},\ H_1=\begin{bmatrix}\frac{3}{5\sqrt{2}} & 0\\ \frac{9}{20} & \frac{1}{\sqrt{2}}\end{bmatrix},\ H_2=\begin{bmatrix}0 & 0\\ \frac{9}{20} & -\frac{3}{10\sqrt{2}}\end{bmatrix},\ H_3=\begin{bmatrix}0 & 0\\ -\frac{1}{20} & 0\end{bmatrix} \quad \dots (3)$$

also, G_k for GHM system are four wavelet matrices G_0, G_1, G_2, and G_3:

$$G_0=\begin{bmatrix}-\frac{1}{20} & -\frac{3}{10\sqrt{2}}\\ \frac{1}{10\sqrt{2}} & \frac{3}{10}\end{bmatrix},G_1=\begin{bmatrix}\frac{9}{20} & -\frac{1}{\sqrt{2}}\\ -\frac{9}{10\sqrt{2}} & 0\end{bmatrix},\ G_2=\begin{bmatrix}\frac{9}{20} & -\frac{3}{10\sqrt{2}}\\ \frac{9}{10\sqrt{2}} & -\frac{3}{10}\end{bmatrix},G_3=\begin{bmatrix}-\frac{1}{20} & 0\\ -\frac{1}{10\sqrt{2}} & 0\end{bmatrix} \quad \dots (4)$$

The low pass filter H_k and high pass filter G_k consist of coefficients corresponding to the dilation equation (1) and wavelet equation (2). However in the multiwavelet setting these coefficients are *n by n* matrices, and during the convolution step they must multiply vectors (instead of scalars). This means that multifilter banks need *n* input rows. The most obvious way to get input rows from a given signal is to repeat the signal. Two identical rows go into the multifilter bank. This procedure is called "Repeated row" which introduces oversampling of the data by a factor of two.

3-Discrete Multiwavelet Transform Computation for 1-D and 2-D Signals

By using an over-sampled scheme of preprocessing (repeated row), the discrete multiwavelet transform (DMWT) matrix is doubled in dimension compared with that of the input, which should be a square matrix NxN where N must be power of two. Transformation matrix dimension equal input signal dimension after preprocessing. To compute a single-level 1-D discrete multiwavelet transform, the next steps should be followed:

1. Checking input dimensions: Input vector should be of length N, where N must be power of two.

2. Constructing a transformation matrix: using GHM low and high pass filters matrices given in (3) and (4), the transformation matrix can be written as follows:

$$W=\begin{bmatrix}H_0 & H_1 & H_2 & H_3 & 0 & 0 & \cdots\\ G_0 & G_1 & G_2 & G_3 & 0 & 0 & \cdots\\ 0 & 0 & H_0 & H_1 & H_2 & H_3 & \cdots\\ 0 & 0 & G_0 & G_1 & G_2 & G_3 & \cdots\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots\end{bmatrix} \quad \dots (5)$$

After substituting GHM matrix filter coefficients values as given by (5), a 2Nx2N transformation matrix results.

3. Preprocessing the input signal by repeating the input stream with the same stream multiplied by a constant α. For GHM system functions $\alpha = 1/\sqrt{2}$.

4. Transformation of input vector which can be done as follows:

a. Apply matrix multiplication to the 2Nx2N constructed

transformation matrix by the 2N preprocessing input vector.

b. Permute the resulting 2Nx1 matrix rows by arranging the row pairs 1,2 and 5,6 ….,2N-3,2N-2 after each other at the upper half of the resulting matrix rows, then the row pairs 3,4 and 7,8,…, 2N-1,2N below them at the next half.

Finally, a 2Nx1 DMWT matrix results from the Nx1 original matrix using repeated row.

To compute a single-level 2-D discrete multiwavelet transform, the next steps should be followed [13]:

1. Checking input dimensions: Input matrix should be a square matrix NxN matrix, where N must be power of two.

2. Constructing a transformation matrix W using GHM low and high pass filters matrices given in (3) and (4).

3. Preprocessing rows: doubles the number of the input matrix rows. So if the 2-D input is NxN matrix elements, after row preprocessing the result is 2NxN matrix. The odd rows of the resultant matrix are the same original matrix rows values. While the even rows are the original signal rows values multiplied by α. For GHM system functions $\alpha = 1/\sqrt{2}$.

4. Transformation of input rows: can be done as follows:

a. Apply matrix multiplication to the 2Nx2N constructed transformation matrix by the 2NxN preprocessing input matrix.

b. Permute the resulting 2NxN matrix rows by arranging the row pairs 1,2 and 5,6 ….,2N-3,2N-2 after each other at the upper half of the resulting matrix rows, then the row pairs 3,4 and 7,8,…, 2N-1,2N below them at the next half.

5. Preprocess columns: to repeat the same procedure used in preprocessing rows

a. Transpose the row transformed 2NxN matrix resulting from step 4.

b. Repeat step 3 to the Nx2N matrix (transpose of the row transformed 2NxN matrix) which results in 2Nx2N column preprocessed matrix.

6. Transformation of input columns: is applied next to the 2Nx2N column preprocessed matrix as follows:

a. Apply matrix multiplication to the 2Nx2N constructed transformation matrix by the 2Nx2N column preprocessed matrix.

b. Permute the resulting 2Nx2N matrix rows.

7. To get the final transformed matrix:

a. Transpose the resulting matrix from column transformation step.

b. Apply coefficients permutation to the resulting transpose matrix. Coefficient permutation is applied to each of the basic four subbands of the resulting transpose matrix so that each subband permute rows then permute columns.

Finally, a 2Nx2N DMWT matrix results from the NxN original matrix using repeated row.

A general example for computing 1-D and 2-D DMWT using an over sampled scheme of preprocessing involves the following steps:

1. Let 8-component vector be the input 1-D signal, $X = [x_0 \;\; x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5 \;\; x_6 \;\; x_7]$

2. For an 8x1 input 1-D signal, X, construct a 8x8 transformation matrix, W, using GHM low and high pass filters,

$$W = \begin{bmatrix} H_0 & H_1 & H_2 & H_3 & 0 & 0 & 0 & 0 \\ G_0 & G_1 & G_2 & G_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & H_0 & H_1 & H_2 & H_3 & 0 & 0 \\ 0 & 0 & G_0 & G_1 & G_2 & G_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_0 & H_1 & H_2 & H_3 \\ 0 & 0 & 0 & 0 & G_0 & G_1 & G_2 & G_3 \\ H_2 & H_3 & 0 & 0 & 0 & 0 & H_0 & H_1 \\ G_2 & G_3 & 0 & 0 & 0 & 0 & G_0 & G_1 \end{bmatrix}$$

As GHM filters, *H*'s and *G*'s are 2x2 matrices, the transformation matrix, *W*, dimension after substituting filters coefficients value will be 16x16 matrix with same dimension of the input matrix after repeated-row preprocessing.

3. Apply repeated row preprocessing to the input X, which results in P matrix.

$$P=[x_0 \quad \alpha x_0 \quad x_1 \quad \alpha x_1 \quad x_2 \quad \alpha x_2 \quad x_3 \quad \alpha x_3 \quad x_4 \quad \alpha x_4 \quad x_5 \quad \alpha x_5 \quad x_6 \quad \alpha x_6 \quad x_7 \quad \alpha x_7]$$

4. Transformation of input vector which can be done as follows:

a. Let $[Z]=[W]\times[P]^T$
$Z=[z_0 \quad z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6 \quad z_7 \quad z_8 \quad z_9 \quad z_{10} \quad z_{11} \quad z_{12} \quad z_{13} \quad z_{14} \quad z_{15}]$

b. Permute *Z* which results in A matrix,
$A=[z_0 \quad z_1 \quad z_4 \quad z_5 \quad z_8 \quad z_9 \quad z_{12} \quad z_{13} \quad z_2 \quad z_3 \quad z_6 \quad z_7 \quad z_{10} \quad z_{11} \quad z_{14} \quad z_{15}]$

numerical example:

$$X=\begin{bmatrix}5\\16\\3\\7\end{bmatrix}\Rightarrow P=\begin{bmatrix}5\\3.5355\\16\\11.3137\\3\\2.1213\\7\\4.9497\end{bmatrix}\Rightarrow Z=W.P^t=\begin{bmatrix}11.738\\14.75\\-1.25\\-7.9903\\5.9397\\6.75\\-0.25\\-2.6163\end{bmatrix}\Rightarrow A=\begin{bmatrix}11.738\\14.75\\5.9397\\6.75\\-1.25\\-7.9903\\-0.25\\-2.6163\end{bmatrix}$$

For a 2-D signal the following steps will be involved:

1. Let 4x4 matrix be the input 2-D signal,

$$X=\begin{bmatrix}x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3}\\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3}\\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3}\\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3}\end{bmatrix}_{N\times N}$$

2. For an 4x4 matrix input 2-D signal, *X*, construct a 4x4 transformation matrix, *W*, using GHM low and high pass filters,

$$W=\begin{bmatrix}H_0 & H_1 & H_2 & H_3\\ G_0 & G_1 & G_2 & G_3\\ H_2 & H_3 & H_0 & H_1\\ G_2 & G_3 & G_0 & G_1\end{bmatrix}_{2N\times 2N}$$

As GHM filters, *H*'s and *G*'s are 2x2 matrices, the transformation matrix, *W*, dimension after substituting filters coefficients value will be 8x8 matrix with same dimension of the input matrix after repeated-row preprocessing.

3. Apply row preprocessing to the input matrix X, using repeated row preprocessing which results in PR matrix.

$$X=\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}_{N\times N} \Rightarrow PR=\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ \alpha x_{0,0} & \alpha x_{0,1} & \alpha x_{0,2} & \alpha x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ \alpha x_{1,0} & \alpha x_{1,1} & \alpha x_{1,2} & \alpha x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ \alpha x_{2,0} & \alpha x_{2,1} & \alpha x_{2,2} & \alpha x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \\ \alpha x_{3,0} & \alpha x_{3,1} & \alpha x_{3,2} & \alpha x_{3,3} \end{bmatrix}_{2N\times N}$$

4. Apply row transformation

a. Let $[Z]=[W]\times[PR]$

b. Permute Z which results in P matrix,

$$Z=\begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & z_{0,3} \\ z_{1,0} & z_{1,1} & z_{1,2} & z_{1,3} \\ z_{2,0} & z_{2,1} & z_{2,2} & z_{2,3} \\ z_{3,0} & z_{3,1} & z_{3,2} & z_{3,3} \\ z_{4,0} & z_{4,1} & z_{4,2} & z_{4,3} \\ z_{5,0} & z_{5,1} & z_{5,2} & z_{5,3} \\ z_{6,0} & z_{6,1} & z_{6,2} & z_{6,3} \\ z_{7,0} & z_{7,1} & z_{7,2} & z_{7,3} \end{bmatrix}_{2N\times N} \Rightarrow P=\begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & z_{0,3} \\ z_{1,0} & z_{1,1} & z_{1,2} & z_{1,3} \\ z_{4,0} & z_{4,1} & z_{4,2} & z_{4,3} \\ z_{5,0} & z_{5,1} & z_{5,2} & z_{5,3} \\ z_{2,0} & z_{2,1} & z_{2,2} & z_{2,3} \\ z_{3,0} & z_{3,1} & z_{3,2} & z_{3,3} \\ z_{6,0} & z_{6,1} & z_{6,2} & z_{6,3} \\ z_{7,0} & z_{7,1} & z_{7,2} & z_{7,3} \end{bmatrix}_{2N\times N}$$

5. Apply columns preprocessing

a. Transpose [P] matrix,

$$[P]^t=\begin{bmatrix} z_{0,0} & z_{1,0} & z_{2,0} & z_{3,0} & z_{4,0} & z_{5,0} & z_{6,0} & z_{7,0} \\ z_{0,1} & z_{1,1} & z_{2,1} & z_{3,1} & z_{4,1} & z_{5,1} & z_{6,1} & z_{7,1} \\ z_{0,2} & z_{1,2} & z_{2,2} & z_{3,2} & z_{4,2} & z_{5,2} & z_{6,2} & z_{7,2} \\ z_{0,3} & z_{1,3} & z_{2,3} & z_{3,3} & z_{4,3} & z_{5,3} & z_{6,3} & z_{7,3} \end{bmatrix}_{N\times 2N}$$

b. Preprocess $[P]^t$ which results in 2Nx2N column preprocessed matrix $[PC]$.

$$PC=\begin{bmatrix} z_{0,0} & z_{1,0} & z_{2,0} & z_{3,0} & z_{4,0} & z_{5,0} & z_{6,0} & z_{7,0} \\ \alpha z_{0,0} & \alpha z_{1,0} & \alpha z_{2,0} & \alpha z_{3,0} & \alpha z_{4,0} & \alpha z_{5,0} & \alpha z_{6,0} & \alpha z_{7,0} \\ z_{0,1} & z_{1,1} & z_{2,1} & z_{3,1} & z_{4,1} & z_{5,1} & z_{6,1} & z_{7,1} \\ \alpha z_{0,1} & \alpha z_{1,1} & \alpha z_{2,1} & \alpha z_{3,1} & \alpha z_{4,1} & \alpha z_{5,1} & \alpha z_{6,1} & \alpha z_{7,1} \\ z_{0,2} & z_{1,2} & z_{2,2} & z_{3,2} & z_{4,2} & z_{5,2} & z_{6,2} & z_{7,2} \\ \alpha z_{0,2} & \alpha z_{1,2} & \alpha z_{2,2} & \alpha z_{3,2} & \alpha z_{4,2} & \alpha z_{5,2} & \alpha z_{6,2} & \alpha z_{7,2} \\ z_{0,3} & z_{1,3} & z_{2,3} & z_{3,3} & z_{4,3} & z_{5,3} & z_{6,3} & z_{7,3} \\ \alpha z_{0,3} & \alpha z_{1,3} & \alpha z_{2,3} & \alpha z_{3,3} & \alpha z_{4,3} & \alpha z_{5,3} & \alpha z_{6,3} & \alpha z_{7,3} \end{bmatrix}_{2N\times 2N}$$

6. Transformation of input columns
a. Let $[B]=[W]\times[PR]$

b. Permute the resulting 2Nx2N matrix to get [*BB*] matrix.

$$B=\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} & b_{0,6} & b_{0,7} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} \\ b_{4,0} & b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} \\ b_{5,0} & b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} \\ b_{6,0} & b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} \\ b_{7,0} & b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} \end{bmatrix} \Rightarrow BB=\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} & b_{0,6} & b_{0,7} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} \\ b_{4,0} & b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} \\ b_{5,0} & b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} \\ b_{6,0} & b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} \\ b_{7,0} & b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} \end{bmatrix}$$

7. To get the final transformed matrix:
a. Transpose [*BB*] to get [*Y*].

b. Apply coefficients permutation to the [*Y*] matrix to get [*YY*] matrix.

$$Y=\left[\begin{array}{cccc:cccc} y_{0,0} & y_{0,1} & y_{0,2} & y_{0,3} & y_{0,4} & y_{0,5} & y_{0,6} & y_{0,7} \\ y_{1,0} & y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} & y_{1,5} & y_{1,6} & y_{1,7} \\ y_{2,0} & y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} & y_{2,5} & y_{2,6} & y_{2,7} \\ y_{3,0} & y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} & y_{3,5} & y_{3,6} & y_{3,7} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{4,0} & y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} & y_{4,5} & y_{4,6} & y_{4,7} \\ y_{5,0} & y_{5,1} & y_{5,2} & y_{5,3} & y_{5,4} & y_{5,5} & y_{5,6} & y_{5,7} \\ y_{6,0} & y_{6,1} & y_{6,2} & y_{6,3} & y_{6,4} & y_{6,5} & y_{6,6} & y_{6,7} \\ y_{7,0} & y_{7,1} & y_{7,2} & y_{7,3} & y_{7,4} & y_{7,5} & y_{7,6} & y_{7,7} \end{array}\right] \Rightarrow YY=\left[\begin{array}{cccc:cccc} y_{0,0} & y_{0,2} & y_{0,1} & y_{0,3} & y_{0,4} & y_{0,6} & y_{0,5} & y_{0,7} \\ y_{2,0} & y_{2,2} & y_{2,1} & y_{2,3} & y_{2,4} & y_{2,6} & y_{2,5} & y_{2,7} \\ y_{1,0} & y_{1,2} & y_{1,1} & y_{1,3} & y_{1,4} & y_{1,6} & y_{1,5} & y_{1,7} \\ y_{3,0} & y_{3,2} & y_{3,1} & y_{3,3} & y_{3,4} & y_{3,6} & y_{3,5} & y_{3,7} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{4,0} & y_{4,2} & y_{4,1} & y_{4,3} & y_{4,4} & y_{4,6} & y_{4,5} & y_{4,7} \\ y_{6,0} & y_{6,2} & y_{6,1} & y_{6,3} & y_{6,4} & y_{6,6} & y_{6,5} & y_{6,7} \\ y_{5,0} & y_{5,2} & y_{5,1} & y_{5,3} & y_{5,4} & y_{5,6} & y_{5,5} & y_{5,7} \\ y_{7,0} & y_{7,2} & y_{7,1} & y_{7,3} & y_{7,4} & y_{7,6} & y_{7,5} & y_{7,7} \end{array}\right]$$

So, [Y] is the final single-level 2-D DMWT matrix.

Numerical example:

$$X=\begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix} \longrightarrow PR=\begin{bmatrix} 16 & 2 & 3 & 13 \\ 11.3137 & 1.414 & 2.1213 & 9.1924 \\ 5 & 11 & 10 & 8 \\ 3.5355 & 7.778 & 7.071 & 5.6569 \\ 9 & 7 & 6 & 12 \\ 6.364 & 4.9497 & 4.2426 & 8.4853 \\ 4 & 14 & 15 & 1 \\ 2.8284 & 9.8995 & 10.6066 & 0.7071 \end{bmatrix}$$

$$Z=\begin{bmatrix} 17.9605 & 6.6468 & 7.2125 & 16.2635 \\ 4.0500 & 11.4500 & 9.9500 & 8.5500 \\ -0.9500 & 0.4500 & -0.0500 & 0.5500 \\ 4.8790 & -4.4548 & -4.0305 & 3.6062 \\ 10.6066 & 12.8693 & 12.3037 & 12.3037 \\ 6.5500 & 11.9500 & 13.4500 & 2.0500 \\ 2.5500 & -2.0500 & -1.5500 & 1.0500 \\ 6.4347 & -6.8589 & -7.2832 & 7.7075 \end{bmatrix} \Rightarrow P=\begin{bmatrix} 17.9605 & 6.6468 & 7.2125 & 16.2635 \\ 4.0500 & 11.4500 & 9.9500 & 8.5500 \\ 10.6066 & 12.8693 & 12.3037 & 12.3037 \\ 6.5500 & 11.9500 & 13.4500 & 2.0500 \\ -0.9500 & 0.4500 & -0.0500 & 0.5500 \\ -0.9500 & -4.4548 & -4.0305 & 3.6062 \\ 2.5500 & -2.0500 & -1.5500 & 1.0500 \\ 6.4347 & -6.8589 & -7.2832 & 7.7075 \end{bmatrix}$$

$$[P]^t = \begin{bmatrix} 17.9605 & 4.0500 & 10.6066 & 6.5500 & -0.9500 & 4.8790 & 2.5500 & 6.4347 \\ 6.6468 & 11.4500 & 12.8693 & 11.9500 & 0.4500 & -4.4548 & -2.0500 & -6.8589 \\ 7.2125 & 9.9500 & 12.3037 & 13.4500 & -0.0500 & -4.0305 & -1.5500 & -7.2832 \\ 16.2635 & 8.5500 & 12.3037 & 2.0500 & 0.5500 & 3.6062 & 1.0500 & 7.7075 \end{bmatrix}$$

$$PC = \begin{bmatrix} 17.9605 & 4.0500 & 10.6066 & 6.5500 & -0.9500 & 4.8790 & 2.5500 & 6.4347 \\ 12.7000 & 2.8638 & 7.5000 & 4.6315 & -0.6718 & 3.4500 & 1.8031 & 4.5500 \\ 6.6468 & 11.4500 & 12.8693 & 11.9500 & 0.4500 & -4.4548 & -2.0500 & -6.8589 \\ 4.7000 & 8.0964 & 9.1000 & 8.4499 & 0.3182 & -3.1500 & -1.4496 & -4.8500 \\ 7.2125 & 9.9500 & 12.3037 & 13.4500 & -0.0500 & -4.0305 & -1.5500 & -7.2832 \\ 5.1000 & 7.0357 & 8.7000 & 9.5106 & -0.0354 & -2.8500 & -1.0960 & -5.1500 \\ 16.2635 & 8.5500 & 12.3037 & 2.0500 & 0.5500 & 3.6062 & 1.0500 & 7.7075 \\ 11.5000 & 6.0458 & 8.7000 & 1.4496 & 0.3889 & 2.5500 & 0.7425 & 5.4500 \end{bmatrix}$$

$$B = \begin{bmatrix} 20.6000 & 8.8671 & 15.9600 & 11.5541 & -0.7495 & 2.9400 & 1.6546 & 3.4600 \\ 4.0729 & 12.6250 & 13.1805 & 13.9750 & 0.5750 & -6.5973 & -2.9750 & -10.3733 \\ -2.5739 & 1.1750 & 0.3111 & 2.0250 & 0.1250 & -2.1425 & -0.9250 & -3.5143 \\ 2.7600 & -2.5244 & -0.8400 & -0.1909 & -0.6152 & 2.2500 & 1.2940 & 2.5500 \\ 14.0400 & 13.4775 & 17.4000 & 14.1846 & 0.1838 & -2.4600 & -1.0889 & -3.9400 \\ 19.0636 & 6.7750 & 11.7663 & 0.6250 & 0.2250 & 5.9185 & 2.1750 & 11.0521 \\ 2.8001 & -1.7750 & -0.5374 & -1.4250 & -0.3250 & 2.3122 & 1.1250 & 3.3446 \\ -1.1600 & -1.7183 & -0.7600 & 4.4336 & -0.7990 & -1.0500 & 0.1202 & -3.7500 \end{bmatrix}$$

$$BB = \begin{bmatrix} 20.6000 & 8.8671 & 15.9600 & 11.5541 & -0.7495 & 2.9400 & 1.6546 & 3.4600 \\ 4.0729 & 12.6250 & 13.1805 & 13.9750 & 0.5750 & -6.5973 & -2.9750 & -10.3733 \\ 14.0400 & 13.4775 & 17.4000 & 14.1846 & 0.1838 & -2.4600 & -1.0889 & -3.9400 \\ 19.0636 & 6.7750 & 11.7663 & 0.6250 & 0.2250 & 5.9185 & 2.1750 & 11.0521 \\ -2.5739 & 1.1750 & 0.3111 & 2.0250 & 0.1250 & -2.1425 & -0.9250 & -3.5143 \\ 2.7600 & -2.5244 & -0.8400 & -0.1909 & -0.6152 & 2.2500 & 1.2940 & 2.5500 \\ 2.8001 & -1.7750 & -0.5374 & -1.4250 & -0.3250 & 2.3122 & 1.1250 & 3.3446 \\ -1.1600 & -1.7183 & -0.7600 & 4.4336 & -0.7990 & -1.0500 & 0.1202 & -3.7500 \end{bmatrix}$$

$$Y = \left[\begin{array}{cccc|cccc} 20.6000 & 4.0729 & 14.0400 & 19.0636 & -2.5739 & 2.7600 & 2.8001 & -1.1600 \\ 8.8671 & 12.6250 & 13.4775 & 6.7750 & 1.1750 & -2.5244 & -1.7750 & -1.7183 \\ 15.9600 & 13.1805 & 17.4000 & 11.7663 & 0.3111 & -0.8400 & -0.5374 & -0.7600 \\ 11.5541 & 13.9750 & 14.1846 & 0.6250 & 2.0250 & -0.1909 & -1.4250 & 4.4336 \\ \hline -0.7495 & 0.5750 & 0.1838 & 0.2250 & 0.1250 & -0.6152 & -0.3250 & -0.7990 \\ 2.9400 & -6.5973 & -2.4600 & 5.9185 & -2.1425 & 2.2500 & 2.3122 & -1.0500 \\ 1.6546 & -2.9750 & -1.0889 & 2.1750 & -0.9250 & 1.2940 & 1.1250 & 0.1202 \\ 3.4600 & -10.3733 & -3.9400 & 11.0521 & -3.5143 & 2.5500 & 3.3446 & -3.75000 \end{array}\right]$$

$$YY = \begin{bmatrix} 20.6000 & 14.0400 & 4.0729 & 19.0636 & -2.5739 & 2.8001 & 2.7600 & -1.1600 \\ 15.9600 & 17.4000 & 13.1805 & 11.7663 & 0.3111 & -0.5374 & -0.8400 & -0.7600 \\ 8.8671 & 13.4775 & 12.6250 & 6.7750 & 1.1750 & -1.7750 & -2.5244 & -1.7183 \\ 11.5541 & 14.1846 & 13.9750 & 0.6250 & 2.0250 & -1.4250 & -0.1909 & 4.4336 \\ -0.7495 & 0.1838 & 0.5750 & 0.2250 & 0.1250 & -0.3250 & -0.6152 & -0.7990 \\ 1.6546 & -1.0889 & -2.9750 & 2.1750 & -0.9250 & 1.1250 & 1.2940 & 0.1202 \\ 2.9400 & -2.4600 & -6.5973 & 5.9185 & -2.1425 & 2.3122 & 2.2500 & -1.0500 \\ 3.4600 & -3.9400 & -10.3733 & 11.0521 & -3.5143 & 3.3446 & 2.5500 & -3.7500 \end{bmatrix}$$

A general computer program computing a single-level DMWT using an over-sampled scheme of preprocessing (repeated row preprocessing) is written using MATLAB v.6.5 for a general NxN 2-D signal (or image). As shown in Fig.(1-a), the original "Lena" image dimensions is 512x512 (NxN). After a single-level of multiwavelets decomposition using an over-sampled scheme of preprocessing, image dimensions will be a matrix of

1024x1024 (2Nx2N) as shown in Fig. (1-b).The upper-left most, L1L1, subband of 256x256 dimension, is zoomed in as in Fig. (1-c).

4- 3-D Discrete Multiwavelet Transformation Algorithm

This transformation is a useful one that changes information from the spatial domain into a local frequency domain. In the discrete case, the integrals can be reduced to simple arithmetic operations. This applies to a one-dimensional signal as well as to the 2-D, 3-D, or higher-dimensional case. This simple 1-D scheme can be lifted to higher dimensional cases. For a 2-D multiwavelet transformation, the algorithm is applied in x-direction first, and then in y-direction. Similarly, in 3-D multiwavelet transformation the structures are defined in 3-D and the transformation algorithm is applied in x-, y- and z-direction successively. One *cycle* for an n-dimensional data set is defined as the completion of the algorithm for all n directions.

To apply this hierarchical scheme to volume data sets (Fig 2-a), a 3-D multiwavelet transformation must be implemented. Therefore, the 2-D scheme as described in the example given above is extended into the z-direction. The size of the array must be a power of two in each dimension. Unused areas can be filled with zeros. If we apply this algorithm to data set *(ctbrain.vols)* shown in Fig. (3) [14], which consists of 512 x 512 x 231 elements, we need to scale the size of the array to the closest powers of two, i.e., 512 x 512 x 256. The algorithm is initially run in x-direction, row by row for all 231 slices. The algorithm splits the volume into two halves, the left half representing the low-frequency coefficients while the right half represents the detail coefficients, as shown in Fig. (2-b). In the second stage of the algorithm, the entire volume is then again transformed in y-direction splitting the volume into four quadrants as shown in Fig. (2-c). For the final run, the volume is transformed in z-direction splitting the volume into eight octants. The upper left front octant contains the low-frequency coefficients that are initially transmitted over the network. A general example for computing 3-D DMWT using an over sampled scheme of preprocessing involves the following steps:

1. Let NxNxM be the input 3-D signal,

$$X = \begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{matrix} \quad \begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{matrix} \quad \begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{matrix} \quad \begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{matrix}$$

2. Apply 2-D DMWT algorithm to each NxN input matrix, which result in a 2Nx2NxM Y matrix.

Y=

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} & a_{0,6} & a_{0,7} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} \\ a_{5,0} & a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} \\ a_{6,0} & a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} \\ a_{7,0} & a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} & b_{0,6} & b_{0,7} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} \\ b_{4,5} & b_{4,6} & b_{4,7} \\ b_{5,5} & b_{5,6} & b_{5,7} \\ b_{6,5} & b_{6,6} & b_{6,7} \\ b_{7,5} & b_{7,6} & b_{7,7} \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} & c_{0,5} & c_{0,6} & c_{0,7} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} & c_{1,6} & c_{1,7} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} & c_{2,6} & c_{2,7} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} & c_{3,6} & c_{3,7} \\ c_{4,5} & c_{4,6} & c_{4,7} \\ c_{5,5} & c_{5,6} & c_{5,7} \\ c_{6,5} & c_{6,6} & c_{6,7} \\ c_{7,5} & c_{7,6} & c_{7,7} \end{bmatrix} \begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} & d_{0,4} & d_{0,5} & d_{0,6} & d_{0,7} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} & d_{1,4} & d_{1,5} & d_{1,6} & d_{1,7} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} & d_{2,4} & d_{2,5} & d_{2,6} & d_{2,7} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} & d_{3,4} & d_{3,5} & d_{3,6} & d_{3,7} \\ d_{4,5} & d_{4,6} & d_{4,7} \\ d_{5,5} & d_{5,6} & d_{5,7} \\ d_{6,5} & d_{6,6} & d_{6,7} \\ d_{7,5} & d_{7,6} & d_{7,7} \end{bmatrix}$$

3. Apply 1-D DMWT algorithm to each 2Nx2N (64 element) in all M matrices in z-direction. Which can be done as follows:

a. For each i,j construct the Mx1 input vector $Y(i,j) = \left[a_{i,j} \quad b_{i,j} \quad c_{i,j} \quad d_{i,j}\right]^T_{1\times M}$ where $i, j = 0,1,2,\cdots,2N$

b. Preprocessing the input vector $Y(i,j)$ by repeating the input stream with the same stream multiplied by a constant α. For GHM system functions $\alpha = 1/\sqrt{2}$.

c. Constructing an 2Mx2M transformation matrix using GHM low and high pass filters.

d. Apply matrix multiplication to the 2Mx2M constructed transformation matrix by the 2Mx1 preprocessing input vector.

e. Permute the resulting 2Mx1 matrix rows by arranging the row

pairs 1,2 and 5,6 ….,2M-3,2M-2 after each other at the upper half of the resulting matrix rows, then the row pairs 3,4 and 7,8,…, 2M-1, 2M below them at the next half.

4. Repeat step 3 for all i, j .
5. Finally, a 2Nx2Nx2M DMW matrix result from the NxNxM original matrix using repeated row preprocessing.

A general computer program computing a single-level 3-D DMWT using an over-sampled scheme of preprocessing (repeated row preprocessing) is written using MATLAB v.6.5 for a general NxNxM 3-D signal (or image). An example test is applied to "Rubic's Cube sequence". In this image sequence a rubic's cube is rotating counterclock wise on a turn table. It consists of 256x240x4 elements as shown in Fig. (4-a). The size of array must be a power of two in each dimension. Unused area can be filled with zeros. The size of the array is scaled to the closest power of two i.e. 256x256x4. The algorithm is initial run in 2-D for all 4 frames as shown in Fig.(4-a). Then a 1-D DMWT is applied in z-direction.

As shown in Fig.(4-a), the original "Rubic's Cube" image dimensions is 256x256x4 (NxNxM). After a single-level 2-D of multiwavelets decomposition using an over-sampled scheme of preprocessing, image dimensions will be a matrix of 512x512x4 (2Nx2NxM). After a single-level 1-D of multiwavelets decomposition using an over-sampled scheme of preprocessing, image dimensions will be a matrix of 512x512x8 (2Nx2Nx2M) as shown in Fig. (4-b). The upper-left-front most subband of 256x256 dimensions is zoomed in as in Fig. (4-c).

Numerical example:

X=

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

1	2	0	1
1	1	2	0
3	0	1	1
2	1	1	0

1	1	0	1
2	1	1	0
0	2	1	1
3	0	1	2

3	2	1	5
2	1	4	1
1	0	1	1
3	1	0	1

4.8000	4.9400	2.0577	5.5932	-0.3465	0.2192	0.3100	-1.2100
2.4200	1.5800	0.1768	1.8031	-0.2475	0.3889	0.6900	0.3100
1.6476	3.9457	1.3450	-0.1800	0.8450	-0.3800	1.7324	1.5556
4.0517	0.8344	0.5950	3.0700	-0.9050	0.8700	-0.1768	-0.0707
-0.7566	-0.4384	-0.3550	-0.8800	0.1450	-0.0800	0.1768	0.2828
0.6576	0.4101	0.2950	1.2700	-0.2050	0.0700	-0.3182	-0.6364
-0.2700	-1.3400	-0.7000	1.3859	-0.5586	0.2546	-0.8300	-1.3200
-0.4300	1.1400	0.2758	1.3718	0.1344	-0.3253	-0.1700	-1.1800

2.4200	0.8400	1.0536	1.3364	-0.3606	0.3465	-0.2700	0.1700
2.1000	2.1800	1.9587	1.7466	-0.0212	-0.0919	-0.4300	-0.3700
2.1072	1.1879	1.3325	0.1575	-0.0175	0.1575	0.0884	0.9016
2.9557	1.5415	-0.6425	2.6825	-0.4925	0.6825	1.1137	0.2298
-0.2970	0.1980	0.4825	-0.1925	0.1325	-0.1925	-0.2652	-0.1591
-0.0141	-0.2970	-0.2425	0.0825	-0.0925	0.0825	-0.0177	-0.0530
-0.9800	-0.0400	0.5975	0.2086	0.1025	-0.3571	-0.8150	-0.9850
-1.2200	-0.6600	1.0996	-0.9864	0.1803	-0.4207	-1.1850	-0.5150

2.4200	1.2600	2.2062	1.0748	-0.1980	0.0849	-0.8400	0.0400
3.9600	1.8200	0.0141	1.7819	-0.4101	0.7920	1.3400	1.3600
1.7466	2.1708	0.8050	0.1050	0.3050	0.0050	1.0253	1.1667
2.1708	0.6859	1.3050	0.3550	-0.1950	0.2550	-0.2475	0.6718
0.3323	0.1909	-0.5450	0.2550	-0.0450	0.1550	0.5303	0.2475
-0.2333	-0.3041	0.4550	0.0050	-0.0450	-0.0950	-0.6010	-0.3889
0.7100	-0.6100	-0.6010	1.0960	-0.4596	0.3889	-0.1300	-0.3700
-0.0100	-0.1900	-0.0354	0.7425	-0.1768	0.0354	-0.3700	-0.6300

Y=

20.6000	14.0400	4.0729	19.0636	-2.5739	2.8001	2.7600	-1.1600
15.9600	17.4000	13.1805	11.7663	0.3111	-0.5374	-0.8400	-0.7600
8.8671	13.4775	12.6250	6.7750	1.1750	-1.7750	-2.5244	-1.7183
11.5541	14.1846	13.9750	0.6250	2.0250	-1.4250	-0.1909	4.4336
-0.7495	0.1838	0.5750	0.2250	0.1250	-0.3250	-0.6152	-0.7990
1.6546	-1.0889	-2.9750	2.1750	-0.9250	1.1250	1.2940	0.1202
2.9400	-2.4600	-6.5973	5.9185	-2.1425	2.3122	2.2500	-1.0500
3.4600	-3.9400	-10.3733	11.0521	-3.5143	3.3446	2.5500	-3.7500

```
YY(:,:,1) =

   21.4197   14.4335    4.9680   19.3280   -2.6320    2.8080    2.3759   -1.1314
   17.4797   17.9973   13.0540   12.4040    0.1340   -0.1960   -0.2630   -0.1754
    9.5190   14.2630   12.8396    6.7515    1.2926   -1.7550   -2.0640   -1.2060
   12.3590   14.3330   14.3882    0.7693    1.9219   -1.3025   -0.2940    4.6740
   -0.6010    0.2630    0.3380    0.3309    0.1047   -0.2560   -0.3840   -0.6860
    1.5390   -1.2070   -2.7521    2.1553   -0.9348    1.0734    1.0260   -0.0460
    3.2117   -2.6941   -6.7860    6.3240   -2.3160    2.4540    2.1722   -1.1964
    3.4210   -3.9810  -10.2840   11.2560   -3.5540    3.3260    2.3674   -3.9796

YY(:,:,2) =

   -1.3350   -1.6060    1.4945   -2.6704    0.2358   -0.3864   -1.4465    0.3815
    1.0790   -1.1760   -2.0439   -0.2266   -0.4458    0.8128    1.2775    1.3175
    0.4356   -0.4741   -1.4277   -1.1990    0.0072    0.4260    1.4188    1.6447
    0.4356   -1.7646   -1.7777    0.8635   -0.6927    0.6885    0.1460   -0.1761
    0.4144    0.2259   -0.4702    0.1835   -0.0353    0.1585    0.5385    0.3330
   -0.5897   -0.1807    0.9397   -0.4690    0.1247   -0.2940   -0.8192   -0.3776
```

```
   -0.1940   -0.0325    0.9627   -0.1492    0.0506   -0.2128   -0.7765   -0.3710
   -1.0460    0.3525    2.3571   -1.8696    0.5823   -0.7453   -1.2085    0.0560

YY(:,:,3) =

    4.4321    2.9274    1.9160    3.6960   -0.5040    0.4360   -0.1358   -0.3451
    3.1056    2.8284    2.0140    2.4940   -0.1260    0.0740   -0.1329   -0.2348
    2.7850    2.8500    1.8897    0.0795    0.3412   -0.0053    0.8225    1.5525
    4.6450    1.8800   -0.3836    3.9580   -0.8715    1.0448    1.0275    0.1975
   -0.6150    0.0100    0.3270   -0.5639    0.1927   -0.2245   -0.1875   -0.0375
    0.2650   -0.1200   -0.1149    0.6205   -0.1785    0.1114   -0.1525   -0.3225
   -1.0847   -0.6081    0.2945    0.7945   -0.1355   -0.2455   -1.1589   -1.5351
   -1.3902   -0.1697    1.2055   -0.3945    0.2355   -0.5545   -1.2452   -1.0105

YY(:,:,4) =

   10.1350    8.6740    2.8557   10.7116   -1.0193    0.9747    1.2185   -1.5335
    6.4690    6.1940    3.7296    4.8044   -0.1170    0.1870    0.4225    0.0725
    3.7166    7.4455    4.7585    1.8247    1.1435   -0.9252    0.8195    0.7237
    6.6157    4.7054    4.8210    2.5498   -0.1440    0.2498   -0.4356    1.1833
   -0.9009   -0.4105   -0.2340   -0.7427    0.1510   -0.1428    0.0099    0.0484
    1.1356    0.1375   -0.5865    1.8423   -0.4515    0.3923    0.1195   -0.5385
    0.7860   -1.9725   -2.7337    2.9957   -1.1710    0.9875    0.0560   -1.3535
    0.8740    0.0425   -3.0681    4.7790   -0.9539    0.7768    0.8590   -2.1115

YY(:,:,5) =

   -3.7550   -2.8660   -0.7117   -3.7452    0.4338   -0.4713   -0.6065    0.3415
   -2.8810   -2.9960   -2.0580   -2.0085   -0.0357    0.0209   -0.0625   -0.0425
   -1.3110   -2.6449   -2.2327   -1.3040   -0.2978    0.4210    0.3935    0.4780
   -1.7352   -2.4505   -3.0827    0.5085   -0.4978    0.4335    0.3935   -0.8478
    0.0820    0.0350    0.0747   -0.0715    0.0097    0.0035    0.0081    0.0856
   -0.3564    0.1234    0.4847   -0.4740    0.1697   -0.1990   -0.2181    0.0113
   -0.9040    0.5775    1.5638   -1.2452    0.5102   -0.6017   -0.6465   -0.0010
   -1.0360    0.5425    2.3925   -2.6121    0.7591   -0.7806   -0.8385    0.6860

YY(:,:,6) =

    4.9738    3.1763    0.0495    4.8795   -0.7305    0.8695    1.1787   -0.1959
    2.7139    4.5764    4.5375    2.8075    0.3575   -0.7225   -1.3216   -1.2594
    2.1740    2.6555    3.5288    1.9290    0.0711   -0.4115   -1.4515   -0.9560
    2.8540    4.1705    2.8076    0.8719    0.5519   -0.3373    0.5885    0.9290
   -0.4960    0.0455    0.7393   -0.1181    0.1100   -0.2666   -0.6365   -0.4710
    0.5640   -0.2695   -1.2548    0.5572   -0.2577    0.4087    0.7635    0.3040
   -0.0170   -0.2298   -1.1805    0.9670   -0.2305    0.2370    0.4320   -0.3861
    0.4978   -1.3541   -2.4645    2.1380   -0.8145    0.7680    0.4660   -0.7948

YY(:,:,7) =
    5.3350    3.7340    0.7980    5.1184   -0.6728    0.7555    0.9085   -0.3235
    4.0490    4.6140    3.5529    3.0013    0.1305   -0.2019   -0.2675   -0.2375
    2.0690    3.4998    3.4135    2.0047    0.2985   -0.5452   -0.9129   -0.8319
    2.5640    3.8711    4.2260   -0.5202    0.7610   -0.6202   -0.2588    1.2541
   -0.1442    0.0279    0.1210    0.1373    0.0060   -0.0628   -0.1669   -0.2344
    0.4780   -0.2726   -0.8815    0.5723   -0.2465    0.3222    0.4377    0.0979
    1.0560   -0.6325   -2.0336    1.6097   -0.6124    0.7329    0.8860   -0.0335
    1.3040   -1.0975   -3.3439    3.4072   -1.0882    1.1020    1.0290   -0.9315

YY(:,:,8) =
    6.1985    2.9614    0.5605    4.8305   -0.9595    1.1405    0.9567    0.3231
    5.5451    6.8646    6.0325    4.2125    0.3125   -0.5575   -1.0119   -0.7205
    3.1860    3.3895    4.8203    3.0261   -0.0658   -0.4670   -2.2210   -1.5465
    3.0060    5.8745    5.2764   -0.9549    1.3096   -0.9832    0.3640    1.9435
    0.0560    0.3995    0.6449    0.5830    0.0014   -0.1524   -0.4860   -0.5815
    0.2960   -0.7855   -1.5507    0.1375   -0.2850    0.4628    0.7890    0.4685
    1.0918   -0.1591   -2.1420    1.6105   -0.4920    0.6905    1.2615    0.1421
    1.3972   -2.5703   -4.2630    3.4845   -1.5130    1.5045    0.8811   -0.9412
```

5- Inverse Discrete Multiwavelet Transform Computation for 1-D and 2-D Signals

To reconstruct the original signal from the discrete multiwavelets transformed signal, the inverse discrete multiwavelets transform (IDMWT) should be used. Reconstruction matrix which is the inverse (or transpose) of the transformation matrix (5) of GHM four mulltifilters matrices (3) and (4) can be used for computing IDMWT. An over – sampled scheme of post-processing should be used in computing IDMWT

To compute a single level 1-D Inverse discrete multiwavelets transform using over-sampled scheme of post-processing, the following steps should be followed:

1. Apply shuffling by arranging the row pairs 1, 2, and 3, 4, …, N-1, N of the 2Nx1 matrix to be the row pairs 1, 2 and 5, 6, …, 2N-1, 2N-2 of the resulting matrix and arranging the row pairs N+1, N+2 and N+3, N+4, …, 2N-1, 2N of the 2Nx1 matrix to be the row pairs 3, 4, and 7, 8,…, 2N-1,2N of the resulting matrix.
2. Multiply a 2Nx2N reconstruction matrix (2Nx2N transformation matrix (5) transpose) with the resulting 2Nx1 shuffled matrix.
3. Apply postprocessing by discarding the even rows 2,4, …, 2N from the row reconstructed 2Nx1 matrix to have an Nx1 original reconstructed 1-D signal matrix.

To compute a single level 2-D Inverse discrete multiwavelets transform using over-sampled scheme of post-processing, the following steps should be followed:

1. Coefficients shuffling, which is applied to the DMWT 2Nx2N matrix four basic subbands individually? For each subband, coefficients shuffling, shuffles the columns first then shuffles the rows.
2. Column reconstruction:
   a. Transpose the postprocessed 2Nx2N resultant matrix.
   b. Apply Shuffling by arranging the row pairs 1, 2, and 3, 4, …, N-1, N of the coefficients shuffled 2Nx2N matrix transpose to be the row pairs 1, 2 and 5, 6, …, 2N-1, 2N-2 of the resulting matrix and arranging the row pairs N+1, N+2 and N+3, N+4, …, 2N-1, 2N of the 2Nx2N postprocessed resultant matrix transpose to be the row pairs 3, 4, and 7, 8,…, 2N-1,2N of the resulting matrix.
   c. Multiply a 2Nx2N reconstruction matrix (2Nx2N transformation matrix (5) transpose) with the resulting 2Nx2N shuffled matrix.
3. Apply postprocessing by discarding the even rows 2,4, …, 2N from the column reconstructed 2Nx2N matrix to have an Nx2N resultant matrix.
4. Row reconstruction
   a. Transpose the postprocessed Nx2N resultant matrix.
   b. Apply Shuffling by arranging the row pairs 1, 2, and 3, 4, …, N-1, N of the 2NxN postprocessed resultant matrix transpose to be the row pairs 1, 2 and 5, 6, …, 2N-1, 2N-2 of the resulting matrix and arranging the row pairs N+1, N+2 and N+3, N+4, …, 2N-1, 2N of the 2NxN postprocessed resultant matrix transpose to be the row pairs 3, 4, and 7, 8,…, 2N-1,2N of the resulting matrix.

c. Multiply a 2NX2N reconstruction matrix (2NX2N transformation matrix (5) transpose) with the resulting 2NXN shuffled matrix.

5. Postrocessing, an over-sampled scheme of postprocessing can be done by discarding the even rows 2, 4, …, 2N from the column reconstructed 2NXN matrix to have an NXN original reconstructed 2-D signal matrix.

A general computer program computing a single-level IDMWT using an over-sampled scheme of preprocessing (repeated row preprocessing) is written using MATLAB v.6.5 for a general NXN 2-D decomposed image.

An example test is applied to the decomposed Lena image shown in Fig. (1-b) to reconstruct the original "Lena" image by using this computer program of the method of computing inverse discrete multiwavelets transform using an over-sampled scheme of postprocessing and the result is shown in Fig. (5).

6- Inverse Discrete Multiwavelet Transform Computation for 3-D Signals

1. Apply 2-D IDMWT algorithm to each resultant matrix from 3-D DMWT.
2. Apply 1-D IDMWT algorithm to each element in all resultant matrices from 1 in z-direction.

A general computer program computing a single-level 3-D IDMWT using an over-sampled scheme of preprocessing (repeated row preprocessing) is written using MATLAB v.6.5 for a general NXNXM 3-D decomposed image.

An example test is applied to the decomposed Rubic's cube image shown in Fig. (4-b) to reconstruct the original "Rubic's cube" image by using this computer program of the method of computing inverse discrete multiwavelets transform using an over-sampled scheme of postprocessing and the result is shown in Fig. (6).

7- Conclusion

This paper presents a new 3-D multiwavelets transform computation methods that verify the potential benefits of multiwavelets and gain a much improvement in terms of low computational complexity. The general procedures with illustrated numerical examples are described in details. The verification of the new and developed methods using a computational aspect was also developed. Following are some concluding remarks obtained:

1. A single level decomposition in the multiwavelet domain is equivalent to two scalar wavelet decompositions. Thus although computation complexity is double for DMWT compared to DWT, the levels of computation are less by half to get the same image quality.
2. Multiwavelets filter banks require a vector-valued input signal. This is another issue which is addressed when multiwavelets are used in the transform process. A scalar-valued input signal must somehow be converted into a suitable vector-valued signal. This conversion is called preprocessing.
3. The most obvious way to get two input rows from a given signal is to repeat the signal using repeated row

preprocessing (over-sampled scheme).

4. Using repeated row preprocessing introduces an oversampling of data by a factor of 2 which doubles the original image dimensions. In the same time the upper-left most subband (L_1L_1) of the decomposed image, which usually the 2nd, 3rd , … level of decompositions are applied to it, has half dimensions of the original.
5. Discrete multiwavelet transform computation algorithm using repeated row preprocessing should be applied to a matrix with a size at least to 4x4.
6. Transformation matrix dimensions used in computing DMWT algorithms should equal the dimensions of the input image after preprocessing.

8- References

[1] Goodman T. N. T, Lee S. L, and Tang, "Wavelets in wandering subspaces" Trans. Amer. Math. Soc, 1993, Vol. 338, No. 1: 639-654.

[2] Goodman T. N. T and Lee S. L, "Wavelets of multiplicity r," Trans. Amer. Math. Soc, 1994, Vol. 342, No. 1: 307-324.

[3] Jia R. Q, Remeschneider S, and Zhou D. X, "Vector subdivision schemes and multiple wavelet" Elsevier Math. Comp, 1998, 67: 1533-1563.

[4] Chui C.K and Lian J, "A study on orthonormal multiwavelets," Appl. Numer. Math, 1996, Vol. 20: 273-298.

[5] Tham J. Y, Shen L, and Lee S. L, "A general approch for analysis and application of discrete multiwavelet tansforms" IEEE Trans. Signal Processing, 2000, Vol. 48,No. 2: 457-464.

[6] Strela V. and Heller P.N, "The application of multiwavelet filterbanks to image processing," IEEE Trans. Image Processing, 2000, Vol.8, No.4: .548-563, 1999.

[7] Cotronei M, Lazzaro D, Montefusco L. B, and Puccio L, "Image Compression Through Embedded Multiwavelet Transform Coding," IEEE Trans. Image Processing, 2000, Vol. 9, No. 2: 184-189.

[8] Chen J., Zhou J., Yu S., Xiao Q. and Jun Xu, "Application of Symmetric Orthogonal Multiwavelets and Prefilter Technique For Image Compression", JCS&T Vol. 3 No. 1, April 2003.

[9] Martin M. B. and Bell A. E. , "New Image Compression Techniques Using Multiwavelets and Multiwavelet Packets", IEEE Transactions on Image Processing, Vol. 10, No. 4, April 2001.

[10] Yew T. J., "Multiwavelets and Scalable Video Compression", Ph.D. Thesis, Department of Electrical and Computer Engineering, National University of Singapore, 2002.

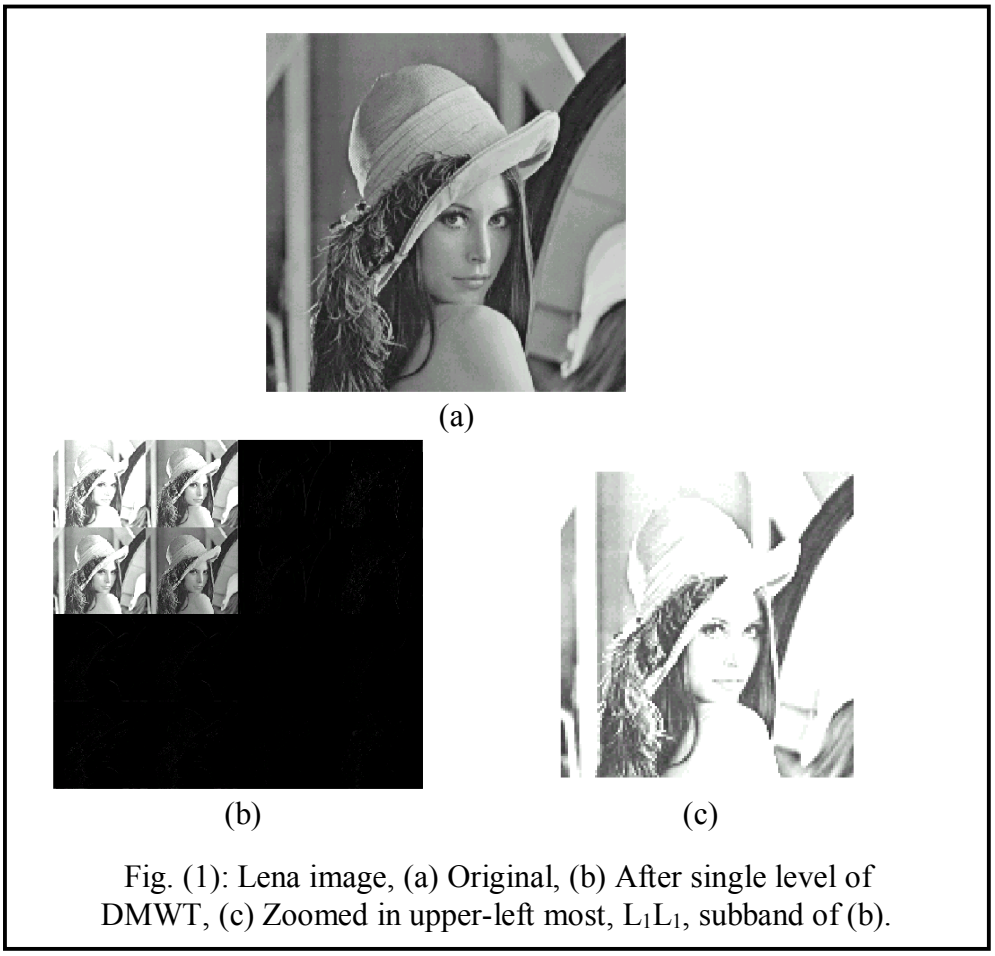
[11] Geronimo J., Hardin D., and Massopust P.R., "Fractal function and Wavelet Expansions Based on Several Function", J.Approx. Theory, Vol. 78, pp. (373-401), 1994.

[12] Strela V., and Walden A.T., "Orthogonal and Biorthogonal Multiwavelets for Signal Denoising and Image Compression", Proc. SPIE, 3391:96-107, 1998.

[13] Jalal Z., "Design and Simulation of DSP Techniques for Video Compression", Ph.D. thesis,

Baghdad University, Electrical Engineering Department, 2004.

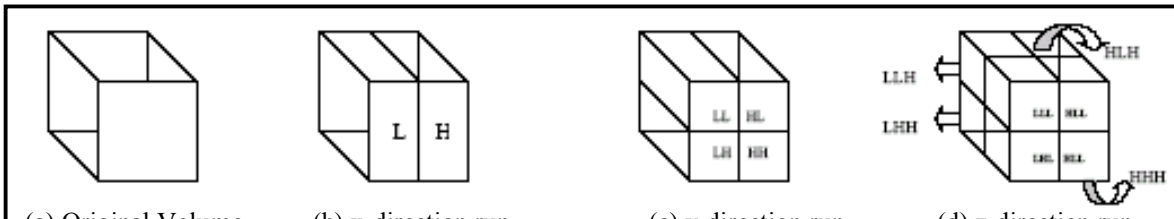
[14] Kourosh J. K. , Hamid S. Z., E. Kost, and Patel S., "Comparision of 2D and 3D wavelet features for TLE lateralization", Proceedings of SPIE, Vol. 5369,Bellingham, WA, 2004.



(a)

(b) (c)

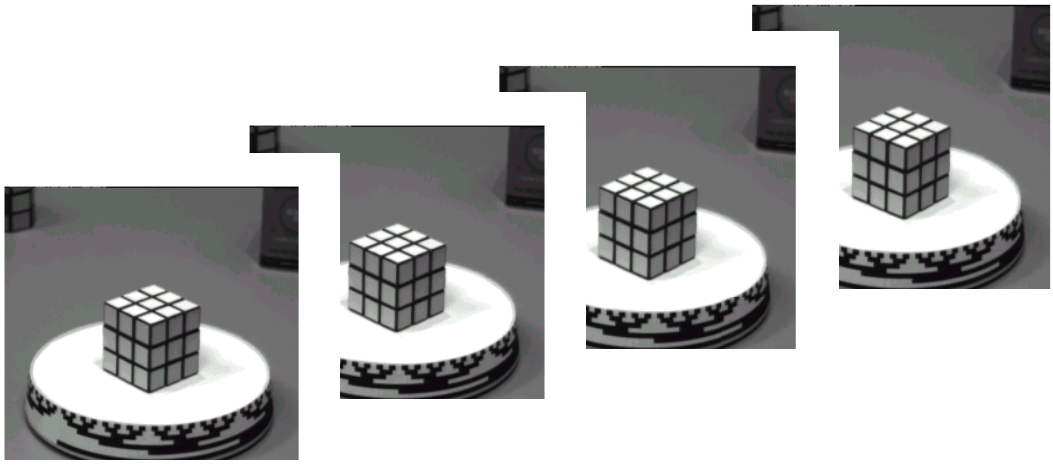
Fig. (1): Lena image, (a) Original, (b) After single level of DMWT, (c) Zoomed in upper-left most, L_1L_1, subband of (b).



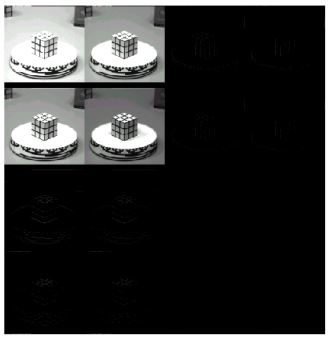
(a) Original Volume (b) x-direction run (c) y-direction run (d) z-direction run

Fig.(2): 3-D Multiwavelet Transform.

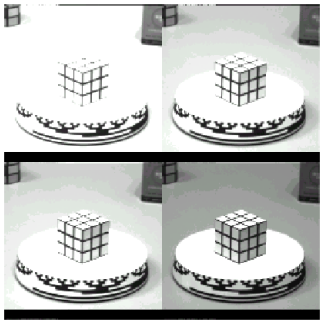
Fig.(3):cross section of the ctbrain.vols data set



(a)



(b)

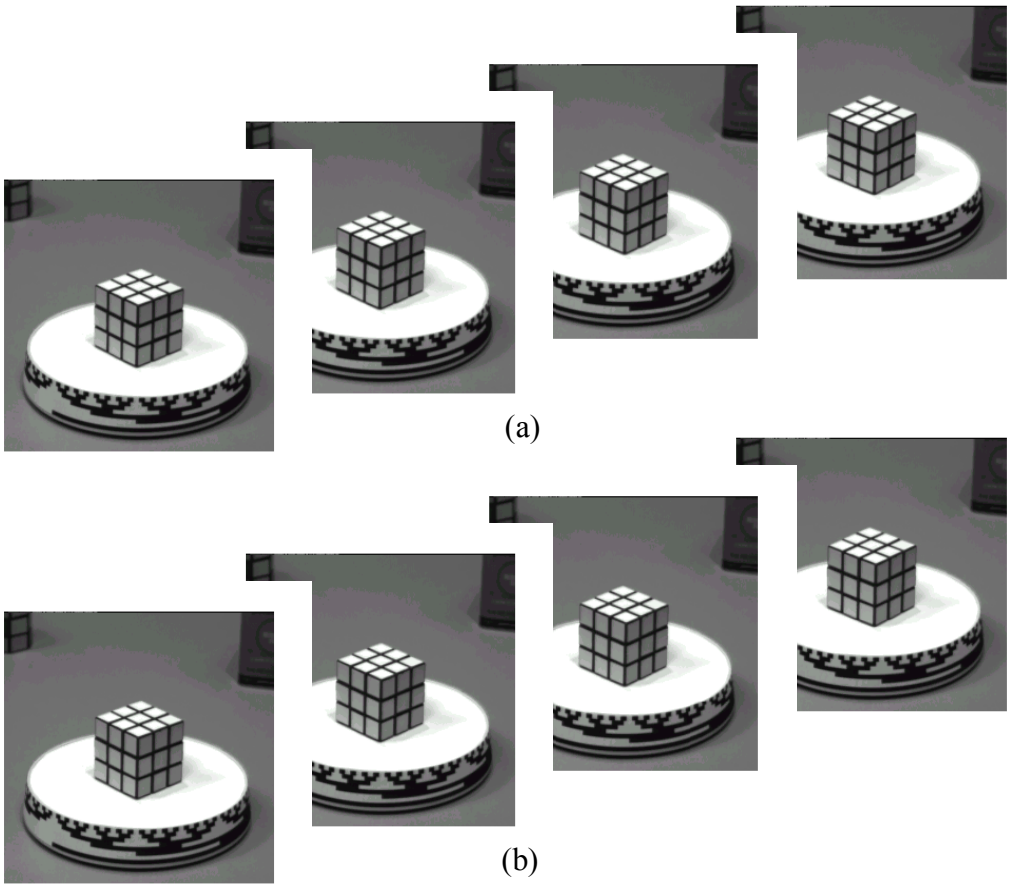


(c)

Fig. (4): Rubic's Cube sequence image (a) Original (b) 1st frame after 3-D DMWT and (c) Zoomed in upper-left front most subband after 3-D DMWT.

(a) (b)

Fig. (5): (a) Reconstructed Lena image and, (b) Original Lena image.



(a)

(b)

Fig. (6): (a) Reconstructed Rubic's cube image and, (b) Original image.