Journal of Electronic Imaging 9(1), 61–71 (January 2000).

Wavelet decomposition/reconstruction of images via direct products

N. C. Griswold
Somit Shah Mathur
Mark Yeary
Ronald G. Spencer
Texas A&M University
Department of Electrical Engineering
College Station, Texas 77843
E-mail: griswold@ee.tamu.edu

Abstract. *The two major aspects of image data compression utilizing wavelet analysis and synthesis are the decomposition of an image and the reconstruction of this image. It has been noticed in this investigation that the pyramid structure of convolution and the down sampling or the up sampling (adding zeros) and convolution have equivalent operations in vector space analysis. That is, the decomposition is equivalent to an outer product expansion. Therefore, tensor products can easily accomplish the synthesis or reconstruction. This is sometimes called the direct product. It is suggested that this method of implementation saves operations and opens the way to utilization of uniform filter banks.* © 2000 SPIE and IS&T. [S1017-9909(00)00401-3]

1 Introduction

It is well known that the fundamental building block of fast implementations with orthogonal transforms is the ability to decompose a matrix of basis vectors into submatrices with fewer operations. Normally the matrix representing the transformation matrix is never actually stored or decomposed. Instead, the equivalent operation is accomplished by the direct product, or tensor product, of a number of lower-dimensional submatrices. The submatrices used are usually lower-dimensional basis sets which, when multiplied together in a direct product operation, would form the orthogonal transform matrix to be used for a mapping operation. In this form, however, fewer operations, typically two operations per row, are possible. In fact, whenever a basis set can be generated by the direct product of two or more lower-dimensional bases sets, then a fast implementation exists. This has been the foundation of work by Cooley and Tukey,[1] Ahmed and Rao,[2] Griswold and Haralick,[3] and more recently, Hou,[4] in the fast implementation of the discrete cosine transform. In this note we apply these concepts to the wavelet decomposition and the reconstruction of an image utilizing an orthogonal wavelet basis.

Many authors[5] have discussed the use of discrete wavelets with fast pyramid-type implementations. In this paper we investigate an alternative vector space approach to the decomposition and reconstruction, using outer products and tensor products for the decomposition and the reconstruction of a data matrix and an image. In this particular note we say nothing about the coding that may take place between the decomposition and reconstruction operation for transmission, but concentrate on the decomposition and perfect reconstruction methods only. It is further suggested that if symmetry is considered, this method can reduce the number of operations now being performed by pyramid structures.

2 Decomposition

Decomposition of an image, we believe, is best understood when viewed as a projection of the data in a higher-dimensional space to a lower-dimensional space. Let us consider one level of decomposition for the sake of clarity. If we have an arbitrary vector, $\mathbf{x}\in \mathbf{L}^2(\mathbf{T})$ (i.e., a finite energy time vector with a norm of $\|\mathbf{x}\|=[\int_T|x(t)|^2dt]^{1/2}$), it can be uniquely approximated in a subspace of $\mathbf{L}^2$. Call this subspace $\mathbf{S}_n$, with an approximation of $\hat{\mathbf{x}}=\Sigma_{i=1}^n\langle \mathbf{x},q_i\rangle b_i$ where q_i,b_i are reciprocal bases and $\langle\cdot,\cdot\rangle$ represents the inner product. This of course is stated in the projection theorem.[6] For wavelets, if ϕ_n and ψ_n are bases (the scaling function and wavelets) and therefore by definition span the space of $\mathbf{V}^n$ and $\mathbf{W}^n$, respectively, and furthermore $\mathbf{V}^{n+1}=\mathbf{V}^n\otimes\mathbf{W}^n$ and $\mathbf{V}^n\perp\mathbf{W}^n$ we have

$$\begin{aligned}v_n(x)&=\sum_i c_{n,i}\varphi_{n,i}(x),\\ u_n(x)&=\sum_i d_{n,i}\psi_{n,i}(x).\end{aligned}\tag{1}$$

This simply states that v_n and u_n are the projections of the original data to the space of $\mathbf{V}^n$ and $\mathbf{W}^n$.

The coefficients of $c_{n,i}$ and $d_{n,i}$ are what we seek and are usually derived from convolution and down sampling in the pyramid structure (see Fig. 1).

In the wavelet decomposition of a sampled data string we use the φ_n or averaging basis as a low pass filter operation and the wavelet basis ψ_n as the high pass filter opera-

Paper 98058 received Nov. 18, 1998; revised manuscript received Aug. 25, 1999; accepted for publication Sep. 24, 1999.
1017-9909/2000/$15.00 © 2000 SPIE and IS&T.

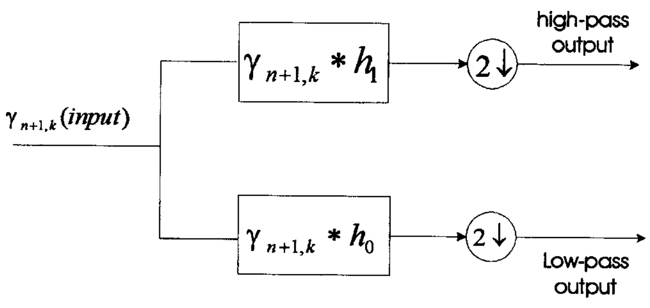


Fig. 1 Typical wavelet decomposition.

tion. This enables the decomposition of the data structure into two segments of a low frequency and high frequency structure. This may be implemented by dyadic convolution, which is known to be equivalent to linear convolution, followed by down sampling by two. To operate on a two-dimensional (2D) image, or data matrix, the two-dimensional scaling function evokes three related two-dimensional wavelets, namely

$$\varphi(x,y)=\varphi(x)\varphi(y), \quad \psi^1(x,y)=\varphi(x)\psi(y),$$
$$\psi^2(x,y)=\psi(x)\varphi(y), \quad \psi^3(x,y)=\psi(x)\psi(y). \tag{2}$$

Typically in practice, two-dimensional operations are carried out using the separability characteristic of the orthogonal basis. That is, by convoluting each row with the basis and then down sampling, or equivalently, by dyadic convolution and then transposing the result and performing the operations on the new rows (original columns).

In reality, for orthogonal or bi-orthogonal wavelets, this operation is equivalent to an outer product expansion of the two-dimensional data structure. If we consider the two-dimensional $M\times M$ matrix or image **V** to be a vector in a $\mathbf{M}^2$-dimensional space it is possible to represent this image in a p-dimensional subspace where p is the rank of the matrix **V**. If we consider the image or data matrix to be real, the matrices **VV**′ and **V**′**V** are non-negative, symmetric, and have the same identical eigenvectors $\{\lambda_m\}$ and there are at most $p\leq M$ nonzero eigenvectors. It is then possible to find p orthogonal $M\times 1$ eigenvectors $\{\varphi_m\}$ of **V**′**V** and p orthogonal $M\times 1$ eigenvectors $\{\psi_m\}$ of **VV**′, that is

$$\mathbf{V}'\mathbf{V}\varphi_m=\lambda_m\varphi_m, \quad \mathbf{V}\mathbf{V}'\psi_m=\lambda_m\psi_m. \tag{3}$$

The matrix **V** has the representation

$$\mathbf{V}=\varphi\Lambda'\psi=\sum_{m=1}^{p}\sqrt{\lambda_m}\psi_m\varphi_m'. \tag{4}$$

This is called the spectral representation or outer product expansion of **V**.[7]

Let us apply this to the decomposition of wavelets. By substituting an outer product of the scaling basis and wavelet basis for the eigenvectors, we can generate our typical low–low, low–high, high–low and high–high projections directly for one level of decomposition under consideration. First, we form the outer products necessary from the basis to be used. These outer products can then be correlated with the data matrix with dyadic shifts. For simplicity, the Haar basis has been chosen for this first introductory example. The Haar basis for dimension $N=2$ is

$$Hr_1=1/\sqrt{2}\begin{pmatrix}1\\1\end{pmatrix}; \quad Hr_2=1/\sqrt{2}\begin{pmatrix}1\\-1\end{pmatrix}. \tag{5}$$

The corresponding outer products, OP, that we need are the combinations of these basis vectors, i.e.

$$\begin{aligned}OP_1&=Hr_1\cdot Hr_1^t=\frac{1}{2}\begin{pmatrix}1&1\\1&1\end{pmatrix}OP_2\\&=Hr_1\cdot Hr_2^t=\frac{1}{2}\begin{pmatrix}1&-1\\1&-1\end{pmatrix},\\OP_3&=Hr_2\cdot Hr_1^t=\frac{1}{2}\begin{pmatrix}1&1\\-1&-1\end{pmatrix}OP_4\\&=Hr_2\cdot Hr_2^t=\frac{1}{2}\begin{pmatrix}1&-1\\-1&1\end{pmatrix}.\end{aligned} \tag{6}$$

The low–low projection then becomes the result of correlating OP_1 with a data matrix **A**.
For our purposes let **A** be given as:

$$\mathbf{A}=\begin{pmatrix}1&2&3&4\\1&2&3&4\\1&2&3&4\\1&2&3&4\end{pmatrix}.$$

The results of the correlation with dyadic shifts are

$$\mathbf{LL}=\begin{pmatrix}3&7\\3&7\end{pmatrix} \quad \mathbf{LH}=\begin{pmatrix}-1&-1\\-1&-1\end{pmatrix}$$
$$\mathbf{HL}=\begin{pmatrix}0&0\\0&0\end{pmatrix} \quad \mathbf{HH}=\begin{pmatrix}0&0\\0&0\end{pmatrix}. \tag{7}$$

As a second example, consider the decomposition (one level) of a more complicated matrix derived from the Laplacian of Gaussian function. This matrix is a horizontal slice of this two-dimensional function with $\sigma=0.2$ and scaled between 0 and 225 to simulate an 8×8 image function. The Laplacian of Gaussian function is given by

$$\nabla^2G=\left(\frac{r^2-\sigma^2}{\sigma^4}\right)\exp\left(\frac{-r^2}{2\sigma^2}\right), \quad \text{where } r^2=x^2+y^2. \tag{8}$$

The corresponding matrix is given in Fig. 2. The corresponding projected values are in Fig. 3.

The same operation could be done with such wavelet derivations as the Daubechies wavelet, as depicted in Fig. 4. In this case the dyadic shift will cause an overlap in the correlation and this overlap will be taken into consideration during the reconstruction.

This same outer product operation is performed on the 256×256 image, "Ryan," using the four-tap Daubechies filter coefficients. Again the outer product is formed for combinations of low pass and high pass coefficients. The resulting LL, LH, HL, HH images for one level of decomposition are in Fig. 5.

3 Reconstruction

The reconstruction of this data formed by an outer product expansion can be easily accomplished by the direct product or the Tensor product. Before demonstrating this fact, consider the fundamentals established by the projection theorem. We have a higher-dimensional space projected to a subspace by selected basis functions. By definition, $\mathbf{V}^{n+1} = \mathbf{V}^n \otimes \mathbf{W}^n$ (the direct sum), and these two subspaces are orthogonal. Let L be a multilinear functional on $\mathbf{V}^r$ and let M be a multilinear functional on $\mathbf{V}^s$, then we can define the function $L \otimes M$ on $\mathbf{V}^{r+s}$. This function is, of course, a Tensor product. The outer product filter functions will be multilinear on their respective projected spaces. Therefore the tensor product of the projected data, with its corresponding basis set used in the decomposition, generates a new subspace of the original data matrix or image. If there are four outer product formed basis sets, there will be four new subspaces formed by the Tensor product of each projected data with its corresponding basis. Because each of the original projections was a subspace of the original, and related to the whole by the direct sum, these new subspaces will also be related to the original by the direct sum. Therefore, we can simply take the tensor product of the projected parts and its corresponding basis and add the results of each of these products. Caution must be taken because the tensor product is not commutative. So always take the product of the projected data with the basis in that order. As an example, consider the Haar once more. The projected data were given by Eq. (7) above. Perform the direct product operation with its corresponding basis to get

$$\mathrm{Rcons}_1 = \mathbf{LL} \otimes OP_1 = \frac{1}{2}\begin{pmatrix} 3 & 3 & 7 & 7 \\ 3 & 3 & 7 & 7 \\ 3 & 3 & 7 & 7 \\ 3 & 3 & 7 & 7 \end{pmatrix}, \tag{9}$$

$$\mathrm{Rcons}_2 = \mathbf{LH} \otimes OP_2 = \frac{1}{2}\begin{pmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \end{pmatrix}. \tag{10}$$

Reconstruction of the remaining two components is zero, so we can simply add these results to get the original matrix back, i.e.

$$\mathrm{Rcons}_1 + \mathrm{Rcons}_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}. \tag{11}$$

The 8×8 matrix example is no more complicated. The reconstruction may be achieved by multiplying each basis used to generate the projection by each data point in the 4×4 projected result and adding the four resulting matrices. Caution must be used to scale everything properly. For the Daubechies four-tap filter example in this paper, the outer product was correlated with the image with dyadic shifts. This allows parts of the data to be used in determining several projections due to the overlap. This overlap is caused by a shifting in steps of two when the basis set is of dimension 4×4. However, this does not create a problem. As we create the tensor products, we simply place the result in an empty matrix in the same position from which the projected data were derived. That means that we overlap our results in an identical fashion. To convince the reader that this works, we have included a reconstruction of our 8×8 matrix as follows:

1. First, we demonstrate the reconstruction of the LL and LH as example

sf the tensor product operation. Then the other decompositions, i.e., high–low and high–high, are handled in the same way. The matrices in Figs. 6 and 7 depict these results.

2. The resulting four matrices are added together with the proper scaling and indexing to get the reconstruction of the 8×8 matrix. Obviously, since we used a floating point to generate the answers we must round off to the nearest integer values for comparison. The data in the matrices of Figs. 2, 8, and 9 confirm this.

Using this same approach, the image Ryan can be reconstructed from its decompositions. These results are depicted in Fig. 10. Note the black border was due to the size of the empty matrix which was filled with zeros. This border can be discarded by simply selecting the image data points from the empty matrix. The decomposition was repeated for bi-orthogonal case (Fig. 11).

4 Software Comparison Between Pyramidal and Outer-Product Approaches to Multi-Resolution Wavelet Analysis and Synthesis of Images

This short paper explores the differences between the conventional pyramidal and outer-product approaches to multiresolution analysis and synthesis of images from the standpoint of storage requirements, number of operations, and simplicity of application.

4.1 *Storage Requirements and Point-Synthesis Equations*

On one level the pyramidal and outer-product approaches can be compared in terms of their storage requirements. Considering only a single level of analysis, it should be obvious that both methods require intermediate storage of four analysis images, LL, LH, HL, and HH. The question is how much additional storage, if any, does the outer-product method require. Although memory is cheap, we would not want the requirement of storing the four images at each reconstruction level. This perception might be generated from the formulation offered by the arithmetic. The problem is not a problem that is inherent to the method itself. In fact, the direct (Tensor) product method of synthesis should be considered academic, seeking to produce understanding

0	0	0	0	0	0	0	0
0	0	2	13	13	2	0	0
0	2	66	255	255	66	2	0
0	13	255	10	10	255	13	0
0	13	255	10	10	255	13	0
0	2	66	255	255	66	2	0
0	0	2	13	13	2	0	0
0	0	0	0	0	0	0	0

Fig. 2 LOG8×8I.

and not necessarily a procedure to be used for the actual implementation. There is another way to reconstruct the image, a way that defines each pixel in a stand-alone sense, and does not require so much additional storage. This method is referred to herein as the point-synthesis method. Finding an expression of a single, reconstructed pixel is synonymous with finding a solution to the storage problem. The only reason that the four full-size reconstruction matrices (images) must be stored is because a given pixel in the final image is defined by information coming from all four such matrices.

It is possible to express the final pixel value in terms of its contributions from the activation of the appropriate element in the four outer-product formed filters from a set of separate correlations that occurred in the analysis. As a result, in a four-tap wavelet/scaling function case such as Daubechies, each pixel is defined by 4×4=16 different correlations, i.e., a LL, LH, HL, and HH correlation in each of four possible positions (translations) of the resulting 4×4 filters. In the four-tap Daubechies case, the outer product produces a set of 4×4 filters, which results in dyadic block overlaps; therefore, four different translations of all four filters incorporate information from the set of pixels into their respective correlations (projections). See Figs. 12 and 13.

Figure 12 shows the overlap of four translations of a 4×4 filter which defines a 2×2 area of pixels (dyadic block) which contribute to each correlation at each of the four translations. Figure 13 shows these four translations separated out for easier viewing and the positions labeled with respect to the position within each translation. For a given shaded pixel, say the upper left hand corner, it must activate coefficients (3,3), (3,1), (1,3), and (1,1) of the upper left, upper right, lower left, and lower right translations, respectively.

The following set of four equations represents the values of each pixel within one dyadic (2×2) block, where the lower right hand corner is denoted by (y,x). Although taking the Trace of a matrix is not as efficient as just multiplying the vectors needed, it serves as a succinct way to convey the idea. A more efficient notation (for actual implementation) is presented later.

$$s_{y-1,x-1}=\text{Trace}(\mathbf{Q}'\mathbf{A}),\quad s_{y-1,x}=\text{Trace}(\mathbf{R}'\mathbf{A}),$$

$$s_{y,x-1}=\text{Trace}(\mathbf{S}'\mathbf{A}),\quad s_{y,x}=\text{Trace}(\mathbf{T}'\mathbf{A}),$$

Where the matrices **Q**, **R**, **S**, and **T** are given by

$$\mathbf{Q}=\begin{bmatrix}\Phi\Phi_{3,3} & \Phi\Psi_{3,3} & \Psi\Phi_{3,3} & \Psi\Psi_{3,3}\\ \Phi\Phi_{3,1} & \Phi\Psi_{3,1} & \Psi\Phi_{3,1} & \Psi\Psi_{3,1}\\ \Phi\Phi_{1,3} & \Phi\Psi_{1,3} & \Psi\Phi_{1,3} & \Psi\Psi_{1,3}\\ \Phi\Phi_{1,1} & \Phi\Psi_{1,1} & \Psi\Phi_{1,1} & \Psi\Psi_{1,1}\end{bmatrix},$$

$$\mathbf{R}=\begin{bmatrix}\Phi\Phi_{3,4} & \Phi\Psi_{3,4} & \Psi\Phi_{3,4} & \Psi\Psi_{3,4}\\ \Phi\Phi_{3,2} & \Phi\Psi_{3,2} & \Psi\Phi_{3,2} & \Psi\Psi_{3,2}\\ \Phi\Phi_{1,4} & \Phi\Psi_{1,4} & \Psi\Phi_{1,4} & \Psi\Psi_{1,4}\\ \Phi\Phi_{1,2} & \Phi\Psi_{1,2} & \Psi\Phi_{1,2} & \Psi\Psi_{1,2}\end{bmatrix},$$

$$\mathbf{S}=\begin{bmatrix}\Phi\Phi_{4,3} & \Phi\Psi_{4,3} & \Psi\Phi_{4,3} & \Psi\Psi_{4,3}\\ \Phi\Phi_{4,1} & \Phi\Psi_{4,1} & \Psi\Phi_{4,1} & \Psi\Psi_{4,1}\\ \Phi\Phi_{2,3} & \Phi\Psi_{2,3} & \Psi\Phi_{2,3} & \Psi\Psi_{2,3}\\ \Phi\Phi_{2,1} & \Phi\Psi_{2,1} & \Psi\Phi_{2,1} & \Psi\Psi_{2,1}\end{bmatrix},$$

$$\mathbf{T}=\begin{bmatrix}\Phi\Phi_{4,4} & \Phi\Psi_{4,4} & \Psi\Phi_{4,4} & \Psi\Psi_{4,4}\\ \Phi\Phi_{4,2} & \Phi\Psi_{4,2} & \Psi\Phi_{4,2} & \Psi\Psi_{4,2}\\ \Phi\Phi_{2,4} & \Phi\Psi_{2,4} & \Psi\Phi_{2,4} & \Psi\Psi_{2,4}\\ \Phi\Phi_{2,2} & \Phi\Psi_{2,2} & \Psi\Phi_{2,2} & \Psi\Psi_{2,2}\end{bmatrix},$$

where the elements of **Q**, **R**, **S**, and **T** are elements of the outer product matrices

Low-Low

0	7.5000	7.5000	0
7.5000	293.0000	293.0000	7.5000
7.5000	293.000	293.0000	7.5000
0	7.5000	7.5000	0

Low-High

0	-5.5000	5.5000	0
-7.50C0	28.0000	-28.0000	7.5000
-7.5000	28.0000	-28.0000	7.5000
0	-5.5000	5.5000	0

High-Low

0	-7.5000	-7.5000	0
-5.5000	28.0000	28.0000	-5.5000
5.5000	-28.000	-28.0000	5.5000
0	7.5000	7.5000	0

High-High

0	5.5000	-5.5000	0
5.5000	-217.0000	217.0000	-5.5000
-5.5000	217.0000	-217.0000	5.5000
0	-5.5000	5.5000	0

Fig. 3 Projected values.

Low-Low

0	3.8394	6.9995	-0.5959	0
3.8394	258.0901	297.6501	18.1153	-1.0288
6.9995	297.6501	210.7065	64.9250	-1.8756
-0.5959	18.1153	64.9250	-13.3717	0.1597
0	-1.0288	-1.8756	0.1597	0

Low-High

0	-1.0288	3.2530	-2.2242	0
-1.0288	-62.0214	25.8758	41.0146	-3.8402
-1.8756	-66.7490	-118.2149	193.8405	-7.0010
0.1597	-5.9615	50.9881	-45.7823	0.5960
0	0.2757	-0.8717	0.5960	0

High-Low

0	-1.0288	-1.8756	0.1597	0
-1.0288	-62.0214	-66.7490	-5.9615	0.2757
3.2530	25.8758	-118.2149	50.9881	-0.8717
-2.2242	41.0146	193.8405	-45.7823	0.5960
0	-3.8402	-7.0010	0.5960	0

High-High

0	0.2757	-0.8717	0.5960	0
0.2757	14.7071	-0.8882	-15.1235	1.0290
-0.8717	-0.8882	-162.7647	167.7782	-3.2537
0.5960	-15.1235	167.7782	-155.4754	2.2247
0	1.0290	-3.2537	2.2247	0

Fig. 4 Daubechies four-tap coefficients of the 8×8 matrix.

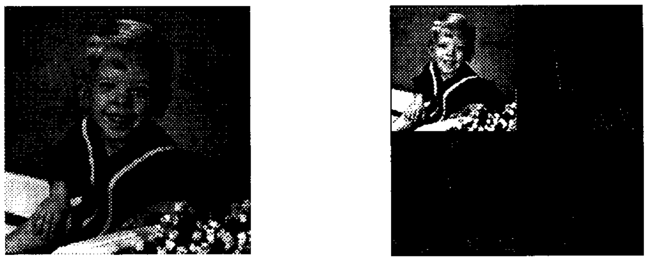


Fig. 5 Daubechies four-tap filter.

3.4904	-7.0044	-21.0236	-21.8957	-26.9529	-16.4089	-2.3950	-1.3428
-7.0044	13.7925	40.8636	44.5310	58.1367	34.6886	3.1850	1.7698
-21.0236	40.8636	119.6850	135.3423	184.5941	108.5812	5.7281	3.1221
-21.8957	44.5310	135.3423	135.1886	156.7188	97.4329	19.6071	11.0449
-26.9529	58.1367	184.5941	156.7188	133.1589	93.6156	47.0832	26.7055
-16.4089	34.6886	108.5812	97.4329	93.6156	62.5157	23.8321	13.4980
-2.3950	3.1850	5.7281	19.6071	47.0832	23.8321	-9.3912	-5.3815
-1.3428	1.7698	3.1221	11.0449	26.7055	13.4980	-5.3815	-3.0836

Fig. 6 Reconstruction of low–low.

-3.4905	5.9763	-1.3027	-1.4518	3.6026	-5.9148	1.3663	1.3432
7.0048	-12.0122	-0.3334	8.0275	-5.5722	5.8365	-1.4034	-1.7703
21.0251	-36.0933	-8.1522	36.4978	-12.7348	2.9371	-0.9535	-3.1231
21.8964	-37.4474	16.8243	-5.8969	-27.4110	54.7161	-12.5199	-11.0476
26.9522	-45.8554	63.2889	-81.1126	-57.5396	154.2418	-34.7990	-26.7118
16.4088	-27.9679	29.5764	-33.8504	-30.0237	75.6271	-17.1094	-13.5012
2.3959	-4.2185	-19.5296	36.4222	8.9512	-37.6336	8.3588	5.3827
1.3433	-2.3664	-11.1668	20.7975	5.1397	-21.5427	4.7855	3.0843

Fig. 7 Reconstruction of Low–High.

0.0000	0.0002	0.0029	-0.0017	-0.0017	0.0029	0.0002	0.0000
0.0002	-0.0007	1.9922	12.9967	12.9980	1.9917	-0.0004	0.0001
0.0029	1.9922	65.9558	254.9636	254.9898	65.9437	1.9981	0.0015
-0.0017	12.9967	254.9636	9.9416	9.9486	254.9388	12.9981	-0.0074
-0.0017	12.9980	254.9898	9.9486	9.9556	254.9649	12.9994	-0.0074
0.0029	1.9917	65.9437	254.9388	254.9649	65.9317	1.9976	0.0015
0.0002	-0.0004	1.9981	12.9981	12.9994	1.9976	-0.0001	0.0001
0.0000	0.0001	0.0015	-0.0074	-0.0074	0.0015	0.0001	0.0000

Fig. 8 Sum of reconstruction of four-subspaces (no round off).

0	0	0	0	0	0	0	0
0	0	2	13	13	2	0	0
0	2	66	255	255	66	2	0
0	13	255	10	10	255	13	0
0	13	255	10	10	255	13	0
0	2	66	255	255	66	2	0
0	0	2	13	13	2	0	0
0	0	0	0	0	0	0	0

Fig. 9 Nearest integer round—it matches original matrix.

$$\Phi\Phi \equiv \varphi\varphi^t, \quad \Phi\Psi \equiv \varphi\psi^t,$$

$$\Psi\Phi \equiv \psi\varphi^t, \quad \Psi\Psi \equiv \varphi\varphi^t$$

and where each vector is a column vector. The matrix **A** is given by

$$\mathbf{A} = \begin{bmatrix} LL_{y/2-1,x/2-1} & LH_{y/2-1,x/2-1} & HL_{y/2-1,x/2-1} & HH_{y/2-1,x/2-1} \\ LL_{y/2-1,x/2} & LH_{y/2-1,x/2} & HL_{y/2-1,x/2} & HH_{y/2-1,x/2} \\ LL_{y/2,x/2-1} & LH_{y/2,x/2-1} & HL_{y/2,x/2-1} & HH_{y/2,x/2-1} \\ LL_{y/2,x/2} & LH_{y/2,x/2} & HL_{y/2,x/2} & HH_{y/2,x/2} \end{bmatrix},$$

where the elements of the matrix are the results of analysis in 2×2 blocks (each column represents a dyadic block from one of the four analysis images).

Although taking the Trace of a matrix is not as efficient as just multiplying the vectors needed, it serves as a succinct way to convey the idea. A more efficient notation for implementation is as follows:

$$s_{y-1,x-1} = \mathbf{Q}^t_{\mathrm{LL}}\mathbf{A}_{\mathrm{LL}} + \mathbf{Q}^t_{\mathrm{LH}}\mathbf{A}_{\mathrm{LH}} + \mathbf{Q}^t_{\mathrm{HL}}\mathbf{A}_{\mathrm{HL}} + \mathbf{Q}^t_{\mathrm{HH}}\mathbf{A}_{\mathrm{HH}},$$

$$s_{y-1,x} = \mathbf{R}^t_{\mathrm{LL}}\mathbf{A}_{\mathrm{LL}} + \mathbf{R}^t_{\mathrm{LH}}\mathbf{A}_{\mathrm{LH}} + \mathbf{R}^t_{\mathrm{HL}}\mathbf{A}_{\mathrm{HL}} + \mathbf{R}^t_{\mathrm{HH}}\mathbf{A}_{\mathrm{HH}},$$

$$s_{y,x-1} = \mathbf{S}^t_{\mathrm{LL}}\mathbf{A}_{\mathrm{LL}} + \mathbf{S}^t_{\mathrm{LH}}\mathbf{A}_{\mathrm{LH}} + \mathbf{S}^t_{\mathrm{HL}}\mathbf{A}_{\mathrm{HL}} + \mathbf{S}^t_{\mathrm{HH}}\mathbf{A}_{\mathrm{HH}},$$

$$s_{y,x} = \mathbf{T}^t_{\mathrm{LL}}\mathbf{A}_{\mathrm{LL}} + \mathbf{T}^t_{\mathrm{LH}}\mathbf{A}_{\mathrm{LH}} + \mathbf{T}^t_{\mathrm{HL}}\mathbf{A}_{\mathrm{HL}} + \mathbf{T}^t_{\mathrm{HH}}\mathbf{A}_{\mathrm{HH}},$$

where the column vectors are given by

$$\mathbf{A}_{\mathrm{IJ}} = \begin{bmatrix} IJ_{y/2-1,x/2-1} \\ IJ_{y/2-1,x/2} \\ IJ_{y/2,x/2-1} \\ IJ_{y/2,x/2} \end{bmatrix} \quad \mathbf{Q}_{\mathrm{IJ}} = \begin{bmatrix} IJ_{3,3} \\ IJ_{3,1} \\ IJ_{1,3} \\ IJ_{1,1} \end{bmatrix} \quad \mathbf{R}_{\mathrm{IJ}} = \begin{bmatrix} IJ_{3,4} \\ IJ_{3,2} \\ IJ_{1,4} \\ IJ_{1,2} \end{bmatrix}$$

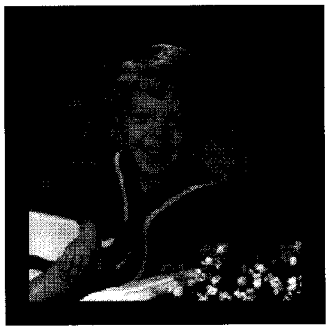
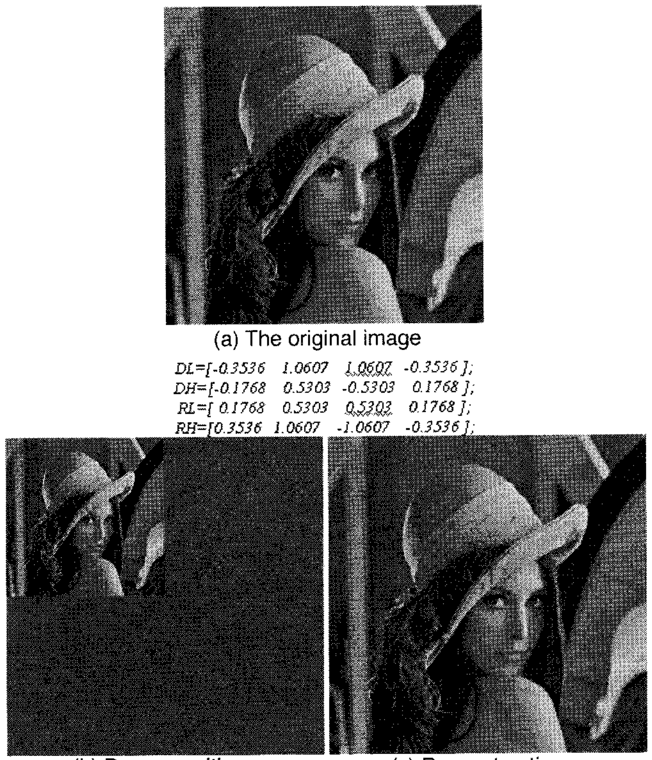


Fig. 10 Reconstructed Ryan by Tensor Products.



```
DL=[-0.3536  1.0607  1.0607  -0.3536 ];
DH=[-0.1768  0.5303  -0.5303  0.1768 ];
RL=[ 0.1768  0.5303  0.5303  0.1768 ];
RH=[0.3536  1.0607  -1.0607  -0.3536 ];
```

Fig. 11 Using bi-orthogonal wavelet 3.1.

$$\mathbf{S}_{\mathrm{IJ}} = \begin{bmatrix} IJ_{4,3} \\ IJ_{4,1} \\ IJ_{2,3} \\ IJ_{2,1} \end{bmatrix} \quad \mathbf{T}_{\mathrm{IJ}} = \begin{bmatrix} IJ_{4,4} \\ IJ_{4,2} \\ IJ_{2,4} \\ IJ_{2,2} \end{bmatrix}.$$

These equations, when demonstrated in MatLab on the four-tap Daubechies scaling functions and wavelets, required roughly the same number of floating point operations as did the program which reconstructed LL, LH, HL, and HH and then summed them up at the end.

Using the above equations for synthesis, the program actually ran slightly faster than the implementation by the previous method, but it was mostly due to the fact that circular shifts were used. By using circular shifts, the analysis matrices were slightly smaller than in the previous implementation (e.g., 128×128 vs 129×129 for a 256 ×256 image) due to using the entire capacity of every correlation. In the previous implementation, "partly empty" correlations at the boundaries occurred. Partly empty correlations result when the length of the filter, *N*, is larger than 2 (larger than one dyadic shift) and circular shifts are not used. Using circular shifts, on the other hand, essentially completes the circle, making maximum use of all correlations and allowing all analysis matrices to be exactly one quarter the size of the original image, while losing no information, as originally intended. [This problem does not appear in the Harr case because the filter length is not longer than one dyadic shift and the two-dimensional outer product formed filters are the same size as the fundamental matrix unit, i.e., a 2×2 matrix. (No overlap.).]

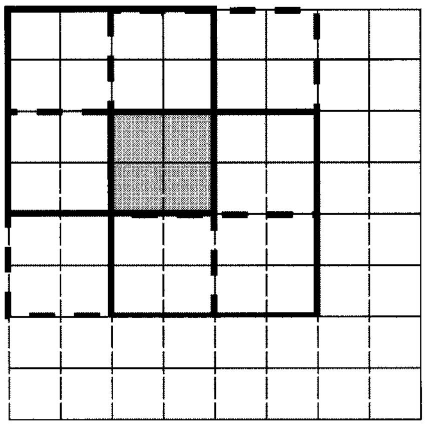


Fig. 12 Overlap of four dyadic shifts of the same 4×4 filter. The intersection of all four translations is a dyadic block also.

It should be noted that a slight storage requirement remains (4 matrices of 16 elements each), but it is not image dependent, and it is much smaller than most images. Furthermore, it is static, i.e., it is not accumulating any values like the four intermediate, full-size working construction matrices would be using the sum of direct product method of synthesis.

4.2 Operation Requirements of the Outer-Product Approach (Intrinsically 2D)

It was found that the outer-product method requires $4 \times K^2$ multiplications for setting up the outer product filters, where K is the number of filter coefficients. In the Daubechies case, $K=4$. For an $M \times N$ image, where M is the number of rows and N is the number of columns, the number of multiply and accumulate operations for the Daubechies case is given by

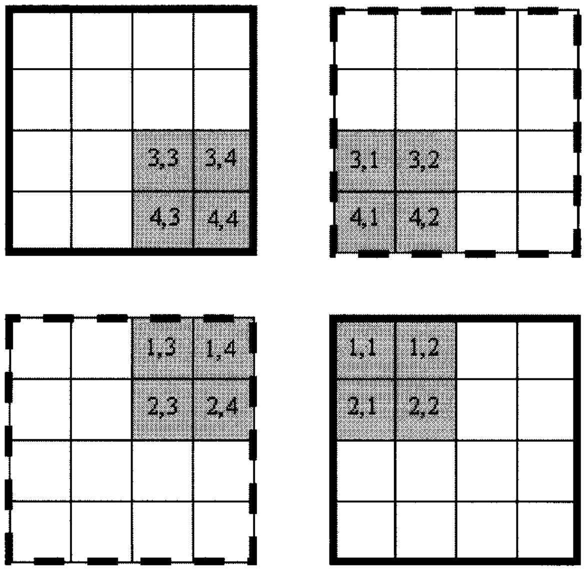


Fig. 13 The four translations of a 4×4 filter that draw from the same dyadic block of pixels separated out for easier viewing. The pixels under consideration are labeled with their position respect to the current translation.

$$\left(\frac{\frac{M}{2}-1 \text{ vert. dyadic shifts}}{\text{image analysis}}\right)\left(\frac{\frac{N}{2}\text{horiz. dyadic shifts}}{1 \text{ vert. dyadic shift}}\right) \times\left(\frac{4 \text{ filters}}{1 \text{ horiz. dyadic shift}}\right)\left(\frac{4 \text{ rows}}{\text{filter}}\right)\left(\frac{4 \text{ cols}}{\text{row}}\right) \times\left(\frac{1 \text{ mult. \& add}}{\text{col}}\right)=\frac{16(M-1)N \text{ mult. \& adds}}{\text{image analysis}}.$$

The number of operations required for direct product synthesis was found to be identical to that of the outer-product analysis. This conclusion was based on the point-decoupled equations for calculating each synthesized pixel value. This gives a total slightly lower than $32 \times M \times N$ multiply and accumulates for a complete, single-level analysis followed by synthesis. For a 256×256 image, this results in 2 097 152 multiply and accumulate operations.

4.3 Operation Requirements of the Pyramidal Approach (Extrinsically 2D)

For a sequence of length L being convolved with a filter of length L, the resulting convolution is $L+M-1$ elements long. Not considering circular convolution, the number of rows in one horizontal analysis is therefore $L+M-1$, but $L-1$ of them are all zeros if the horizontal sweep is the first. Each row then requires a sweep of $L+N-1$ elements, where N is the width of the image. For each of these, there are $2L$ multiply and accumulates. (There are actually fewer than $2L$ at the boundaries but, assuming a large image, these effects are negligible for a rough estimate.) Additionally, for each row there are $(N+L)/2$ down-sample assignments. This tally is given by

$$T_1=M\left[(N+L-1)(2L)\text{MAA}+\frac{N+L}{2}\text{STORE}\right]$$

where MAA stands for multiply and accumulate and STORE stands for a storage assignment.

Unlike the first (horizontal) sweep, the second sweep (taken here to be vertical) requires that all $L+N-1$ columns be analyzed by the one-dimensional (1D) pyramidal structure, due to the results having come from a noncircular shifted horizontal analysis. Like the horizontal sweep, each column requires a sweep of $M+L-1$ elements. For each of these, there are $2L$ multiply and accumulates. Additionally, for each column there are $(M+L)/2$ down-sample assignments. This tally is given by

$$T_2=(N+L-1)\left[(M+L-1)(2L)\text{MAA}+\frac{M+L}{2}\text{STORE}\right].$$

For a 256×256 image being processed with a four-tap Daubechies filter, the total number of operations for one (extrinsic) 2D analysis comes to 1 067 080 multiply and accumulates plus 66 560 storage assignments.

For vertical synthesis, the one-dimensional pyramidal structure must be run in reverse. This procedure requires four up-sample storage assignments for half of $M+L$ rows plus $2L$ multiply and accumulate operations and one addition for each of the $M+L-1$ rows in each of the $N+L-1$ columns. This tally is given by

$$T_3=(N+L-1)\left[(M+L-1)[(2L)\mathrm{MAA}+\mathrm{ADD}]+\left(\frac{M+L}{2}\right)\mathrm{STORE}\right].$$

The corresponding tally for the subsequent horizontal synthesis is given by the following:

$$T_4=(M+L-1)\left[(N+L-1)[(2L)\mathrm{MAA}+\mathrm{ADD}]+\left(\frac{N+L}{2}\right)\mathrm{STORE}\right].$$

For a 256×256 image being processed with a four-tap Daubechies filter, the total number of operations for one (extrinsic) 2D synthesis comes to 1 073 296 multiply and accumulates, 4144 additions, and 66 560 storage assignments. For one complete, single-level, pyramidal analysis and synthesis, 2 140 376 multiply and accumulate operations, 134 680 storage assignments, and 4144 additions are required. These figures are to be contrasted with the total number of multiply and accumulate operations of 2 097 152 in the outer-product approach. Although the number of multiply and accumulates is roughly the same, a substantial savings in the number of additions and up/down-sample storage assignments is realized.

The formulas and numbers given above are theoretical. In practice, there are always additional delays such as additional setup times and multiplexing in an integrated circuit. In software there are additional delays with "for-next" loops and memory addressing. In fact, memory addressing constitutes a considerable deviation from the theoretical limits. The pyramidal structure is much worse than the outer-product structure in terms of memory addressing because the outer-product method performs its analysis in a single step whereas the pyramidal structure must go through a horizontal and vertical sweep, reindexing the values many times. Refer to the code for one dyadic block in the outer-product method below

$$\{\mathrm{Pixel}=(\mathrm{float})((\mathrm{BYTE})^*(lpo\mathrm{DIBB}its+X+(Y^*\mathrm{WidthBytes}))),$$

$$\mathrm{LLSum}+=\mathrm{Pixel}^*\mathrm{LL}[dy+2][dx+2],$$

$$\mathrm{LHSum}+=\mathrm{Pixel}^*\mathrm{LH}[dy+2][dx+2],$$

$$\mathrm{HLSum}+=\mathrm{Pixel}^*\mathrm{HL}[dy+2][dx+2],$$

$$\mathrm{HHSum}+=\mathrm{Pixel}^*\mathrm{HH}[dy+2][dx+2]\},$$

and contrast it with the "core" code for implementing the horizontal and vertical phases of the pyramidal method

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[0],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[0],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[1],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[1],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[2],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[2],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[3],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[3],$$

$$\mathrm{lo}ds[x]=\mathrm{lo}[2^*x],$$

$$\mathrm{hi}ds[x]=\mathrm{hi}[2^*x],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[0],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[0],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[1],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[1],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[2],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[2],$$

$$\mathrm{lo}[x]+=\mathrm{lo}ds[x-1]^*a[3],$$

$$\mathrm{hi}[x]+=\mathrm{lo}ds[x-1]^*b[3],$$

$$\mathrm{lo}ds[x]=\mathrm{lo}[2^*x],$$

$$\mathrm{hi}ds[x]=\mathrm{hi}[2^*x].$$

Synthesis is essentially the same story, i.e., there is much more indexing and shuffling around of data in the pyramidal structure, which leads to delays (in software and hardware) and greater complexity in hardware. (The pyramidal structure is inherently well-suited for 1D processing. Overwhelming complexity comes when applying it to 2D images in a "brute-force" manner.)

4.4 *Timing Results*

Both methods were implemented in a Windows 98 image processing application to become familiar with the methods and to test the timing in software. For the same 256×256 image, the outer-product tested out at about 600 μs per analysis and about the same for synthesis on a 333 MHz Pentium II Deschutes processor with 224 MB random access memory (RAM) and 512k pipelined cache (about 1200 μs for a full analysis/synthesis). On the other hand, the pyramidal method tested out at about 27 ms for analysis and 20 ms for synthesis of the same image (about 47 ms for a full analysis/synthesis). (Measured processing times were subject to interrupts; therefore, multiple runs were conducted and minimum time recorded.) While everything that could possibly be done to reduce the time for both was done, there is still a substantial amount of shuffling of data in the pyramidal approach. Scaled timing results are shown in Fig. 14.

As a note, all for-next control loops were excluded from the timing analysis of each method. But due to the existence of many more control loops in the pyramidal approach than the outer-product approach, many more were excluded from the pyramidal approach. Therefore, in practice, these additional loops will contribute more processing time in the pyramidal case than the outer-product case.

4.5 *Additional Comments*

It is believed that the outer-product method is advantageous to the conventional pyramidal approach when processing two-dimensional images in several ways. The first advantage is simplicity of application. The outer-product method is intrinsically two-dimensional whereas the pyramidal approach is a piecemeal, brute-force application of an inherently 1D method to a 2D problem. The pyramidal approach is inefficient in its complexity whereas the outer-product method is elegant in its simplicity.

Essentially, the outer products encapsulate a certain ''kernel'' of information that is in a natural form for processing 2D images, i.e., a 2D structure. In one way or another, the same information or ''kernel of knowledge'' manifests itself also in the pyramidal approach, but in the repeated application of a 1D form. Therefore, the notion that there is somehow a duplication of processing in the pyramidal approach, which is calculated one time in the beginning of the outer-product approach, is implied.

A somewhat trivial but practical discovery was made that deals with calculating the range of values of the resulting analysis matrices. Since these images must be normalized before displaying them (values above and below certain display parameters are not allowed), a quick, simple and direct method of determining the minimum and maximum possible values of such matrices is very valuable. When implementing the cross-product/tensor product method of analysis, it was quite clear that the extreme values could be calculated from the outer products. When implementing the conventional pyramidal approach, it was not so clear how to obtain such values. Luckily, the same approach yielded acceptable results, so it was concluded that cross products are useful for determining the min and max values for display normalization no matter which method is used. Therefore, calculating min and max values using cross-product arrays is a new way for programmers to normalize the analysis matrices such that they are suitable for display when using the pyramidal approach.

5 Real Time Implementation Using the C6X DSP Chip

On a real time chip like the C6X DSP it is not possible to have the same implementation as the software approach at this time. So for the comparison of the pyramidal approach and the outer/tensor product approach a brute-force approach was used to investigate what could be accomplished. The outer product decomposition and the tensor product reconstruction were tested on the C6X DSP chip directly as the original algorithm indicated and compared to a modified pyramidal approach which consisted of a one-dimensional pyramid for decomposition and a 1D tensor product for reconstruction with different memory arrangements.

5.1 *Storage for 2D Implementation on C6X Outer Product/Tensor Product*

The storage requirements for this implementation were as follows:

Given an $N\times N$ image there are $N\times N$ locations for the original image:

$(N/2)\times(N/2)$ memory locations for the LL image.
$(N/2)\times(N/2)$ memory locations for the LH image.
$(N/2)\times(N/2)$ memory locations for the HL image.
$(N/2)\times(N/2)$ memory locations for the HH image.

The total amount of required memory locations is given by $2\times N\times N$.

5.2 *Storage for 1D Pyramid for Decomposition and 1D Direct Product for Reconstruction*

Again given an $N\times N$ image the amount of storage is given by:

$N\times N$ memory locations for the original image.
$N\times(N/2)$ memory locations for the L image.
$N\times(N/2)$ memory locations for the H image.
$(N/2)\times(N/2)$ memory locations for the LL image.
$(N/2)\times(N/2)$ memory locations for the LH image.
$(N/2)\times(N/2)$ memory locations for the HL image.
$(N/2)\times(N/2)$ memory locations for the HH image.

Since there is no need to keep storing the original $N\times N$ image, after the L and H images have been obtained, the total amount of required memory locations is given by $2\times N\times N$. It should be noted that this is the same as the new approach. However, because the storage has to be ''off chip'' when using the C6X the new algorithm runs slower, thus the modification to a hybrid pyramid/tensor product for this comparison.

5.3 *The Number of Operations*

The number of operations and clock cycles was determined for this comparison and is tabulated below.

Again, the Daubechies four-coefficient wavelet is being used. The wrapping-around technique is used at the end of each row or column to slightly reduce the amount of required storage at the expense of a little increase in computational complexity.

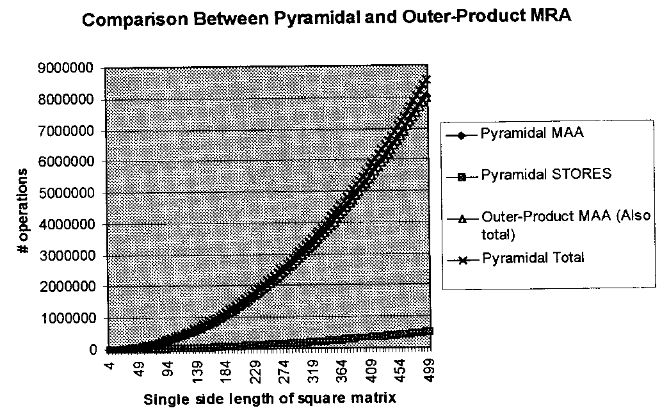


Fig. 14 Scaled timing results of both methods.

5.3.1 *Decomposition*

First step
Size of the output image: $N\times(N/2)$.
Number of output images: 2.
Size of each coefficient matrix: 4.
Number of additions: $4\times N\times(N/2)\times 2=4N^2$.
Number of multiplies: $4\times N\times(N/2)\times 2=4N^2$.
Second step
Size of the output image: $(N/2)\times(N/2)$.
Number of output images: 4.
Size of each coefficient matrix: 4.
Number of additions: $4\times(N/2)\times(N/2)\times 4=4N^2$.
Number of multiplies: $4\times(N/2)\times(N/2)\times 4=4N^2$.
Total number of additions: $8N^2$.
Total number of multiplies: $8N^2$.

5.3.2 *Reconstruction*

First step
Size of the input images: $(N/2)\times(N/2)$.
Number of input images: 4.
Size of each coefficient matrix: 4.
Number of additions: $4\times(N/2)\times(N/2)\times 4=4N^2$.
Number of multiplies: $4\times(N/2)\times(N/2)\times 4=4N^2$.
Second step
Size of the input images: $N\times(N/2)$.
Number of input images: 2.
Size of each coefficient matrix: 4.
Number of additions: $4\times N\times(N/2)\times 2=4N^2$.
Number of multiplies: $4\times N\times(N/2)\times 2=4N^2$.
Total number of additions: $8N^2$.
Total number of multiplies: $8N^2$.
As compared with the first method, the pyramidal approach with direct product reconstruction is evidently better. It not only saves half of the operations both for decomposition and reconstruction, but it is also easier to implement. For the first method, four 4×4 matrices have to be stored corresponding to the 2D LL, LH, HL, and HH wavelet filters. For the second method, only two vectors of length 4 have to be stored. Since these vectors can be stored in CPU registers, this approach is much faster for a DSP implementation.

5.4 *Timing Results on the C6x DSP*

Program	Number of cycles
2D direct product approach. Fully optimized C program running from external memory.	168 260 503
Pyramidal approach with 1D direct product for reconstruction. Fully optimized C program running from external memory.	94 133 453
Pyramidal approach with 1D direct product for reconstruction. Fully optimized C program running from internal DSP memory.	70 995 925
Pyramidal approach with 1D direct product for reconstruction. Fully optimized C program running from internal DSP memory. The wavelet filter coefficients are stored in CPU registers.	42 311 990

The last combination, i.e., the pyramidal approach with 1D direct product for reconstruction, with the wavelet coefficients stored in the CPU registers runs in about 250 ms.

6 Conclusion

The results of this type of operation give us several advantages and disadvantages. This type of procedure can be used readily on DSP since its processing power is manifested by an efficient multiply and accumulate operation. We have shown this method substantially reduces the operations over normal linear convolution. This procedure also lends itself to programming the DSP chip for the particular orthogonal basis to be used. The disadvantage is the storage needed for the reconstructed subspaces. However, with the trend in more inexpensive memory, this is minimal.

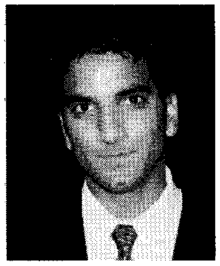
Acknowledgment

The authors would like to thank Javier Davila for running this comparison on the Texas Instruments C6X chip protocol.

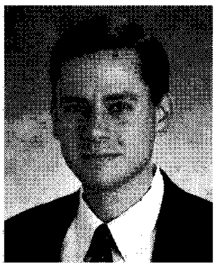
References

1. J. W. Cooley and J. W. Tukey, "An algorithm for machine calculation of complex Fourier series," *Math. Comput.* **19**(90), 297–301 (1965).
2. N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer, New York (1975).
3. R. M. Haralick and N. C. Griswold, "Image data compression: The incomplete fast transform," *IEEE SMC-CHO,* **908**(4), 299–306 (1974).
4. H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Process.* **6**(10), 1455–1461 (1987).
5. P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. Commun.* **31**(4), 532–540 (1983).
6. L. E. Franks, *Signal Theory*, Dowden and Culver, Stroudsburg, PA (1981).
7. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ (1989).

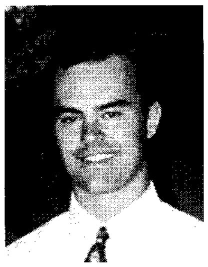
N. C. Griswold obtained his D. Engr. degree from the University of Kansas, Lawrence Kansas in 1976, his MS degree from the University of Cincinnati, Cincinnati, Ohio, 1971 (MSEE), and his Baccalaureate degree in electronic engineering in 1960 (BEE), from Clarkson University in Potsdam, New York. His interests have been focused on image transmission and its applications. These applications have included stereo vision for robotic control, bandwidth compression, pattern recognition and morphological signal processing. Other applications have included digital signal processing and source coding for images.



Somit Shah Mathur will be graduating this spring with a BS in computer engineering at Texas A&M University. He will then pursue and MS in electrical engineering with a specific interest in the digital signal and image processing areas. Currently, he has published two undergraduate research papers with topics pertaining to bit allocation and wavelet decomposition. Somit hopes to participate in future contributions to the field of electrical engineering.



Mark Yeary received his BS (honors), MS and is currently pursuing a PhD in electrical engineering each from Texas A&M University, College Station, Texas. In the past, he served as teaching assistant with the Department of Electrical Engineering, and received an "outstanding teaching assistant" award from the IEEE local student chapter during the academic years of 1994-1995 and 1995-1996. He also received a second place prize from the IEEE local chapter for his entry in the graduate student paper contest in 1996. As a student of Texas A&M University, he was a charter member and officer of the Engineering Scholars Program and has also been a recipient of the Dean's Outstanding Student Award. He was also an NSF/FIE 1998 New Faculty Fellow. He has worked for IBM as a member of a microprocessor development team. He is a member of Tau Beta Pi and Eta Kappa Nu. He is interested in digital signal processing, adaptive filters, and hardware implementation of DSP systems.



Ronald G. Spencer received his BSEE degree from GMI Engineering & Mgt. Inst. (Kettering University) in 1991 and his MS and PhD degrees from Texas A&M University in bioengineering and electrical engineering in 1994 and 1999, respectively. Currently, he is assistant professor at Texas A&M in the Analog & Mixed-Signal Center. His present interests include analog and mixed-mode VLSI of bio-inspired sensors and other adaptive circuits and systems for image processing, face recognition, and intelligent processing.