A Low-power, Low-memory System for Wavelet-based Image Compression

James S. Walker
Department of Mathematics
University of Wisconsin–Eau Claire

Truong Q. Nguyen
Department of Electrical and Computer Engineering
University of California–San Diego

Ying-Jui Chen
Department of Civil and Environmental Engineering
Massachusetts Institute of Technology

Abstract

We describe a wavelet-based image compression system which has the following advantages: low-power utilization, low RAM requirements, embedded bit-plane compression, scalable decompression, and region-of-interest selectivity. These advantages are illustrated in an application to very high compressions of underwater camera images. The proposed system performs at a comparable level to a system based on the celebrated Daub 9/7 system, yet employing 256 times less RAM and a 16-bit dynamic range (with 8-bit images) instead of a 32-bit dynamic range.

Introduction

In this paper we describe a new wavelet-based image compression system. This system has the following advantages: low-power utilization, low RAM requirements, embedded bit-plane compression, scalable decompression, and region-of-interest (ROI) selectivity. We shall illustrate all of these advantages in an application to very high compressions of underwater camera images.

The paper is organized as follows. In section 1, we provide a brief overview, via block diagrams, of the basic structure of our wavelet transform based image compression system. A brief synopsis of 1D and 2D wavelet transforms, with emphasis on discrete processing, is given in section 2. In that section, we shall provide some simple examples illustrating the fundamental concepts of wavelet transforms of images. Our compression system is based on a particular type of compressed encoding of wavelet transforms called Wavelet Difference Reduction (WDR). In section 3, we explain how WDR encoding works. We follow this description of WDR encoding with a description in section 4 of how to apply WDR encoding in a low-memory, low-power application.

Our coding scheme enjoys the advantages of scalability and ROI selectivity. These advantages are described in section 5 and illustrated on several images in section 6. Section 6, the final section of the paper, describes experimental results in applying our compression algorithm to a suite of underwater camera images. These underwater camera images were required to be compressed at very high compression ratios (400:1, 200:1, 100:1, and 50:1), and yet our algorithm produced very high-fidelity decompressions. In fact, it performed at a comparable level to a system based on the celebrated Daub 9/7 system (used in JPEG 2000 [10]) yet employing 256 times less RAM and a 16-bit dynamic range (with 8-bit images) instead of a 32-bit dynamic range.

1 Description of Image Compression

The basic scheme for compressing and decompressing images is shown in Figure 1 below. Compression consists of two steps to generate a compressed bitstream. The first step is a wavelet transform of the image and the second step is the compressed encoding of the image's wavelet transform. Decompression simply consists of reversing these two steps, decoding the compressed bit stream to produce an (approximate) image transform and then applying an inverse wavelet transform to create the decompressed image.

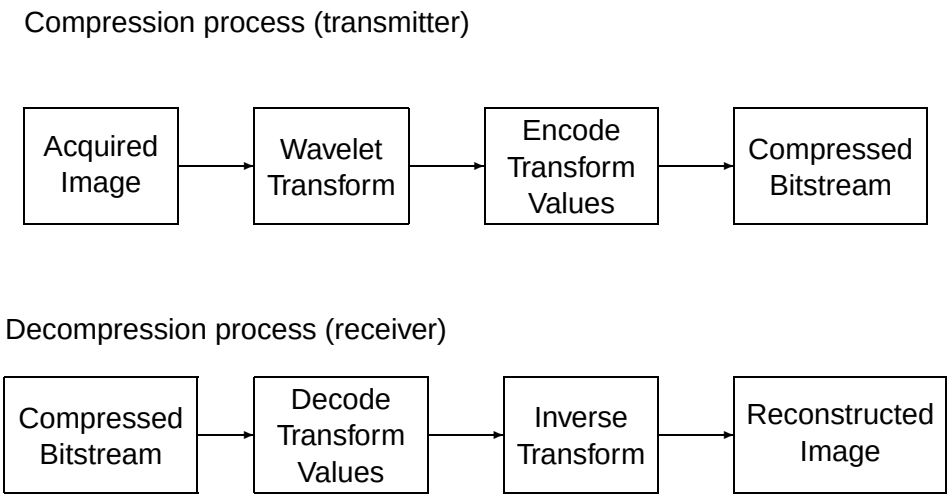


Figure 1: Compression and Decompression.

We shall now discuss each of the components of our particular method of wavelet transform based image compression.

2 Wavelet transforms

Wavelet transforms have been extensively studied over the last decade [4, 5, 6, 8]. A wavelet transform combines both lowpass and highpass filtering in a spectral decomposition of signals,

along with an extremely fast implementation. Before considering wavelet transforms of 2D images, it is helpful to first consider wavelet transforms of 1D signals. Given a 1D signal $s_0[n]$, its 1-level wavelet transform is the mapping $s_0[n] \mapsto (s_1[2n] \mid d_1[2n])$ defined by

$$s_1[2n] = \sum_{k=-M}^{M} \alpha_k s_0[2n+k]$$
$$d_1[2n] = \sum_{k=-N}^{N} \beta_k s_0[2n+k+1].$$

The signals $s_1[2n]$ and $d_1[2n]$ are respectively lowpass and highpass filterings of $s_0[n]$. These filterings have also been *downsampled:* they are defined over the indices $\{2n\}$ rather than $\{n\}$. Viewed as sampled signals, they are sampled at half the rate as s_0. The coefficients $\{\alpha_k\}$ are the *lowpass coefficients* and the coefficients $\{\beta_k\}$ are the *highpass coefficients.*

A simple example of these ideas is the Daub 5/3 wavelet system [3]. For this system, the lowpass and highpass coefficients are defined by

$$\begin{aligned} &(M=2): \quad \alpha_{-2} = \frac{-1}{8},\ \alpha_{-1} = \frac{1}{4},\ \alpha_0 = \frac{3}{4},\ \alpha_1 = \frac{1}{4},\ \alpha_2 = \frac{-1}{8} \\ &(N=1): \quad \beta_{-1} = \frac{-1}{2},\ \beta_0 = 1,\ \beta_1 = \frac{-1}{2}. \end{aligned} \tag{1}$$

These coefficients have some basic properties which are shared by other wavelet systems. One important property is that they define an invertible transform; we shall discuss this further later. Perhaps just as importantly, the highpass coefficients satisfy $\sum \beta_k = 0$ and $\sum k\beta_k = 0$. Consequently, if s_0 is linear (or approximately linear) over the indices $2n$, $2n+1$, $2n+2$, then $d_1[2n] = 0$ (or $d_1[2n] \approx 0$). When s_0 is obtained from samples of a piecewise smooth function, the highpass filtering d_1 will be essentially zero-valued (except near transitions between pieces of the piecewise smooth function). This provides the foundation for compression. When the transform $s_m \mapsto (s_{m+1} \mid d_{m+1})$ is iterated on the lowpass outputs $s_1, s_2, \ldots$, then several levels of transform will produce large numbers of zero values (or nearly zero values) in the highpass outputs $d_2, d_3, \ldots$. Such high redundancy of zero values, in $d_1, d_2, d_3, \ldots$, allows for significant compression.

It is also worth noting that the Daub 5/3 lowpass coefficients satisfy $\sum \alpha_k = 1$ and $\sum k\alpha_k = 0$. These equations imply the approximation $s_1[2n] \approx s_0[2n]$, at most indices n, when s_0 is sampled from a piecewise smooth signal. This approximation is useful in detection and denoising algorithms. The equation $\sum \alpha_k = 1$ also tells us that each lowpass output is an average: $s_1[2n]$ is an average over the signal values $\{s_0[2n+k]\}_{k=-2}^{2}$, while $s_2[4n]$ is an average over the lowpass values $\{s_1[4n+2k]\}_{k=-2}^{2}$ and hence also over the signal values $\{s_0[4n+k]\}_{k=-6}^{6}$, and so on. Notice that $s_2[4n]$ is an average over a wider range of original signal values than $s_1[2n]$ (more than twice as many values, a doubling in scale). Similarly, the highpass output $d_2[4n]$ is a response to more than twice the range (twice the scale) of signal values as $d_1[2n]$. This doubling of scale occurs in passing to each new level of the transform.

Another important wavelet system for image compression is the Daub 9/7 system [3]. This system involves more complicated highpass and lowpass coefficients. In particular, these coefficients are not simple rational numbers—as in the Daub 5/3 system. The additional requirement of using 32-bit floating point numbers for implementing the Daub 9/7 system is one reason that our image

compression algorithm employs the Daub 5/3 system (other advantages of the Daub 5/3 system will be described below). The Daub 9/7 highpass coefficients satisfy $\sum \beta_k = 0$, $\sum k\beta_k = 0$, and $\sum k^2\beta_k = 0$. The extra zero sum $\sum k^2\beta_k = 0$ implies even greater compression. If a signal is well-approximated by piecewise quadratics, then a huge number of (approximately) zero-values will appear in its wavelet transform. This is especially important in image compression, where approximation of an image by piecewise quadratic surfaces is justified based on standard illumination models [6].

A wavelet transform for 1D signals can be easily generalized to 2D images by applying it separately in each dimension. The 1^{st} level of a discrete wavelet transform of a $J \times K$ matrix $\mathbf{F}$, where J and K are both even, is obtained in two steps: (1) transform each row of $\mathbf{F}$ by a 1D wavelet transform obtaining a matrix $\widetilde{\mathbf{F}}$; (2) transform each column of $\widetilde{\mathbf{F}}$ by the same 1D transform. Steps 1 and 2 are independent and may be performed in either order. It is important to note that the symmetry of the Daub 5/3 transform coefficients (and Daub 9/7 coefficients) allows for these transforms to be calculated using symmetric extensions of the matrix rows and columns.

Step 1 of this transform process produces J rows of 1D transforms:

$$\mathbf{F} \mapsto \begin{pmatrix} s_1^1 \,|\, d_1^1 \\ s_2^1 \,|\, d_2^1 \\ \vdots \\ s_J^1 \,|\, d_J^1 \end{pmatrix}.$$

Step 2 then produces the following 1^{st} level transform of the matrix $\mathbf{F}$:

$$\mathbf{F} \mapsto \left(\begin{array}{c|c} \mathbf{A}^1 & \mathbf{V}^1 \\ \hline \mathbf{H}^1 & \mathbf{D}^1 \end{array}\right)$$

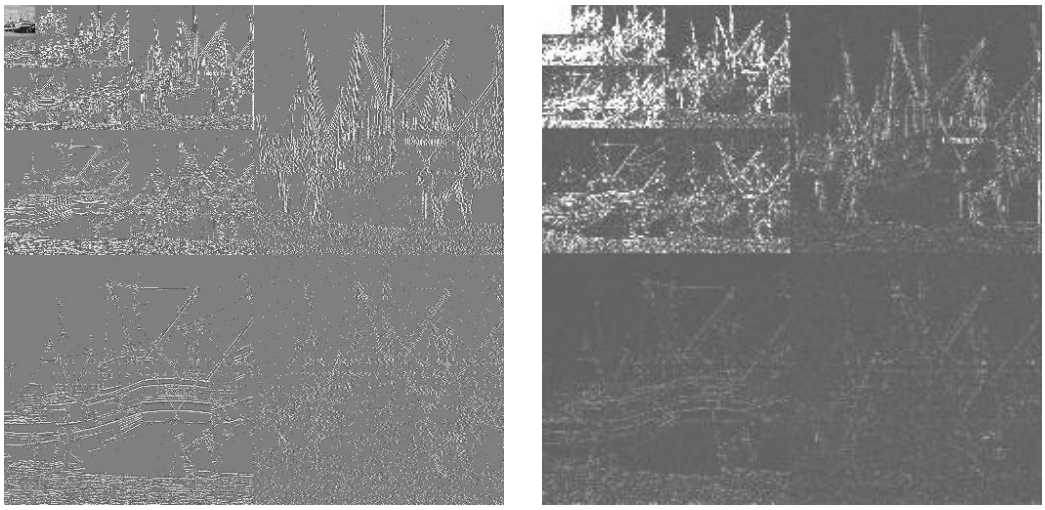
where $\mathbf{A}^1$, $\mathbf{V}^1$, $\mathbf{H}^1$, and $\mathbf{D}^1$ are each $J/2 \times K/2$ sub-matrices. For example, in Figure 2 we show the 1^{st} level of a Daub 5/3 wavelet transform of the "boats" test image. The row-lowpass/column-lowpass submatrix $\mathbf{A}^1$ is shown in the upper left quadrant of Figure 2(b). It is clearly related to the original image (recall the approximation $s_1[2n] \approx s_0[2n]$ mentioned above). The row-lowpass/column-highpass submatrix $\mathbf{H}^1$ is shown in the lower left quadrant of Figure 2(b). Wherever there are horizontal edges in an image, the highpass filterings along columns are able to detect these edges. That is why horizontal edges appear most prominently in $\mathbf{H}^1$. Similarly, the row-highpass/column-lowpass submatrix $\mathbf{V}^1$—which is shown in the upper right quadrant of Figure 2(b)—emphasizes vertical edges. The last submatrix, the row highpass/column highpass submatrix $\mathbf{D}^1$ shown in the lower right quadrant of Figure 2(b), emphasizes diagonal edges.

The wavelet transform can be iterated on the row-lowpass/column-lowpass outputs. Doing this on $\mathbf{A}^1$ produces submatrices $\mathbf{A}^2$, $\mathbf{V}^2$, $\mathbf{H}^2$, $\mathbf{D}^2$. This is called a 2-level transform. As with 1D signals, the 2^{nd} level submatrices are responses to the 2D image values of twice the range of pixels (twice the scale) as the 1^{st} level submatrices. In Figure 3(a), we show a 4-level Daub 5/3 transform of the "boats" image. The most important thing to observe about the wavelet transform in Figure 3(a) is the large number of very small magnitude (approximately zero) values—indicated by grey background regions—in the submatrices $\mathbf{V}^m$, $\mathbf{H}^m$, $\mathbf{D}^m$ for $m = 1, 2, 3, 4$.

For the Daub 5/3 system the proportion of these small transform values can be greatly increased by multiplying the transform by an appropriate set of *weight factors* for each submatrix. The set

(a) “boats” image (b) 1-level Daub 5/3 transform

Figure 2: Example of an image and its 1-level Daub 5/3 wavelet transform.



(a) (b)

Figure 3: Daub 5/3 transform and the Daub 5/3 Int.-to-Int. transform of the “boats” image. (a) The 4-level Daub 5/3 transform. (b) The locations of larger values of the Daub 5/3 Int.-to-Int. transform indicated by brighter pixels.

of weight factors used is shown in Figure 4. Multiplying the submatrices of a 4-level Daub 5/3 transform by this set of weight factors produces a nearly orthogonal transform. The value of having a nearly orthogonal transform is that the different transform values are accorded equal energy, when energy is measured by summing squares, for a randomly selected image. (The Daub 9/7 transform is very nearly orthogonal by itself, hence no weight factors are needed.)

The equations defining the Daub 5/3 transform (also the Daub 9/7 transform) can be expressed

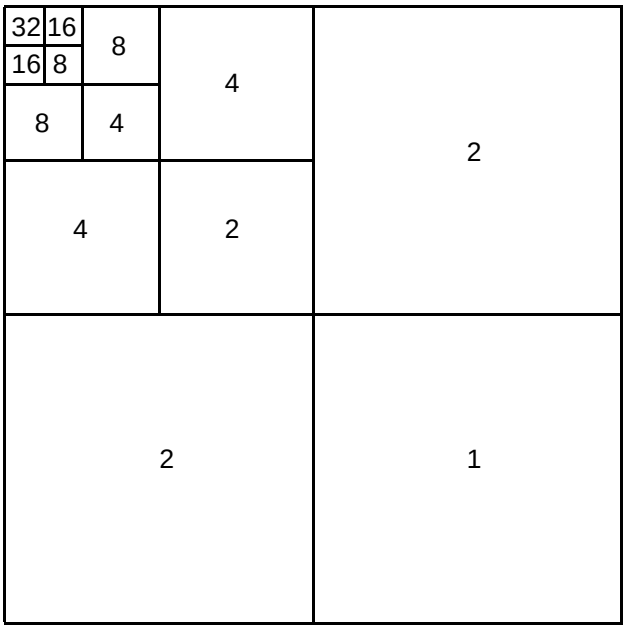


Figure 4: Weight factors for multiplying 4-level Daub 5/3 transforms.

in a more efficient form, known as the *lifting* form [9, 2]. The lifting form of the Daub 5/3 transform equations (1) is

$$\begin{aligned} d_1[2n] &= s_0[2n+1] - \frac{1}{2}(\, s_0[2n] + s_0[2n+2]\,) \\ s_1[2n] &= s_0[2n] + \frac{1}{4}(\, d_1[2n-2] + d_1[2n]\,). \end{aligned} \tag{2}$$

These equations have a simple set of inverse equations (thus defining the inverse of the Daub 5/3 transform):

$$\begin{aligned} s_0[2n] &= s_1[2n] - \frac{1}{4}(\, d_1[2n-2] + d_1[2n]\,) \\ s_0[2n+1] &= d_1[2n] + \frac{1}{2}(\, d_0[2n] + s_0[2n+2]\,). \end{aligned} \tag{3}$$

The equations in (2) are considerably more efficient than the original equations in (1). About half as many operations are needed to perform the calculations in (2). There is a similar efficiency inherent in (3).

More importantly, however, is that the equations (2) and (3) have *integer-to-integer* versions:

$$\begin{aligned} d_1[2n] &= s_0[2n+1] - \left\lfloor \frac{1}{2}(\, s_0[2n] + s_0[2n+2]\,) + \frac{1}{2} \right\rfloor \\ s_1[2n] &= s_0[2n] + \left\lfloor \frac{1}{4}(\, d_1[2n-2] + d_1[2n]\,) + \frac{1}{2} \right\rfloor \end{aligned} \tag{4}$$

and

$$
\begin{aligned}
s_0[2n] &= s_1[2n] - \left\lfloor \frac{1}{4}(d_1[2n-2] + d_1[2n]) + \frac{1}{2} \right\rfloor \\
s_0[2n+1] &= d_1[2n] + \left\lfloor \frac{1}{2}(d_0[2n] + s_0[2n+2]) + \frac{1}{2} \right\rfloor .
\end{aligned}
\tag{5}
$$

The equations (4) and (5) are used respectively to implement the *Daub 5/3 Int.-to-Int.* wavelet transform and its inverse. When applied to images, the weighting factors shown in Figure 4 are applied after the Daub 5/3 Int.-to-Int. transform based on (4), and their reciprocal factors are applied before the inverse Daub 5/3 Int.-to-Int. transform based on (5).

An illustration of a 4-level Daub 5/3 Int.-to-Int. transform is shown in Figure 3(b). In that figure, the locations of the largest transform values are shown as the brightest pixels. The large dark grey areas indicate nearly zero values, implying that the Daub 5/3 Int.-to-Int. transform should yield good compression. In fact, it has been shown that this transform provides essentially the best image compression results among the principal integer-to-integer transforms [1]. We shall see below in subsection 6 that this transform performs nearly as well as the celebrated Daub 9/7 transform in compressing underwater camera images.

The two major advantages of using a Daub 5/3 Int.-to-Int. transform are the following:

1. *Extremely fast implementation.* Equations (4) and (5) (and the weighting factors and their reciprocals) can be implemented using only integer additions and bit-shifts—which are extremely fast operations. The Daub 9/7 transform cannot be implemented using only integer operations; it requires floating point additions and multiplications. There is an integer-to-integer transform related to the Daub 9/7 transform, but it requires multiplications as well as additions, and has been shown to not perform as well for image compression as the Daub 5/3 Int.-to-Int. transform [1].

2. *Low power requirements.* When transforming an 8-bit grey-scale image (as from an underwater camera), a 4-level Daub 5/3 Int.-to-Int. transform will produce values that can be stored in 16-bit integers. (This is not the case for a Daub 9/7 transform, which requires 32-bit floating point variables.)

3 WDR encoding

In the image compression scheme shown in Figure 1, the step following the wavelet transform is an encoding of the wavelet transform to generate a compressed bitstream. The method we have employed is a public-domain algorithm known as *Wavelet Difference Reduction* [11, 6]. We shall now briefly describe the WDR algorithm; more complete details can be found in [5].

The WDR algorithm consists of five parts, as shown in Figure 5. In the Initialize part, an initial threshold value T_0 is chosen so that all transform values have magnitudes that are less than T_0, and at least one has magnitude greater than or equal to $T_0/2$. The purpose of the loop indicated in Figure 5 is to encode significant transform values by the method of *bit-plane encoding.* A binary expansion, relative to the quantity T_0, is computed for each transform value. The loop constitutes the procedure by which these binary expansions are calculated. As the threshold is successively

reduced by half, the Significance Pass and Refinement Pass compute the next bit in the binary expansions of the transform values. See [5] for a more complete description of bit-plane encoding.

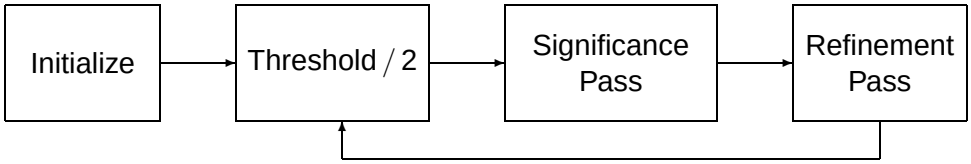


Figure 5: Schematic diagram of WDR Compression

We shall now describe each part of the WDR algorithm in more detail. The Initialize part, as described above, involves choosing an initial threshold $T = T_0$. During this part an assignment of a *scan order* is made. For an image with P pixels, a scan order is a one-to-one and onto mapping, $\widehat{F}_{i,j} = x_k$, for $k = 1, \ldots P$, between the transform values $(\widehat{F}_{i,j})$ and a linear ordering (x_k). This initial scan order is a zigzag from higher to lower levels [7], with row-based scanning through the horizontal coefficients, column-based scanning through the vertical coefficients, and zigzag scanning through the diagonal coefficients.

After the Initialize part, the threshold $T = T_{k-1}$ is updated to $T = T_k$, where $T_k = T_{k-1}/2$.

The next part of the algorithm is the Significance Pass. In this part, *new* significant transform values x_m satisfying $T \leq |x_m| < 2T$ are identified. Their index values m are encoded using the *difference reduction method* of Tian and Wells [11, 6]. The difference reduction method essentially consists of a binary encoding of the number of steps to go from the index of the last significant value to the index of the present significant value. The output from the Significance Pass consists of the signs of significant values along with sequences of bits, generated by difference reduction, which concisely describe the precise locations of significant values. The best way to see how this is done is to consider a simple example.

Suppose that the threshold is $T = 32$ and that the new significant values are $x_2 = +34$, $x_3 = -33$, $x_8 = +47$, $x_{12} = +40$, and $x_{32} = -54$. The indices for these significant values are 2, 3, 8, 12, and 32. Rather than working with these values, WDR works with their successive differences: 2, 1, 5, 4, and 20. In this latter list, the first number is the *starting index* and each successive number is the *number of steps* needed to reach the next index. The binary expansions of these successive differences are $(10)_2$, $(1)_2$, $(101)_2$, $(100)_2$, and $(10100)_2$. Since the most significant bit for each of these expansions is always 1, this bit can be dropped and the signs of the significant transform values can be used instead as separators in the symbol stream. When this most significant bit is dropped, we shall refer to the binary expansion that remains as the *reduced binary expansion*. Notice, in particular, that the reduced binary expansion of 1 is empty. The resulting symbol stream for this example is then $+0 - +01 + 00 - 0100$, and the *quantized values* q_m assigned (implicitly) by this encoding are $q_2 = +32$, $q_3 = -32$, $q_8 = +32$, $q_{12} = +32$, and $q_{32} = -32$.

Following the Significance Pass, there is a Refinement Pass. The Refinement Pass is a process of refining the precision of *old* quantized transform values q_n, which satisfy $|q_n| \geq 2T$. Each

Step 1 (*Initialize*). Choose an initial threshold T_0 so that *all* transform values satisfy $|x_m| < T_0$ and at least one transform value satisfies $|x_m| \geq T_0/2$.

Step 2 (*Update threshold*). Let $T_k = T_{k-1}/2$.

Step 3 (*Significance pass*). Perform the following procedure while scanning through insignificant values for higher thresholds:

```
Initialize step-counter C = 0
Let C_old = 0
Do
    Get next insignificant index m
    Increment step-counter C by 1
    If |x_m| ≥ T_k then
        Output sign x_m and set q_m = sgn(x_m) · T_k
        Move m to end of sequence of significant indices
        Let n = C − C_old
        Set C_old = C
        If n > 1 then
            Output reduced binary expansion of n
    Else if |x_m| < T_k then
        Let q_m retain its initial value of 0.
Loop until end of insignificant indices
Output end-marker
```

The end-marker is a plus sign followed by the reduced binary expansion of $n = C+1-C_{\text{old}}$ and a final plus sign.

Step 4 (*Refinement pass*). Scan through significant values found with higher threshold values T_j, for $j < k$ (if $k = 1$ skip this step). For each significant value x_m, do the following:

```
If |x_m| ∈ [|q_m|, |q_m| + T_k), then
    Output bit 0
Else if |x_m| ∈ [|q_m| + T_k, |q_m| + 2T_k), then
    Output bit 1
    Replace value of q_m by q_m + sgn(q_m) · T_k.
```

Step 5 (*Loop*). Repeat steps 2 through 4 (exiting at any point if bit budget is exceeded).

Figure 6: The WDR encoding algorithm.

refined value is a better approximation of an exact transform value. The precision of quantized values is increased to make them at least as accurate as the present threshold. For example, if an old significant transform value's magnitude $|x_n|$ lies in the interval $[64, 128)$, say, and the present threshold is 32, then it will be determined if its magnitude lies in $[64, 96)$ or $[96, 128)$. In the latter case, the new quantized value becomes $96\,\mathrm{sgn}(x_n)$, and in the former case, the quantized value remains $64\,\mathrm{sgn}(x_n)$. The Refinement Pass adds another bit of precision in the binary expansions of the scaled transform values $\{x_k/T_0\}$.

We provide in Figure 6 a pseudo-code listing for the WDR method. It is important to note that this procedure may be exited at any point once the total number of bits exceeds a pre-assigned amount (a *bit budget*).

In order to decompress a WDR compressed image, the above steps are recapitulated using the bits output by WDR to rebuild a quantized transform. An inverse wavelet transform is then applied to create the decompressed image. More details on WDR compression can be found in [5].

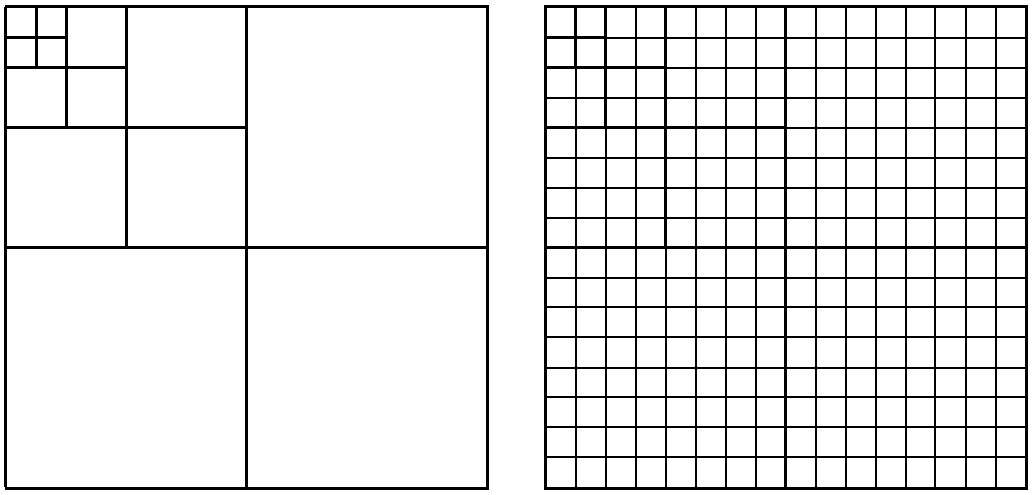


Figure 7: Block-based wavelet transform. (a) Arrangement of submatrices in a 4-level wavelet transform. (b) Division of transform into 256 blocks.

4 Low-memory, block-based WDR

One drawback of the wavelet-based image compression method described above is that it requires significant memory resources. This is a result of the entire wavelet transform and its scanning order being held in full in RAM. There is, however, a simple method of reducing memory requirements: the *block-based* wavelet transform, along with a WDR algorithm for compressing this block-based transform.

For a block-based wavelet transform, the wavelet transform is computed by successively loading from external memory into RAM a small number of rows at a time, computing their 1D transforms and then outputting those transforms to externally stored memory. A similar sequential op-

eration is done on the columns, and also with multiplying by weights. In this way, very little RAM is needed for computing the image transform (and also its inverse, which is handled similarly).

The term *block-based* comes from the way in which the encoding of the transform is handled in order to remain consistent with the desire to use a small amount of RAM. In Figure 7 we show the way in which a 4-level wavelet transform is organized into 256 tiny blocks (a 16×16 array of blocks). When performing the WDR encoding, these blocks are read in sequentially along rows from top to bottom *within each submatrix* according to the following order of submatrices:

$$\mathbf{A}^4, \mathbf{V}^4, \mathbf{H}^4, \mathbf{D}^4, \mathbf{V}^3, \mathbf{H}^3, \mathbf{D}^3, \mathbf{V}^2, \mathbf{H}^2, \mathbf{D}^2, \mathbf{V}^1, \mathbf{H}^1, \mathbf{D}^1.$$

Each block lies in a particular submatrix, so the scanning order used for that block is the one appropriate to its type (row-wise for horizontally detailed submatrices $\mathbf{H}^m$, column-wise for vertically detailed submatrices $\mathbf{V}^m$, and zigzag for diagonally detailed submatrices $\mathbf{D}^m$ and for the all-lowpass submatrix $\mathbf{A}^4$). The resulting reduction of RAM requirements by a factor of 256 allows for WDR encoding to be carried out efficiently in situations where memory is severely limited.

When used in conjunction with a Daub 5/3 Int.-to-Int. transform, this block-based method enjoys the following advantages (two of which were noted above):

1. Extremely fast implementation.

2. Low power requirements.

3. Low internal memory requirements.

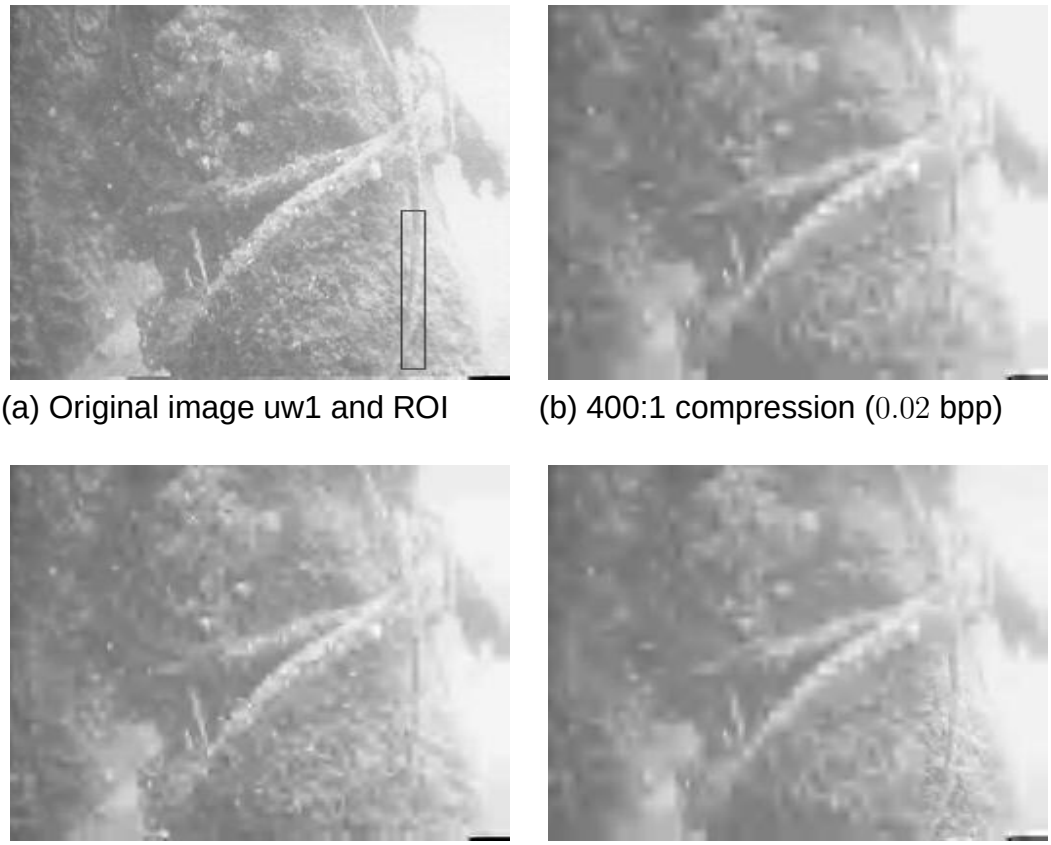
We shall see in subsection 6 below that this block-based Daub 5/3 Int.-to-Int. transform provides excellent compressions of test images.

5 Scalability and ROI Selectivity

Two particularly important properties of WDR compression are its scalability and Region-of-Interest (ROI) selectivity. By scalability, we mean that any bit-rate, less than or equal to the compressed image's bit-rate, can be used for decompressing a WDR compressed image. Scalability is an important feature for transmitting compressed images over low-capacity channels.

When transmitting an image over a low-capacity channel, the compressed image is typically transmitted using data packets. The first packet produces a low-resolution image at the receiver, and subsequent packets enhance this image. The WDR algorithm, because it uses bit-plane encoding, easily fits into such a transmission scheme. Moreover, between successive packets, the receiver could request the compressor to selectively enhance a region-of-interest (ROI). The WDR algorithm allows for just such an ROI selectivity.

In Figure 8 we show some images illustrating such a system. In Fig. 8(a) there is an underwater camera image. Because transmission capacity is low, this image must be transmitted in packets that are, say, 400 times smaller in size than the total bits for the image. In Fig. 8(b), we show the image at the receiver after the first packet has been transmitted. This image is a 400:1 compression of the source image. Suppose now that the receiver requests that the next packet focus exclusively on enhancing the region inside the rectangle shown in Fig. 8(a). The image shown in Figure 8(d) is the result of using only bits in the next packet which enhance details in the ROI. While in Figure 8(c)



(a) Original image uw1 and ROI (b) 400:1 compression (0.02 bpp)

(c) Update (0.04 bpp) (d) Update (0.04 bpp), ROI only

Figure 8: ROI selectivity of block-based algorithm.

we show the 200:1 compression obtained by the next packet, when no ROI is selected. The ROI in Figure 8(d) is much more detailed—it more closely resembles the original ROI—than the ROI in Figure 8(c). The contrast between the ROIs in Figures 8(c) and (d) is significantly greater on a CRT display—see the webpage listed below in (6).

6 Underwater camera image compressions

We complete our discussion of image compression with some examples of compressions of underwater camera images. The test images that we used (labeled uw1, uw2, uw3, uw4, and uw5) can be found at the following webpage:

http://www.uwec.edu/academic/curric/walkerjs/UnderwaterImages/ (6)

Larger sized versions of all of the images shown in Figures 8 to 11 can also be found at this webpage.

To make an objective comparison, we used PSNR as a numerical measure of error between original and decompressed images. The three methods of compression compared are all based on WDR applied to one of three transforms: a Daub 9/7 wavelet transform, a Daub 5/3 Int. to Int.

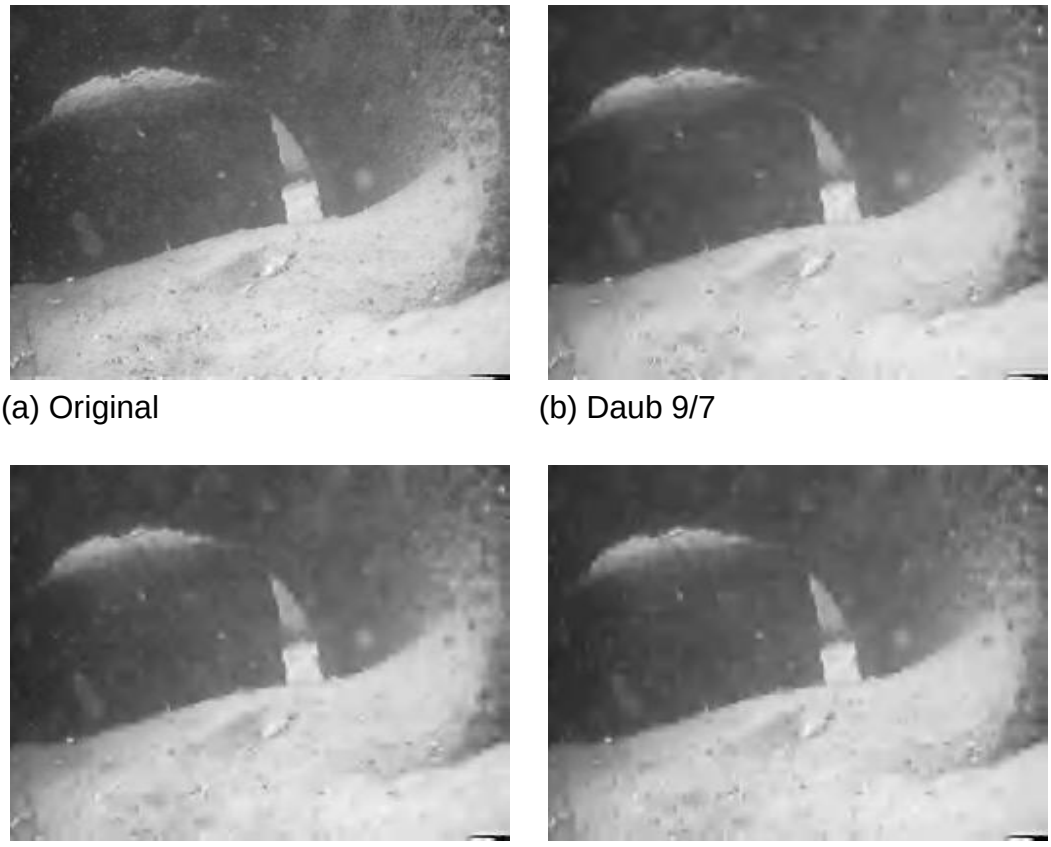
Image	Daub 9/7	Daub 5/3 Int.-to-Int.	Block-based Daub 5/3 Int.-to-Int.
400:1 compressions			
uw1	29.3	28.9	29.1
uw2	36.2	36.6	36.4
uw3	27.4	27.2	27.4
uw4	30.1	30.7	30.8
uw5	36.8	36.6	36.5
200:1 compressions			
uw1	31.5	31.0	31.1
uw2	40.2	39.1	38.8
uw3	29.7	29.3	29.6
uw4	33.5	33.1	33.0
uw5	41.3	40.1	40.1
100:1 compressions			
uw1	34.0	33.4	33.4
uw2	43.0	41.7	40.8
uw3	32.6	32.1	32.3
uw4	36.3	35.5	35.3
uw5	44.1	42.2	41.7
50:1 compressions			
uw1	37.1	36.4	36.1
uw2	45.6	43.3	41.2
uw3	36.2	35.5	35.5
uw4	39.5	38.5	38.2
uw5	46.5	43.5	41.6

Table 1: Comparison of PSNR compression performance.

transform, and a block-based Daub 5/3 Int. to Int. transform. Besides PSNR data, we shall also provide some visual comparisons between original and decompressed images.

In Table 1, we summarize the PSNRs obtained using four different compression ratios (400:1, 200:1, 100:1, and 50:1) on five different underwater camera images. The PSNRs for the 400:1 decompressions show that all three methods are essentially equivalent at this high compression ratio. As the compression ratio is reduced, however, the Daub 9/7 compressions exhibit higher PSNRs. In Figures 9 to 11 we show the decompressed images for three different images (*uw2*, *uw4*, and *uw5*) which show large variations between PSNRs for the three methods. These images show that visually there is essentially no difference between the three methods. This is largely due to the fact that the PSNR values for these lower compression ratios (especially 100:1 and 50:1) are relatively large.

The PSNR data summarized in Table 1 and the visual examples in Figures 9 to 11 provide strong evidence that the block-based Daub 5/3 Int. to Int. transform performs as well as both of the other transforms (including the celebrated Daub 9/7 transform).



(a) Original (b) Daub 9/7

(c) Daub 5/3 Int. to Int. (d) Block-based Daub 5/3 Int. to Int.

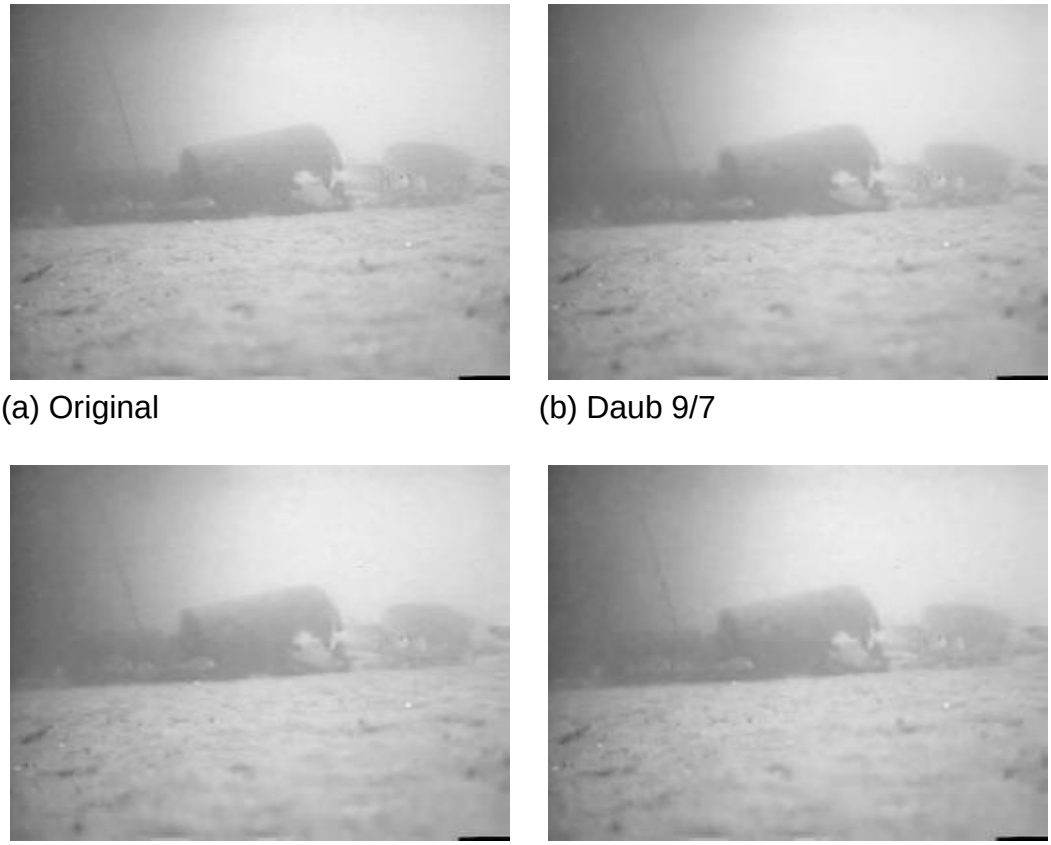
Figure 9: 200:1 compressions of underwater camera image uw4.

Conclusion

We have outlined the basic theory behind wavelet transform image compression and the WDR compression procedure, and compared three different methods. Of these methods, only the one using the block-based Daub 5/3 Int.-to-Int. transform exhibits all of the following advantages:

1. Extremely fast implementation.
2. Low power requirements.
3. Low RAM requirements.
4. Scalability.
5. Region-of-Interest selectivity.
6. Excellent fidelity (even using very high compression ratios).

Thus making it a particularly powerful system for underwater image compression.



(a) Original (b) Daub 9/7

(c) Daub 5/3 Int. to Int. (d) Block-based Daub 5/3 Int. to Int.

Figure 10: 100:1 compressions of underwater camera image uw2.

References

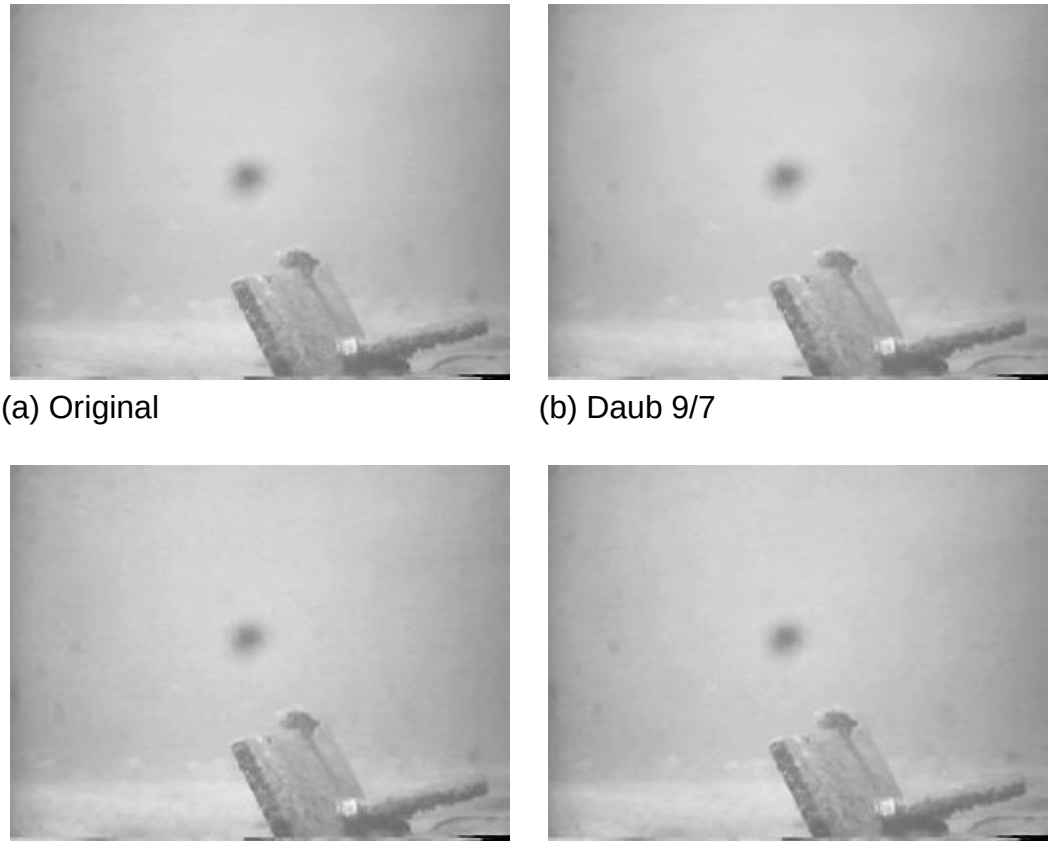
[1] Adams, M. D., and Kossentri, F. Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis. *IEEE Trans. on Image Proc.*, **9,** 1010–1024, 2000.

[2] Calderbank, A. R., Daubechies, I., Sweldens, W., and Yeo, B.-L. Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, **5**, 332-369, 1998.

[3] Cohen, A., Daubechies, I., and Feauveau, J.-C. (1992). Biorthogonal bases of compactly supported wavelets. *Commun. on Pure and Appl. Math.,* **45**, 485–560.

[4] Mallat, S. (1999). *A Wavelet Tour of Signal Processing, Second Edition.* Academic Press, New York.

[5] Nguyen, T. Q., and Walker, J. S. (2001). Wavelet-based image compression. *In* "Handbook of Transforms and Data Compression," Rao, K. R., and Yip, P. C. (Eds.), CRC Press, Boca Raton.



(a) Original (b) Daub 9/7

(c) Daub 5/3 Int. to Int. (d) Block-based Daub 5/3 Int. to Int.

Figure 11: 50:1 compressions of underwater camera image uw5.

[6] Resnikoff, H. L., and Wells, R. O. *Wavelet Analysis. The Scalable Structure of Information.* Springer, New York, NY, 1998.

[7] Shapiro, J. M. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Proc.*, **41**, 3445–3462, 1993.

[8] Strang, G., and Nguyen, T. Q. (1996). *Wavelets and Filter Banks.* Wellesley-Cambridge Press, Boston.

[9] Sweldens, W. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, **3,** 186–200, 1996.

[10] Taubman, D. S., and Marcellin, M. W. (2002). *JPEG2000. Image Compression Fundamentals, Standards, and Practice.* Kluwer, Boston.

[11] Tian, J., and Wells, R. O. Embedded image coding using wavelet-difference-reduction. *Wavelet Image and Video Compression,* P. Topiwala, ed., 289–301. Kluwer Academic Publ., Norwell, MA, 1998.