zero-runs resulting from a wider zero bin, allowing more bits to be allocated to nonzero coefficients, and is also exhibited by the deadzone quantizer. Fig. 6 shows example zero-run length versus $\beta$ for Lena at two compression ratios, and applies to any quantizer with an expanded zero bin. The improvement relative to the deadzone quantizer is due to the ability of the smoothness-constrained quantizer to place the reconstruction levels closer to the centroids of the bins. The maximum PSNR improvement of the smoothness-constrained quantizer relative to the uniform quantizer averages 0.25 dB for most images in the USC image database, with $\beta$ between 1.2 and 1.4. Several results for compression ratios ranging from 16:1 to 128:1 are shown in Table I. Deadzone quantizers also exhibit an increase in PSNR relative to a uniform quantizer, though the improvement is slightly less than that of the smoothness-constrained quantizers (less than 0.1 dB) and the maximum PSNR’s occur at optimal $\beta$’s that are higher, between 1.4 and 1.9.

Though the optimal $\beta$ to produce the maximum PSNR differs for the smoothness-constrained and deadzone quantizers, the change in smoothness as computed by (6) and plotted in Fig. 4 is similar at the optimal $\beta$’s. As such, the visual results for the smoothness-constrained and deadzone quantizers are similar when optimal $\beta$’s are used, and are better than those for the uniform quantizer, as shown in Fig. 7. However, the smoothness-constrained quantizer can produce a slight PSNR improvement and does not require $\beta$ to be known at the decoder.

V. CONCLUSIONS

Changes in visual smoothness caused by wavelet coefficient quantization are quantified mathematically by analyzing changes in the local Hölder regularity. A smoothness-constrained quantizer limits the amount by which the regularity can decrease. Choosing the central quantization bin to be 1.2 to 1.4 times the width of the regular quantization stepsize produces improved visual results as well as an average PSNR increase of 0.25 dB relative to the uniform quantizer. Though a deadzone quantizer can produce similar smoothness decreases though with a larger central bin width, the PSNR performance of the smoothness-constrained quantizer will be slightly higher. The smoothness constraint can be applied to any quantization scheme, including successive approximation quantizers and lattice vector quantizers, with a computationally inexpensive encoder side modification.

REFERENCES


1057–7149/99$10.00 © 1999 IEEE
architectures of both the SPIHT and CREW coders are essentially the same [2]–[4]. The encoder [Fig. 1(a)] first performs a wavelet transform on the image. The resulting coefficients are then repeatedly scanned, first to create a set of symbols corresponding to a signed significance map (e.g., the dominant symbols in [1]) and then to send refinement bits for each coefficient previously determined to be significant (the subordinate symbols). Each time these two steps are performed, a threshold \( T \) is halved so that the next coefficient bit plane will be evaluated (the initial value of \( T \) must be greater than or equal to \( 1/2 \) the magnitude of the largest wavelet coefficient). The symbols created by the scanning process are then losslessly compressed using an arithmetic encoder to generate the actual bit stream [8]. The process continues until either a target bit rate or a target reconstruction error is achieved with the transmission of a stop symbol being required in the latter case. As shown in Fig. 1(b), the decoder simply reverses these steps—it arithmetically decodes the compressed bit stream and then interprets the dominant and subordinate symbols to build up an approximation of the wavelet coefficients. Again, the process continues until either a predetermined number of bits have been read or a stop symbol has been received. At this point, wavelet synthesis is performed to create the output image.

While the "computational complexity" of this compression algorithm is determined entirely by that of its wavelet transform, many noncomputational operations are required to scan the coefficients and thus generate the embedded bit stream. In fact, as the bit rate increases, this scanning complexity grows in an approximately linear fashion. At a bit rate of 1 b/pixel, it requires between eight and twelve scanning passes to compress a typical natural image using an embedded algorithm, and one distinct layer of resolution is generated for each pass. In contrast, a nonembedded coder might require only two sweeps through the wavelet coefficients—one to analyze the coefficients and a second to code them. Depending on the number of levels of error protection available, the typical embedded coder may deliver many more resolution layers than can be effectively used by the channel coder while its nonembedded counterpart cannot deliver enough (i.e., it has only one layer). Our goal, then, is to trade excess layers of embedding for speed. Fig. 2 illustrates the utility of this concept. Here, the Lena image has been compressed with varying degrees of coarseness (e.g., the number of bit planes incorporated into each resolution layer) and at different bit rates. We note that going from a coarseness of one (eight layers) to four (two layers) halves the encoder’s execution time at 1 b/pixel with no adverse effect on the reconstruction quality (given by the peak signal to noise ratio or PSNR) of the decoded image. In Fig. 2, the stated execution times are for the image encoder running on a Unix-based Sparc 10 workstation.

B. Varying Embedding Coarseness

It should first be noted that subordinate symbol generation is performed in almost the same way regardless of the embedding coarseness—if \( N \) bit planes are embedded together, then \( N \) refinement bits are transmitted for each significant coefficient during the subordinate pass. Thus, the process of generating dominant symbols is the key to achieving a large complexity reduction. Shapiro has shown how these symbols can be efficiently generated for conventional EZW [9]. Consider the wavelet coefficient mapping shown in Fig. 3 where frequency increases from the upper left corner toward the lower right corner and the boxed \( z \)'s denote the members of a single zerotree. The first step in Shapiro’s fast encoding technique is to scan the coefficients in this mapping from the highest frequency band (lower right) to the lowest, forming an array that contains the leading bit of the current coefficient’s magnitude OR’d with those of its children—we will call it \( z_{+1}[m] \). Since these values propagate upward (i.e., toward the lower frequencies), one need only AND a value in this array with the threshold \( T \) (assuming it to be a power of two) to determine if the associated wavelet coefficient is a zerotree root. Consequently, if this AND operation results in zero, a zerotree root or ZTR symbol is transmitted for that pass and none of that coefficient’s children are explicitly encoded.

Only minor modifications to this process are required to form the larger symbols associated with a coarsely embedded representation. The initial OR’ing step is unchanged, but to determine coefficient significance at a given power-of-two threshold \( T \) we now AND the array with a new quantity called \( t_{mask} \). To build \( t_{mask} \), we start with \( T \) and set the \((num_{bp}-1)\) bits immediately to the right of its single nonzero entry to one where \( num_{bp} \) is the number of bit planes to be embedded into the current resolution layer. For example, if \( num_{bp} = 3 \) and \( T = 32 \) (0010 0000b), then \( t_{mask} = 56 \) (0011 1000b). Defining \( coef[m] \) as the array containing the
Fig. 3. Partitioning of wavelet subbands. Shade of gray indicates partition mapping and boxed “x’s” denote zerotree relationships.

Assume: $T = 32$, $\text{num}_{\text{bp}} = 3$, $\Rightarrow$ shift = 3, $t_{\text{mask}} = 56$

Cases: a) $\text{cof}[m] = 6$, $z_{\text{m}}[m] = 29$; b) $\text{cof}[m] = -22$, $z_{\text{m}}[m] = 29$
 c) $\text{cof}[m] = 22$, $z_{\text{m}}[m] = 29$; d) $\text{cof}[m] = 6$, $z_{\text{m}}[m] = 7$

Table I

<table>
<thead>
<tr>
<th>sym</th>
<th>sign</th>
<th>bit plane 2</th>
<th>bit plane 1</th>
<th>rsym</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>ZTR</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>0</td>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>6</td>
<td>+</td>
<td>1</td>
<td>0</td>
<td>+2</td>
</tr>
<tr>
<td>7</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>+3</td>
</tr>
</tbody>
</table>

Assume: $T = 32$, $\text{num}_{\text{bp}} = 3$

Cases: a) sym = 1; b) sym = 3; c) sym = 10; d) sym = 0

Fig. 5. Block diagram illustrating symbol decoding process.

TABLE I

<table>
<thead>
<tr>
<th>Symbol Translation Table for $\text{num}_{\text{bp}} = 2$. Bit Plane 2 Is Most Significant</th>
</tr>
</thead>
</table>

Wavelet coefficients and shift as $(\log_2(T) - \text{num}_{\text{bp}} + 1)$, we can form the symbol $\text{sym}$ at location $m$ as shown in Fig. 4. To illustrate this process, the intermediate results for four different sets of coefficient and $z_{\text{m}}[m]$ values are shown in the dashed boxes of the figure; the final symbols output to the arithmetic encoder are also listed. Note that the quantity ‘r’ in the block diagram corresponds to the numeric value being input to that block and processed. Also, the blocks to the left and right of the final adder simply contribute fixed values to the sum whenever they are activated by their input control lines. When a coarse embedding is used, a larger reduction in both $T$ and $t_{\text{mask}}$ is required after each pass. Specifically, after the completion of the dominant pass, we set $T = (T \gg \text{num}_{\text{bp}})$ and $t_{\text{mask}} = (t_{\text{mask}} \gg \text{num}_{\text{bp}})$ where “$\gg$” is the right-shift operator.

Decoding the symbols created by the proposed encoder is easy because of their organized structure. The ZTR symbol is always coded as zero while the isolated zero symbol is always one. Furthermore, the relationship between positive and negative significant symbols is always given by

$$ r_{\text{sym}} = -r_{\text{sym}} $$

where $\text{sym}$ is the actual transmitted symbol and $r_{\text{sym}}$ is the raw symbol which corresponds directly to the signed significance map. Table I shows the symbol translation table when $\text{num}_{\text{bp}} = 2$.

To decode a symbol and form the initial approximation for its corresponding wavelet coefficient $\text{sym}$, we exploit the regular structure of (1) as illustrated by the block diagram in Fig. 5. Here, the first block checks to see if the coefficient is positive significant and, if so, computes its approximation. The second diamond-shaped block tests for negative significance and calculates the coefficient’s approximation if required. The second diamond-shaped block tests for negative significance and calculates the coefficient’s approximation if required. The final test determines if the coefficient is the root of a zerotree—if so, its descendants are not processed during the current pass, and their approximation values remain zero.

To demonstrate the operation of the symbol decoder, we use here the same four test cases that are included in Fig. 4. Again, refinement bits
the coarseness of the embedding could be adjusted in both the C. Adaptation of Embedding work [1], [9].

Note that the processing of the ZTR and isolated zero symbols is also the same regardless of the embedding coarseness. Therefore, we do not discuss it in detail here, referring the reader instead to other work [1], [9].

C. Adaptation of Embedding

In the previous section, we introduced the mechanisms through which the coarseness of the embedding could be adjusted in both the encoder and decoder. Here, we consider how it should be adjusted. Experimentally, we have found that the rate-distortion performance of the system is degraded whenever the encoding terminates in the middle of a coarsely embedded pass. The exact amount of degradation depends largely on two factors: the number of bit planes incorporated into the final resolution layer and where the encoder terminated within the pass. If our goal is fixed rate coding, we cannot control this second factor, but we can alter the first by adapting the embedding process. Specifically, we would like to dynamically alter the number of bit planes incorporated into each layer so as to guarantee that the final (partial) layer contains only one bit plane. For a single still image, this would be very difficult because one does not know a priori how many bit planes will be processed for any given bit rate. To adapt the embedding for video, however, one can assume that the encoder will process the same number of bit planes during the next frame as it did during the current one. Thus, a single byte of side information can be transmitted to the decoder with the current frame, allowing it to synchronize its embedding parameters for the next frame with those of the encoder. While the target embedding coarseness used to generate the results of Section IV is 3-bit planes per layer (bpl), the actual embedding coarseness varies so as to ensure that the last pass is always performed on a single bit plane.

III. CACHE-BASED ZEROTREE PROCESSING

Fig. 6 summarizes our basic cache-optimized processing scheme. First, we perform a wavelet transform on the entire image in exactly the same way as other wavelet-based compression algorithms. Note that the data cache is not a major I/O bottleneck here because the regular structure of the transform calculation allows data access to be very effectively pipelined on most processor architectures. Next, we divide the wavelet coefficients up into to separate groups which we call partitions. This process is shown in Fig. 3 where each shade of gray denotes the partition to which that coefficient is mapped with four different partitions shown here. As one can see, the partitioning

are processed in essentially the same way as with conventional EZW except that multiple bits are received for each significant coefficient. Note that the processing of the ZTR and isolated zero symbols is also the same regardless of the embedding coarseness. Therefore, we do not discuss it in detail here, referring the reader instead to other work [1], [9].

are processed in essentially the same way as with conventional EZW except that multiple bits are received for each significant coefficient. Note that the processing of the ZTR and isolated zero symbols is also the same regardless of the embedding coarseness. Therefore, we do not discuss it in detail here, referring the reader instead to other work [1], [9].

is always performed in such a way that the parent-child relationships required for the zerotree root calculations are preserved—a constraint that limits the minimum partition size to $2^5$ where $S$ is the maximum depth of the wavelet decomposition. The size of each partition is selected to ensure that all of the coefficients in the partition along with additional scratch memory can fit into the on-chip data store or cache. For example, if the original image has 8-b pixels, then the transform coefficients can generally be represented as 16-b integers with precision sufficient for lossy compression. Selecting 5 equal to five (a typical number for EZW compression) implies that 2048 bytes of cache memory are required to store a minimum size partition. In addition, at least three more bits are required for each coefficient as scratch memory, adding another 384 bytes to the total (1024 bytes might actually be used here to avoid the additional operations needed to manipulate single bits). Once the coefficients have been partitioned, each set is coded separately using the EZW algorithm until of its bit allocation is exhausted. Consequently, a separate embedded bit stream is created for each partition resulting in what one might call a piecewise embedded image representation. If a fully embedded bit stream is required, a bit interleaver can be added to the output of the arithmetic encoder without introducing a significant amount of computational overhead but at the expense of system latency: the reconstructed video sequence will be delayed by at least one frame.

Realistically, some portions of an image are more complex than others and should therefore be allocated more bits to improve the overall rate-distortion performance of the system. The challenge is to adjust the bit allocation in such a way that it does not slow down the encoder. Our solution is to adapt the bit allocation for the next frame based on statistics gathered in the current frame (for the first frame of the sequence, we use a uniform bit allocation). After processing a frame, we allocate one more byte to each partition whose reconstruction error was above the median value and take one byte away from partitions with reconstruction errors below the median. Thus, the total bit rate always remains the same from frame to frame. To determine the reconstruction error in each partition, we simply use the total number of bit planes encoded which is exactly equivalent. Fig. 7 shows the organization of the bit steam created for a frame.

IV. RESULTS

For the results shown in Fig. 8, the input is a 30 frame aerial video sequence, and individual fields of size 256 x 512 are coded. To achieve both low complexity and acceptable rate-distortion performance, we use a lifted 5/3 [also called (2,2)] biorthogonal wavelet transform with five scales [10]. In the figure, the mean squared error (MSE) is plotted versus the frame number for two different bit rates and embedding approaches. Our target coarseness for the adaptive embedding is 3-bpl which results in a representation having two to three full layers of embedding as opposed to the six or seven layers produced by conventional EZW. On the average, adaptive embedding actually decreased the average MSE of the sequence slightly compared to EZW (~1.3% and ~0.8% for bit rates of 0.2 and 0.4 b/pixel, respectively), in addition to reducing the encoder’s execution time by 18.5% (0.2 b/pixel) and 26.3% (0.4 b/pixel). A similar decrease also occurs in the decoder’s execution time, although it is somewhat smaller because zerotree decoding is inherently
faster than zerotree encoding. Unless otherwise noted, all speed comparisons have been performed using a Sun Sparc 10 computer running under the Unix operating system.

Fig. 9 illustrates the rate-distortion tradeoffs inherent in cache-based zerotree coding. In this case, a longer portion of the same video sequence is used (200 frames), but the wavelet transform remains unchanged. The compression ratio is fixed at 40:1, resulting in a bit rate of 0.2 b/pixel for each frame. In the figure we compare the performance of five different cases: conventional EZW, one zerotree/partition with fixed uniform bit allocation (one zt/p), two zerotrees/partition with fixed allocation (two zt/p), one zt/p with adaptive allocation (one zt/p,a), and two zt/p with adaptive allocation (two zt/p,a). Clearly, all of the cache-based algorithms performed more poorly than classic EZW. This is to be expected with such small (but realistic) partition sizes. What is impressive, however, is how much the adaptive bit allocation reduces the MSE of the reconstructed sequence. With one zerotree/partition the reduction averages about 25% and with two zt/p it is 21%. The increase in the average MSE of the adaptive coders versus EZW is 31% for one zt/p and 17% for two zt/p. While the MSE’s of the adaptive systems and their nonadaptive
counterparts are initially the same (on the first frame they all use uniform bit allocations), one notes that the adaptive systems quickly converge to their final MSE ranges—in about 50 frames.

For the results shown in Fig. 10, we combine adaptive embedding with the two zt/p,a cache-based coder. Our target for adaptive embedding is again 3-bpl, and the bit rate is fixed at 0.2 b/pixel per frame. We note that adaptive embedding slightly increases the MSE when compared to the same cache-based algorithm implemented using conventional embedding (+2.6%, on the average). This occurs because the adaptive model in the arithmetic encoder does not process enough of the larger coarsely embedded symbols to accurately converge to the source statistics. In terms of encoder speedups versus EZW, cache-based zerotree processing alone results in a 19% speed increase; adding adaptive embedding as well provided an additional 3% increase. Using instead a Texas Instruments 320C80 processor, the combined approach actually increased the encoding speed by 7% over that of cache-based processing alone.

V. CONCLUSIONS

We have discussed here two new methods of speeding up embedded video compression and have found them both to be very effective. By tailoring these methods (alone or in combination) to a processor’s architecture and a communication channel’s requirements, the execution times of both the encoder and decoder can be minimized.

ACKNOWLEDGMENT

The author thanks Dr. C. Podilchuk and the anonymous reviewers for their invaluable comments.

REFERENCES