Embedded Image Coding Using ZeroBlocks of Subband/Wavelet Coefficients and Context Modeling

Shih-Ta Hsiang
Center for Image Processing Research and Engineering, Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute, Troy, NY 12180-3590
E-mail: hsiang@ieee.org

Abstract

In this paper, we present a new embedded wavelet image coding system using quadtree splitting and context modeling. It features low computational complexity and high compression efficiency, thanks to jointly utilization of two powerful embedded coding techniques — set partitioning and context modeling. With effective exploitation of strong statistical dependencies among the quadtree nodes built up from subband coefficients, the proposed algorithm substantially improves the coding efficiency of the existing set-partition coders for both lossy and lossless image compression. For example, our experimental results show that the new algorithm respectively outperforms zerotree-based SPIHT and quadtree-based SPECK by 0.7 dB and 1.0 dB at 0.5 bpp on average.

I. INTRODUCTION

With desirable features such as SNR scalability and exact rate control, embedded wavelet image coding, pioneered by Shapiro [1], has attracted a lot of research interests in recent years. Due to the high energy compaction nature of the transform coding, it is typical that a large region of transform coefficients are quantized to zero after coarse quantization. Hence, some special zero coding schemes can substantially improve the compression efficiency of a transform coding system. For embedded wavelet image coding, two techniques — hierarchical set partitioning and context modeling of subband coefficients — have been found particularly successful.

The conventional set partition coders [2], or widely known as zerotree coders [1], utilizes the zerotree structure to take advantage of the nature of energy clustering of subband/wavelet coefficients in both frequency and space. For this class of coders, a group of zero pixels are indicated by a special zero-root symbol. During bitplane coding process, a hierarchical set partitioning rule is employed to split off significant pixels (with respect to the threshold in the current bitplane coding pass), while maintaining areas of insignificant pixels as collections of zero-roots and isolated zero pixels. In this way, a group of leading zeros of subband/wavelet coefficients can be compactly coded. Moreover, instead of all subband coefficients, only a small number of elements in maintained lists [2] need to be processed in individual bitplane coding passes. Hence, processing speed for this class of coders is very fast, particularly at low bit rate. The distinguished compression and speed performances is demonstrated by well-respected embedded wavelet coder SPIHT [2].

Recently, some quadtree-based algorithms, also known as zeroblock coders, have been proposed in literature [3], [4], [5] for wavelet image coding. The core of this approach is also based on the idea of hierarchical set partitioning, as illustrated in Fig. 1. The bottom quadtree level, or pixel level, consists of magnitudes of all subband
coefficients. Each quadtree node of the next higher level is then set to the maximum value of its four child nodes, as illustrated in Fig. 1 (a). By recursively grouping each $2 \times 2$ vector this way, the number of pixels covered by one quadtree node exponentially grows with its level number. To locate a significant pixel, we can perform quadtree splitting recursively up to the bottom quadtree level, as shown in Fig. 1 (b). An active area can be fast zoom in through several quadtree splitting steps. Given the coded quadtree splitting information, the same significance map can be reproduced by the decoder. The excellent performance for embedded image coding using this scheme is presented in [5].

Embedded wavelet image coding based on context modeling was first presented by Taubman and Zakhor in LZW [6]. In this scheme, individual pixels of subband bitplane are coded using context-based arithmetic coding. With carefully designed context models, strong correlation of subband coefficients subbands can be effectively exploited. Although some simple context modeling schemes are also employed in [1], [2] and most other zerotree coders, the limited context information utilized in those algorithms are insufficient to accurately predict the status of the current node. On the other hand, with sophisticated context modeling scheme, some algorithms in literature [7], [8] have been able to outperform the best zero-tree/block coders in PSNR performances. Nevertheless, these algorithms are typically based on the conventional sequential approach for bitplane coding. That is, all subband coefficients are processed one by one following a predefined scanning order, raster-scan for instance. As a result, unlike zerotree coders, these coders typically need to process all sub-band/wavelet coefficients at least once to finish coding of a full bitplane, hence with an implied higher computational cost.

Given the distinctive features of these two zero coding schemes, it is certainly desirable to have a unified framework which combines these two techniques and takes advantage of their respective strength at the same time. This paper presents a new embedded wavelet image coding system EZBC (Embedded coding using ZeroBlocks of wavelet coefficients and Context modeling). The framework for bitplane processing is also based on the core idea of set partitioning. Nevertheless, we adopt the emerging quadtree-based approach rather than the conventional zerotree-based method.

EZBC is motivated by the experimental observation that strong dependency exists not only among wavelet coefficients but also among nodes at individual quadtree levels within and across subbands. Fig. 2 displays the MSB (Most Significant Bit) maps of the quadtrees built up from individual subbands of the decomposed Lena image. Interesting enough, another pyramidal image description is provided by the individual quadtree levels, in addition to the original subband pyramid generated by wavelet transform. Self-similarity across both resolution and quadtree levels is clearly shown. The image features such as edges and contours can be easily identified in different hierarchies of the wavelet transform domain. Such strong dependency is exploited by context-based arithmetic coding in EZBC. With special care given to the design of context modeling scheme, we demonstrate the compression performance of the set partition coder can be substantially improved.

The following advanced coding techniques are efficiently combined by EZBC:
- Quadtree-based set partitioning scheme for compact, efficient and flexible data representation;
- Carefully designed context modeling scheme for entropy coding;
- Efficient sub-bitplane interleaving for improved R-D performance;
- Context-based dequantization to further exploit the collected source statistics accumulated during the decoding process.

The remainder of the paper is organized as follows. The next section details our algorithm for processing of subband coefficient bitplanes. The context modeling scheme is described in Section III. Our dequantization strategy is introduced in Section IV. Section V presents the experimental evaluation of the proposed algorithm for lossy, lossless and resolution scalable coding. The conclusions are given finally.

II. Bitplane Processing

The bitplane coding process begins with establishment of the quadtree representations for individual subbands. The bottom quadtree level, or pixel level, consists of magnitude of each subband coefficient. Each quadtree node of the next higher level is then set to the maximum value of its four corresponding nodes at the current level, as illustrated in Fig. 1 (a). By recursively grouping each $2 \times 2$ vector this way, the top quadtree node in the end just corresponds to the maximum magnitude of all the coefficients from the same subband.

Similar to conventional bitplane coders, we progressively encode subband coefficients from the most significant bitplane toward the least significant bitplane. Such
a bitplane coding method is equivalent to the successive approximation quantization scheme with threshold 2^n for coefficient bitplane level n. Each bitplane pass performs two basic operations:

- Test all the insignificant quadtree nodes against the current thresholds;
- Refine coefficient values for all the coded pixels from the previous passes.

A quadtree node tests significant if it is greater or equal to the current bitplane threshold 2^n. Once testing significant, it is split into four descendent nodes. Such testing and splitting procedure is recursively performed until the bottom (pixel) level, as shown in Fig. 1 (b). Whenever a pixel tests significant, its sign is coded immediately. Significance test, sign and refinement bits are all coded by context-dependent arithmetic coding, to be detailed in next section.

Similar to SPIHT and other hierarchical bitplane coders, we use lists for tracking the set-partitioning information. Two arrays of lists, LIN and LSP, are defined for each subband. A quadtree node, QT_k[l](i, j), from subband k and quadtree level l is added to the end of LIN_k[l] once it tests insignificant. LSP_k contains a full list of significant pixels from subband k.

To have efficiently embedded bitstream, it is imperative that the coded data units are embedded in the order of their importance. This concept is generally referred as embedding principle and is investigated in [9], [10], [11]. In each bitplane coding pass, quadtree nodes in LIN_k[l] are visited from the bottom level to the top level, a similar order to SPECK. The coefficient refinement are executed last. Within the same quadtree level, the subbands are coded from coarse to high resolutions. Although such embedding order is not explicitly optimized for the best R-D performance, our empirical data show the relative performance loss are actually not significant. The effectiveness of this embedding scheme is evidenced by the smooth R-D curve later presented in our experimental results. With assistance of the list facility, EZBC does not need to scan individual pixels more than once during each bitplane pass, in clear contrast to conventional sequential bitplane coders. The price to pay is relatively high memory usage.

The complete algorithm for bitplane processing is presented as follows.

A. Algorithm

A.1 Definition

- c(i, j), m(i, j): quantized subband coefficient and its MSB at position (i, j).
- QT: quadtree representation for coefficients from the same subband with QT_k[l](i, j) corresponding to a quadtree node at position (i, j) band k, level l.

\[
\begin{align*}
QT_k[0](i, j) & \equiv |c_k(i, j)| \\
QT_k[l](i, j) & \equiv \max\{QT_k[l-1](2i, 2j), QT_k[l-1](2i, 2j+1), \\
& QT_k[l-1](2i+1, 2j), QT_k[l-1](2i+1, 2j+1)\}
\end{align*}
\]

- D_k, D_{max}: depth of the quadtree for subband k and the maximum quadtree depth
- K: total number of subbands
- LIN_k[l]: list of insignificant nodes from quadtree level l of subband k
- LSP_k: list of significant pixels from subband k
• $S_n(i, j)$: significance test of node $(i, j)$ at bitplane $n$

$$S_n(i, j) \equiv \begin{cases} 1, & \text{if } n \leq m(i, j), \\ 0, & \text{otherwise}. \end{cases}$$

A.2 Coding Steps

1. initialization:

\[
\begin{align*}
\text{LIN}_k[l] &= \begin{cases} (0, 0), & l = D_k, \\ \emptyset, & \text{otherwise,} \end{cases} \\
\text{LSP}_k &= \emptyset, \quad \forall k, \\
\theta &= \left\lfloor \log_q \left( \max \{ |c_k(i, j)| \} \right) \right\rfloor; \\
\end{align*}
\]

2. for $l = 0 : D_{\text{max}}$

   for $k = 0 : K - 1$, CodeLIN($k, l$);

3. for $k = 0 : K - 1$, CodeLSP($k$);

4. decrement $\theta$ and go back to step 2.

A.3 Pseudo Code

CodeLIN($k, l$)

• for each $(i, j)$ in LIN$_k[l]$,
  - code $S_n(i, j)$;
  - if $(S_n(i, j) = 0)$, node $(i, j)$ remains in LIN$_k[l]$;
  - else
    • if $(l = 0)$, then code sign bit of $c(i, j)$ and add $(i, j)$ to LSP$_k$;
    • else CodeDescendants($k, l, i, j$).

CodeDescendants($k, l, i, j$)

• for each node $(x, y)$ in \{(2i, 2j), (2i, 2j + 1), (2i + 1, 2j), (2i + 1, 2j + 1)\} of quadtree level $l - 1$, band $k$
  - code $S_n(x, y)$;
  - if $(S_n(x, y) = 0)$, add $(x, y)$ to LIN$_k[l - 1]$;
  - else
    • if $(l = 1)$, code the sign bit of $c_k(x, y)$ and add $(x, y)$ to LSP$_k$;
    • else CodeDescendants($k, l - 1, x, y$).

CodeLSP($k$)

• for each pixel $(i, j)$ in LSP$_k$, code bit $n$ of $|c_k(i, j)|$.

III. CONTEXT-BASED ARITHMETIC CODING

In contrast to the conventional pixel-wise bitplane coders, EZBC deals with nodes from individual quadtree levels. As shown earlier in Fig. 2, strong statistical de-
dependencies exist among bitplanes, resolution scales, and quadtree levels. Such dependencies are exploited by context-based arithmetic coding. Four types of binary symbols are coded in EZBC: (1) significance of nodes from LIS (in routine CodeLIN); (2) significance of descendants (in routine CodeDescendants); (3) sign of significant coefficients (4) coefficient refinement. Unlike most other set-partition coders, the lists in EZBC are separately maintained for individual subbands and quadtree levels. Therefore, the separate context models are allowed to be built up for the nodes from different subbands and quadtree levels. In this way, distinctive statistical characteristics of quadtree nodes from different orientations, subsampling factors and amplitude distributions will not be mixed up. Whereas, in the actual 3D-EZBC implementation, the same set of context models are shared among quadtree level 2 and upper to reduce the actual model cost.

Fig. 3 illustrates the context for conditional entropy coding of significance test. To utilize the intraband correlation, we include eight nodes of the first-order neighbor from the same quadtree level. This spatial context has been commonly used in many conventional bitplane coders, e.g. [9], [11]. Whereas, for the information across scales, the node from the parent subband of the next lower quadtree level is used. This choice is made because at the same quadtree level the dimension of the corresponding region, with respect to the input image, of each node is doubled in the parent subband as a result of subsampling during the transform stage.

Statistics for coding significance test of individual descendent nodes (conducted in routine CodeDescendants) is also dependent on their relative positions in their corresponding 2 x 2 blocks, as illustrated in Fig 4. It is directly attributed to the mechanism of quadtree buildup and decomposition — Once a parent node tested significant, at least one of its four immediate descendents is also significant. Hence, if no siblings (descendent nodes from the same 2 x 2 region) have yet tested significant,
the chance of testing significant becomes higher for the next descendant to be tested. If all three earlier scanned siblings are insignificant, the fourth node is significant for sure and no bit is coded. To take advantage of this statistical characteristics, significance coding of descendants is additionally conditioned on the relative position of the current descendant.

Rather than all coded bitplanes, we only utilize the binary valued significance map \( \sigma_n(i, j) \) for context modeling, where \( \sigma_n(i, j) = 1 \) if node \((i, j)\) has become significant, \( \sigma_n(i, j) = 0 \) otherwise. This scheme is equivalent to performing context quantization [12] with the quantizer step size equal to the threshold of the current bitplane coding pass. The model adaptation cost can thus be substantially reduced. To further improve models efficiency, configurations of \( \sigma_n \) in the context region are grouped into several model classes based on their statistical features. The look-up tables which map the configurations to the corresponding model labels are established accordingly.

This strategy for efficient context selection is first presented in EBCOT [11] and is already included in the new international standard, JPEG 2000 [13]. The look-up tables used by EZBC can be found in [14].

We also adopt the similar approach to EBCOT and JPEG 2000 for sign coding. The sign bit for the current pixel is predicted from sign and significance of its eight immediate neighboring pixels. Rather than the sign bit itself, the correctness of prediction is entropy coded. Coefficient refinement is also entropy coded and is conditioned on configuration of \( \sigma_n \) and \( \sigma_{n-1} \) in the context.

IV. CONTEXT-BASED DEQUANTIZATION

It is known the optimal representation levels of a scalar quantizer are equal to the centroids of their corresponding decision intervals. However, reconstruction values are typically set to the midpoints of the quantization intervals in practice either for simplicity of implementation or for lack of knowledge about the source statistics. Such a choice is optimal for symbols with uniform probability distribution over the individual decision intervals. For bitplane coding in particular, it is implicitly assumed that the positive and negative coefficients within the dead zone are equally likely and the reconstructed values of all the insignificant pixels are set to 0 as a result.

The proposed dequantization algorithm is based on the two empirical observations:
1) Subband coefficients exhibit strong spatial correlation for both their sign and magnitude.
2) The bitplane statistics bear resemblances from plane to plane and grow into less skewed probability distribution gradually.

Observation 1 has been utilized in EZBC for context-based arithmetic coding of the subband coefficients. Similarly, those coded significant pixels can provide context information for reconstruction of their neighboring coefficients. Especially, the sign of significant pixels are useful for predicting the sign of their neighboring insignificant pixels (which are quantized to zero.) Coupled with observation 2, we can approximate the source statistics for the given context from the corresponding probability models accumulated during decoding process.

The proposed dequantization algorithm is partially inspired by [15], which requires to compute a partial LL band reconstruction during each bitplane pass. Unlike their work, our algorithm only utilized the statistical information accumulated at the de-
coder. Those insignificant pixels that do not have any significant neighbors are still quantized to zero. Non-zero pixels can be easily located in already maintained lists. Hence, additional computational cost is actually minimum. The resulting PSNR improvement ranges from 0 - 0.4 dB with an average around 0.1 dB. Due to the space limitation, the interested readers are referred to [14] for more detail.

V. Experimental Results

In this section, we evaluate the coding performance of EZBC for both lossy and lossless image compression. The comparisons are made to several state-of-the-art image coders. All experimental results presented here are obtained by actually running the codec softwares under evaluation, except for those of SPECK which are from [5]. Ten standard test images are utilized in simulations. The popular images Lena, Barbara, and Goldhill are from the USC image data base. The other are the standard JPEG 2000 test images. All of them are in gray-scale.

A. Lossy Coding

EZBC is compared with two highly regarded zero-tree/-block coders, SPIHT and SPECK (using arithmetic coding for both), and with the JPEG 2000 test coder(J2000) in generic scalable (GS) mode [16]. The dyadic wavelet decomposition with Daubechies 9/7 filters was adopted for all these coders. The PSNR results for coding images Cafe, Bike, and Woman over a wide range of bitrates are shown in Tables I along with the average PSNR performance differences between each coder and EZBC. The average PSNR performance differences of all test images are also summarized in Tables I. Rate-distortion comparison of EZBC, SPIHT and the JPEG 2000 test coder is displayed in Fig. 5 for image Hotel.

![Rate-Distortion performance comparison for EZBC, SPIHT, and JPEG 2000 coders, Hotel.](image_url)

B. Lossless Coding

When RWT (reversible wavelet transform) is employed for subband transform, EZBC can encode/decode the image up to the lossless quality levels. Table II sum-
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0.0625</th>
<th>0.125</th>
<th>0.25</th>
<th>0.5</th>
<th>1.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cafe</td>
<td>EZBC</td>
<td>19.23</td>
<td>21.02</td>
<td>23.61</td>
<td>27.36</td>
<td>32.64</td>
<td>39.79</td>
</tr>
<tr>
<td></td>
<td>J2000/GS</td>
<td>19.03</td>
<td>20.73</td>
<td>23.09</td>
<td>26.74</td>
<td>31.87</td>
<td>39.00</td>
</tr>
<tr>
<td></td>
<td>SPIHT</td>
<td>18.95</td>
<td>20.67</td>
<td>23.03</td>
<td>26.49</td>
<td>31.74</td>
<td>38.91</td>
</tr>
<tr>
<td></td>
<td>SPECK</td>
<td>18.93</td>
<td>20.61</td>
<td>22.87</td>
<td>26.31</td>
<td>31.47</td>
<td>38.75</td>
</tr>
<tr>
<td>Bike</td>
<td>EZBC</td>
<td>23.82</td>
<td>26.54</td>
<td>29.87</td>
<td>33.77</td>
<td>38.39</td>
<td>44.51</td>
</tr>
<tr>
<td></td>
<td>J2000/GS</td>
<td>23.73</td>
<td>26.27</td>
<td>29.53</td>
<td>33.57</td>
<td>38.00</td>
<td>43.96</td>
</tr>
<tr>
<td></td>
<td>SPIHT</td>
<td>23.44</td>
<td>25.89</td>
<td>29.12</td>
<td>33.01</td>
<td>37.70</td>
<td>43.80</td>
</tr>
<tr>
<td></td>
<td>SPECK</td>
<td>23.31</td>
<td>25.59</td>
<td>28.84</td>
<td>32.69</td>
<td>37.33</td>
<td>43.10</td>
</tr>
<tr>
<td>Woman</td>
<td>EZBC</td>
<td>25.80</td>
<td>27.75</td>
<td>30.47</td>
<td>34.25</td>
<td>39.00</td>
<td>44.76</td>
</tr>
<tr>
<td></td>
<td>J2000/GS</td>
<td>25.54</td>
<td>27.31</td>
<td>29.94</td>
<td>33.52</td>
<td>38.31</td>
<td>43.96</td>
</tr>
<tr>
<td></td>
<td>SPIHT</td>
<td>25.43</td>
<td>27.33</td>
<td>29.95</td>
<td>33.59</td>
<td>38.28</td>
<td>43.99</td>
</tr>
<tr>
<td></td>
<td>SPECK</td>
<td>25.50</td>
<td>27.34</td>
<td>29.88</td>
<td>33.46</td>
<td>38.07</td>
<td>43.73</td>
</tr>
<tr>
<td>Average Differences (Cafe+Bike +Woman)</td>
<td>EZBC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>J2000/GS</td>
<td>-0.18</td>
<td>-0.33</td>
<td>-0.46</td>
<td>-0.58</td>
<td>-0.62</td>
<td>-0.71</td>
</tr>
<tr>
<td></td>
<td>SPIHT</td>
<td>-0.34</td>
<td>-0.47</td>
<td>-0.62</td>
<td>-0.76</td>
<td>-0.77</td>
<td>-0.79</td>
</tr>
<tr>
<td></td>
<td>SPECK</td>
<td>-0.37</td>
<td>-0.59</td>
<td>-0.79</td>
<td>-0.97</td>
<td>-1.05</td>
<td>-1.16</td>
</tr>
<tr>
<td>Average Differences (All)</td>
<td>EZBC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>J2000/GS</td>
<td>-0.43</td>
<td>-0.53</td>
<td>-0.52</td>
<td>-0.54</td>
<td>-0.64</td>
<td>-0.80</td>
</tr>
<tr>
<td></td>
<td>SPIHT</td>
<td>-0.38</td>
<td>-0.54</td>
<td>-0.62</td>
<td>-0.70</td>
<td>-0.73</td>
<td>-0.82</td>
</tr>
</tbody>
</table>

Table I summarizes the coding results for lossless image compression. The coding results for the three scalable coders are shown on the first three columns. It is shown that EZBC outperforms the other two for all the images tested.

The coding results for three representative lossless image coders are shown on the last three columns as a comparison to the more traditional lossless image coding method. These three coders are all based on the classical predictive approach and their codestreams are not scalable.

VI. SUMMARY AND CONCLUSIONS

This work presents our new embedded wavelet image coder EZBC. We demonstrate that compression efficiency of the set-partition coders can be substantially improved once the context information is effectively exploited. Unlike the conventional set partition coders allowing zerotrees spanning several subbands of the different resolutions, zero coefficients from the same block region associated with the individual quadtree node are all within the same subband. Hence, the zero block coders are more efficient for some desirable applications such as resolution scalability. Interband dependency is still made good use via context modeling in EZBC.

REFERENCES


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Barbara</td>
<td>4.711</td>
<td>4.701</td>
<td>4.493</td>
<td>5.663</td>
<td>4.626</td>
<td>4.863</td>
</tr>
<tr>
<td>Cafe</td>
<td>5.277</td>
<td>5.364</td>
<td>5.112</td>
<td>5.865</td>
<td>4.925</td>
<td>5.091</td>
</tr>
<tr>
<td>Woman</td>
<td>4.419</td>
<td>4.504</td>
<td>4.276</td>
<td>5.034</td>
<td>4.287</td>
<td>4.451</td>
</tr>
<tr>
<td>Aerial?</td>
<td>5.331</td>
<td>5.441</td>
<td>5.188</td>
<td>5.589</td>
<td>5.177</td>
<td>5.289</td>
</tr>
<tr>
<td>Hotel</td>
<td>4.579</td>
<td>4.616</td>
<td>4.374</td>
<td>5.063</td>
<td>4.249</td>
<td>4.382</td>
</tr>
<tr>
<td>Finger</td>
<td>5.597</td>
<td>5.545</td>
<td>5.286</td>
<td>5.846</td>
<td>5.455</td>
<td>5.663</td>
</tr>
<tr>
<td>US</td>
<td>3.156</td>
<td>3.239</td>
<td>3.096</td>
<td>3.317</td>
<td>2.562</td>
<td>2.630</td>
</tr>
<tr>
<td>Mean</td>
<td>4.653</td>
<td>4.711</td>
<td>4.488</td>
<td>5.119</td>
<td>4.421</td>
<td>4.568</td>
</tr>
</tbody>
</table>


