Abstract

This paper presents an embedded and memory efficient image compression algorithm which exploits both inter and intra-band correlation of wavelet coefficients. The Set Partitioning In Hierarchical Tree (SPIHT) is a zero-tree based coder which exploits inter-band correlation among the bands of same orientation, while the Set-Partitioning Embedded Block Coder (SPECK) is a zero-block based coder which exploits intra-band correlation. However, they have extensively large memory requirement due to the use of three/two linked lists whose entries increase from one-bit-plane to the next. We propose an algorithm that is based on block-set partitioning and quad-splitting using two re-usable lists. The main list is initialized at the beginning of each bit-plane and is exhausted within the same bit-plane. This makes our proposed algorithm highly memory efficient. The experimental results show that the compression efficiency of proposed method is comparable to any state-of-art image coder while reducing the memory requirement by 50-60% in comparison to the SPIHT algorithm.

1. Introduction

Ever since the pioneer paper on the Embedded Zero-tree Wavelet (EZW) [1] coding algorithm, many of its extensions and improvements have been suggested. SPIHT [2] is the most successful improvement over EZW and has become a standard benchmark in the field of image compression. The wavelet image coders developed in the last few years can be divided into two groups; hierarchical tree (or zero-tree) encoders [1,2] and zero-block encoders [3,4]. The zero-tree coders exploit the inter-band correlation among the subbands of the same orientation while the zero-block coders are based on inter-subband correlation, e.g. SPECK [3]. Both SPIHT and SPECK use pixel/set lists, which significantly increase the memory requirement. The Embedded Zero Block Coding (EZBC) [4] is based on the SPECK algorithm, but uses context-based adaptive arithmetic coding to improve its performance. However, it has much higher complexity as compared to SPECK. EBCOT [6] restricts the memory usage through encoding of pixel blocks but is of higher complexity due to the use of adaptive arithmetic coding, multiple coding pass & use of rate distortion optimizers.

In this paper we propose a novel algorithm for image compression with reduced memory requirement while retaining all the desirable features of both zero-tree and zero-block coding algorithms. The proposed coder referred as Embedded Zero Block Tree Coder (EZBTC), is based on the significance testing of block-trees (all descendent blocks of a root block) and quad-tree splitting within a block. Though it combines the features of SPIHT and SPECK algorithms, but follows an altogether a different approach. Both SPECK and SPIHT use linked lists, whose elements from the previous bit-plane are used as reference in the next bit-plane, whereas in our approach encoding of each bit-plane is independent of that of previous bit-plane(s) and the lists are re-initialized for every bit-plane encoding. Another difference is that both SPIHT and SPECK execute two passes (sorting and refinement) for coding of each bit-plane, whereas proposed EZBTC algorithm uses only a single pass. The most attractive feature of our coder is that it uses two re-usable lists. In one of the lists, entries are re-initialized in each bit-plane, whereas second list is used only for quad-tree partitioning and block coding. This is contrary to the SPIHT and SPECK algorithms. A similar attempt to use the features of both SPIHT and SPECK is made by Wheeler and Pearlman [5]. They divided transformed subbands into spatial and subband blocks, which were encoded independently by SPIHT and SPECK respectively. But it no longer retains the progressiveness property and requires rate-distortion optimization.

The remainder of the paper is organized into the following sections. Section 2 describes the details of EZBTC algorithm. Section 3 compares the memory requirements of the proposed EZBTC algorithm with...
that of the SPIHT algorithm. Experimental results are presented in Section 4 followed by concluding statements and future scope in Section 5.

2. Proposed Algorithm

The main purpose of the proposed Embedded Zero-Block-Tree (EZBTC) algorithm is to exploit inter-band correlation similar to SPIHT along with intra-band correlation similar to SPECK in such a way that both these algorithms are fused together while retaining the progressiveness of the generated bit-stream with least memory requirements.

2.1. Nomenclatures

Following notations and nomenclatures are used in this algorithm.

- $I$: image of size $M \times N$.
- $N_d$: number of wavelet decomposition level.
- $c_{i,j}$: wavelet coefficient with coordinates $(i, j)$, $1 \leq i \leq M$ and $1 \leq j \leq N$.
- $B_{k,l}^{m,n}$: an arbitrary block that consists of set of indexed wavelet coefficients $\{c_{i,j}\}_{k \leq i < k + m, \ l \leq j < l + m}$, where $(k, l)$ being top left corner coordinate of the block.
- $B_{0,0}^{m,n}$: block containing LL-subband of size $m \times n$,
  \[ \{m = M/2^N & n = N/2^N \} \]
- $Q_{k,l}^{m,n}$: set of quad-blocks of parent block $B_{k,l}^{m,n}$, i.e.,
  \[ \{B_{k,l/2}^{m/2,n/2}, B_{k,(l/2)+1}^{m/2,n/2}, B_{k+l/2}^{m/2,n/2}, B_{k+l}^{m/2,n/2} \} \]
- $O_{k,l}^{m,n}$: set of offspring blocks of $B_{k,l}^{m,n}$, i.e.,
  \[ \{B_{2k,2l}^{m,n}, B_{2k+1,2l}^{m,n}, B_{2k,2l+1}^{m,n}, B_{2k+1,2l+1}^{m,n} \} \]
- D($B$): set of all descendant blocks of root block $B$.
- SBT($B$): block tree significance test function defined as
  \[ SBT(B) = \begin{cases} 1 & \text{if } \max(|c_{i,j}|) \geq 2^r, c_{i,j} \in D(B) \\ 0 & \text{otherwise} \end{cases} \]
  (where $r$ is an integer)
- SB($B$): block significance test function
  \[ SB(B) = \begin{cases} 1 & \text{if } \max(|c_{i,j}|) \geq 2^r, c_{i,j} \in B \\ 0 & \text{otherwise} \end{cases} \]

The proposed algorithm uses two ordered auxiliary lists, which are initialized at the beginning of encoding of each bit-plane and are used for that bit-plane only. This is contrary to SPIHT that uses three lists initialized at the beginning and appended for encoding of every next bit-plane.

Description of lists used in our algorithm is as follows:

1. **LIBS** (List of Insignificant Block Sets) contains insignificant block sets of varying sizes with one or more member. This list is initialized with three lower quads of LL-subband at the beginning of each bit-plane (encoding pass).

2. **LQB** (List of Quad partitioned Blocks) contains the addresses of blocks while performing quad partitioning. This list is used only when a significant block is to be partitioned and coded. Every time a significant block is coded, LQB is initialized with addresses of quads of that block and updated with addresses of new partitioned blocks. After block coding is completed, the entries of LQB with significant descendents are transferred to LIBS. Thus LQB may be re-initialized and re-used many times within a bit-plane. Entries in LIBS and LQB may be of type A or B and coded differently in each case.

Within each bit-plane, after initialization the encoder algorithm exploits tree block hierarchy before recursive quad partitioning at block level until significant coefficients are found in a block.

A complete description of algorithm is as follows:

**2.2. Algorithm overview**

1. **Initialization**:
   1.1 Output $r = \left\lfloor \log_2|c_{i,j}| \right\rfloor \forall i, j \in I$;
   1.2 Set $\text{LQB}=\text{NULL}$; add $B_{0,0}^{m/2,n/2}$, $B_{m/2,2,n/2}$ and $B_{m/2,n/2}$ to LIBS as entries of type ‘A’.

2. **Sorting and Refinement pass**:
   - For each element $B_{k,l}^{m,n}$ in LIBS;
     - if type ‘A’
       - output $\text{STB}(B_{k,l}^{m,n})$;
       - if $\text{STB}(B_{k,l}^{m,n})=1$
         - replace $B_{k,l}^{m,n}$ by $B_{2k,2l}^{m,n}$ as type ‘B’ in LISB.
     - if type ‘B’
       - output $\text{SB}(B_{k,l}^{m,n})$;
       - if $\text{SB}(B_{k,l}^{m,n})=1$
         - add $Q_{k,l}^{m,n}$ into LQB as entries of type ‘B’.
     - call $	ext{codec_block}()$.
   - else replace $B_{k,l}^{m,n}$ by $B_{2k,2l}^{m,n}$ as type ‘B’ entry in LIBS.
3. Quantisation step:
   decrement \( r \) by 1 and go to step 1.2.

```c
codec_block()
{
1. for each entry of type ‘B’ in LQB;
   o if (m=1) && (n=1)
      output \( SB(B^{m,n}_{k,j}) \) and change its
      type as ‘A’;
   o if \( SB(B^{m,n}_{k,j})=1 \), output sign bit of
      \( B^{m,n}_{k,j} \) and subtract current threshold
      from the magnitude of \( B^{m,n}_{k,j} \).
   o else
      output \( SB(B^{m,n}_{k,j}) \).
   if \( SB(B^{m,n}_{k,j}) =0 \), change entry as type
      ‘A’;
2. for all \( B^{m,n}_{k,j} \in LQB \),
   if \( O^{m,n}_{k,j} \neq NULL \) && \( SB(O^{m,n}_{k,j}) \neq 0 \), then
   append LIBS with all ‘A’ type entries of LQB.
3. reset LQB to NULL;
}
```

3. Memory Requirements

In this section we will compare the memory requirements for the auxiliary lists in EZBTC with that of SPIHT. At any stage, the size of the memory needed depends upon the number of entries in the corresponding lists. The comparisons are made in terms of the memory requirements at the end of each pass (or bit-plane) as well as the maximum (or worst case) memory requirements.

3.1. SPIHT

In SPIHT three linked lists are used namely; LIP, LSP and LIS. Each entry in LIP and LSP is a single coordinate of a wavelet coefficient whereas LIS also requires type (A or B) information to distinguish nodes.

Let:

- \( N_{LIP} \) = number of entries in LIP.
- \( N_{LSP} \) = number of entries in LSP.
- \( N_{LIS} \) = number of entries in LIS.
- \( c \) = number of bits to store addressing information of a coefficient.

\( M_{SPIHT} \) = total memory required in SPIHT (in bits). Then

\[
M_{SPIHT} = c \cdot N_{LIP} + (c+1) \cdot N_{LIS} + c \cdot N_{LSP}
\]

(1)

Where an extra bit per element in LIS is needed for defining the type of entry.

In the worst case,

\[
N_{LIP} + N_{LSP} = MN
\]

\[
N_{LIS} = MN/4
\]

(coefficients having no descendents (highest frequency subbands) will never enter into LIS.). Thus

\[
\max(M_{SPIHT}) = (5c + 1) \frac{MN}{4}
\]

(2)

3.2. EZBTC

In our proposed algorithm, we have used only two lists namely LIBS and LQB. In each list, as entries correspond to address of blocks, so each element needs twice as much memory as in SPIHT. However, since the maximum number of entries in each of LIBS and LQB will only be a quarter of the total number of coefficients, the effective number of bits per elements will be \( 2c-1 \). Further an extra bit is needed for identifying the ‘type’ of elements in the lists.

If \( N_{LIBS} \) and \( N_{LQB} \) be the numbers of entries in LIBS and LQB respectively and \( M_{EZBTC} \) denotes the total memory required in EZBTC (in bits), then

\[
M_{EZBTC} = (2c-1) (N_{LIBS} + N_{LQB})
\]

(3)

In the worst case,

\[
N_{LIBS} = N_{LQB} = MN/4
\]

Thus,

\[
\max(M_{EZBTC}) = (2c - 1) \frac{MN}{4}
\]

(4)

The worst case Memory Reduction Factor (MRF) over SPIHT is

\[
MRF = \frac{5c + 1}{2c - 1}
\]

(5)

For example, for any \( 512 \times 512 \) image, \( c = \log_2(512) = 9 \), MRF=2.71, i.e. EZBTC has reduced the maximum working memory requirement by a factor of 2.71 over the SPIHT algorithm.

4. Simulation Results

In order to compare the performance (memory requirements and coding efficiency) of EZBTC, experiments are performed for the \( 512 \times 512 \) grayscale Lena and Barbara images. For these experiments, we used a five level decomposition with the 9/7 biorthogonal filter.

4.1 Memory Analysis:

The bit-plane wise memory requirement of EZBTC and SPIHT for the first nine passes of
‘LENA’ image is compared in Fig. 1. At the end of each pass, based on the number of entries in the corresponding lists, $M_{\text{SPIHT}}$ and $M_{\text{EZBTC}}$ are evaluated according to Eqn. 1 and 3 respectively and are plotted in this figure. It can be seen that the memory requirements in EZBTC has reduced by considerable amount when compared with SPIHT.

4.2 Rate-distortion Analysis:

The results in terms of PSNR of decoded image at various bit-rates (up to 1.0 bpp) are compared with that of SPIHT and SPECK. These results are given in Table 1 & 2 for Lena and Barbara images respectively. It should be noted that these results are without the use of any arithmetic coding. As expected, EZBTC shows comparable performance to that of other state-of-art coders.

![Bit-plane wise memory analysis](image)

**Fig. 1: pass-wise memory comparison of EZBTC with SPIHT**

5. Conclusions

In this paper, we have presented a novel approach for wavelet image coding. The proposed algorithm is based on the hierarchical block-set partitioning and block splitting. So far most of the coder developed, exploit either zero-tree or zero-block concept, whereas the proposed algorithm has flavors of both. The most attractive feature of the coder is its high memory efficiency with excellent rate-distortion performance. In future, we aim to extend this algorithm for color image and video coding.

6. References


<table>
<thead>
<tr>
<th>Coding method</th>
<th>0.125bpp</th>
<th>0.25bpp</th>
<th>0.5bpp</th>
<th>1.0bpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIHT</td>
<td>30.25</td>
<td>33.46</td>
<td>36.68</td>
<td>39.88</td>
</tr>
<tr>
<td>SPECK</td>
<td>30.36</td>
<td>33.56</td>
<td>36.79</td>
<td>39.98</td>
</tr>
<tr>
<td>EZBTC</td>
<td>30.34</td>
<td>33.52</td>
<td>36.70</td>
<td>39.88</td>
</tr>
</tbody>
</table>

**Table 1: PSNR at various rates for Lena image**

<table>
<thead>
<tr>
<th>Coding method</th>
<th>0.125bpp</th>
<th>0.25bpp</th>
<th>0.5bpp</th>
<th>1.0bpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIHT</td>
<td>24.39</td>
<td>26.92</td>
<td>30.71</td>
<td>35.78</td>
</tr>
<tr>
<td>SPECK</td>
<td>24.70</td>
<td>27.48</td>
<td>31.26</td>
<td>36.18</td>
</tr>
<tr>
<td>EZBTC</td>
<td>24.74</td>
<td>27.50</td>
<td>31.24</td>
<td>36.13</td>
</tr>
</tbody>
</table>

**Table 2: PSNR at various rates for Barbara**