Embedded Descendent-Only Zerotree Wavelet Coding for Image Compression

Wai Chong Chia, Li-Minn Ang, and Kah Phooi Seng

Abstract—**The Embedded Zerotree Wavelet (EZW) coder which can be considered as a degree-0 zerotree coder, and Set Partitioning In Hierarchical Trees (SPIHT) coder which can be considered as a degree-2 zerotree coder are two well-known image compression techniques. However, the performance of a degree-1 zerotree coder has not been well investigated. In this paper, an Embedded Descendent-Only Zerotree Wavelet (EDOZW) coder which can be considered as a degree-1 zerotree coder is proposed. The EDOZW coder is a modified version of SPIHT coder with Arithmetic Coding (SPIHT-AC). When coding the entry in the List of Insignificant Set (LIS) in SPIHT, it is necessary to first determine whether the entry is Type A or Type B. Then, a different scheme is used to handle the entry. But in contrast to SPIHT, it is no longer necessary to differentiate the type of entry in EDOZW. Only one type of entry is used in EDOZW and a recursive algorithm can be used to encode all the entries easily. This will help to reduce the overall complexity. Moreover, the EDOZW also achieved better performance at high bit rates. The EDOZW can serve as an alternative solution for image compression other than EZW and SPIHT.**

Index Terms— **Degree-1 zerotree coder, Degree-k zerotree coder, EZW, image compression, SPIHT.**

I. Introduction

The Embedded Zerotree Wavelet (EZW) coder [1] and Set Partitioning In Hierarchical Trees (SPIHT) coder [2] are gaining popularity in the image compression field, due to their superior performance. In fact, similarities do exist in both of the coding method, such as the use of wavelet transform and zerotree coding.

In both of the coding methods, a wavelet transform is used to decompose an image into different frequency subbands. Then, the coefficients in different frequency subbands are linked together to form a tree structure. Although the tree structure adopted by EZW and SPIHT is slightly different, it does not significantly affects the performance. The EZW and SPIHT also use a similar magnitude test to determine whether the coefficient is significant or insignificant. If the value of a coefficient is larger than the predefined threshold, it will be considered as significant.

On the other hand, the zerotree coding is based on the idea that if the root of the tree is insignificant, it is very likely that all the descendents are also insignificant [3]. If this condition is true, all the coefficients under the tree can be coded by using a single symbol to achieve compression.

A zerotree can be viewed as a tree structure that all the descendents are insignificant beyond a certain level. Depends on how many top k levels that a zerotree are having non-zero values, a zerotree can be classified as various type of degree-k zerotree [3]. The structure of degree-0 to degree-2 zerotree is illustrated in Fig. 1.

For example, EZW is considered as a degree-0 zerotree coder because it can only encode an occurrence of degree-0 zerotree with one symbol. On the other hand, SPIHT can encode the occurrence of degree-1 and degree-2 zerotree with one symbol. Additional information on type of zerotree and how it differentiates the EZW and SPIHT coder can be found in [3] and [4].

From the previous results, we can notice that the performance of SPIHT is higher than EZW. However, it does not imply that a degree-2 zerotree coder is always better than a degree-0 zerotree coder. Depending on the tree structure and distribution of the significant coefficients, a degree-k zerotree coder may send more redundant information than a degree-(k-1) zerotree coder. This condition will be explained more detail in later section.

Unlike the case for degree-0 and degree-2 zerotree coder, the performance of a degree-1 zerotree coder has not been well investigated. Hence, an Embedded Descendent-Only Zerotree Wavelet (EDOZW) coder which can be considered as a degree-1 zerotree coder is proposed in this paper. The EDOZW coder is a modified version of the SPIHT coder that can achieve comparable result with lower complexity. In contrast to SPIHT, a simple recursive algorithm can be used to encode all the entries in the List of Insignificant Set (LIS). It is no longer necessary to differentiate whether the entry is Type A or Type B.

The paper is organized in the following manner. Firstly, a brief introduction on SPIHT coding is given in Section II. Next, the proposed EDOZW coder will be explained in detail in Section III. Simulation results and discussions are presented in Section IV. Finally, we will conclude the paper in Section V.

II. SPIHT

In SPIHT, a tree structure called Spatial Orientation Tree (SOT) is used to link the coefficients in different subbands. The relationship between the root and the descendents in SOT is illustrated in Fig. 2. Coefficients in the LL band without the '*' label are considered as the root of the tree, and all the coefficients will have either 4 offspring or no offspring.

The algorithm uses 3 types of list, which named as List of Significant Pixel (LSP), List of Insignificant Pixel (LIP), and

Manuscript received October 1, 2008.

W. C. Chia, L-M. Ang, and K. P. Seng are with the School of Electrical & Electronic Engineering, The University of Nottingham Malaysia Campus, Jalan Broga, 43500, Semenyih, Selangor, Malaysia (phone: 603-89248000; fax: 603-89248002; correspond e-mail: [keyx7cwc / kezklma / kezkps] @nottingham.edu.my).

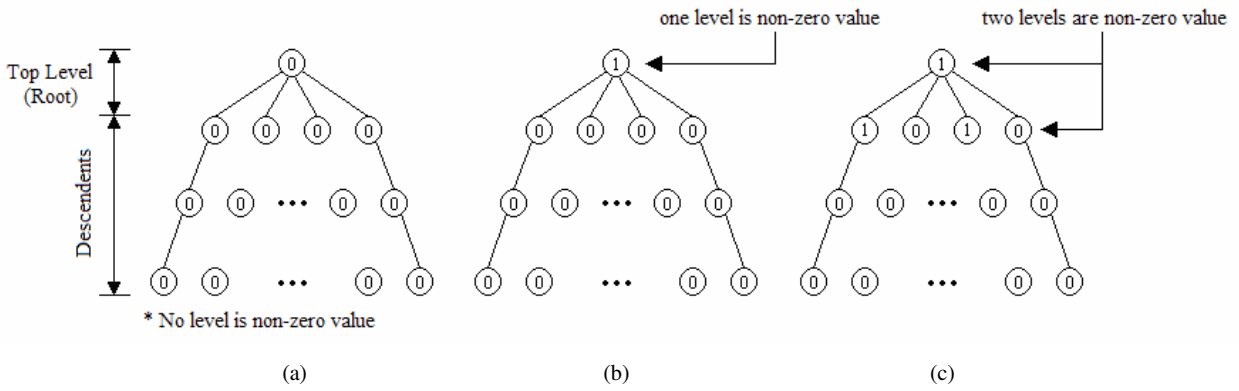


Figure 1. Structure of (a) Degree-0, (b) Degree-1, and (c) Degree-2 zerotree.

LIS to control the coding process. At the initialization phase, the LSP is left empty and all the coordinates in the LL band are added to the LIP. The entries in the LIS is similar to LIP, except that the coordinates with the '*' label are excluded. Then, a threshold is chosen for the magnitude test.

In the sorting phase, the magnitude test is first performed on all the entries in the LIP. If the entry in the LIP is significant in the current pass, it will be coded and shifted to the LSP. Otherwise it will remain in the LIP and tested again in the next pass.

The entries in the LIS can be categorized into 2 types, which defined as Type A entry and Type B entry. Type A entry contains the set of coordinates for all its descendents and Type B entry contains the set of coordinates for all its descendents with its 4 direct offspring excluded. In other word, Type B entry contains the set of coordinates start from its grand descendents to the end of the tree. Throughout the paper, we will consider all the coordinates in this set as the grand descendents. Since the coding of Type A and Type B entry is important for further explanation, we will explain the process more in detail.

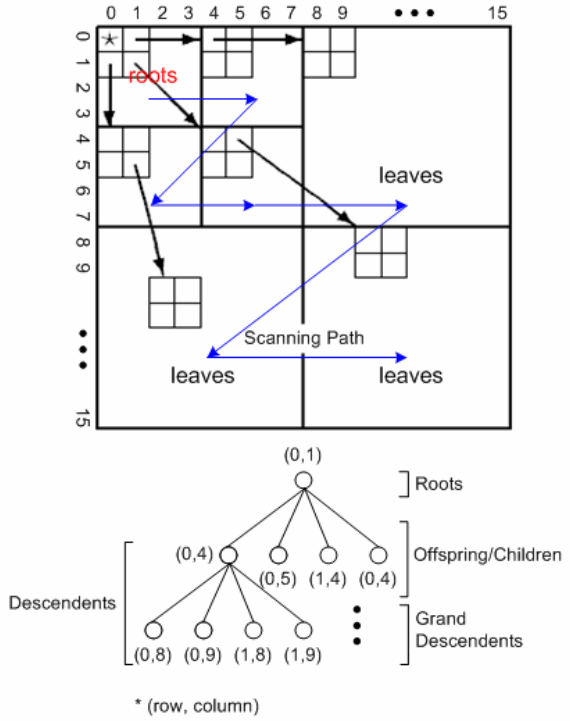


Figure 2. The SOT structure adopted by SPIHT.

When coding a Type A entry, if any coefficient within the set is significant, a '1' is sent as the output, and a magnitude test will be performed on its 4 direct offspring. The significant offspring will be coded and added to the end of the LSP, while the insignificant offspring will be added to the end of the LIP. If none of the coefficient within the set is significant, a '0' is sent as the output. This output bit that used to indicate the significance of a Type A entry will be labeled as *DESC* bit in this paper.

For a Type B entry, if any coefficient within the set is significant, a '1' is sent as the output. Then, the coordinate for the 4 direct offspring will be added to the end of the LIS and marked as new Type A entry. Otherwise, a '0' is sent as the output. This output bit that used to indicate the significance of a Type B entry will be labeled as *GDESC* bit in this paper.

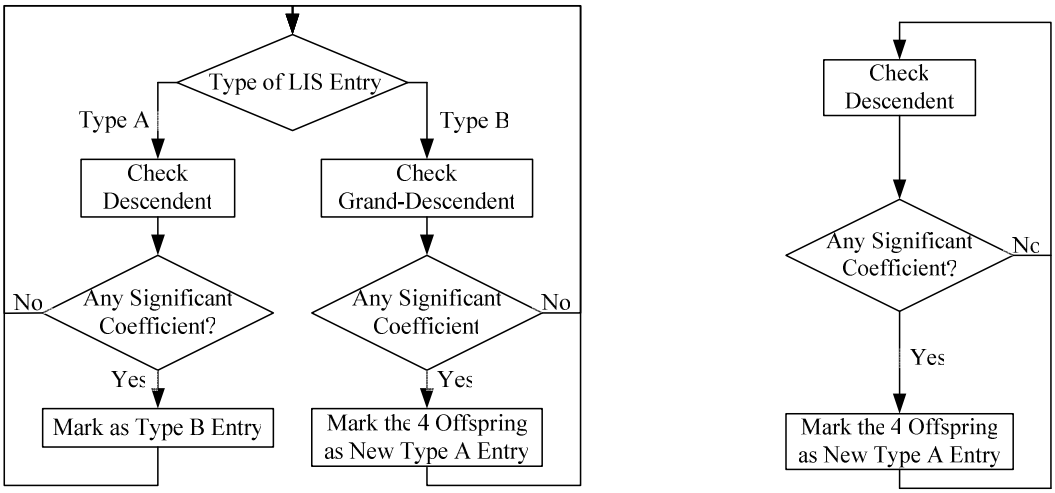
In the refinement phase, the refinement bit for each entry in the LSP except for new entries from the current pass is coded. Then, the threshold value is reduced and the coding process described above is performed again.

III. EDOZW

One of the advantages of EDOZW is no longer necessary to differentiate the Type A and Type B entry in the LIS. All the entries inside the LIS are processed in the same way, and the same algorithm can be used to encode all the entries. Hence, it becomes a recursive algorithm that is easier to implement as illustrated in Fig. 3. This also implies that the complexity is lower as compared to SPIHT. It should be noted that the flow chart shown in Fig. 3 only considered the main flow of the algorithm. The encoding of various output bits is not shown.

In EDOZW, the sorting phase of SPIHT is modified. For each entry in the LIS, if any coefficient within the set is significant, the entry is considered as significant and the 4 offspring will be coded in the same way as SPIHT. Then, the coordinate of the 4 offspring will be added to the end of the LIS immediately. In contrast, SPIHT will first mark the entry as a Type B entry and perform a magnitude test all the grand descendents. Only when any one of the grand descendent is significant, the coordinate of the 4 offspring is added to the end of LIS.

For example, consider the tree structure shown in Fig. 4(a). We will explain the coding process of EDOZW and SPIHT in terms of bit-plane wise. The '1' and '0' are used to represent the significant and insignificant coefficient respectively.



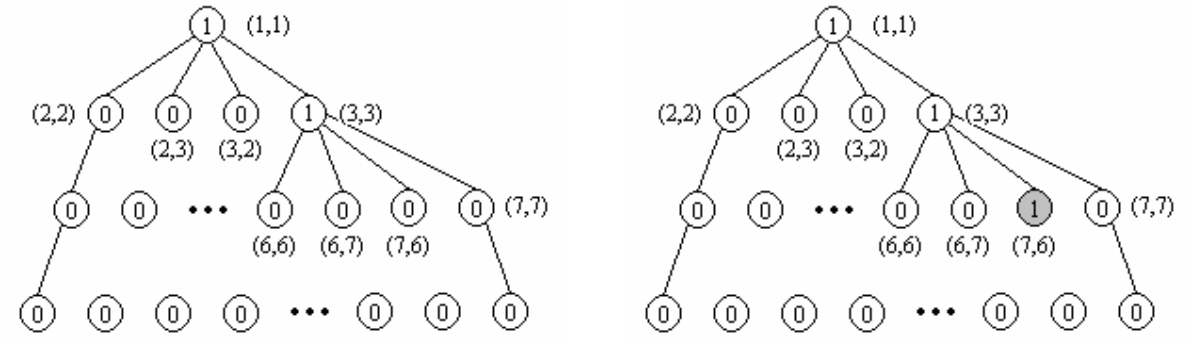
(a) (b)

Figure 3. The flow chart on encoding the LIS's entries in (a) SPIHT and (b) EDOZW.

Initially, the LIS content in EDOZW is {(1, 1)}. Since the set is significant, a '1' bit is sent as output. Then, the 4 offspring will be coded in the same way as SPIHT, and the output will be '0 0 0 1 1'. In this case, we use '1' to indicate a positive coefficient and '0' to indicate a negative coefficient. Next, the coordinates of the 4 offspring will be added to the end of the LIS, and this (1, 1) entry is discarded. Now, the LIS content is changed to {(6, 6); (6, 7); (7, 6); (7, 7)}. The coding process will continue until all entries in the LIS are checked. Since none of the grand descendent is significant, the output for this 4 entries will be '0 0 0 0'.

In the same case for SPIHT, the (1,1) entry which initially marked as a Type A entry is not directly discarded. In fact, it will be marked as a Type B entry, and the LIS content is changed to {Type B(1,1)}. The coordinate of the 4 offspring is not added to the LIS, because all the grand descendents are insignificant in this pass. Only a '0' (*GDESC*) bit is sent to indicate that all the grand descendents are insignificant. The output bits for EDOZW and SPIHT are listed under the example tree structure shown in Fig. 4(a). Since SPIHT is a degree-2 coder, it can use a symbol (bolded bit in Fig. 4(a)) to represent a degree-2 zerotree. In contrast, extra 3 bits are required to represent the same zerotree in EDOZW. We will define the occurrence of this type of tree structure as Case 1.

Now consider another tree structure shown in Fig. 4(b), whereas one of the grand descendents has become significant. The output bits for EDOZW and SPIHT are listed under the tree structure shown in Fig. 4(b). Notice that the number of bits sent for EDOZW is lesser than SPIHT. This is due to the GDESC bit (bolded bit in Fig. 4(b)) in SPIHT. The GDESC bit is required to indicate that one of the grand descendents is significant. Only when the GDESC bit is '1', the tree is breaks and the coordinates of the 4 offspring are added to the end of LIS as new Type A entry. But in EDOZW, the tree is directly breaks without considering the grand descendents. Hence, the GDESC bit is not needed. We will define the occurrence of this type of tree structure as Case 2. On the other hand, the complete EDOZW algorithm is shown in Algorithm I.



Method	Output Bits
EDOZW	1 0 0 0 11 0 0 0 0
SPIHT	1 0 0 0 11 **0**

(a)

Method	Output Bits
EDOZW	1 0 0 0 11 0 0 0 1 0 0 11 0
SPIHT	1 0 0 0 11 **1** 0 0 0 1 0 0 11 0

(b)

Fig. 4. Example tree structure of (a) Case 1 and (b) Case 2 illustrated in terms on bit-plane coding.

Algorithm I:

1) **Initialization Phase:**
- Set LSP as an empty list.
- Add all the coordinates in the lowest frequency subband to the LIP.
- Add all the coordinates with descendents (without the '*' label) in the lowest frequency subband to the LIS.
- Set the threshold value.

2) **Sorting Phase:**
- For each entry (i, j) in the LIP:
  - If the entry (i, j) is significant, output '1' and its sign bit. Then, shift the entry (i, j) to the LSP. Otherwise output '0'.
- For each entry (i, j) in the LIS:
  - Check all the descendents of entry (i, j). If any descendent is significant, output '1' and entry (i, j) is considered as significant.
  - If entry (i, j) is **significant**:
    - For each offspring (k, l) of entry (i, j):
      - If offspring (k, l) is significant, output '1' and its sign bit. Then, shift the offspring (k, l) to the LSP. Otherwise output '0' and shift offspring (k, l) to the LIP.
    - **Add the 4 offspring to the end of the LIS as new entries directly**.

3) **Refinement Phase:**
- Except for new entries in the LSP, output the refinement bits for each entry (i, j) in the LSP.

4) **Update:**
- Decrease the threshold value and repeat the steps above again.

IV. Simulation Results & Discussions

The performance of the EDOZW coder with AC will be compared to EZW coder with AC and SPIHT coder with AC. All the 3 coder are using the same AC presented in [5]. The result for EZW coder with AC is directly obtained from [1]. Hence, only the result reported in [1] is used for comparison. On the other hand, the result for SPIHT coder with AC is obtained from [6].

Simulation is carried out on several gray-scale test images which include *Lena*, *Barbara*, *Goldhill*, *Baboon*, *Sailboat*, and *Zelda*. These test images are all in the size of 512 x 512. For simulation, we use 9/7 tap wavelet filter [7] to perform the wavelet transform, and 5 decomposition levels is adopted for all the cases. All the simulation results are summarized in Table 1.

A. EDOZW at Low Bit Rates

For low bit rate (<=1.00Bit/Pixel), the performance of EDOZW is not as good as SPIHT. At this level, the threshold used for magnitude test is not very small. Therefore, many of the descendents are still considered as other insignificant, and

Table 1. The PSNR (db) for various gray-scale test images coded with different image coding method.

Bit Rate (Bit/Pixel)	**Lena**			**Barbara**			**Baboon**		
	EZW	EDOZW	SPIHT	EZW	EDOZW	SPIHT	EZW	EDOZW	SPIHT
0.10	-	30.1813	**30.2284**	-	24.2391	**24.2564**	-	22.6121	**22.6352**
0.25	33.17	34.0786	**34.1102**	26.77	27.5145	**27.5818**	-	24.5347	**24.5487**
0.50	36.28	37.1931	**37.2064**	30.53	31.3609	**31.3955**	-	26.9878	**27.0265**
1.00	39.55	**40.4191**	40.4051	35.14	36.3995	**36.4144**	-	30.5927	**30.6013**
1.50	-	42.9183	**42.9187**	-	**39.9949**	39.9456	-	**33.4454**	33.4042
2.00	-	**45.1885**	45.0627	-	**42.6905**	42.6483	-	**36.3193**	36.2943
2.50	-	**48.1995**	47.9055	-	**45.2991**	45.1185	-	**38.9224**	38.7857
3.00	-	**50.7131**	50.0568	-	**48.0763**	47.7476	-	**41.9059**	41.7568
Bit Rate (Bit/Pixel)	**Goldhill**			**Sailboat**			**Zelda**		
	EZW	EDOZW	SPIHT	EZW	EDOZW	SPIHT	EZW	EDOZW	SPIHT
0.10	-	27.9158	**27.9382**	-	26.5447	**26.5930**	-	34.1882	**34.1925**
0.25	-	30.5291	**30.5597**	-	29.9759	**30.0629**	-	**37.5174**	37.4990
0.50	-	33.0772	**33.1272**	-	32.8586	**32.9043**	-	**39.6748**	39.6608
1.00	-	36.5157	**36.5518**	-	36.1435	**36.1811**	-	**42.1437**	42.1341
1.50	-	**39.2235**	39.2005	-	38.7708	**38.7927**	-	**44.4148**	44.3012
2.00	-	**42.0418**	42.0164	-	41.7198	**41.7682**	-	**47.0256**	46.8379
2.50	-	**44.4975**	44.3293	-	44.4206	**44.2544**	-	**49.7380**	49.1866
3.00	-	**47.6848**	47.3573	-	**47.5531**	47.2525	-	**52.7161**	51.7010

the occurrence of Case 1 is higher than Case 2. In this case, the number of bits sent in EDOZW is higher. However, the difference in terms of PSNR between EDOZW and SPIHT is less than 0.1 dB. With the help of AC, the redundant formation in EDOZW can be compensated. As shown in the list of output bits in Table 1, all the redundant bits are '0'. In this case, the probability model for AC will heavily skew to '0' and AC can code the '0' bit efficiently.

B. EDOZW at High Bit Rates

For high bit rate (>1.00Bit/Pixel), the performance of EDOZW is better than SPIHT. At this level, the threshold used for magnitude test is very small. Hence, many of the descendents are considered as significant. In this case, the occurrence of Case 2 is higher than Case 1. The number bits sent in EDOZW will be lower than SPIHT, since EDOZW does not required to send a GDESC bit to indicate whether the grand descendents of an entry is significant or insignificant. This also shows that a degree-*k* coder is not always better than a degree-(*k*-1) coder.

V. Conclusions

The EDOZW which considered as a degree-1 zerotree coder achieves a result that is comparable to SPIHT. At low bit rate, the performance of EDOZW is not as good as SPIHT. However, the difference in terms of PSNR is less than 0.1 dB. At high bit rate, the performance of EDOZW is always better than SPIHT, and the gain can be as high as 0.65 dB. Moreover, the complexity of EDOZW is also lower than SPIHT. Since it is no longer necessary to differentiate the type of root, a simple recursive algorithm can be used to complete the coding process. This also helps reduce the overall processing time and power consumption. The EDOZW can serve as an alternative solution where high performance coder with lower complexity is preferable.

References

[1] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, vol. 41, no. 12, December 1993, pp. 3445-3462.

[2] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, June 1996, pp. 243-250.

[3] Y. Cho and W. A. Pearlman, "Quantifying the coding performance of zerotress of wavelet coefficients: degree-k zerotree," *IEEE Trans. on Signal Processing*, vol. 55, no. 6, June 2007, pp.2425-2431.

[4] Y. Cho and W. A. Pearlman, "Quantifying the coding power of zerotrees of wavelet coefficients: a degree-k zerotree model," *IEEE Int. Conf. on Image Processing*, vol. 3, Sept. 2005, pp. III-53-56.

[5] I. H. Witten, R. Neal and J. G. Cleary, "Arithmetic coding for data compression," *Comm. ACM*, vol. 30, June 1987, pp. 520-540.

[6] A. Said and W. A. Pearlman, "SPIHT image compression programs," June 1996. [Online] Available: http://www.cipr.rpi.edu/Research/SPIHT/EW_Code/SPIHT.zip

[7] M. Antonini, M. Barlaud, P. Mathiew, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Processing*, vol. 1, Apr 1992, pp. 205 – 220.