AN ERROR RESILIENT AND MEMORY EFFICIENT SCHEME FOR WAVELET IMAGE CODING[1]

Ekram KHAN
Department of Electronics Engineering,
Aligarh Muslim University, Aligarh, India

E-mail: ekhan@lycos.com

Irshad A. ARSHAD
Department of Mathematics and Statistics,
Allama Iqbal Open University, Islamabad, Pakistan

E-mail: irshad_arshad@hotmail.com

Taru VARSHNEY
Department of Electronics Engineering,
Aligarh Muslim University, Aligarh, 202002, India

E-mail: taru_varsh@yahoo.com

Abstract: Set- Partitioning in Hierarchical Trees (SPIHT) is a state-of-art zero-tree based image coder with excellent rate-distortion performance in the noise free environments. However in presence of noise they are extremely sensitive to the bit errors. Even a single bit error may lead a substantial degradation in the quality. Our investigations reveal that the use of three linked lists in SPIHT increases the inter-bitplane dependencies of bits and therefore causes the errors to propagate down the bitstream. In this paper we propose a modified coder which uses only a single list. The list is initialized, processed and exhausted within each bit-plane, thereby removing dependencies of inter-plane bits. The proposed coder is known as Single List SPIHT (SL-SPIHT) image coder. Additionally, this coder is memory efficient as compared to the SPIHT. Simulation results show that under the same channel conditions, SL-SPIHT can improve the quality of reconstructed image by 5-6 dB compared to that of SPIHT.

Key words: Wavelet transform; Image coding; Error resilience; memory efficient

1. Introduction

Real time transmission of images through handheld mobile/portable devices (with limited memory, processing power and battery life) requires an image coding algorithm that can compress images efficiently with reduced memory requirements. The contemporary modern image coding methods (e.g. SPIHT [3]0) support a wide range of functionality but requires significant amount of memory during their execution. Also, wireless channels are highly prone to bit errors; therefore the devolvement of error resilient techniques is an additional challenge for memory constrained environments.

Among the many methods of compression, Discrete Wavelet Transform (DWT) approach has become a popular technique [2-5]. Due to energy compaction nature of wavelet transform coding, it is common that a large number of wavelet coefficients are quantized to zero value. The concept of zero-tree structure was first introduced by Lewis and

Knowles [1] for efficient representation of zero wavelet coefficients after perceptual thresholding. This data structure was later elegantly combined with the bit-plane coding scheme by Shapiro for embedded wavelet image coding, popularly known as embedded zero-tree wavelet (EZW) [2]. Intensive efforts have been drawn into this research field ever since and many zero-tree image coders are being developed 000. The SPIHT [3] and virtual SPIHT (VSPIHT) [5] algorithms are the simplest and most efficient variations of EZW. These techniques have excellent rate distortion characteristics in noise free environments. However in the presence of noise they are extremely sensitive to bit errors. Even a single bit error may lead to loss of synchronization between the encoder and the decoder beyond erroneous bit. As a result the quality of the reconstructed image degrades substantially unless proper error correcting methods are used. Thus the transmission of zerotree coded images over noisy channels is still a challenging problem.

One of the problems with SPIHT and VSPIHT coders are the large memory requirement due to the uses three continuously growing lists. Also, these lists are responsible for propagation of errors down the bitsream. Since information accumulated in list during any pass is used for coding of significant/insignificant pixels/sets in the subsequent passes. In case any error occurred, it will lead to wrong entries in the lists, thereby severely affecting the decoding process.

Recently, Arora et. al. proposed a memory efficient wavelet image coding scheme [2] which uses a single list for encoding and decoding. Here onward, this algorithm is referred as Single List SPIHT (SL-SPIHT). The emphasis of the paper was more on memory reduction, similar to work done in [7] [8]. In this paper we will explore the error resiliency feature of SL-SPIHT.

As compared to SPIHT, the SL-SPIHT uses only one list to store the significance information of the sets. The use of single and re-usable list in SL-SPIHT not only reduces the memory requirement but also improves the error resiliency. Since the list is re-initialized at the beginning of each pass, in SL-SPIHT algorithm hence the entries are not used in the subsequent passes. This is beneficial as it makes the algorithm pass by pass independent, puts a check on the consecutively growing size of the lists and makes a foundation to study the error resiliency feature of the SL-SPIHT algorithm.

Rest of the paper is organized as follows. Section II briefly reviews the SPIHT algorithm and its error resiliency feature. The SL-SPIHT algorithm is described in Section III. Simulation Results are presented in Section IV, where memory analysis and error resiliency performance of SL=-SPIHT algorithm is compared with that of SPIHT algorithm. Finally, paper is concluded in Section V.

3. Review of SPIHT algorithm

SPIHT Algorithm

The SPIHT algorithm is based on hierarchical set partitioning, which can be thought of as a divide-and-conquer strategy. The SPIHT algorithm views wavelet coefficients as a collection of spatial orientation trees, with each tree consisting of coefficients from all sub-bands that correspond to the same spatial location in an image. A partitioning rule is used to divide a given set into smaller subsets so that significant coefficient can be efficiently isolated.

The SPIHT consists of two main stages namely sorting and refinement. For practical implementation, SPIHT maintains three linked lists viz. the list of insignificant pixels (LIP), the list of significant pixels (LSP) and the list of insignificant sets (LIS). At the initialization stage, SPIHT initializes the LIP with all the pixels in the highest level of the pyramid (i.e. LL subband), the LIS with all the coefficients at the highest level of the pyramid except those, which don't have descendents, and LSP as an empty set. During the sorting pass, the algorithm first traverses through the LIP, testing the magnitude of its elements against the current threshold and representing their significance by 0 or 1. Whenever a coefficient is found significant, its sign is coded and it is moved to LSP. The algorithm then examines the LIS and performs a magnitude check on all coefficient of set. If a particular tree/set is found to be significant, it is partitioned into its subsets (children and grandchildren) and tested for significance. Otherwise a single bit is appended to the bit stream to indicate an insignificant set (or zero-tree). After each sorting pass SPIHT outputs refinement bits at the current level of bit significance of those pixels which had been moved to LSP at higher threshold, resulting in the refinement of significant pixels with bits that reduce maximum error. This process continues by decreasing current threshold by factor of two until desired bit rate is achieved.

Error Analysis

The SPIHT coder generates different type of bits that have different degree of vulnerability to the errors. The effect of error in some bits is more severe, damaging the image globally by disturbing the synchronization between the encoder and decoder. On the other hand, some bits are less sensitive to errors, and damage the image locally without disturbing the synchronization.

● *Significance bits:* During the LIP and LIS testing, pixels or trees are checked for significance. Error in significant bits will result in the propagation of error down to the bit-stream during the decoding process. A significant coefficient or set may be deemed insignificant and vice-versa. This will cause the decoder to update the list with the wrong nodes and can cause a large degradation in the decoded image quality. Thus a single bit error in the significant bits has global effect and may damage the entire image.

● *Sign bits:* The sign bit of a wavelet coefficient is generated, immediately after its significance test during sorting pass, when it is found significant. An error in this bit will change only the phase of the coefficient and does not disturb the synchronization between the encoder and the decoder. The effect of an error in the sign bit corrupts the image locally only during the inverse wavelet transform.

● *Refinement bits:* The refinement bits are generated during refinement pass. An error in these bits has less severe effect and distorts the image locally only. The effect of error in a refinement bit is limited to a single coefficient and does not disturb the progression of the decoding process after it occurs.

● Based on the degree of their sensitivity, the bits generated by SPIHT algorithm can be classified into two classes:

● Critical Bits (CB) and

● Non-Critical Bits (NCB).

The critical bits are those, which causes the loss of synchronization between the encoder and decoder. A single bit error in critical bit causes the failure of the reconstruction process after that point. It consists of significant bits generated during LIP and LIS tests. The non-critical bits on the other hand cause less severe errors. The effect of error in a non-critical bit is limited to a single coefficient and does not disturb the progression of the decoding process after it occurs. The non-critical bits consist of the sign and refinement bits.

3. SL-SPIHT algorithm

The SL-SPIHT algorithm was first proposed by Arora at. al. 0 to reduce the working memory requirement of SPIHT algorithm. In this algorithm, the functions of all three lists are performed by a single list. The list is re-initialized at the beginning of each bit-plane to avoid the inter-dependency on the previous bit-planes.

Working of SL-SPIHT Algorithm

The SL-SPIHT algorithm works as follows.

After the 'N_d' level of wavelet decomposition of input image, depending upon the value of the largest wavelet coefficient, the initial threshold is calculated as 2^n where $n = \left\lfloor \log_2 \left(\max_{\forall(i,j)} |c_{i,j}| \right) \right\rfloor$. The LL-sub-band is coded separately. The coefficients of LL-sub-band are represented in (n+1) bit sign-magnitude form. In the first (most significant) bit-plane, the sign bit and the most significant bit from magnitude of the LL sub-band coefficients are embedded in the bitstream. In the subsequent bit-planes, only the nth most significant bits of LL-sub-band coefficient are transmitted. In order to encode the remaining sub-band coefficients, they are linked through spatial orientation trees with their roots in the LL-sub-band. To define a uniform child-parent relationship, it is assumed that first quarter of LL-sub-band coefficients have no descendents and remaining three-quarter of the coefficients have their children in sub-bands of correspondingly same orientation.

A single list referred as **L**ist of **P**ixel **S**ets (LPS) is used in this algorithm. At the beginning of each bit-plane, LPS is initialized with address of those coefficients of LL-subband that have decedents. For each entry in LPS, test the significance of the set. For insignificant sets, a single bit is appended to the bitstream to indicate a zerotree. However, if a particular set is found to be significant, it is then partitioned into two its subsets (children and grandchildren). First the significance of each of the children is tested and if a particular child is found significant, its sign bit is transmitted and current threshold is subtracted from its magnitude. Then the significance of grandchildren is tested. If the set of grandchildren (provided they exist) is found to be significant, the addresses of children are added at the end of LPS. After testing every entry of LPS (including those which are added in the current pass), it is re-initialized and threshold is reduced by a factor of two. The entire process is repeated for the next pass (bit-plane). It should be noted that same coefficients might become significant in more than one pass, depending upon its absolute value. In order to avoid the multiple transmission of its sign bit, sign bits are masked (or flagged) once they are transmitted at the first significant test of the corresponding coefficient. If in the subsequent passes, the same coefficient again tests significant at lower thresholds, the transmission of sign bit is avoided as it has already been masked.

At the decoder, the reverse process is performed. At the arrival of first significant bit of a coefficient, it is reconstructed as $\pm 1.5 \times 2^n$. The decoder adds or subtract 2^{n-1} to its current reconstructed value depending upon whether it inputs significant bit in the current pass or not.

It is observed that the working memory requirement in any pass is proportional to the maximum number of entries in LPS during that coding pass. Further, since in SL-SPIHT, LPS is re-initialized, the same memory can be used in the next and subsequent passes, whereas in SPIHT, the lists keep growing progressively thereby increasing memory requirements. Additionally, the entire encoding and decoding of a bit-plane is performed in a single pass, whereas SPIHT requires two passes, sorting and refinement.

Error Analysis of SL-SPIHT

Another attractive feature of the SL-SPIHT algorithm is its improved error resilience. This can be explained as follows. Since SL-SPIHT uses a single re-useable list. The LPS list is initialized, processed and exhausted within each pass, making encoding/decoding of each pass independent to its preceding passes. This is in contrast to the SPIHT algorithm, where the continuously growing lists (LIP, LIS and LSP) make encoding/decoding of each pass dependent over the previous passes. This is because, list entries are processed, updated in a pass (bit-plane), which will be used to code significant/insignificant information in the next pass. This inter-pass dependency of bitstream makes SPIHT coder highly sensitive to the channel errors.

On the other hands, in SL-SPIHT the LPS is initialized at the beginning of each pass, so that even if the list entries during earlier passes are corrupted due to channel errors, it will not affect the decoding of subsequent passes. Therefore, if the decoder can be synchronized again (by using pass-by-pass markers), then there will be decoding error in that pass. This will lead to the reconstruction of good quality image even in presence of errors. Therefore, the use of single re-usable list to achieve pass by pass independency and use of markers to synchronize will lead to better error resiliency in SL-SPIHT.

The use of markers at the end of each pass (bit-plane) will have no impact on the decoding of SPIHT in presence of errors, but it has significant impact on the decoding of SL-SPIHT bitstream. For example, consider a case when error occurs in a bit and all its following bits until the end of the pass are zero. This situation is more likely to happen in early passes when threshold is high & most of the trees are insignificant (zerotrees) and all the refinement bits are also zero. In this case since all the bits are of same magnitude (i.e.0) and correspond to same type of information (generally zerotree significance) the use of marker will be very much advantageous. As in this case the only error observed is due to loss of synchronization which is again regained in the next pass by the use of marker and since the single list is reinitialized at the beginning of each pass hence the problem of wrong entries in the list is also avoided. This will lead to a large improvement in the quality of the decoded image.

4. Simulation results

In order to highlight the features of SL-SPIHT, we have considered two gray level test images, BARBARA and BOAT, each of depth 8 bits/pixel (bpp) and of size 512×512. Each image is wavelet transformed with 5 levels of decomposition using 9/7 bi-orthogonal filter. First we compare the memory requirements of SL-SPIHT with that of SPIHT, followed by the error resiliency.

Memory Requirements

In this section we will compare the memory requirements of the auxiliary list(s) in SL-SPIHT with that of SPIHT and NLS 0 algorithms. There are two types of memory requirement in zerotree coders: fixed memory to store wavelet coefficients and variable working memory to store list entries. For the comparison purpose, we consider only variable working memory requirements. Though NLS does not use any lists, but needs fixed size arrays and state tables. For an image of size $M \times N$, the working memory requirement in NLS 0 is $M_{NLS} = \left(\frac{MN}{4} + \frac{MN}{16}\right)W + \frac{MN}{2}$ bytes, where W is the number of bytes used to store each wavelet coefficient (say two bytes). This working memory is always fixed and is independent of number of bit-planes to be executed. However, in SPIHT and the proposed coder, the size of required working memory at any instant depends upon the current number of entries in the list(s). For the purpose of comparisons, memory requirements at the end of each pass (or bit-plane) and the maximum (or worst case) memory requirement are considered.

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.

Total memory required in SPIHT, M_{SPIHT} (in bits).

$M_{SPIHT} = c\, N_{LIP} + (c-1)\, N_{LIS} + c\, N_{LSP}$ (1)

Where

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 [2*log-$_2$(max(M,N))]

Each element in LIS requires (c-2) bits for addressing, since it contains coefficients with descendents and an extra bit is required for defining the 'type' of entries.

In the worst case,

$N_{LIP} + N_{LSP} = MN$

$N_{LIS} = MN/4$ (as the coefficients having no descendents or the highest frequency sub-bands will never enter into LIS.) Thus the maximum working memory requirement in SPIHT is

$$M_{SPIHT}^{\max} = (5c-1)\frac{MN}{4} \qquad (2)$$

SL-SPIHT

Since SL-SPIHT uses only one list namely LPS. In the list, entries correspond to address of a wavelet coefficient (except the coefficients of LL-sub-band and those which have no offsprings), so each element can be stored by using only (c-2) bits. If N_{LPS} be the numbers of entries in LPS and $M_{SL\text{-}SPIHT}$ (in bits) denotes the total working memory required in the SL-SPIHT algorithm, then

$$M_{SL\text{-}SPIHT} = (c-2)\, N_{LPS} \qquad (3)$$

In the worst case,

$$N_{LPS} = MN/4$$

Thus, the maximum working memory requirement in SL-SPIHT is

$$M_{MESH}^{\max} = (c-2)\frac{MN}{4} \qquad (4)$$

For example, for any 512×512 image, $c=2*\log_2(512)=18$ bits, W= 2 bytes (say),

$M_{SPIHT}^{\max} =$ 729088 bytes

$M_{NLS}^{\max} =$ 294912 bytes

$M_{MESH}^{\max} =$ 131072 bytes

Thus, $M_{SPIHT}^{\max} : M_{NLS}^{\max} : M_{MESH}^{\max} = 5.56 : 2.25 : 1$. It should be noted that NLS uses fixed working memory equal to $M_{NLS}^{\max}$. Thus it can be seen that the SL-SPIHT algorithm has reduced memory requirement by factors of 5.56 and 2.25 in comparisons to SPIHT and NLS respectively.

At the end of every pass, the number of elements in the list is counted. Based on this, the memory requirement for SPIHT and SL-SPIHT are calculated according to Eqs. (1) and (3) respectively. Table 1 compares the memory requirements for two test images at the end of first 10 passes. It can be observed that SL-SPIHT has about 65-75% memory saving as compared to SPIHT, which is reasonably significant.

Table 1. Memory Requirements for SPIHT and SL-SPIHT for BARBARA and BOAT Images (All values are in K Bytes)

Pass	BARBARA		BOAT	
count	SPIHT	SL-SPIHT	SPIHT	SL-SPIHT
1	1.032	0.384	1.076	0.384
2	1.176	0.384	1.598	0.480
3	1.864	0.480	3.842	1.072
4	4.473	1.200	11.919	3.576
5	15.135	4.408	29.118	8.192
6	51.891	14.440	59.114	17.248
7	101.157	27.744	101.125	28.424
8	168.327	45.816	169.339	57.088
9	260.939	72.040	337.228	94.408
10	413.191	114.632	539.563	129.528

Error Resiliency Performance

In order to compare the error resiliency of SPIHT and SL-SPIHT, we have considered the same set of gray level images. A marker (corresponding to number of bits in each pass) is placed at the end of each pass in the bitstreams of both coders.

The encoded bits are transmitted over Binary Sequential Channel (BSC) with Bit Error Rate (BER) ranging from 10^{-4} to 10^{-1}. The quality is measured in terms of Peak Signal to Noise Ratio (PSNR). All the results are averaged for 20 independent channel conditions. Fig. 1(a) and (b) show the average image quality for different bit rates for BARBARA and BOAT images respectively. Both the images are coded at 0.25 bpp.

It can be observed from these Figures that SL-SPIHT algorithm consistently out performs SPIHT. The relative improvement is more at lower BERs than at higher BERs. More specifically, SL-SPIHT has up to 5-6 dB improvement over SPIHT for BARBARA image and about 2-3 dB for BOAT Image. This is due to the fact that use of marker in SL-SPIHT results in significant improvement as compared to the SPIHT as discussed earlier. Hence it is capable of totally nullifying the effect of error in special case when all the bits at the end of the pass are zero. This effect leads to improvement in quality in SL-SPIHT as compared to SPIHT both with markers.

5. Conclusion

In this paper, we have shown that SL-SPIHT algorithm is not only memory efficient as reported in 0, but also error resilience. Both these features are obtained due to the use of single re-usable list in SL-SPIHT as opposed to three continuously growing lists in SPIHT. It was observed that SL-SPIHT may result up to 75% of memory saving and up to 5-6 dB quality improvement under the same channel conditions. This type of coders is very useful for real time image transmission over wireless channel through hand held mobile devices such as digital cameras of mobile sets (which has scarcity of memory).

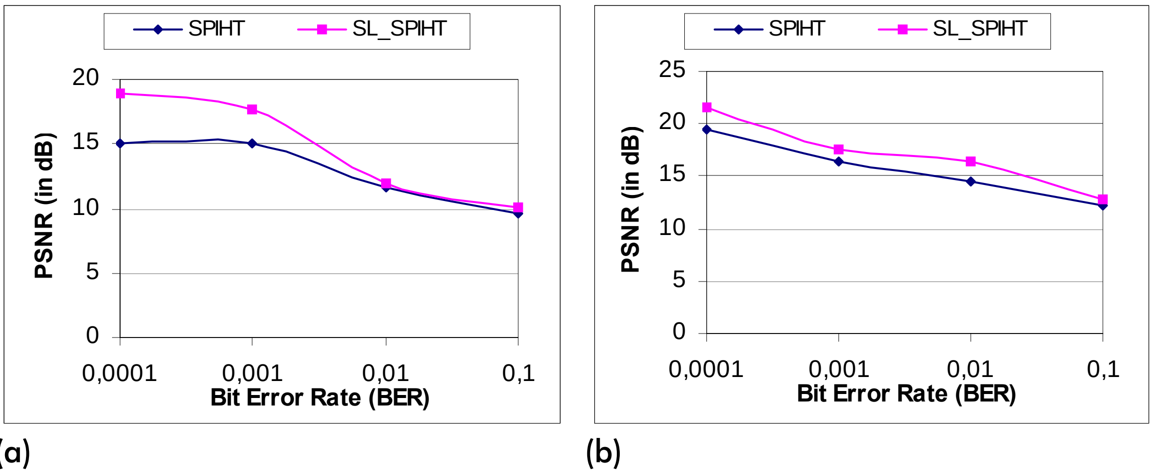

Figure 1. PSNR versus BER comparisons of SPIHT and SL-SPIHT for (a) BARBARA and (b) BOAT Images

References

1. Lewis, A. S. and Knowles, G. **Image Compression using the 2-D wavelet Transform**, *IEEE Transaction on Image Processing*, vol. 1, Feb 1991, pp. 244-250.
2. Shapiro, J. M. **Embedded Image Coding using Zerotrees of Wavelet Coefficients,** *IEEE Trans. Acoustic, Speech and Signal Processing*, vol. 41, Dec. 1993, pp. 3445-3462.
3. Said, A. and Pearlman, W. A., **A New Fast and Efficient Codec Based on Set Partitioning in Hierarchical Trees**, *IEEE Trans. Circuits Systems for Video Technology*, vol. 6, pp. 243-250, March 1996.
4. Xiong, Z. Ramchandran, K. and Orchard, M. T. **Space-frequency Quantization for Wavelet Image Coding**, *IEEE Trans. Image Processing*, vol. 6, May 1997, pp. 677-693.
5. Khan, E. and Ghanbari, M. **Very Low Bit rate Video Coding Using Virtual SPIHT**, *IEE Electronics Letters*, vol. 37, Jan 2001, pp. 40-42.
6. Arora, H., Singh, P., Khan, E. and Ghani, F. **Memory Efficient Set Partitioning in hierarchical tree for wavelet image compression**, *Proc. IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP-2005), vol. 2, May 2005, pp. 385-388.
7. Wheeler, F. W. and Pearlman, W. A. **SPIHT image compression without lists**, *IEEE International Conference on Acoustics, Speech and Signal Processing, ICAASP 2000*, Vol. 6, September 2000, pp 2047-2050.
8. Wheeler, F. W. and Pearlman, W. A. **Low memory packetized SPIHT image compression**, *33rd Asilomar Conference on Signals, Systems and Computers*, Vol. 2, October 1999, pp 1193-1197.

[1] **Acknowledgment**
This research is partly supported by AICTE research grant *F. No. 8023/RID/BOR/RPS-60/2005-06*.