EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

An improved EZBC algorithm based on block bit length

Renlong Wang[1], Shuangchen Ruan[1*], Chengxiang Liu[1], Wenda Wang[2] and Li Zhang[3]

Abstract

Embedded ZeroBlock Coding and context modeling (EZBC) algorithm has high compression performance. However, it consumes large amounts of memory space because an Amplitude Quadtree of wavelet coefficients and other two link lists would be built during the encoding process. This is one of the big challenges for EZBC to be used in real time or hardware applications. An improved EZBC algorithm based on bit length of coefficients was brought forward in this article. It uses Bit Length Quadtree to complete the coding process and output the context for Arithmetic Coder. It can achieve the same compression performance as EZBC and save more than 75% memory space required in the encoding process. As Bit Length Quadtree can quickly locate the wavelet coefficients and judge their significance, the improved algorithm can dramatically accelerate the encoding speed. These improvements are also beneficial for hardware.

PACS: 42.30.Va, 42.30.Wb

Keywords: block bit-length, zeroblock, EZBC, Quadtree, DWT

1. Introduction

At present, the typical embedded wavelet-based image coding methods includes Embedded Zerotree Wavelet coder (EZW) [1] proposed by Shapiro, Set Partitioned in Hierachical Tree (SPIHT) [2] proposed by Said etc., Set Partition Embedded block algorithm (SPECK) [3] proposed by Asad Islam, Embedded Block Coding with Optimal Truncation (EBCOT) [4] used in JPEG2000 [5], and Embedded ZeroBlock Coding and context modeling (EZBC) [6] proposed by Shih-Ta Hsiang etc.

Among these algorithms, EZW and SPIHT were based on the local characteristic in spatial-frequency domain of wavelet transform. They utilized the similarity of wavelet coefficients of the same spatial location in different subbands. SPECK was based on the characteristic that the energy concentrated in low-frequency subband and the insignificant coefficients mainly concentrated in high-frequency subbands after wavelet transform, it utilized the correlation of insignificant coefficients in the same subband. EBCOT used the correlation of wavelet coefficients in the same subband to build the high efficiency context and adopted the Arithmetic Coder. EZBC utilized the correlation of coefficients in the same sub-band and coefficients in different subbands at the same time and adopted a simple and efficient quadtree coding structure and context-based bitplane encoding method [6,7], so it can achieve higher compression performance than SPIHT and EBCOT. EZBC algorithm is widely used in digital image compression and scalable video coding.

However, during the encoding process EZBC built an Amplitude Quadtree $Q_k[l](i, j)$ and two link lists: the insignificant nodes link list *LIN* and the significant pixels link list *LSP*. Both Amplitude Quadtree $Q_k[l](i, j)$ and the link lists consumed a large amounts of memory space and this was one of the big challenges for the EZBC to be used in real time or hardware applications. As it has a large amount of operations of link lists such as adding and deleting nodes during encoding, this made its coding efficiency decline greatly when good image quality is required. The algorithm proposed in reference [8] presented the significance state-table of Quadtree coefficients to record the significance state change of all the QuadTree corresponding coefficients in the coding of Quadtree bitplane, which removed the *LIN* and *LSP*. Some memory was saved but $Q_k[l](i, j)$

* Correspondence: scruan@szu.edu.cn
[1]Shenzhen Key Laboratory of Laser Engineering, College of Electronic Science and Technology, Shenzhen University, Shenzhen, 518060, China
Full list of author information is available at the end of the article

was still adopted in the algorithm. Instead of making any radical change, it increased the complexity of the algorithm on the contrary.

In order to overcome these disadvantages, an improved EZBC algorithm, Bit Length EZBC (BL-EZBC) based on bit length of coefficients, was put forward in this article. It used Bit Length Quadtree instead of Amplitude Quadtree of wavelet coefficients and other two link lists, so it can save more than 75% memory space required in the encoding process. As Bit Length Quadtree can quickly locate the wavelet coefficients, judge their significance, and avoid a large amount operations of link lists used in EZBC, this algorithm can accelerate the encoding speed effectively. The test results indicates that this algorithm achieves the same signal to noise ratio as EZBC and gains much higher encoding speed, saves 75% memory usage than EZBC.

2. Bit Length Quadtree

The bitplane coding process in BL-EZBC begins with establishment of the Bit Length Quadtree representations for the individual subbands. The bit length here refers to the significant bit length of the absolute value of quantized wavelet coefficient. The value of the Bit Length Quadtree node $B_k[l](i, j)$ at position (i, j), quadtree level l and subband k is defined by

$$B_k[l](i,j) = \begin{cases} 0, & l \equiv 0,\ |c_k(i,j)| < 1 \\ 1 + \lfloor \log_2(|c_k(i,j)|) \rfloor & l \equiv 0,\ |c_k(i,j)| \geq 1 \\ \max \begin{Bmatrix} B_k[l-1](2i,2j) \\ B_k[l-1](2i+1,2j) \\ B_k[l-1](2i,2j+1) \\ B_k[l-1](2i+1,2j+1) \end{Bmatrix} & l > 0 \end{cases} \quad (1)$$

where $C_k(i, j)$ is the quantized subband coefficient at position (i, j). $|x|$ indicates the absolute value of x; $\lfloor x \rfloor$ indicates the rounding operation on x. For example, if $x = \lfloor 3.8 \rfloor$, then $x \equiv 3$ where " = " means assignment and "≡" indicates the equality test.

Below is an example for $B_k[l](i, j)$.

If $C_k(i, j) = (\pm 9)$, then $B_k[0](i, j) = 4$. That is, the significant bit length of ± 9 is equal to 4.

Each bottom quadtree node is assigned to the bit length of the subband coefficient at the same position. The quadtree node at the next level is then set to the maximum of the four corresponding nodes at the current level, as illustrated in Figure 1a. The top quadtree node is just equal to the maximal bit length of all subband coefficients. Similar to the conventional bitplane coders, we progressively encode subband coefficients from the MSB toward the LSB. In the Bit Length Quadtree, a node is significant if the value of the node $B_k[l](i, j)$ is great than the index of the bitplane. A significant pixel (coefficient) is located by the testing and splitting operation recursively performed on the significant nodes up to the pixel (bottom) level of a quadtree, as shown in Figure 1b.

Instead of using Amplitude Quadtree $Q_k[l](i, j)$ and two link lists in EZBC, the bit length of the Quadtree in BL-EZBC is built up with the bit length of the absolute value of the subband coefficients. It corresponds to the index of the bitplane. It means that Bit Length Quadtree can quickly locate the wavelet coefficients and judge their significance. Thus, it can accelerate the encoding speed effectively. And the lists of insignificant nodes *LIN* and significant nodes *LSP* required in EZBC are not necessary in BL-EZBC. What iss more, the usual bit length of wavelet coefficients is 16. That is, each node of Amplitude Quadtree $Q_k[l](i, j)$ uses 16 bit length memory. However, in BL-EZBC, each node of the Bit Length Quadtree $B_k[l](i, j)$ uses 4 bit length memory. For 4 bit length, the max value it can represent is 15. It means that the corresponding max absolute value of wavelet coefficients is $2^{15} - 1 = 32767$. Generally, the absolute value of the wavelet coefficients is less than 32767. That is, 4 bit length of each node of Bit Length Quadtree can represent the bit length of wavelet coefficients. This saves a large amount of memory space required in the coding process.

3. The coding process of BL-EZBC

First, define parameters below,

- C_k (i, j): the quantized wavelet coefficient of subband k at position (i, j).

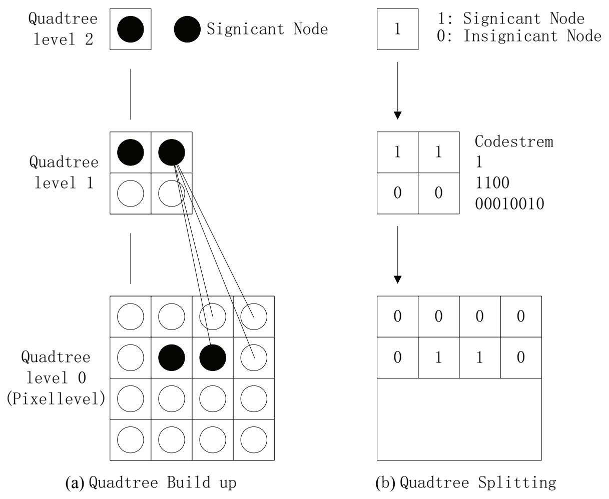

Figure 1 Illustration of quadtree build up and decomposition. (a) Quadtree build up and **(b)** Quadtree splitting.

- $B_k[l](i, j)$: Bit Length Quadtree B representation for the bit length of coefficients from the same subband with node of $B_k[l](i, j)$ corresponding to a quadtree node at position (i, j), Suband k, and level l. Its value is defined by (1).
- D_k: depth of the quadtree of the subband k.
- D_{max}: the maximum quadtree depth among all subbands.
- K: total number of subbands.
- X_k: Horizontal offset of the subband k referring to the original image, left-to-right as a positive direction.
- Y_k: Vertical offset of the subband k referring to the original image, top-to-down as a positive direction.
- *CodeBL*(k, l): Function for coding the insignificant node of level l in Bit Length Quadtree of subband k. Its parent node should be signficant in the last bit-plane if it exists.
- *CodeDescendant*(k, l, i, j): function for processing all the descendent nodes of $B_k[l](i, j)$ after it just tested significant against the current threshold.
- *CodeLSP*(k): function for refinement of the coefficients of subband k.

```
The coding process,
Initialization, $n = \max_{(k)}\{B_k[D_k - 1](0,0)\}$;
Coding the highest bitplane,
for $k$ = 0: $K$-1, Code BL($k$, $D_k$-1);
$n$–;
Coding the remaining bitplanes.

  for (;$n$ > 0; $n$–){
      for $l$ = 0: $D_{\max}$-1
          for $k$ = 0: $K$-1, Code BL($k$, $l$);
  for $k$ = 0: $K$-1, Code LSP($k$);

}
Below are the functions in pseudocode,
CodeBL ($k$, $l$)
{

  • if ($l$ <$D_k$ - 1)
      * for all ($i$, $j$) in quadtree level $l$+1, subband $k$.
      That is, all the nodes in level $l$+1 of Bit Length
      Quadtree of subband $k$
          ⋆ if ($B_k[l+1](i, j)$ >$n$), CodeDescendant ($k$, $l$+1,
          $i$, $j$);
  • else if ($l \equiv D_k$ - 1)
      * if ($B_k[l](0, 0)$ <$n$), output 0;
      * else if ($B_k[l](0, 0) \equiv n$){
          output 1;
          ⋆ if ($l \equiv 0$)
          -output the sign of $C_k$ (0+$X_k$, 0+$Y_k$);
          ⋆ else
          - CodeDescendant ($k$, $l$, 0, 0);
      }

}
CodeDescendant ($k$, $l$, $i$, $j$)
{

  • for ($x$, $y$)∈ {(2$i$, 2$j$), (2$i$, 2$j$+1), (2$i$+1, 2$j$), (2$i$+1, 2$j$
  +1)}. That is, the four child nodes in level $l$-1 that
  mapping to the node ($i$, $j$) in level $l$, subband $k$
      * if ($B_k[l-1](x, y)$ <$n$), output 0;
      * else if ($B_k[l-1](x, y) \equiv n$){
          output 1;
          ⋆ if ($l \equiv 1$)
          -output the sign of $C_k$ ($x$+$X_k$, $y$+$Y_k$);
          ⋆ else
          -CodeDescendant ($k$, $l$-1, $x$, $y$);
  }

}
CodeLSP ($k$)
{

  • for all ($i$, $j$) in quadtree level 0, subband $k$. That is,
  all the nodes in level 0 of Bit Length Quadtree of
  subband $k$, scanning first in row then column.

 * if ($B_k[0](i, j)$ >$n$), output the value of | $C_k$ ($I$ + $X_k$, $j$ +
$Y_k$) | in bitplane $n$.
  }
```

4. Experimental results and analysis

The improved algorithm BL-EZBC and the original EZBC were verified and compared in Pentium(R) D CPU 2.80 GHz computer with 512 × 512 × 8 bits Standard grayscale images of Lena, Goldhill and Barbara. 9/7 Wavelet filters boundary symmetrical extension was used. The context model was the same as which used in EZBC algorithm, the algorithm data in reference [8] were quoted here. Table 1 listed out the PSNR comparison. Table 2 shows the memory usage comparison during coding. Table 3 shows the coding time comparison with different thresholds of BL-EZBC and EZBC.

From Table 1, the same PSNR was achieved by using BL-EZBC, algorithm in reference [8] and EZBC. However, EZBC used Amplitude Quadtree $Q_k[l](i, j)$ and two link lists (LIN and LSP) to finish the coding process and utilized the significance of the neighbor nodes and the node in parent subband to construct the context, so a large amount of memory was required to store Amplitude Quadtree $Q_k[l](i, j)$ and two link lists. Therefore, more memory was required for more complex image and higher coding bit rate. These increased the

Table 1 PSNR(dB) comparison.

Image	Rate (bpp)	Ref [8]	EZBC	BL-EZBC
Lena	1.0	40.58	40.63	40.63
	0.5	37.44	37.49	37.49
	0.25	34.32	34.41	34.42
Goldhill	1.0	36.78	36.93	36.93
	0.5	33.37	33.45	33.45
	0.25	30.71	30.77	30.76
Barbara	1.0	37.17	37.35	37.37
	0.5	32.08	32.27	32.28
	0.25	28.20	28.37	28.39

complexity of the hardware. Although linked lists *LIN* and *LSP* were removed from the algorithm in reference [8], Amplitude Quadtree $Q_k[l](i, j)$ and significance state-table of Quadtree coefficients were still adopted, which also occupied a lot of memory and increased the algorithm complexity. Bit Length Quadtree was presented to replace the Amplitude Quadtree in the improved algorithm, which was used to complete the coding process and construct the context, so the memory usage was greatly reduced during the coding. As shown in Table 2, by using the improved algorithm BL-EZBC, more than 75% memory was saved compared to the original EZBC algorithm. Therefore, the memory usage was significantly reduced.

As EZBC used a large amount of operations of link lists such as adding and deleting nodes during encoding, this made its coding efficiency decline greatly when good image quality was required. Instead of using Amplitude Quadtree $Q_k[l](i, j)$ and two link lists in EZBC, the bit length of the Quadtree in BL-EZBC is built up with the bit length of the absolute value of the subband coefficients. It corresponds to the index of the bitplane. It means that Bit Length Quadtree can quickly locate the wavelet coefficients, judge their significance. Thus, it can accelerate the encoding speed effectively. As shown in Table 3, if the threshold less than 32, the coding time of BL-EZBC was significantly less than EZBC. If the threshold is great than or equal to 32, the coding time of BL-EZBC was less than EZBC too. However, it was not so obvious. A larger threshold means less bitplanes would be scanned, the size of link lists was smaller, and the operations of link lists were also fewer. As the larger threshold means the worse image quality, in most situations in order to achieve better image quality the threshold was not so large. When good image quality was required, the coding speed of BL-EZBC is significantly faster than EZBC.

Table 2 Memory usage comparison (Bytes).

Image	Rate (bpp)	Ref [8]	EZBC	BL-EZBC
Lena	1.0	726K	844K	171K
	0.5	726K	765K	171K
	0.25	726K	730K	171K
Goldhill	1.0	726K	853K	171K
	0.5	726K	767K	171K
	0.25	726K	726K	171K
Barbara	1.0	726K	861K	171K
	0.5	726K	773K	171K
	0.25	726K	736K	171K

Table 3 Coding time comparison with different threshold with BL-EZBC and EZBC (ms).

Threshold	1	2	4	8	16	32	64
Lena							
BL-EZBC	156	109	63	47	31	20	15
EZBC	406	312	187	93	46	31	20
Goldhill							
BL-EZBC	172	125	94	62	46	26	16
EZBC	453	375	265	156	78	32	21
Barbara							
BL-EZBC	171	125	93	78	47	31	20
EZBC	422	344	234	140	94	47	31

5. Conclusion

An improved algorithm BL-EZBC based on EZBC was proposed in this article. A new model Bit Length Quadtree was used to complete the coding process and construct the context. It can achieve the same compression performance as EZBC but the memory usage was greatly reduced during the coding process. Bit Length Quadtree can quickly locate the wavelet coefficients, judge their significance, and avoid a large amount operations of link lists used in EZBC. Thus, it can accelerate the encoding speed effectively. These improvements are also beneficial for the hardware.

Abbreviations

EBCOT: Embedded Block Coding with Optimal Truncation; EZBC: Embedded ZeroBlock Coding and context modeling; EZW: Embedded Zerotree Wavelet coder; SPECK: Set Partition Embedded block algorithm; SPIHT: Set Partitioned in Hierachical Tree.

Acknowledgements

This work was supported in part by the Shenzhen Double Hundred Person Project and the Shenzhen University Dr start-up fund research (000201) of China.

Author details

[1]Shenzhen Key Laboratory of Laser Engineering, College of Electronic Science and Technology, Shenzhen University, Shenzhen, 518060, China [2]College of Automation, Harbin Engineering University, Harbin, 15001, China [3]College of Information Engineering, Shenzhen University, Shenzhen, 518060, China

Competing interests

The authors declare that they have no competing interests.

Received: 9 March 2011 Accepted: 10 October 2011
Published: 10 October 2011

References

1. JM Shapiro, Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans Signal Process. **41**(12), 3445–3462 (1993). doi:10.1109/78.258085
2. A Said, WA Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Trans Circuits Syst Video Technol. **6**(3), 243–250 (1996). doi:10.1109/76.499834
3. A Islam, WA Pearlman, An embedded and efficient low-complexity hierarchical image coder, in *Visual Communication and Image Processing '99*, vol. 3653. (San Jose, California, SPIE, 1999), pp. 294–305
4. D Taubman, High performance scalable image compression with EBCOT. IEEE Trans Image Process. **9**(7), 1158–1170 (2000). doi:10.1109/83.847830
5. M Rabbani, R Joshi, An overview of the JPEG2000 still image compression standard. Signal Process Image Commun. **17**(1), 3–48 (2002). doi:10.1016/S0923-5965(01)00024-8
6. S-T Hsiang, *Highly Scalable Subband/Wavelet Image and Video Coding*, (Rensselaer Polytechnic Institute, New York, 2002)
7. ST Hsiang, JW Woods, Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling, in *IEEE ISCAS2000*, (IEEE, Geneva, Switzerland, 2000), pp. 662–665
8. W Du, J Sun, M Sima, Improved EZBC algorithm with low complexity. IEICE Electron Express. **1**(15), 447–452 (2004). doi:10.1587/elex.1.447

doi:10.1186/1687-6180-2011-84
Cite this article as: Wang *et al.*: An improved EZBC algorithm based on block bit length. *EURASIP Journal on Advances in Signal Processing* 2011 2011:84.