Utilizing Repeated Adjacencies of Vector Quantization Indices in Image Compression

Mohammed F. Abdel-Latif, Tarik K. Abdel-Hamid, Magdy M. Doss, and H. Selim
Electrical Engineering Department, Faculty of Engineering, Assiut University
Assiut, Egypt.
E-mail: farouk@aun.edu.eg

Abstract *- Image compression using Vector Quantization (VQ) results in highly correlated indices. The correlation between these indices is used to reduce the bits needed to represent them. This is done by many index compression algorithms such as the Hu and Chang [1], search order coding (SOC)[2], and Switching Tree Coding (STC) [3]. In this paper a new algorithm for VQ index compression is introduced; it utilizes the local statistics of each image and the repeating pattern of its adjacent indices. The proposed algorithm improves the index compression performance of the basic VQ, with a relatively slight increase of complexity.*

Keywords *- VQ Index compression, Lossless Coding, Image compression*

1. INTRODUCTION

1.1 Vector Quantization

VQ is one of the promising image compression methods. It is a strong competitor because of its simple hardware implementation; it allows using simple decoders consisting only of a look- up table. Also VQ has excellent rate-distortion performance. It can reach the rate distortion bound when increasing the vector dimension. In VQ, the image to be compressed is divided into non-overlapping square blocks which are converted into vectors with dimensions 1xk, where, for example, k=16 for 4x4 pixels blocks. The full search algorithm or any equivalent method [4] is used to find the best match codevector from a previously designed codebook in terms of minimum Euclidean distance between the input vector and the codevectors. The index of the best match codevector is saved as a representative of the input vector. The bits needed to store the index are much less than those needed to store the input vector itself, this gives the compression in image. The same procedure is performed to all the blocks of the image; the result is a 2D index map of indices.

Using large blocks increases the memory requirements and complexity costs, since they are exponential in block size. So practical VQ is limited to small block dimensions (4x4 pixels blocks are common) which does not exploit all the correlation between adjacent pixels.

To increase the efficiency of VQ, variants are introduced such as shape gain VQ, Removed mean VQ, and Predictive VQ. These methods decorrelate the image blocks before applying VQ to the resulting vectors. Classified VQ, finite state VQ, and Adaptive VQ use the correlation found between adjacent blocks to enhance the VQ performance by using local codebooks for each region of the image [2].

1.2 VQ index compression

To increase the VQ compression ratio, index compression algorithms are introduced. They exploits the correlation between VQ indices. An early work on VQ index compression was Address VQ by Nasrabadi and Feng [5], where 2x2 blocks of adjacent indices are compressed using lossless VQ. This method results in high compression ratios, but it has an algorithm of relatively high complexity.

Also in [6], Wu et al modeled and utilized the correlation between indices of unconstrained VQ by context modeling and conditional entropy coding of VQ indices (CECOVI). However, they need to estimate several high order conditional probabilities and find suitable weighting functions for each index. Moreover, they need to implement arithmetic coding for high order probability models [7].

Hsieh and Tsai introduced in [2] the SOC method for index compression, where the current index is compared with the previous indices in a predefined search path, when a matched index is found, the corresponding search order is sent to the decoder.

In the STC algorithm, four relations between adjacent indices are defined in a 2D index map. Based on these relations, the current index is coded with one of four prefixes defined by three trees. These trees define different prefix codes according to the four types of relations.

In [1], Hu and Chang introduced a low complexity index compression algorithm. After the 2D index map is generated, the algorithm is applied to each index in raster scan order, each index is checked to see whether any of its adjacent entries (upper or left) in the index map have the same index value or not, if the same index value is found in either of the two positions only short binary code will be used to indicate which of the two entries matches the current index. If both entries do not have the same index value as the current index, then it is checked to see if the relative offset between it and one of the two adjacent indices does not exceed a certain threshold. If this is valid the relative distance between indices is sent, else the full index value is sent. The algorithm relies on the repeated value indices to achieve saving in bit rate.

One performance key in the algorithm is to use codebook with vectors that are arranged according to their mean values. This reduces the average difference in value between any two successive indices due to the correlation inherently found between the adjacent blocks in natural images.

0-7803-8689-2/04/$20.00 ©2004 IEEE

2. THE PROPOSED ALGORITHM

VQ is mainly a mapping process that projects the image blocks into a reduced set of codevectors. For natural images, there is some kind of pattern repeating where similar image components tend to repeat in a certain arrangement. In VQ the image is divided into small blocks which inherent the natural images pattern repetition property. When mapping the image blocks into the codevectors, the resulting indices are found to confirm the repeated patterns. This could be employed for a new index compression algorithm. This algorithm utilizes the local statistics of each image and the repeating pattern of its adjacent indices. In the proposed algorithm the 2D index map is scanned from up to down and from left to right. A search is done for the repeated patterns of spatially adjacent indices pairs. The correlation between VQ indices can be exploited by generating for each individual image a "Next" (or "Right") table as shown in Fig. 1. It is a 1xN index vector, where N is the codebook size, which keeps for each index a pointer for the most probable index that comes bellow (or right to) it.

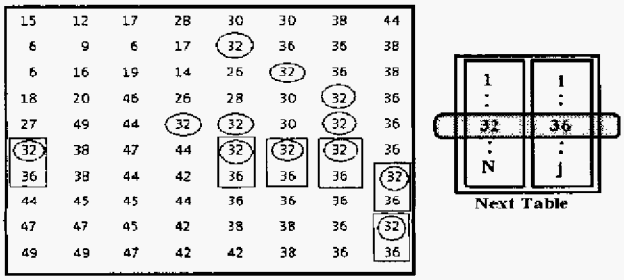

Fig. 1. "Next" table generation process

The current index is to be checked to see if it has the same value as the index in the "Next " (or "Right") table of the adjacent upper (or left) index; in this case a short binary sequence is sent. If the current index does not match the previous condition, then the difference in index value between the current index and the adjacent indices (upper or left) is checked, if it is less than a predefined threshold then this difference is sent, else the full index must be sent. The codebook vectors should be arranged according to their mean values to reduce the average difference in value between adjacent indices.

The "Next" table for every individual image is constructed by finding for every index the spatially lower index with highest repetition rate. Since this table is constructed on individual image basis, it needs to be sent to the receiver each time we encode an image and it represents an extra overhead. Also the "Right" table is constructed the same way, but it keeps for each index the most probable index which comes just right to it. Three methods for the proposed algorithm are studied.

Method 1 (Fig. 2): Compares the current index value with the value of "most probable next index" of the upper adjacent index and the value of the left adjacent index as follows:

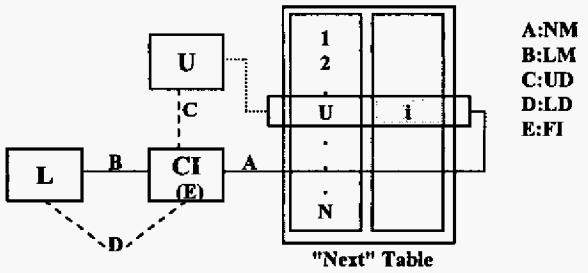

Fig. 2. Comparing the current index CI with i of the "Next" table.

1- The indices in the 2D index map are scanned from top to bottom and from left to right. The "Next" table is constructed as described previously.
2- The scanned current index **CI** is checked to determine if it has the same value **i** as the most probable next index of its upper index **U**, if it does match (NM), the re-encoded index will be code A, and go to step 6.
3- **CI** is checked to determine if it has the same value **L** as the adjacent left index, if it does match (UM), the re-encoded index will be code B, and go to step 6.
4- The differences between **CI** and each of **L** and **U** are checked to see if they exceed a certain threshold or not (UD). If any of them does not exceed this threshold, the corresponding re-encoded index will be [code C + index difference (**CI-L**)] or [code D + index difference (**CI-U**)], respectively. Whereby the symbol difference is sent as a signed number (needs an extra bit), and go to step 6.
5- When the index does not match any of the previous conditions, then the re-encoded index will be [code E + full index (FI)].
6- If it is not the last index in the index map, go to step 2 with the next scanned index, else end.

Method 2 (Fig. 3): Compares the current index value with the value of the "most probable right index" of the left index and the value of the upper adjacent index. The method is similar to method 1, but differing only in the indices to be compared with. Steps 1, 2 and 3 are modified as follows:

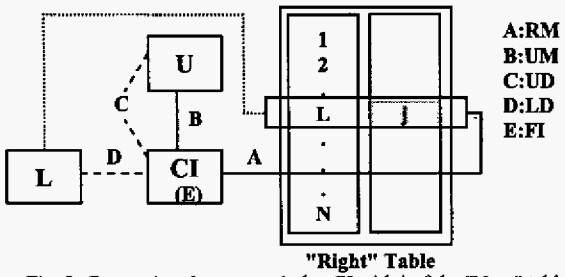

Fig. 3. Comparing the current index CI with j of the "Next" table.

1- The indices in the 2D index map are scanned from top to bottom and from left to right. The "Right" table is constructed as described previously.
2- The current index **CI** is checked to determine if it has the same value **j** as the most probable right index of its left index **L**, if it does match (RM), the re-encoded index will be code A, and go to step 6.

3- **CI** is checked to determine if it has the same value **U** as the adjacent upper index, if it does match (UM), the re-encoded index will be code B, and go to step 6.

Method 3 (Fig. 4): Compares the current index value with the value of the "most probable next index" of the upper adjacent index and the value of the "most probable right index" of the left adjacent index

The method is similar to method 1, but differing only in the indices to be compared with. The steps 1, 2 and 3 are modified as follows:

1- The indices in the 2D index map are scanned from top to bottom and from left to right. The "Next" and "Right" tables are constructed as described previously.
2- The current index **CI** is checked to determine if it has the same value **i** as the most probable next index of the upper index **U**, if it does match (NM), the re-encoded index will be code A, and go to step 6.
3- **CI** is checked to determine if it has the same value **j** as the most probable right index of the left index **L**, if it does match (LM), the re-encoded index will be code B, and go to step 6.

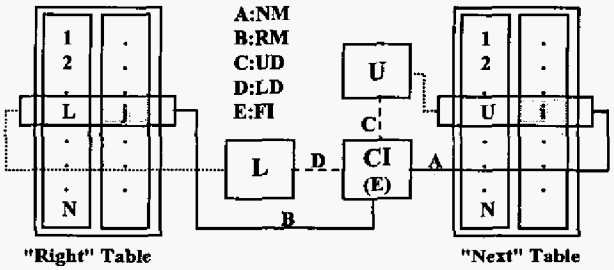

Fig. 4. Comparing the current index CI with i and j of the "Next" and "Right" tables.

A, B, C, D, and E are binary codes that are determined by Huffman coding.

The total bit rate needed to encode the image is given by:

$$BR = A\_size \times A\_HuffLen + B\_size \times B\_HuffLen + C\_size \times (C\_HuffLen + \log_2(Thr) + 1) + D\_size \times (D\_HuffLen + \log_2(Thr) + 1) + E\_size \times (E\_HuffLen + \log_2(N)) + TableCost + HuffmanTab\ leCost\ /(N \times N)$$

Where: *A_size, B_size, C_size, D_size, and E_size*: Are the number of indices which follow the symbol A, B, C, D, and E, respectively.

Thr: the index difference threshold.

A_HuffLen, B_HuffLen, C_HuffLen, D_HuffLen, and E_HuffLen are the lengths of binary codes A, B, C, D, and E, respectively.

Table Cost: the bits needed to store the "Next" table, "Right" table, or both depending on the case.

(Table Cost = Nx log (N))

For the third method we send two tables so the table cost will be doubled.

Huffman Table Cost: the bit size needed to store the binary sequences corresponding to symbols A, B, C, D, and E.

3. SIMULATION RESULTS

Simulation results indicate the increased compression ratio of the newly proposed algorithm over other algorithms. Five gray images, with size 512x512 pixels, and resolution of 256 gray levels are used to generate the codebooks used in the simulation. The splitting codebook generation method is used to obtain the initial codebook, and the Linde-Buzo-Gray Method (LBG) is used to enhance it [4]. Four codebooks with dimensions 4x4 and sizes 64, 128, 256, and 512 are generated to explore the performance in wide range of bit rates. The standard test images Lena, gold hill, F-16, and Elaine, which are not part of the training set, are used to judge the algorithm.

The proposed algorithm is compared with three compression methods. The first compression method is the Huffman coder [8]. To increase its efficiency, the correlation is removed by applying differential pulse code modulation (DPCM) to the indices then coding the output differences using variable length binary codes generated by Huffman method. The Huffman codes are generated on individual image basis, i.e. each image has its own set of binary codes depending on its local statistic. This gives higher compression ratio than using global Huffman codes that are designed using a group of images. The bit rate obtained using DPCM cascaded by Huffman indicates the minimum bit rate could be obtained using lossless coding. The second method is Hu and Chang algorithm. The search order coding (SOC) index compression is the third method used for comparison.

The index difference threshold used in the proposed algorithm is 16, which is found to give better results than other values of threshold.

Tables 1, 2, 3, and 4 display the bit rates (in bpp) of the encoded indices for the four different codebook sizes.

It is clear that the new algorithm with its three methods give better results than the other algorithms almost in all images with the different codebooks.

It is clear from the results that the average bit rate of the three proposed methods is almost the same. For individual images there is a difference in bit rate that depends on the

Table 1. Comparing the bit rate (in bpp) of different index compression methods for codebook with size 64

Method \ Image	Lena	GH	F-16	Elaine	Ave.
Huff.+DPCM	0.2509	0.2376	0.2019	0.2357	**0.2315**
SOC	0.2495	0.2695	0.2510	0.2511	**0.2553**
Hu & Chang	0.2358	0.2565	0.2249	0.2487	**0.2415**
Next	0.2224	0.2396	0.1966	0.2355	**0.2235**
Right	0.2082	0.2465	0.2030	0.2300	**0.2219**
Next+Right	0.2229	0.2394	0.1949	0.2351	**0.2231**

Table 2. Comparing the bit rate (in bpp) of different index compression methods for codebook with size 128

Method \ Image	Lena	GH	F-16	Elaine	Ave.
Huff.+DPCM	0.3066	0.2968	0.2541	0.2959	**0.2884**
SOC	0.2839	0.3201	0.2880	0.2916	**0.2959**
Hu & Chang	0.2673	0.2934	0.2577	0.2822	**0.2752**
Next	0.2582	0.2794	0.2374	0.2705	**0.2614**
Right	0.2456	0.2815	0.2412	0.2673	**0.2589**
Next+Right	0.2576	0.2792	0.2353	0.2706	**0.2607**

Table 3. Comparing the bit rate (in bpp) of different index compression methods for codebook with size 256

Method \ Image	Lena	GH	F-16	Elaine	Ave.
Huff.+DPCM	0.3759	0.3643	0.3207	0.3673	**0.3570**
SOC	0.3395	0.3883	0.3393	0.3601	**0.3568**
Hu & Chang	0.3250	0.3538	0.3084	0.3481	**0.3338**
Next	0.3170	0.3471	0.2990	0.3383	**0.3254**
Right	0.3103	0.3484	0.3038	0.3383	**0.3252**
Next+Right	0.3184	0.3476	0.2987	0.3393	**0.3260**

Table 4 - Comparing the bit rate (in bpp) of different index compression methods for codebook with size 512

Method \ Image	Lena	GH	F-16	Elaine	Ave.
Huff.+DPCM	0.4553	0.4482	0.3846	0.4547	**0.4357**
SOC	0.3950	0.4609	0.3870	0.4312	**0.4185**
Hu & Chang	0.3907	0.4376	0.3539	0.4355	**0.4044**
Next	0.3782	0.4267	0.3482	0.4167	**0.3925**
Right	0.3774	0.4275	0.3518	0.4150	**0.3929**
Next+Right	0.3867	0.4317	0.3532	0.4197	**0.3978**

image orientation and if the repeated patterns in the horizontal or vertical direction are found more. For all images the third method does not improve the performance over the first and second method due to the need to store two tables instead of one as an overhead.

Table 5 indicates the percentage reduction in bit rate when comparing the proposed algorithm with Hu and Chang method. The improvement over Hu and Chang method decreases with increasing the codebook size.

Table 6 indicates the average percentage of the different cases in the re-encoded indices resulting from different images using method 1. Note that the compression is gained from the high percentage of UM and LM cases, which are re-encoded by only few bits.

When analyzing the "Next" and the "Right" tables we find that there are some code vectors, which are the most probable next index or the most probable right index to themselves, this is especially present at the background areas where the same code vectors are repeated to construct them. Their percentage from all the code vectors decreases with increasing the codebook size. This is expected, since the increase in codebook size decreases the correlation between indices.

Table 5. Percentage improvement of bit rate over the Hu and Chang method

Image \ N	64	128	256	512
Lena	5.7035	3.3938	2.4628	3.1891
GoldHill	6.6011	4.7584	1.8741	2.4862
F-16	12.5681	7.8686	3.0502	1.6134
Elaine	5.2787	4.1457	2.8100	4.3114
average	**7.5378**	**5.0416**	**2.5493**	**2.9000**

Table 6. Symbols percentages of method 1 for image Lena

Image \ N	64	128	256	512
NU	40.3998	34.7687	28.3380	25.4187
LM	15.9973	14.3982	12.1967	10.1898
UD	39.6954	38.5321	33.8257	24.6198
LD	2.9895	7.2760	10.8435	11.4349
FI	0.9180	5.0250	14.7961	28.3368

4. CONCLUSION

This paper presents a new lossless compression algorithm for VQ indices. It uses the local statistics of the image and the repeated patterns of the indices for index compression. Three different methods were used, they gave comparable bit rates in the range of 0.22, 0.25, 0.32 and 0.39 when using codebook sizes 64, 128, 256, and 512, respectively. It is clear that using the first or the second method is better since they are less complex than the third one which needs an estimation of both the "Next" and the "Right" tables. The overall performance is better than Hu and Chang method by 7.5, 5, 2.5, and 2.9 % for codebook sizes 64, 128, 256, and 512, respectively.

REFERENCES

[1] Yu-Chen Hu and Chin-Chen Chang, "Low complexity index-compressed vector quantization for image compression," IEEE Trans. on Consumer Electronics, vol. 45, No. 1 pp. 219–224, Feb. 1999.

[2] C. H. Hsieh and J. C. Tsai, "Lossless compression of VQ index with search order coding," IEEE Trans. on Image processing, vol. 5, No. 11, pp. 1579–1582, 1996.

[3] M. H. Sheu, S.C. Tsai, and M. D. Shieh "A lossless index coding algorithm and VLSI design for vector quantization," The First IEEE Asia Pacific Conference AP-ASIC '99, pp. 198 – 201, Aug. 1999

[4] A. Gersho and R. M. Gray, "Vector Quantization and Signal Compression," Kluwer Academic Publishers, 1992.

[5] N. M. Nasrabadi and Y. Feng, "Image compression using address-vector quantization," IEEE Trans. Communication, vol. 38, pp. 2166–2173, Dec. 1990.

[6] X. Wu, J. Wen and W.H. Wong, "Conditional Entropy Coding of VQ Indexes for Image Compression," Proc. IEEE Data Compression Conference, IEEE Computer Society Press, pp.347-356, March 1998

[7] Yun Gong, Michael K. H. Fan, "Image compression using lossless coding on VQ indexes", Proc. IEEE Data Compression Conference, pp.583, March 2000.

[8] D. A. Huffman, "A method for the construction of minimum redundancy codes," Proc. IRE, Vol. 40, pp. 1098-1101, 1952.