Lossless Contour Compression Using Morphology, Chain Coding, and Distribution Transform

Ching-Wen Hsiao, Jian-Jiun Ding *, and Po-Jen Chen
* Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
Email: r02942031@ntu.edu.tw, jjding@ntu.edu.tw, r03942106@ntu.edu.tw
TEL: +886-2-33669652

***Abstract*— Chain coding is widely used in image compression to encode the boundaries of objects efficiently. Although chain codes are effective, they still need large amount of memory to store the codes. Therefore, an efficient encoding technique for chain codes is required. In this paper, we propose an algorithm to encode contours losslessly. First, the morphological operation is applied to shrink the contours if the process is invertible. Then, the modified Angle Freeman chain code is used to represent the contours, and the distribution transform is applied to rearrange the binary stream and the proposed improved adaptive arithmetic code is adopted for encoding. Simulations show that the proposed algorithm can much reduce the data size required for encoding contours.**

I. INTRODUCTION

In image processing, shape is an important feature for object recognition, template matching, and image analysis. To represent the shape, Freeman proposed chain code methods [1] that consider the correlation between successive points. Chain code is a sequence of symbols, where the symbol comes from a set including finite symbols. The set in the Freeman Eight Directional Chain Code (*FEDCC*) [1] includes eight possible directions, while the set in the Freeman Four Directional Chain Code (*FFDCC*) [1] contains four possible directions.

After that, the Angle Freeman chain code (*AF8*) [3] not only considers the relative directions, but apply the Huffman coding to compress the chain code. In 2010, Hermilo Sanchez-Cruz further proposed the modified directional Freeman Chain code in eight directions by a set of nine symbols (*MDF9*) 0, which modifies the contour at first and adapts the symbols to represent contours efficiently.

There was another kind of chain code based on cell instead of pixel proposed in 1999. The chain code was named the Vertex Chain Code (*VCC*) [4], which uses only three symbols to represent the discrete contour composed of regular cells. After that, some modified vertex chain codes were proposed to decrease the storage of the chain code, such as the Compressed Vertex Chain Code (*C_VCC*) 0. In 2014, Borut Zalik and Niko Lukac used the move-to-front transform and adaptive run-length encoding [7] to compress *VCC* and other chain codes in a lower bit rate.

In this paper, we propose an algorithm to compress the contours losslessly based on the transformation of the chain code and entropy coding. First, the morphological operation is used to shrink the contours slightly. Second, we modify the Angle Freeman chain code [3] with chain division and symbol substitution. Moreover, we use the Huffman code as an intermediate code, and apply the distribution transform to make the distribution of binary streams suitable for our improved adaptive binary arithmetic coding. Simulations show that, with the proposed algorithm, the total data sizes required for encoding contours can be much reduced.

II. PRELIMINARY

Before introducing the proposed algorithm, we first review the Freeman chain code [1], the Angle Freeman chain code [3] and the adaptive arithmetic coding method with context modeling [9]. Freeman chain codes are based on the characteristics that successive contour points are adjacent to each other, which means that we can begin with a contour point, find its direction to next point, and then encode the direction chain code until the contour is complete. Taking the 8-directional Freeman chain code [1] in Fig. 1 for example, the symbols 0-7 can be applied to represent the absolute direction from the present point to next possible point.

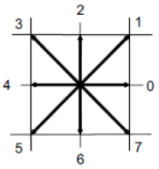


Fig. 1 The 8-directional Freeman chain code [1].

Since there is some correlation between succesive directions, the Angle Freeman chain code [3] calculates the angle differences between successive directions. In Fig. 2, the eight cases of the Angle Freeman chain code are illustrated. For instance, the symbol 0 means the present direction is the same as the previous direction, while symbol 1 indicates that the present direction is rotated 45 degrees from the previous direction.

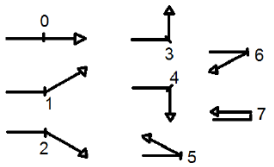


Fig. 2 The eight cases of the Angle Freeman chain code [3].

The encoding technique of Adaptive binary arithmetic coding [9] is similar to that of binary arithmetic coding, except for the adaptation of frequency table. The frequency table will be adjusted according to the input symbol, which is more

flexible than the static frequency table. Besides, by using context modeling, the input symbols will be classified into different cases based on the previous encoded symbols. With the reasonable contexts, the distribution of frequency tables would be more suitable for encoding, and then the coding efficiency of adaptive binary arithmetic coding would be improved.

III. Proposed Lossless Contour Compression

In this section, we describe the basic idea of our proposed algorithm for encoding contours. The flowchart of the proposed lossless contour compression algorithm is shown in Fig. 3. We apply the techniques of the morphological operation, the modified Angle Freeman chain code, Huffman coding, the distribution transform, and improved adaptive binary arithmetic coding. Each of these techniques will be illustrated in the following subsections.

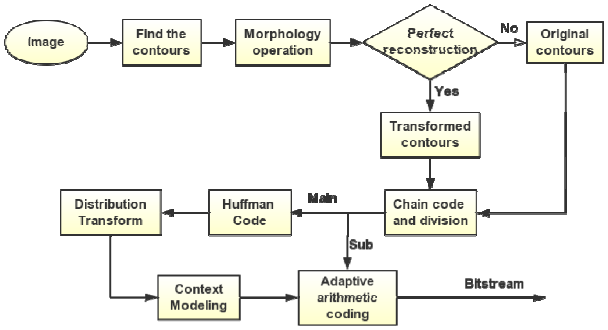


Fig. 3 The flowchart of the proposed lossless contour compression algorithm.

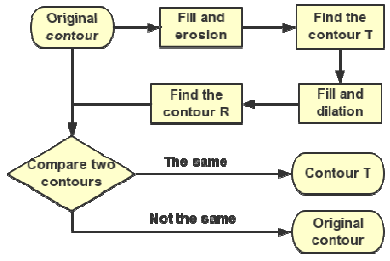


Fig. 4 The flowchart of the morphological operation.

A. Morphological operation

The objective of the morphological operation is shrinking the contours to decrease the length of chain code, but the contours can be reconstructed perfectly after operation. The flowchart of the morphological operation is shown in Fig. 4. We will first change the original contour into transformed contour, which is denoted as *T*, by two morphology processes (hole filling and erosion). Then the contour *T* is used to reconstruct the contour, which is denoted as *R*, by using hole filling and dilation. If the contour *R* is the same as the original contour, the contour *T* will be used to run the next step, otherwise, the original contour will be chosen to go on.

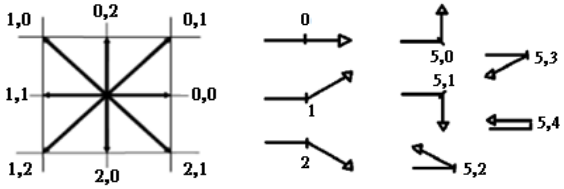
As we know, Erosion is a morphological process to shrink the image, while dilation is applied to expand the image. As a result, the selection of the structure element is very important because the erosion is not the inverse of the dilation. We choose the structure element as a 2 x 1 rectangle, and simulate with the images in [10] and [11]. With the morphological operation, six bits in average will be further saved.

B. Modified Angle Freeman Chain Code

When testing with the Angle Freeman chain code [3], we found that the symbols appearing in a chain code are almost 0, 1, 2, i.e. the angle difference between successive directions are mostly 0 degree and 45 degrees to the right or left.

First, to decrease the symbol diversity, we use another symbol (ex: 5) to substitute symbols except for 0, 1, 2, and use a sub-chain to distinguish them. As shown in Fig. 5(b), to represent the relative direction except for 0, 45, -45 degrees, symbol 5 will be recorded in the main-chain, and the second symbol would be stored in the other chain, which is named the sub-chain.

Besides, in the first code, because there is no previous direction, the chain code will follow the scheme in Fig. 5(a) to prevent from the appearing of symbols other than 0, 1, and 2. After the steps above, we will get the main-chain code composed of {0, 1, 2, 5}. Here, two other symbols (3 and 4) are added to substitute the combination of {1, 2} and {2, 1} because symbol 1 and 2 usually appear successively.



(a) (b)

Fig. 5 The modified chain code used in the proposed method.

C. Huffman Code

The concept of the Huffman code is using variable-length code to construct a table according to the probabilities of symbols, and then transform all the symbols into a binary stream. The Huffman code may have one disadvantage, which is that every input symbol needs at least one bit to represent.

However, the Huffman code here is used as an intermediate code. There are two reasons to apply the Huffman code. One is transforming the chain code into binary stream based on the statistics of the chain code. Not only will the binary stream be shorter than the one using fixed-length code, but it can reconstruct the original chain code easily according to the code table. The other one is that the binary stream will have some regularity when assigning variable-length code, which may result in the effectiveness of the distribution transform in subsection *D*. To be mentioned, the bit to determine whether the morphological operation in subsection *A* is applied will be added at the beginning of the binary stream.

Taking the "butterfly" image in Fig. 6 for example, we will calculate the probabilities of symbols occurring in the chain code, and record the descending order of the probabilities. The probability table is shown in Table I. According to this table, the chain code will be transformed into the binary stream according to the corresponding Huffman binary codes.

TABLE I. THE PROBABILITY TABLE OF THE "BUTTERFLY" IMAGE IN FIG. 6

Symbol	0	1	2	3	4	5
Probability	0.545	0.051	0.049	0.214	0.137	0.004
Binary code	0	1110	11110	10	110	11111

D. Distribution Transform

After transforming into the binary code in subsection *C*, there are lots of zeros and ones. The issue we are concerned about is how we can get higher compression rate. For instance, if there are two sequences, one is "10110", and the other is "11100". Both sequences have the same amount of 0's and 1's, but the latter sequence will have a better compression result due to the data distribution.

Accordingly, to make the binary stream more compressible, we combine two transforms, which are the Burrows-Wheeler transform and differential coding. We name the combination of these two transforms the "distribution transform" for convenience. The goal of the distribution transform is to make the binary stream filled with runs of the same bit, and to increase the probability of zeros.

The Burrows-Wheeler transform is a transform to rearrange the elements in a string based on sorting technique. The steps are shown in Table II with the example of the sequence "10110". First, we construct a conceptual matrix, whose rows are the circular rotations of the sequence. Second, the matrix is sorted by rows in ascending order. Third, we will get the last column of the sorted matrix, and output the transformed sequence (11100) with a primary index, which is an important index to recover the sequence. The inverse Burrows-Wheeler transform also uses the sorting technique. Starting with the primary index for first index, the original sequence (10110) can be perfectly reconstructed from the transformed sequence (11100).

After applying the Burrows-Wheeler Transform, the elements will appear as successively as possible if there is some regularity in the original binary stream. For example, if the binary stream contains the combination of "10" frequently. After sorting the circular rotations, the rotations starting with "0" will be grouped together, which will make the last bit of those rotations be "1". In consequence, "1" will be grouped together, and the distribution of the same bit will become denser. That is the reason the chain code is transformed by fixed Huffman code in subsection *C*. Some combinations will occur frequently, such as the combination of {1,0} and {1,1,0}, which results in the effectiveness of the Burrows-Wheeler transform.

The second part of the distribution transform is differential coding, which is used to increase the probabilities of zeros. The previous bit is compared with the present bit, as shown in (1). If the previous bit is the same as the present bit, the output bit is set as 0. On the other hand, if the previous bit is different from the present bit, the output bit is set as 1.

$$y_i = x_{i-1} \oplus x_i \ , \tag{1}$$

where $\oplus$ indicates module-2 addition, x_i is the present bit, and y_i is the transformed bit. y_1 is set as the inverse bit of x_1 because after the Burrows-Wheeler transform, the first bit of the binary stream is always 1 unless the binary stream is all zeros. For example, the sequence "11100" will become "00010" after the differential coding.

The other example is the "butterfly" image in Fig. 6. There are 1011 zreos and 872 ones totally in original binary stream. If we call a series of repeated bits a "run", there are 916 runs originally. After the Burrows-Wheeler transform, the amount of zeros and ones remains the same, but the number of runs becomes 570. After differential codeing , the number of runs becomes 639, but the amount of zeros becomes 1314, and the amount of ones becomes 569. After the distribution transform, we can find that not only does the runs decrease, but the amount of zeros increase a lot.

TABLE II. AN EXAMPLE FOR DEMONSTRATING THE BURROWS-WHEELER TRANSFORM

Burrows-Wheeler Transform					
Input	All rotations	Sort all rows into ascending order	Take last column	Output last column	Output the primary index (the position of (2))
10110	10110 (1) 01101 (2) 11010 (3) 10101 (4) 01011 (5)	01011 (5) 01101 (2) 10101 (4) 10110 (1) 11010 (3)	0101**1** (5) 0110**1** **(2)** 1010**1** (4) 1011**0** (1) 1101**0** (3)	11100	2

E. Context Modeling for Adaptive Binary arithmetic coding

Before adaptive binary arithmetic coding, different contexts are used to improve the compression efficiency. The contexts are based on the regularity after the distribution transform and the high probability of zeros.

First, we may find many "impulses" after the distribution transform, like the sequence "00010", where one "1" will occur between two runs of zeros. Also, to prevent from the case that the Burrows-Wheeler transform does not work well, it may result in two "1"s between two runs of zeros, such as "00011000", after differential coding. Therefore, the following contexts are adopted to meet the needs, where the contexts are trained from the compression result of images in Fig. 6.

(i) Context 1-5: if n bits before the previous bit are successive zeros and the previous bit is the same as the present bit, where n = 28, 18, 1, 3, 5.
(ii) Context 6-8: if k bits before the previous bit are successive zeros and the previous bit is different from the present bit, where k = 1, 3, 5, respectively.
(iii) Context 9-11: if k bits before the previous bit are successive ones and the previous bit is the same as the present bit, where k = 1, 3, 5, respectively.
(iv) Context 12-14: if k bits before the previous bit are successive ones and the previous bit is different from the present bit, where k = 1, 3, 5, respectively.
(v) Context 15: if the amount of zeros appearing in the previous 19 bits is more than 15.
(vi) Context 16: the case other than context 1-15.

At the beginning of adaptive binary arithmetic coding, we will construct sixteen frequency tables, with the probability of zeros higher than that of ones. The impulses might be handled by context 6-8 and 12-14. Taking the sequence "00010" for

example, when the encoded bit is '1', it will go into context 6 ('00**1**'), and the latter bit is '0', but it will go into context 7 ('0001**0**'). The positive edge and the negative edge of the impulse run into different contexts. If the assumption works well, where the amount of complex contexts is larger than that of simple contexts, it may result in the disparity of probabilities in frequency tables, which is suitable for encoding. Also, the classification of context 1-5, 9-11 is used to handle the case that two identical bits occurring successively after a run of zeros or ones.

IV. SIMULATIONS

In this section, several simulations are performed to compare the compression performance of proposed algorithm with other chain code methods, including Freeman Eight Directional Chain Code (*FEDCC*) [1], Difference Freeman Chain Code (*DFCCE*) [2], Angle Freeman Chain Code (*AF8*) [3], Compressed Vertex Chain Code (*C_VCC*) 0, the modified directional Freeman Chain Code in eight directions by a set of nine symbols (*MDF9*) 0, Vertex Chain Code [4] using move-to-front transform and adaptive run-length coding [7] (*MTFT*).

The dataset 1 includes eleven contour images in [10], shown in Fig. 6, and the dataset 2 includes five hundred contour images randomly picked from [11]. To be mentioned, binary arithmetic coding is used to encode the other chain codes, except for *MTFT*. The results are shown in Tables III, IV, where *Avg* is the abbreviation for average. Since adaptive binary arithmetic coding is applied, there is no overhead information in Tables III and IV except for MTFT.

Note that, in the dataset 1, using the proposed algorithm can save 26.9% when comparing with *AF8*, while in the dataset 2, 14.2% of bits can be saved by the proposed algorithm.

All the simulation results show that the proposed algorithm has the best coding performance for compressing the contours.

V. CONCLUSION

In this paper, an effective algorithm to compress the contours losslessly was proposed. The algorithm is based on the transformation of the chain code and entropy coding. We apply the techniques of (i) the morphological operation, (ii) the modified Angle Freeman chain code, (iii) the distribution transform, and (iv) context modeling for adaptive binary arithmetic coding. Simulations in Section IV show that, for contour images, the proposed lossless compression can significantly reduce the data size requirement when comparing with other existing chain code methods.

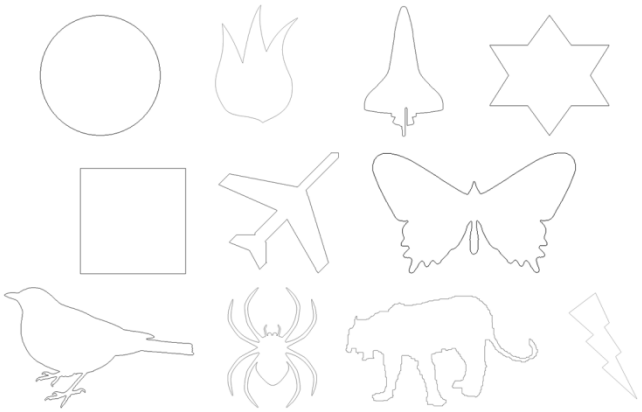


Fig. 6 The images in Zalic's dataset [10] (*P1*: circle, *P2*: fire, *P3*: shuttle, *P4*: star, *P5*: square, *P6*: plane, *P7*: butterfly, *P8*: bird, *P9*: spider, *P10*: tiger, *P11*: thunder).

TABLE III. COMPARISON OF THE DATA SIZES (BY NUMBER OF BITS) REQUIRED FOR ENCODING **CONTOURS IN DATASET 1** [10] USING DIFFERENT CHAIN CODE METHODS AND THE PROPOSED ALGORITHM.

	FEDCC [1]	*DFCCE* [2]	*AF8* [3]	*C_VCC* 0	*MDF9* 0	*MTFT* [7]	**Proposed Method**
P1	2368	1264	680	1320	912	1069	**521**
P2	5744	3416	1928	3424	2576	2711	**1648**
P3	2552	1384	1456	1240	1112	1234	**1055**
P4	3128	1568	1672	1352	1160	1569	**511**
P5	2288	136	152	128	192	328	**140**
P6	4216	2312	2576	2624	1992	2321	**1695**
P7	4496	2256	2432	1616	1784	1547	**863**
P8	4832	2808	2304	2800	2432	2897	**2176**
P9	11680	6504	3896	7616	5048	6421	**3467**
P10	3648	2336	1456	1616	1568	1507	**768**
P11	9680	4440	4360	3592	5124	5440	**3912**
Avg	4967	2584	2083	2484	2173	2459	**1523**

TABLE IV. COMPARISON OF THE DATA SIZES (BY NUMBER OF BITS) REQUIRED FOR ENCODING **CONTOURS IN DATASET 2** [11] USING DIFFERENT CHAIN CODE METHODS AND THE PROPOSED ALGORITHM.

	FEDCC [1]	*DFCCE* [2]	*AF8* [3]	*C_VCC* 0	*MDF9* 0	*MTFT* [7]	**Proposed Method**
Avg	4041	2418	2315	2319	2288	2664	**1987**

REFERENCE

[1] H. Freeman, "Computer processing of line drawing images," *ACM Copmut.* Surveys, vol. 6, pp. 57-59, 1974.

[2] H. Freeman, "Application of the generalized chain coding scheme to map data processing," *Proceedings of IEEE Pattern Recognition and Image Processing*, pp. 220-226, 1978.

[3] Y. K. Liu and B. Zalik, "An efficient chain code with Huffman coding," *Pattern Recognition*, vol. 38, pp. 553-557, 2005.

[4] E. Bribiesca, "A new chain code," *Pattern Recognition*, vol. 32, pp. 235-251, 1999.

[5] Y. K. Liu, W. Wei, P.-J. Wang, and B. Zalik, "Compressed vertex chain codes," *Pattern Recognition*, vol. 40, pp. 2908-2913, 2007.

[6] H. Sanchez-Cruz, "Proposing a new code by considering pieces of discrete straight lines in contour shapes," *Journal of Visual Communication and Image Representation*, vol. 21, pp. 311-324, 2010.

[7] B. Žalik and N. Lukač, "Chain code lossless compression using move-to-front transform and adaptive run-length encoding," *Signal Processing: Image Communication*, vol. 29, pp. 96-106, 2014.

[8] M. Burrows and D. Wheeler, "A block sorting lossless data compression algorithm," Digital Equipment Corporation, Palo Alto, CA, Technical Report 124, 1994.

[9] A. Moffat, Alistair, R. M. Neal, and I. H. Witten. "Arithmetic coding revisited," ACM Transactions on Information Systems, vol. 16, pp. 256-294, 1998.

[10] Zalik's chain codes dataset, http://gemma.uni-mb.si/chaincodes/

[11] MPEG7 shape dataset, http://www.dabi.temple.edu/~shape/MPEG7/dataset.html

[12] C. C. Lu and J. G. Dunham. "Highly efficient coding schemes for contour lines based on chain code representations," *IEEE Trans. Commun.*, vol. 39, issue 10, pp. 1511-1514, 1991.