Image Compression with Modified LZ77 Coding

Hideo Nagumo, Mi Lu, and Karan Watson
Department of Electrical Engineering, Texas A&M University
College Station, TX 77843-3128, USA
E-mail: {nagumo, mlu, watson}@eesun1.tamu.edu

Abstract

We present in this paper a new coding method for lossless gray-scale image compression. Our method is based on modifications of LZ77-based compression and estimation of prediction errors of pixels. Our method gives compression comparable to JPEG lossless mode and its speed is about twice that of JPEG lossless mode with arithmetic coding.

1 Introduction

We present in this paper a new coding method for lossless gray-scale image compression. Our method is based on modifications of LZ77-based compression and estimation of prediction errors of pixels. Our method gives compression comparable to JPEG lossless mode and its speed is about twice that of JPEG lossless mode with arithmetic coding.

JPEG lossless mode employs one of seven different prediction schemes of pixels and Huffman or arithmetic coding [1]. Huffman coding suffers from the fact that it cannot assign fractional number of bits to a pixel and is not efficient when compressing large monotone regions. Arithmetic coding does not have this problem, but it suffers from its slow speed.

There is another popular lossless image compression method called FELICS, which was developed by Howard and Vitter [2]. This method uses context-based prediction and Rice-Golomb coding of prediction errors. It gives compression comparable to JPEG lossless mode with about five times the speed. However, like Huffman coding, it cannot assign less than one bit to a pixel value [3]. Also it cannot employ different prediction schemes.

With our method, at first we estimate the number of bits to express the prediction error of a pixel in the lookahead buffer. Then, we compute the actual prediction error. If the prediction error can be expressed with the estimated number of bits, we declare that this pixel has approximately matched the prediction. We encode the sequence of approximately matched pixels in the lookahead buffer with the length of the sequence followed by their prediction errors expressed with their estimated number of bits.

2 Compression Algorithm

To describe our compression method, we use the following notations.

I: value of a pixel to be encoded. $\hat{I}$: prediction of I.
E: prediction error of I ($E = I - \hat{I}$).
$\hat{E}$: estimation of E.
B: number of bits to express E. $\hat{B}$: estimation of B.
$I_{i,j-1}, I_{i-1,j}, I_{i-1,j-1}$: values of immediate left pixel, immediate above pixel, and immediate left above pixel of the pixel to be encoded.

2.1 LZ77-based Compression

In an LZ77-based compression, the encoder has a sliding window that is divided into a search buffer and a lookahead buffer [4]. A search buffer contains a portion of the recently encoded input string, and a lookahead buffer contains the next portion of the string to be encoded. During compression, the encoder searches the longest pattern in the search buffer that matches the pattern starting at the first symbol in the lookahead buffer. If the length of the pattern match is greater than zero, this pattern is encoded with the position and the length. Otherwise, literal value of the first symbol in the lookahead buffer is coded.

2.2 Modifications on LZ77

Gray-scale images have the property that neighboring pixels are highly correlated. Therefore, we have modified LZ77-based compression so that it compares the pixels in the lookahead buffer with their predictions and accepts approximate pattern match.

Instead of searching for the pattern matches in the search buffer, we compare each pixel in the lookahead buffer with its prediction and encode the lengths of the pattern matches. Since we have only one pattern (the pattern made up of the predictions of the pixels in the lookahead buffer) to compare the pattern in the lookahead buffer with, we do not need to encode the positions of the pattern matches in this case.

When we compare the pixel to be encoded with its prediction, if their difference (*prediction error*) is within a certain range, we declare that the pixel *approximately matches* its prediction. When a pattern match is found with this method, we have to encode the prediction errors as well as the length of the match.

The range the prediction errors can fall in is determined by the number of bits to express the prediction

0-7803-2912-0

errors. We estimate the number of bits to express the prediction error of a pixel using its neighboring pixels that have been encoded earlier. Thus the decoder can also estimate the number of bits without requiring any side information.

With our method, the size of the lookahead buffer determines the number of bits to express the length of the pattern match. We change the lookahead buffer size adaptively so that different regions of an image with different characteristics can be compressed efficiently.

Our compression method works as follows. At first, we find the approximate pattern match between the pattern in the lookahead buffer and the pattern made up of the predictions of the pixels in the lookahead buffer. We estimate the number of bits to express the prediction error for a pixel in the lookahead buffer. Then, we compute the prediction of the pixel using its neighboring pixels. After that, we compute the actual prediction error, and determine if the prediction error can be expressed with the estimated number of bits. If so, we declare that the pixel has approximately matched its prediction. We continue finding approximate matches of pixels from the first pixel in the lookahead buffer towards the end of the lookahead buffer until there is a pixel that does not approximately match its prediction or until the maximum length for the pattern match is reached.

Once an approximate pattern match is found, we encode it with the length of the pattern and the prediction errors. The information of a pattern match is always followed by the information of the next pixel (with the exception of the case where a pattern match is ended because the end of the raster-scan line is reached) so that we can advance at least one position even if the first pixel in the lookahead buffer does not approximately match. After an approximate pattern match and the next pixel are encoded, we adjust the lookahead buffer size.

2.3 Estimation of Prediction Errors

The process of obtaining the estimation of the number of bits to express the prediction errors critically affects the performance of our compression method. We empirically chose the following method.

We first determine the estimation of the prediction error ($\hat{E}$) for I using the equation

$$\hat{E} = (|I_{i-1,j} - I_{i-1,j-1}| + |I_{i,j-1} - I_{i-1,j-1}|)/2.$$

Then, we map $\hat{E}$ to the number of bits $\hat{B}$ using the mapping table

$\hat{E}$	$\hat{B}$	$\hat{E}$	$\hat{B}$
0	0	16, $\cdots$, 31	5
1	1	32, $\cdots$, 63	6
2, 3	2	64, $\cdots$, 127	7
4, $\cdots$, 7	3	128, $\cdots$	8
8, $\cdots$, 15	4.		

Since we use only the neighboring pixels that have been encoded earlier to compute the estimation of the prediction error, the decoder can also compute the same estimation without any side information.

2.4 Computation of Prediction

We can combine any prediction scheme with our coding method. To compare our method with JPEG lossless mode, we used the four two-dimensional prediction schemes of JPEG lossless mode [1]. They are called prediction scheme 4, 5, 6, and 7.

After computing the prediction of a pixel, we subtract it from the pixel value to obtain the prediction error E, and then find the number of bits to express it. The range for the n-bit two's complement number is -2^{n-1} to $2^{n-1}-1$, and any number in this range can be uniquely represented using n bits. Thus the number of bits to express the prediction error B is given by the following equation.

$$B = \begin{cases} 0 & \text{if } E = 0 \\ \lceil \log(E+1) \rceil + 1 & \text{if } E > 0 \\ \lceil \log(-E) \rceil + 1 & \text{if } E < 0 \end{cases}$$

In our program, we used a table to map E to B. If $E \le \hat{E}$, we declare that the pixel has approximately matched its prediction.

2.5 Encoding Prediction Errors

For encoding a prediction error, we have the following two cases.

1. The prediction error is within the range -2^{n-1} to $2^{n-1}-1$.
2. The prediction error is within the range -2^{n-1} to $2^{n-1}-1$, but not within the range -2^{n-2} to $2^{n-2}-1$.

In the first case, we can simply truncate the prediction error to n bits to encode it. To decode the error in this case, we sign-extend the n-bit value to form a signed integer.

To describe the process of encoding for the second case, we divide the range for the n-bit two's complement number into four equal parts, and investigate the fixed most significant two bits. The lower bits can assume any value.

-2^{n-1} to $-2^{n-2}-1$	: 10
-2^{n-2} to -1	: 11
0 to $2^{n-2}-1$	: 00
2^{n-2} to $2^{n-1}-1$	: 01

For the second case, we use the two ranges -2^{n-1} to $-2^{n-2}-1$ and 2^{n-2} to $2^{n-1}-1$. We can uniquely express any value in these two ranges without the most significant bit (MSB). Thus in this case, we can truncate the prediction error to $n-1$ bits to encode it. In these ranges, the MSB and the second MSB are complement to each other. Therefore, to decode the error, we can perform opposite sign-extension on the $(n-1)$-bit value to form an integer. That is, if the MSB of the $(n-1)$-bit value is 0, we know the error is negative, so we fill the upper bits with 1's; if the MSB of the $(n-1)$-bit value is 1, we know the error is positive, so we fill the upper bits with 0's.

Encoding the prediction error of a pixel in an approximately matched pattern belongs to the first case. Therefore, we encode the prediction error by truncating it into $\hat{B}$ bits. To decode the prediction error, the decoder computes $\hat{B}$, picks $\hat{B}$ bits from the input string to the decoder, and then sign-extends it.

2.6 Encoding Pixel after Pattern Match

An approximate pattern match is always followed by the information about the next pixel. However, the next pixel does not necessarily match its prediction, and when it does not, the decoder has to know the number of bits to express the prediction error B. Therefore, when encoding the next pixel, we have to encode the information about B so that the decoder can derive B from this information and $\hat{B}$, which the decoder can obtain locally.

There are three different conditions to consider for encoding the next pixel, depending on whether the maximum length for the pattern match is reached, and whether the next pixel approximately matches its prediction.

1. The maximum length for the pattern match is not reached.
2. The maximum length for the pattern match is reached, and the next pixel does not match its prediction.
3. The maximum length for the pattern match is reached, and the next pixel matches its prediction.

In the first condition, the decoder can know from the condition that the next pixel did not match its prediction, or it can know that $B \geq \hat{B}+1$. We encode the information about B with m bits of 0's followed by one bit of 1. For example, if $m=0$ then $B=\hat{B}+1$, if $m=1$ then $B=\hat{B}+2$, and so on. Thus $m=B-\hat{B}-1$, and the decoder can compute B using the equation $B=\hat{B}+1+m$. In this condition, the decoder knows that the prediction error is within the range -2^{B-1} to $2^{B-1}-1$, but not within the range -2^{B-2} to $2^{B-2}-1$. This is the second case in Section 2.5. Thus we encode the prediction error by truncating it to $B-1$ bits, and the decoder recovers it by performing the opposite sign-extension. In total, we need $m+1+B-1$ bits or $2B-\hat{B}-1$ bits to encode the next pixel in the first condition.

If the maximum length for the pattern match is reached as in the second and third condition, the decoder cannot know whether the next pixel has approximately matched its prediction or not until it receives the next bit. In the second condition, we encode the information about B with $m=B-\hat{B}$ bits of 0's followed by one bit of 1. Here, m is at least one because $B>\hat{B}$. This is the second case in Section 2.5. Thus we encode the prediction error by truncating it to $B-1$ bits, and the decoder recovers it by performing the opposite sign-extension. In total, we need $m+1+B-1$ bits or $2B-\hat{B}$ bits to encode the next pixel in the second condition.

In the third condition, we encode the information about B with one bit of 1 without any preceding 0's ($m=0$). With this 1, the decoder can distinguish the third condition from the second. This is the first case in Section 2.5. Thus we encode the prediction error by truncating it to B bits, and the decoder recovers it by performing the sign-extension. In total, we need $1+B$ bits to encode the next pixel in the third condition.

2.7 Lookahead Buffer Size

In our compression method, the size of the lookahead buffer is one more than the maximum length for the pattern match, because we can encode at most the pixels in the pattern match with the maximum length and the next pixel in one step. When we use n bits to express the length of the pattern match, the range of the length is 0 to 2^n-1, and the lookahead buffer size is 2^n. If all the pixels in the lookahead buffer approximately match their predictions we double the size of the buffer. If the approximate pattern match is shorter than half of the buffer, we decrease the size of the buffer to $2^{\lceil \log_2(p+1) \rceil}$ where p is the number of pixels in the patten match (in case $p=0$, the size of buffer becomes two).

3 Experimental Results

We compare our method (modified LZ77) with JPEG lossless mode with arithmetic coding, and with the UNIX *compress* program. We run the compression programs on a Sun SPARC station 20. Table 1 shows the compression ratios, which are expressed as original size divided by compressed size. For our method and for JPEG, we used the four two-dimensional prediction schemes (4, 5, 6, 7) of JPEG lossless mode. Table 2 shows the encoding and decoding throughput, which is expressed as thousands of pixels processed per second. For the results of our method and of JPEG in Table 2, we used the prediction scheme 7 of JPEG lossless mode. Although it is not shown in the table, the average encoding and decoding throughput for the UNIX *compress* are 252.3 and 436.2 respectively.

As we can see from the tables, compression of our method is comparable to that of JPEG lossless mode with arithmetic coding, and its speed is about twice that of JPEG lossless mode. Our method is particularly efficient for images that are mixtures of completely different regions like the image earth.

Table 1: Compression ratios.

image (size)	modified LZ77				lossless JPEG				compress
	P4	P5	P6	P7	P4	P5	P6	P7	
chart (256 x 256)	5.55	5.30	5.34	5.04	4.73	4.73	4.20	4.10	6.01
clock (256 x 256)	1.97	1.94	1.99	1.97	1.88	1.89	1.98	1.93	1.30
earth (256 x 256)	2.28	2.33	2.33	2.38	1.95	1.98	1.98	2.01	2.03
girl (256 x 256)	1.69	1.73	1.74	1.78	1.66	1.73	1.73	1.77	1.40
girl2 (256 x 256)	1.89	1.98	1.95	2.04	1.94	2.08	1.98	2.08	1.59
girl3 (256 x 256)	1.72	1.72	1.76	1.76	1.78	1.73	1.85	1.75	1.19
house (256 x 256)	1.73	1.77	1.69	1.76	1.76	1.81	1.74	1.75	1.28
jelly (256 x 256)	2.56	2.49	2.54	2 55	2.47	2.62	2.71	2.71	1.85
moon (256 x 256)	1.37	1.42	1.44	1.43	1.43	1.49	1.51	1.57	1.08
plane (256 x 256)	2.07	2.10	2.12	2 15	2.04	2.15	2.16	2.22	1.48
town (256 x 256)	1.29	1.32	1.30	1.34	1.31	1.35	1.33	1.35	< 1
tree (256 x 256)	1.46	1.50	1.48	1.52	1.40	1.46	1.42	1.46	1.07
aerial (512 x 512)	1.49	1.52	1.49	1.52	1.45	1.49	1.46	1.47	1.10
baboon (512 x 512)	1.16	1.20	1.18	1.24	1.18	1.23	1.21	1.25	< 1
couple (512 x 512)	1.64	1.66	1.65	1.65	1.71	1.71	1.70	1.66	1.17
f16 (512 x 512)	1.77	1.79	1.78	1.81	1.76	1.82	1.80	1.81	1.29
lenna (512 x 512)	1.50	1.53	1.56	1.56	1.54	1.58	1.62	1.63	1.07
peppers (512 x 512)	1.41	1.48	1.49	1.56	1.44	1.51	1.52	1.58	1.11
splash (512 x 512)	1.76	1.80	1.80	1.84	1.76	1.82	1.83	1.86	1.32
tiffany (512 x 512)	1.50	1.55	1.56	1.60	1.51	1.56	1.59	1.62	1.19
average	1.89	1.91	1.91	1.93	1.84	1.89	1.87	1.88	1.59

Table 2: Encoding and decoding throughput.

image	modified LZ77		lossless JPEG	
	encode	decode	encode	decode
chart	163.8	159.8	62.4	47.8
clock	115.0	104.0	55.1	47.1
earth	126.0	121.4	54.2	45.5
girl	109.2	104.0	53.7	46.8
girl2	113.0	107.4	60.1	52.4
girl3	105.7	97.8	55.5	46.5
house	105.7	99.3	53.7	45.5
jelly	126.0	119.2	67.6	59.6
moon	97.8	93.6	49.3	42.0
plane	117.0	109.2	62.4	53.7
town	96.4	91.0	40.2	33.4
tree	100.8	95.0	44.0	35.8
aerial	89.8	88.0	44.7	37.6
baboon	80.4	80.4	35.0	28.1
couple	92.6	93.6	52.0	44.4
f16	97.5	96.4	54.3	47.1
lenna	91.0	90.7	50.2	43.6
peppers	89.8	90.4	48.6	41.7
splash	100.0	97.5	56.5	49.2
tiffany	93.3	90.4	50.2	43.5
average	105.5	101.5	52.5	44.6

4 Conclusions

We presented a new coding method for lossless gray-scale image compression, which was based on modifications of LZ77-based compression and estimation of prediction errors of pixels.

Our method is suitable for hardware implementation because there are many processes that can be performed in parallel. Therefore, we plan to implement our method in hardware. We also plan to extend this research by combining context based prediction schemes with our coding method.

References

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG : still image data compression standard.* Van Nostrand Reinhold, 1992.

[2] P. G. Howard and J. S. Vitter, "Fast and efficient lossless image compression," in *Proceedings IEEE Data Compression Conference*, pp. 351–360, IEEE Computer Society Press, 1993.

[3] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes : compressing and indexing documents and images.* Van Nostrand Reinhold, 1994.

[4] K. Sayood, *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc., 1996.