CS--1994--14

Fast Progressive Lossless Image Compression

Paul G. Howard, Jeffrey S. Vitter

Department of Computer Science

Duke University

Durham, North Carolina 27708-0129

March 25, 1994

Fast Progressive Lossless Image Compression

Paul G. Howard and *Jeffrey Scott Vitter*

To appear in

Image and Video Compression
IS&T/SPIE Symposium on Electronic Imaging: Science & Technology
San Jose, California
February 6 10, 1994

Paper number 2186 12

Fast progressive lossless image compression

Paul G. Howard* and Jeffrey Scott Vitter†

*AT&T Bell Laboratories, Visual Communications Research, Holmdel, New Jersey 07733 3030
†Duke University, Department of Computer Science, Durham, North Carolina 27708 0129

ABSTRACT

We present a method for progressive lossless compression of still grayscale images that combines the speed of our earlier FELICS method with the progressivity of our earlier MLP method. We use MLP's pyramid-based pixel sequence, and image and error modeling and coding based on that of FELICS. In addition, we introduce a new prefix code with some advantages over the previously used Golomb and Rice codes. Our new progressive method gives compression ratios and speeds similar to those of non-progressive FELICS and those of JPEG lossless mode, also a non-progressive method.

The image model in Progressive FELICS is based on a simple function of four nearby pixels. We select two of the four nearest known pixels, using the two with the middle (non-extreme) values. Then we code the pixel's intensity relative to the selected pixels, using single bits, adjusted binary codes, and simple prefix codes like Golomb codes, Rice codes, or the new family of prefix codes introduced here. We estimate the coding parameter adaptively for each context, the context being the absolute value of the difference of the predicting pixels; we adjust the adaptation statistics at the beginning of each level in the progressive pixel sequence.

1. INTRODUCTION

Lossless compression of still images is required in a number of scientific and engineering disciplines, notably space science, medical imagery, and nondestructive testing of materials. Progressive compression methods provide a gradual increase in precision or spatial resolution over the entire image, in contrast to raster-scan methods, in which pixels are coded to full precision and resolution along the rows of the image. In effect, a progressive coding consists of a typically small lossy part followed by a lossless part. The parts can be separated, allowing a user to browse a set of lossily-compressed images prior to ordering a particular image for lossless transmission or display.

We have recently developed a very fast non-progressive (raster-scan based) lossless image compression method called FELICS[1]; it gives compression comparable to that of the lossless mode of the JPEG standard for image compression (also a non-progressive method) while running three to five times as fast. We have also developed a progressive method called MLP[2,3], based on a hierarchical pixel sequence. MLP consistently gives the best compression ratios reported in the literature for lossless compression of grayscale images, about 6 to 10 percent better than JPEG lossless mode; however, it is compute-intensive and runs very slowly.

In this paper we show that we can combine the hierarchical pixel sequence of MLP with the fast

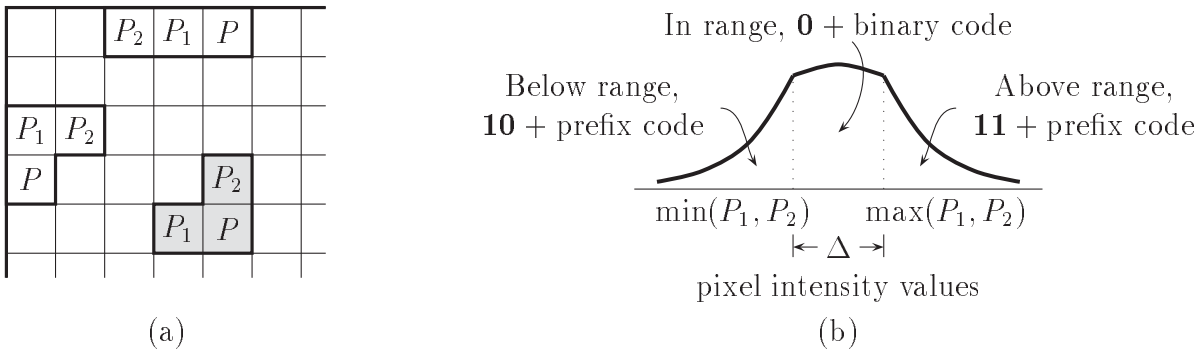


Figure 1: FELICS coding process. (a) Coding context, consisting of the two nearest neighbors, P_1 and P_2. The shading illustrates the normal case of a pixel in the interior of the image. (b) Coding for different intensity ranges relative to the larger and smaller of the two context pixels. The probability of an in-range event is about equal to that of an out-of-range event, and above-range and below-range events are about equally likely.

image and error modeling and coding of FELICS. The result is a progressive coder which we call Progressive FELICS; it runs up to twice as fast as JPEG lossless mode while providing slightly better compression ratios. In Section 2 we provide background for Progressive FELICS by briefly describing both the original non-progressive FELICS method and the MLP method. In Section 3 we discuss the image modeling aspects of Progressive FELICS, and in Section 4 we describe the coding aspects, including the new family of prefix codes. In Section 5 we give experimental results.

2. BACKGROUND

Our *Fast, Efficient, Lossless Image Compression System* (FELICS) obtains its speed by using a simplified but fairly accurate model of the image together with a combination of a number of very fast, slightly suboptimal prefix codes. The image model in FELICS is based on a pixel's two nearest previously coded neighbors. We use a raster-scan pixel sequence, so for each pixel away from the edges the nearest pixels are the immediate neighbors to the left and above. (See Figure 1.) We treat the absolute value of the difference between the neighbors as the context for the pixel. About half the time the pixel's intensity is between that of its neighbors, so we can use one bit to indicate that fact, followed by a binary code (adjusted if the absolute difference is not one less than a power of 2) to specify the exact value. Otherwise we use one bit to specify that the pixel's intensity is outside the range of its neighbors, a second bit to indicate whether it is above or below the range, and a simple prefix code (a Golomb or Rice code, described in Section 4) to tell how far out of range the intensity value lies. Golomb and Rice codes each form a family with a single parameter that indicates how peaked the distribution is. We have devised a provably good method, described in Section 4.1, for adaptively estimating the parameter[1]; we apply the method separately to each context.

In our *Multi-Level Progressive* image compression method (MLP), the model is based on using nearby pixels in all directions. We code in levels such that before coding each level we already know

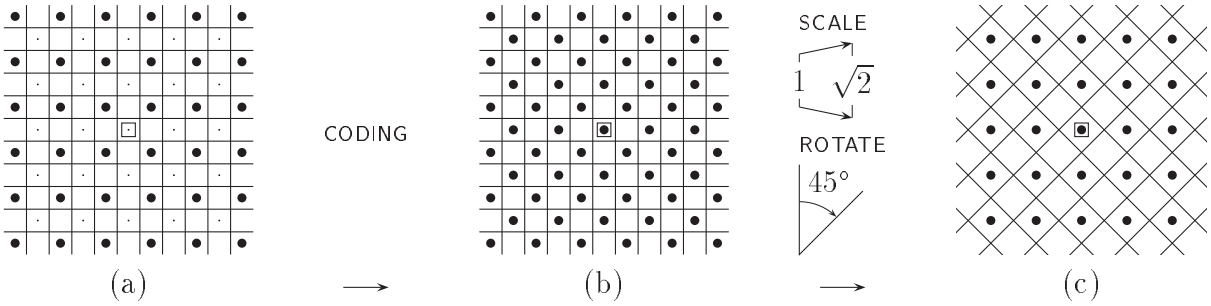


Figure 2: MLP coding process. (a) At the beginning of a level, the known pixels form a square grid. The midpoints of the grid squares (small dots) will be predicted in this level. (b) After coding the midpoints, the known pixels form a checkerboard pattern. (c) After scaling by $\sqrt{2}$ and rotation by 45°, the known pixels form exactly the same pattern as in (a). In the next level we again code the midpoints, namely all the remaining pixels.

the intensities of the pixels at the lattice points of a square grid, and we are about to code the pixels at the midpoints of the grid squares, thus doubling the number of known pixels. (See Figure 2.) For the next level we scale the coordinate system and rotate it by 45 degrees, returning to the situation of knowing the intensities of the pixels on a square grid. The MLP pixel sequence is similar to that of Knowlton[4] and others[5] [8]. In MLP we use a linear combination of 16 nearby pixels to predict the intensity of a pixel and to model the error of the prediction, and we use arithmetic coding to code the prediction error using the distribution supplied by the model.

3. PROGRESSIVE FELICS: IMAGE MODELING

In Progressive FELICS we retain the hierarchical pixel sequence of MLP to obtain a progressive coding. The image model is based on a simple function of just four nearby pixels. We select two of the four nearest known pixels and proceed as in FELICS, coding the pixel's intensity using single bits, adjusted binary codes, and simple prefix codes like Rice codes.

The selection of two of the four neighbors can be done in a number of ways. We could use the maximum and minimum values, the two middle values, an appropriately selected pair of spatially adjacent values, or a pair of spatially opposite values. We shall see in Section 5.1 that empirically the best choice is to use the two pixels with the middle (non-extreme) values, ties being broken in any consistent way. Pixels near the edges require special treatment.

After selecting the two predicting pixels, we use their absolute difference as the context. We use one bit to indicate whether the current pixel's value is in range. Then we use an adjusted binary code for the exact value within the range, or an above-range/below-range bit followed by a prefix code for an out-of-range value. The prefix codes are described in Section 4.

As in FELICS, we estimate the coding parameter separately for each context. Since the difference between the nearby pixels is correlated with the local variance of the image intensity, we find that contexts based on smaller differences correspond to more sharply-peaked distributions. However,

Table 1: The beginnings of Golomb and Rice codes for a few parameter values. The codes can be extended to all non-negative values of n, and codes can be constructed for all $m > 0$ and all $k \geq 0$. In this table a midpoint ($\cdot$) separates the high-order (unary) part from the low-order (binary or adjusted binary) part of each codeword.

Golomb Rice	$m = 1$ $k = 0$	$m = 2$ $k = 1$	$m = 3$	$m = 4$ $k = 2$	$\cdots$	$m = 6$	$\cdots$	$m = 8$ $k = 3$
$n = 0$	**0·**	**0·0**	**0·0**	**0·00**		**0·00**		**0·000**
1	**10·**	**0·1**	**0·10**	**0·01**		**0·01**		**0·001**
2	**110·**	**10·0**	**0·11**	**0·10**		**0·100**		**0·010**
3	**1110·**	**10·1**	**10·0**	**0·11**		**0·101**		**0·011**
4	**11110·**	**110·0**	**10·10**	**10·00**		**0·110**		**0·100**
5	**111110·**	**110·1**	**10·11**	**10·01**		**0·111**		**0·101**
6	**1111110·**	**1110·0**	**110·0**	**10·10**		**10·00**		**0·110**
7	**11111110·**	**1110·1**	**110·10**	**10·11**		**10·01**		**0·111**
8	**111111110·**	**11110·0**	**110·11**	**110·00**		**10·100**		**10·000**
9	**1111111110·**	**11110·1**	**1110·0**	**110·01**		**10·101**		**10·001**
⋮	⋮	⋮	⋮	⋮		⋮		⋮

because of the progressivity of this new method, we must modify the adaptive parameter estimation scheme. Because the predicting pixels become closer together with each level of the progressive coding, a given intensity difference has different implications for the underlying local image variance as we move through the levels. In Progressive FELICS we take this effect into account by adjusting the statistics used for each context's adaptive parameter estimation. At the start of each level we divide all the statistics by a scaling factor s, in effect giving less weight to data from earlier levels.

4. PROGRESSIVE FELICS: CODING

In FELICS we use Rice coding[9] to code the values of out-of-range pixels. To encode a non-negative integer n using the Rice code with parameter k, we first divide the binary representation of n into high-order and low-order parts, the low-order part containing k bits. Then we output the unary code for the high-order part, followed by the k low-order bits with no coding. Rice codes are very easy to implement and are theoretically tractable.

Rice codes are a subset of Golomb codes[10]. To encode non-negative integer n by the Golomb code with parameter m, we output $\lfloor n/m \rfloor$ in unary, followed by n mod m in a binary code, adjusted to avoid wasting code space if m is not a power of 2. If m is a power of 2 (say 2^k), we have the Rice code with parameter k. Golomb codes provide a denser choice of models, and hence usually give slightly better compression than Rice codes. Although the coding is somewhat more complicated, and more statistics must be maintained for parameter estimation, Golomb coding is only slightly slower than Rice coding, especially if the code tables are precomputed. Both families of codes are illustrated in Table 1.

Table 2: The beginnings of the new subexponential codes for a few parameter values. The codes can be extended to all non-negative values of n, and codes can be constructed for all $k \geq 0$. In this table a midpoint ($\cdot$) separates the high-order (unary) part from the low-order (binary) part of each codeword.

Subexponential	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$n = 0$	**0·**	**0·0**	**0·00**	**0·000**	**0·0000**	**0·00000**
1	**10·**	**0·1**	**0·01**	**0·001**	**0·0001**	**0·00001**
2	**110·0**	**10·0**	**0·10**	**0·010**	**0·0010**	**0·00010**
3	**110·1**	**10·1**	**0·11**	**0·011**	**0·0011**	**0·00011**
4	**1110·00**	**110·00**	**10·00**	**0·100**	**0·0100**	**0·00100**
5	**1110·01**	**110·01**	**10·01**	**0·101**	**0·0101**	**0·00101**
6	**1110·10**	**110·10**	**10·10**	**0·110**	**0·0110**	**0·00110**
7	**1110·11**	**110·11**	**10·11**	**0·111**	**0·0111**	**0·00111**
8	**11110·000**	**1110·000**	**110·000**	**10·000**	**0·1000**	**0·01000**
9	**11110·001**	**1110·001**	**110·001**	**10·001**	**0·1001**	**0·01001**
10	**11110·010**	**1110·010**	**110·010**	**10·010**	**0·1010**	**0·01010**
11	**11110·011**	**1110·011**	**110·011**	**10·011**	**0·1011**	**0·01011**
12	**11110·100**	**1110·100**	**110·100**	**10·100**	**0·1100**	**0·01100**
13	**11110·101**	**1110·101**	**110·101**	**10·101**	**0·1101**	**0·01101**
14	**11110·110**	**1110·110**	**110·110**	**10·110**	**0·1110**	**0·01110**
15	**11110·111**	**1110·111**	**110·111**	**10·111**	**0·1111**	**0·01111**
16	**111110·0000**	**11110·0000**	**1110·0000**	**110·0000**	**10·0000**	**0·10000**
⋮	⋮	⋮	⋮	⋮	⋮	⋮

In this paper we introduce a third family of simple prefix codes. Golomb and Rice codes are well-suited to the decaying exponential distributions that often occur in image coding. However, in such exponential codes the code length increases linearly with n, leading to the possibility of very long codewords for outlier values. In our new subexponential prefix code family, the codewords are identical to those of the corresponding Rice codes for $n < 2^{k+1}$, but for larger values of n the codeword lengths increase logarithmically as in Elias codes[11] instead of linearly as in Rice codes. For most images the total code length is slightly larger for these subexponential codes than for Rice or Golomb codes, but for four of our 28 test images ('crowd' and Donaldsonville Bands 1, 2, and 3) the code length is smaller. Coding speed is generally very slightly slower than Rice coding and slightly faster than Golomb coding. The subexponential codes have some interesting theoretical properties as well, allowing a simplification of the proof that our parameter estimation mechanism works well.

To code non-negative integer n using the subexponential code with parameter k, we compute

$$b = \begin{cases} k & \text{if } n < 2^k; \\ \lfloor \log_2 n \rfloor & \text{if } n \geq 2^k \end{cases} \quad \text{and} \quad u = \begin{cases} 0 & \text{if } n < 2^k; \\ b - k + 1 & \text{if } n \geq 2^k. \end{cases}$$

We output u as a unary number ($u + 1$ bits), and then output the low-order b bits of n directly.

Hence the total number of bits is given by

$$u + b + 1 = \begin{cases} k + 1 & \text{if } n < 2^k; \\ 2\lfloor \log_2 n \rfloor - k + 2 & \text{if } n \geq 2^k. \end{cases}$$

Examples of this code appear in Table 2. It can easily be shown that for a given value of n, the code lengths for adjacent values of k differ by at most 1.

4.1. Estimating the coding parameter

We briefly describe our method for estimating the coding parameter for Golomb and Rice codes and for our subexponential code, and give an informal proof that it works well for the subexponential codes.

For each context Δ we maintain a cumulative total, for each reasonable parameter value, of the code length we would have if we had used that parameter to encode all values encountered so far in the context. Then we simply use the parameter with the smallest cumulative code length to encode the next value encountered in the context. The total expected excess code length is only $O(\sqrt{N})$ bits for a context that occurs N times in the image.

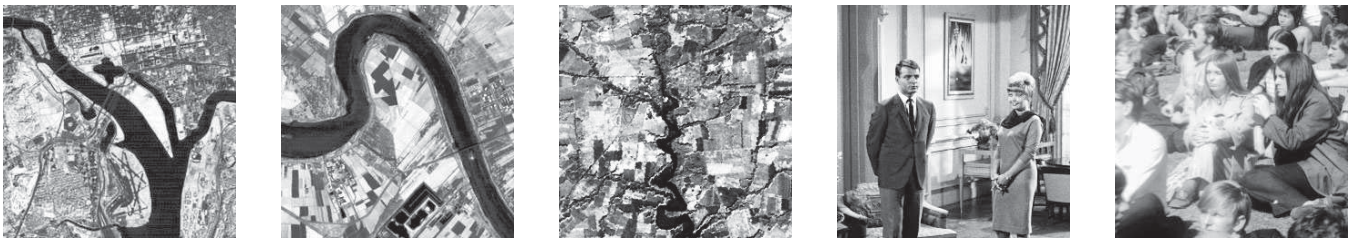
THEOREM 1. *For a stationary source, using our parameter selection algorithm in conjunction with subexponential coding gives an expected code length that exceeds the expected code length given by the optimal parameter by at most $O(\sqrt{N})$ bits, where N is the number of samples coded.*

Proof sketch: After a short startup period during which any parameter may be best, we alternate between using the right parameter and the next-best wrong one. When we are using the right parameter, we get optimal code length, by definition. We start using the wrong parameter when its cumulative code length is one bit shorter than that of the right parameter. We stop using it and switch back to the right parameter when the right parameter has a cumulative code length one bit shorter. (The differences cannot be more than one bit because the code lengths used in coding a given n differ by at most 1 for adjacent values of k; this is the property of the subexponential codes that makes the proof easier.) While we are using the wrong parameter, we accumulate exactly two bits of excess code length relative to the right parameter. Thus the excess code length is the same as the number of parameter changes. By renewal theory the expected number of changes is $O(\sqrt{N})$, and hence so is the expected excess code length. □

This theorem is more general than the corresponding theorem for Rice coding[1] since it does not depend on the probability distribution of the source.

5. TUNING AND EXPERIMENTAL RESULTS

Our test data consists of seven photographic images and 21 Landsat Thematic Mapper images, seven from each of three data sets (Washington, D.C., Donaldsonville, Louisiana, and Ridgely, Maryland). Ten of these images are illustrated in Figure 3. In Section 5.1 we support our choices for various algorithm parameters, and in Section 5.2 we compare the resulting method with other lossless image compression methods in the literature.



W4 D4 R4 couple crowd

lax lena man woman1 woman2

Figure 3: Some of the test images, shown here to identify the images and to give some idea of their characteristics. The first three images are Band 4 of the Washington, Donaldsonville, and Ridgely Landsat Thematic Mapper data sets. The Ridgely images are 468×368 pixels; all others are 512×512 pixels. In this figure all but three images have been cropped to 256×256 pixels. The exceptions are 'lax', cropped to 192×192, and 'woman1' and 'woman2', left at 512×512.

We use the following definitions in this section.

$$\begin{aligned}
\text{Compression ratio} &= \frac{\text{original size}}{\text{compressed size}}; \\
\text{Bits/pixel} &= \frac{8 \times \text{compressed size}}{\text{original size}}; \\
\text{Inefficiency} &= 100 \times \log_e \frac{\text{compressed size}}{\text{best compressed size}}; \\
\text{Throughput} &= \frac{\text{original size}}{1000 \times \text{encoding time in seconds}}.
\end{aligned}$$

Inefficiency is expressed in *percent log ratio*, denoted by $\frac{\circ}{\circ}$. Since $\log_e(1+x) \approx x$ for small x, a difference of any given small percent log ratio means almost the same thing as the same difference expressed as an ordinary percentage. Because of the logarithm in the definition, inefficiencies are additive, and can be compared by subtraction. Throughput is expressed in thousands of pixels encoded per second on a SPARCstation 10, Model 40 (40 MHz clock, about 109.5 mips). Although the results are not listed here, decoding throughput for Progressive FELICS is about the same as encoding throughput.

5.1. Selection of algorithm parameters

We must make a number of choices to fully specify the Progressive FELICS algorithm:

- the pixel pairs to be used as a context,
- the prefix code (Golomb, Rice, or subexponential) to be used in the coding step, and
- the scaling factor s to be applied to the parameter estimation statistics at the end of each level, as described at the end of Section 3.

In addition, we investigate speeding up the coding by ceasing to gather statistics for coding parameter estimation once the cumulative code length for the best parameter choice reaches a specified "freezing level" f. Tables in this subsection contain values averaged over all 28 images.

Table 3: Context selection, using Golomb coding with $s = 8$ and $f = \infty$.

Context selection method	Comp. ratio	Bits/pixel	Inefficiency
Two middle values	2.18	3.68	–
Most widely separated pair	2.08	3.85	4.6
Second most widely separated pair	2.08	3.84	4.4
Third most widely separated pair	2.06	3.89	5.7
Fourth most widely separated pair	2.13	3.76	2.3
Fifth most widely separated pair	2.03	3.95	7.1
Least widely separated pair	1.99	4.01	8.7
Most widely separated pair of adjacent pixels	2.08	3.84	4.3
Second most widely separated pair of adjacent pixels	2.06	3.89	5.7
Third most widely separated pair of adjacent pixels	2.02	3.95	7.3
Least widely separated pair of adjacent pixels	1.98	4.04	9.5
Most widely separated pair of opposite pixels	2.03	3.94	6.9
Least widely separated pair of opposite pixels	2.13	3.75	2.0

We first decide which of the many possible pairs of values to use as the two-pixel context. Table 3 compares 13 different possible contexts, using Golomb coding and reasonable choices for the other parameters. Using the two intermediate values gives the best compression for every test image except 'lax'; it is also one of the faster methods.

Table 4: Selection of coding method, using the two middle values as context, with $s = 12$ and $f = \infty$.

Coding method	Comp. ratio	Bits/pixel	Inefficiency	Throughput
Golomb	2.18	3.67	–	104
Rice	2.17	3.68	0.1	109
Subexponential	2.16	3.70	0.6	108

In Table 4 we see that Golomb coding gives slightly better compression (less than $1\frac{0}{0}$) than either Rice coding or subexponential coding, and that it runs slightly slower (about $4\frac{0}{0}$). Subexponential coding is about $0.2\frac{0}{0}$ better than Golomb coding for the 'crowd' image, and a few bytes better for bands 1, 2, and 3 of the Donaldsonville data set. Overall, the best choice seems to be Golomb coding for maximum compression or Rice coding for maximum speed, but the differences are small.

Table 5: Selection of scaling factor s be be applied at the end of each level, using the two middle values as context, Golomb coding, and $f = \infty$.

Scaling factor	Comp. ratio	Bits/pixel	Inefficiency
1	2.1641	3.6967	0.603
2	2.1743	3.6793	0.133
4	2.1765	3.6756	0.033
6	2.1770	3.6748	0.010
8	2.1771	3.6746	0.005
12	2.1772	3.6744	–
16	2.1772	3.6744	–
24	2.1771	3.6746	0.003

The differences are even smaller when choosing among possible values of the end-of-level scaling factor s, as seen in Table 5. Almost any value greater than 1 works fine; for general use we recommend using $s = 12$.

Table 6: Selection of freezing level f, using the two middle values as context, Golomb coding, and $s = 12$.

Freezing level	Comp. ratio	Bits/pixel	Inefficiency	Throughput
128	2.16	3.70	0.8	134
256	2.16	3.70	0.6	132
512	2.17	3.69	0.4	130
1024	2.17	3.69	0.3	126
2048	2.17	3.68	0.1	122
4096	2.18	3.68	0.1	118
∞	2.18	3.67	–	105

Finally we select the freezing level f. This is a pure tradeoff: a lower freezing level gives more speed but less compression by halting the adaptation of the coding parameter statistics. Table 6 shows that by choosing $f = 1024$ we can obtain about 20 percent more throughput at a cost of only $0.3\frac{0}{0}$ compression inefficiency.

5.2. Comparative results

In Tables 7 and 8 we compare Progressive FELICS with non-progressive FELICS, the lossless mode of the JPEG standard (using two-point prediction), UNIX *compress*, and the MLP method. The

Table 7: Summary of compression results for different image coding methods. High compression mode for the FELICS methods is based on Golomb coding and omission of the freezing heuristic. Fast mode uses Rice coding and freezing with $f = 1024$.

Coding method	Comp. ratio	Bits/pixel	Inefficiency	Throughput
Progressive FELICS				
High compression mode	2.18	3.67	5.7	106
Fast mode	2.17	3.69	6.1	132
Non-progressive FELICS				
High compression mode	2.16	3.71	6.7	192
Fast mode	2.14	3.74	7.4	314
JPEG lossless mode	2.15	3.72	7.1	61
UNIX *compress*	1.63	4.92	34.9	286
MLP	2.31	3.47	–	7

FELICS versions in Table 8 use Golomb coding and no freezing. Progressive FELICS compresses better than both non-progressive FELICS and JPEG lossless mode for most images. MLP consistently gives the best compression ratios, but it runs very slowly. Progressive FELICS is slower than non-progressive FELICS because of its more complicated pixel sequence and context selection, but it is still about twice as fast as JPEG lossless mode.

6. CONCLUSION

Progressive FELICS is a lossless compression method that combines the fast image and error modeling and coding of our earlier FELICS method with the progressive pixel sequence of our earlier MLP method. Progressivity has obvious browsing applications; it also leads to excellent compression. In fact, Progressive FELICS gives about one percent better compression than non-progressive FELICS. We have given details of pixel sequence, image and error modeling, and coding using simple prefix codes. In addition, we have introduced a new prefix code with some advantages over the previously used Golomb and Rice codes.

7. ACKNOWLEDGMENTS

Support was provided in part by Air Force Office of Strategic Research grant F49620 92 J 0515, by Army Research Office grant DAAH04 93 G 0076, and by associate memberships in CESDIS. Some of this research was performed while the first author was at Duke University.

The authors acknowledge Allan R. Moser of E. I. duPont de Nemours and Company (Inc.) for assisting us with experimental evaluation of JPEG lossless mode compression.

Table 8: File-by-file comparison of progressive FELICS (high compression mode) with other lossless image compression methods in the literature. The figures are compression ratios.

	Progressive FELICS	Non-progressive FELICS	JPEG lossless	*compress*	MLP
W1	2.13	2.12	2.07	1.70	2.21
W2	2.68	2.71	2.67	2.21	2.83
W3	2.32	2.33	2.28	1.92	2.44
W4	1.85	1.85	1.81	1.46	1.93
W5	1.73	1.72	1.68	1.34	1.79
W6	5.24	5.01	7.92	5.36	9.76
W7	2.15	2.14	2.10	1.70	2.24
D1	2.33	2.32	2.26	1.79	2.41
D2	3.01	3.01	3.07	2.36	3.24
D3	2.58	2.54	2.58	1.99	2.72
D4	1.90	1.87	1.85	1.34	1.98
D5	1.87	1.85	1.82	1.34	1.94
D6	5.59	5.39	9.25	6.14	11.26
D7	2.22	2.19	2.17	1.65	2.31
R1	2.40	2.37	2.28	1.79	2.46
R2	2.99	2.97	2.94	2.26	3.18
R3	2.53	2.51	2.45	1.86	2.66
R4	2.31	2.30	2.24	1.76	2.42
R5	1.87	1.85	1.78	1.34	1.94
R6	2.18	2.14	2.08	1.58	2.25
R7	5.17	5.09	7.43	5.50	8.78
couple	1.61	1.63	1.54	1.17	1.64
crowd	1.90	1.83	1.87	1.31	2.02
lax	1.35	1.36	1.31	1.04	1.38
lena	1.83	1.77	1.72	1.14	1.89
man	1.71	1.69	1.64	1.15	1.75
woman1	1.66	1.64	1.58	1.30	1.66
woman2	2.33	2.25	2.28	1.40	2.50

8. REFERENCES

1. P. G. Howard and J. S. Vitter, "Fast and Efficient Lossless Image Compression," in *Proc. Data Compression Conference*, J. A. Storer and M. Cohn, eds., pp. 351 360, Snowbird, Utah, Mar. 30-Apr. 1, 1993.
2. P. G. Howard and J. S. Vitter, "New Methods for Lossless Image Compression Using Arithmetic Coding," *Information Processing and Management*, 28:6, pp. 765 779, 1992.
3. P. G. Howard and J. S. Vitter, "Error Modeling for Hierarchical Lossless Image Compression," in *Proc. Data Compression Conference*, J. A. Storer and M. Cohn, eds., pp. 269 278, Snowbird, Utah, Mar. 24-26, 1992.
4. K. Knowlton, "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," *Proc. of the IEEE*, 68:7, pp. 885 896, July 1980.
5. N. Garcia, C. Munoz and A. Sanz, "Image Compression Based on Hierarchical Coding," *SPIE Image Coding*, 594, pp. 150 157, 1985.
6. H. H. Torbey and H. E. Meadows, "System for Lossless Digital Image Compression," *Proc. of SPIE Visual Communication and Image Processing IV*, 1199, pp. 989 1002, Nov. 8-10, 1989.
7. T. Endoh and Y. Yamakazi, "Progressive Coding Scheme for Multilevel Images," *Picture Coding Symp.*, Tokyo 1986.
8. P. Roos, M. A. Viergever, M. C. A. van Dijke and J. H. Peters, "Reversible Intraframe Compression of Medical Images," *IEEE Trans. Medical Imaging*, 7:4, pp. 328 336, Dec. 1988.
9. R. F. Rice, "Some Practical Universal Noiseless Coding Techniques," Jet Propulsion Laboratory, JPL Publication 79 22, Pasadena, California, Mar. 1979.
10. S. W. Golomb, "Run-Length Encodings," *IEEE Trans. Inform. Theory*, IT 12:4, pp. 399 401, July 1966.
11. P. Elias, "Universal Codeword Sets and Representations of Integers," *IEEE Trans. Inform. Theory*, IT 21:2, pp. 194 203, Mar. 1975.