

AD-A277 905



NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

DTIC
ELECTE
APR 11 1994
S G D

A SIMPLE, LOW OVERHEAD DATA
COMPRESSION ALGORITHM FOR
CONVERTING LOSSY COMPRESSION
PROCESSES TO LOSSLESS

by

Walter D. Abbott, III

December, 1993

Thesis Advisor:

Ron J. Pieper

Approved for public release; distribution is unlimited.

94-10799



9390

DTIC QUALITY INSPECTED S

94 4 8 030

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 1993.		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE A SIMPLE, LOW OVERHEAD DATA COMPRESSION ALGORITHM FOR CONVERTING LOSSY COMPRESSION PROCESSES TO LOSSLESS.			5. FUNDING NUMBERS	
6. AUTHOR(S) Abbott, Walter D., III				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) In this thesis, a hybrid lossless compression model is tested which employs a combination of both a lossy compression method and one or more lossless image compression methods to produce an overall lossless image compression. The hybrid model decomposes the original image into a browse and a residual image. The hybrid model is tested and evaluated using various combinations of lossy and lossless image compression methods. The lossy compression method used in the model is JPEG (Joint Photographic Experts Group). The lossless compression methods used are Huffman, Arithmetic, LZW, lossless JPEG, and Diagonal coding. The compression results achieved using the hybrid compression model are compared to the compression achieved using the corresponding direct lossless compression. Additionally, the hybrid model is evaluated as to the advantages that the decomposition of the image into browse and residual images provide to the user.				
14. SUBJECT TERMS Image Compression, Data Compression, Hybrid Compression, Lossless, Lossy, Browse, Residual.			15. NUMBER OF PAGES 94	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. Z39-18

DTIC QUALITY IS ASSURED

Approved for public release; distribution is unlimited.

A Simple, Low Overhead Data Compression Algorithm
for Converting Lossy Processes to Lossless

by

Walter D. Abbott, III
Lieutenant, United States Navy
B.S., The Citadel, 1985

Submitted in partial fulfillment
of the requirements for the degree of

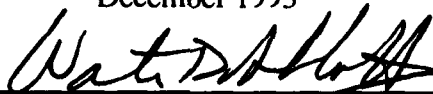
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

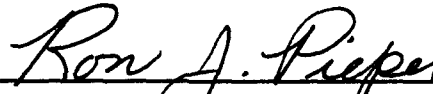
December 1993

Author:

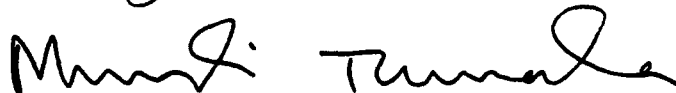


Walter D. Abbott, III

Approved by:



Ron J. Pieper, Thesis Advisor



Murali Tummala, Second Reader



Michael A. Morgan, Chairman
Department of Electrical & Computer Engineering

ABSTRACT

In this thesis, a hybrid lossless compression model is tested which employs a combination of both a lossy compression method and one or more lossless image compression methods to produce an overall lossless image compression. The hybrid model decomposes the original image into a browse and a residual image. The hybrid model is tested and evaluated using various combinations of lossy and lossless image compression methods. The lossy compression method used in the model is JPEG (Joint Photographic Experts Group). The lossless compression methods used are Huffman, Arithmetic, LZW, lossless JPEG, and Diagonal coding. The compression results achieved using the hybrid compression model are compared to the compression achieved using the corresponding direct lossless compression. Additionally, the hybrid model is evaluated as to the advantages that the decomposition of the image into browse and residual images provide to the user.

Accession For	
NTIS	CRA&I <input checked="" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	REVIEW OF LITERATURE.....	1
B.	OVERVIEW OF THE THESIS.....	2
II.	HYBRID LOSSLESS COMPRESSION MODEL.....	4
III.	COMPRESSION TECHNIQUES.....	8
A.	LOSSLESS AND LOSSY TECHNIQUES.....	8
B.	HUFFMAN CODING.....	8
C.	ARITHMETIC CODING.....	11
D.	LIMPEL-ZIV (LZ) COMPRESSION.....	14
E.	RUN LENGTH ENCODING.....	16
F.	BIT PLANE ENCODING.....	17
G.	PREDICTIVE ENCODING.....	18
H.	JPEG.....	19
IV.	COMPARISON OF COMPRESSION METHODS.....	27
A.	OVERVIEW.....	27
B.	TEST IMAGES.....	27
C.	LOSSY JPEG.....	30
D.	SECONDARY COMPRESSION.....	32
E.	COMPARISON OF LOSSLESS COMPRESSION METHODS.....	34
F.	CONCLUSIONS.....	36

V.	DIAGONAL CODING.....	42
A.	INTRODUCTION.....	42
B.	RESIDUAL IMAGE HISTOGRAM.....	42
C.	DIAGONAL CODING.....	44
D.	CONCLUSIONS.....	49
VI.	HYBRID MODEL OPTIMIZATION.....	54
VII.	CONCLUSIONS.....	60
	APPENDIX A.....	63
	APPENDIX B.....	71
	LIST OF REFERENCES.....	86
	INITIAL DISTRIBUTION LIST.....	87

I. INTRODUCTION

A. REVIEW OF LITERATURE

Memory requirements to store, transmit, and display images have rapidly grown as the need for higher resolution images has increased. As a result of this explosion of data associated with images, various image compression algorithms have been developed. These compression algorithms capitalize on the redundancies inherent in images to reduce the number of bits required to represent them. This results in savings in the memory needed for image storage or in the channel capacity required for image transmission [1], [2], [3].

Image compression can be divided into two groups: lossy and lossless. Images may be compressed using a lossy or lossless compression method depending on the amount of compression and image resolution desired by the user. Lossy compression methods achieve high compression but produce an image which is of lower resolution than the original image. Lossless compression methods achieve low compression but produce an exact replica of the original image.

Some of the standard lossless compression methods are Huffman [1], [2], Arithmetic [1], [2], the Ziv and Lempel algorithms [2], [4], Predictive encoding [1], [2], Bit-plane encoding [1], [5], and Run-length encoding [6]. Each of these compression methods have many variations which are reported in the literature. A non-standard lossless compression method is Diagonal coding [7]. Lossy compression methods consist

primarily of the Joint Photographic Experts Group (JPEG) algorithm [1], [2], [8], [9] and Fractal encoding [4], [10].

Comparisons of the performance of lossy and lossless compression methods reveal that some compression methods achieve better performance results in terms of compression ratios and root mean square error than others [11].

Lossy and lossless methods may be combined together to produce a lossless compressed image. Such an arrangement takes advantage of the high compression ratios achieved by the lossy methods and the error-free compression of the lossless methods [1], [10].

B. OVERVIEW OF THE THESIS

The current chapter introduces the literature used in the thesis and discusses the structure of the thesis. The various methods of image compression are presented.

Chapter II describes the goal of the thesis and discusses the proposed hybrid lossless compression model. The decomposition of an image into browse and residual images is introduced. The evaluation criteria used to evaluate the hybrid model and the lossless and lossy compression methods is defined.

Chapter III describes the lossy and lossless compression methods used to evaluate the hybrid lossless compression model. The lossy algorithm used in the evaluation is the lossy JPEG. The lossless algorithms used in the evaluation are Huffman, Arithmetic, LZW, and lossless JPEG.

Chapter IV discusses the results of using secondary compression to compress the lossy compressed image in the hybrid model. A comparative analysis of the compression achieved using direct lossless compression and the compression achieved using the hybrid lossless compression model is performed for each of the lossless compression methods.

Chapter V introduces another lossless compression method called Diagonal coding. It is compared to the other lossless compression methods evaluated in the hybrid model.

The optimization of the model is discussed in Chapter VI with emphasis on the combination of lossy and lossless compression methods that result in high overall compression and a visually acceptable browse image.

Chapter VII contains the general conclusions reached from the comparative analysis of Chapters IV, V, and VI.

Appendix A contains the tabulated numeric data gathered during the research of the thesis. The source code for the Diagonal coding compression algorithm (encoding and decoding) is contained in Appendix B.

II. HYBRID LOSSLESS COMPRESSION MODEL

In many practical situations involving images, a small degree of error in the pixel values can be tolerated without a significant effect on the display. This suggests that there are advantages to a decomposition of images into a lossy component and an error component. A hybrid compression model which employs the browse and residual concept has recently appeared in the literature [7].

A hybrid image compression model was tested which utilizes both lossy and lossless image compression techniques to produce an overall lossless image compression. The model is tested and evaluated using the JPEG algorithm, the industry standard for lossy image compression, and various popular lossless compression techniques. The total compression achieved using the model is compared to the compression achieved using standard lossless image compression techniques. Figure II.1 displays a block diagram of the hybrid lossless compression model.

The model was evaluated using 8-bit (256 levels), 256x256 (65536 bytes) pixel grey-scale images in raw pixel grey map (rpgm) format. Three different images were used in the tests. The images were all structurally different from each other in order to test the model over a broad range of images.

An image is first compressed using a lossy compression process. The lossy compression technique used in the testing of the model is the Joint Photographic Expert Group (JPEG) algorithm. After compression using JPEG, the compressed image is further compressed (secondary compression) using a lossless compression method.

Several different lossless compression algorithms are tested and evaluated. The lossy compressed image is decompressed and compared on a pixel by pixel basis to the original image. The decompressed image is termed the browse image data as it can be used for browsing an image and preliminary analysis of the image. Browsing enables a user to

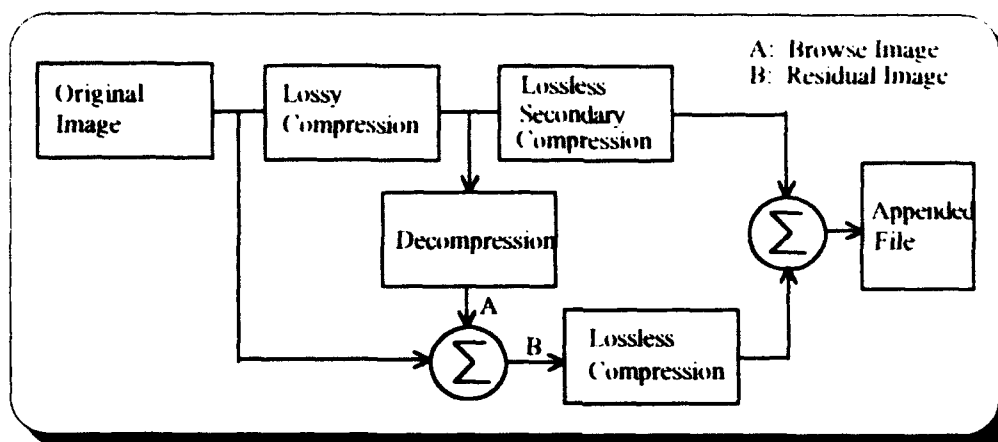


Figure II.1: Hybrid Lossless Compression Model

determine whether a lossless representation of the original image is required or if the lossy browse image data is adequate for their needs. The difference between the original image and the decompressed image is the error image or residual image. The residual image is compressed with a lossless compression routine. Once again, several different lossless compression algorithms are tested and evaluated. Lastly, the compressed browse image file and the compressed residual image file are appended together into a single file. Decoding consists of separating the appended file into the respective compressed browse and residual image files and applying the appropriate decompression algorithm to

each. Figure II.2 displays a block diagram of the decoding process. Both the browse and residual image files are first decompressed using the same lossless compression routines which were used to compress them. The resulting browse image file is then decompressed using the lossy JPEG algorithm. The residual image file is added on a pixel by pixel basis to the decompressed browse image file to obtain the original image.

The hybrid lossless compression model combines the inherent advantages of both lossy and lossless compression algorithms to achieve the lossless result. The high compression achieved by the lossy JPEG algorithm combined with the error-free lossless algorithms results in a significantly compressed image, which upon decoding, is an exact replica of the original image.

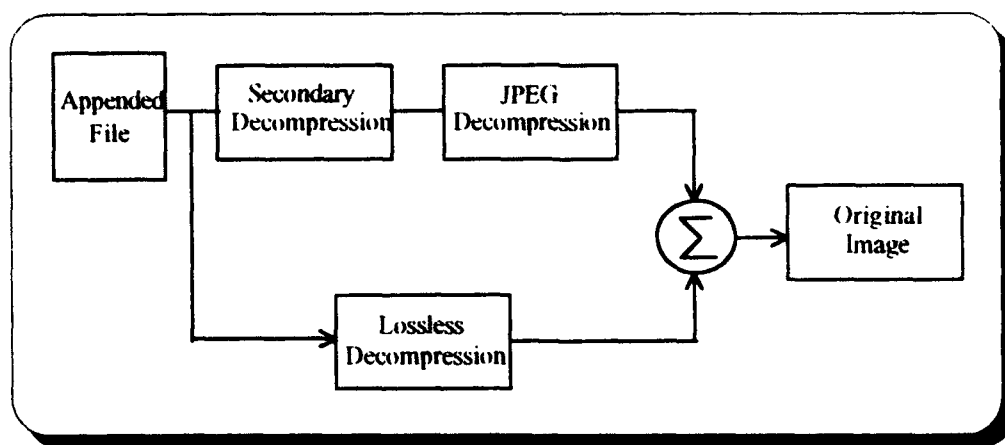


Figure II.2: Diagram of Decoding Process

Various combinations of the lossy JPEG and lossless algorithms were evaluated in the model and compared. The evaluation criteria used in the comparisons was the total

compression ratio (CR) achieved using a particular lossless compression algorithm in combination with the JPEG lossy algorithm. Compression ratio is the percent compression achieved as a result of compressing a file [2, p. 10]:

$$CR = (1 - (\text{Compressed Image Size} / \text{Original Image Size})) \times 100. \quad (\text{II.1})$$

A file whose file size does not change when compressed will have a compression ratio of 0 percent. A file which is compressed to one-third of its original size will have a compression ratio of 67 percent. Therefore, perfect compression occurs at 100 percent. A file whose compressed file size is greater than its original file size will have a negative compression ratio. The overall compression ratio achieved by the hybrid lossless compression model is a combination of the compressed browse image CR and the compressed residual image CR. Application of Equation II.1 to browse, residual, and overall compression ratios leads to:

$$CR_{\text{overall}} = [CR_{\text{browse}} - 50] + [CR_{\text{residual}} - 50] \quad (\text{II.2})$$

where CR_{browse} and CR_{residual} are the compression ratios of the compressed browse and residual images.

The overall compression ratios achieved using the model with different combinations of lossless techniques and JPEG are compared with each other and with the compression ratios achieved using standard lossless compression techniques. Additionally, the benefits of breaking up the image into browse and residual images are evaluated and compared to the standard lossless compression methods.

III. COMPRESSION TECHNIQUES

A. LOSSLESS AND LOSSY TECHNIQUES

Compression techniques can be divided into lossless and lossy methods. A lossless method always produces a decompressed image that is identical, pixel-for-pixel, to the original image. On the other hand, lossy methods produce a decompressed image that is not identical to the original image. The degree of difference between the lossy decompressed image and the original image depends upon the compression ratio desired. The higher the compression ratio, the greater the difference between the decompressed image and the original image. Lossless compression methods typically attain small compression ratios of about 50% or less while lossy methods can achieve much higher compression ratios.

B. HUFFMAN CODING

Huffman coding is a lossless compression method that assigns variable-length codes to symbols based on the probability of each symbol's occurrence in a file. It is based on the premise that if the probability of symbols in a file are known, and the probability distribution is a non-uniform distribution, variable-length codes can be assigned to each symbol which will result in compression of the file. When using this type of coding, a symbol that has a very high probability of occurrence generates a code with very few bits. A symbol with a low probability generates a code with a larger number of bits. Generating codes that vary in length according to the probability of the symbol they are encoding makes data compression possible. Each variable-length code

can be uniquely decoded. Huffman coding achieves the minimum amount of redundancy possible in a fixed set of variable-length codes; however, this doesn't mean that Huffman coding is an optimal coding method. It means that it provides the best approximation for coding symbols when using fixed-length codes [2, p. 18].

A binary tree is constructed from the individual symbols in a file. Each symbol is a child node in the tree. A weight is assigned to each child node. The assigned weight is either the frequency or the probability of the symbol occurring in the file. Therefore, symbols with a low probability of occurrence have lower weights assigned. The binary tree is built by combining the two lowest weight child nodes, creating a parent node, and assigning a weight to the parent node. The parent node's assigned weight is the sum of the two child node weights. A bit value of 1 is assigned to the path taken from the parent node to the child node with the lowest weight. The path from the parent node to the other child node is assigned a bit value of 0. The process is repeated until only one node is left. This node is designated the root of the binary tree. The variable-length codes are generated by traversing the binary tree from the child node which represents the symbol of interest to the root. The bits in the generated code are arranged in the order from root to child node. Table III.1 contains a list of five different symbols and their frequency of occurrence in a file. The table also contains the unique variable-length Huffman codes assigned to each symbol. Figure III.1 displays the Huffman binary tree for the file in Table III.1. Huffman codes have the unique prefix attribute, meaning that no code is a prefix to another code. As a result, the codes can be correctly decoded despite being variable length. Using Figure III.1, the Huffman code for 'MAZES' would be

1101001110101 or 13 bits long. If each letter in 'MAZES' requires eight bits to represent it, then a total of 40 bits would be required. In this case, Huffman coding produces a compression ratio of 67.5%.

SYMBOL	FREQUENCY	CODE
A	8	100
E	15	0
M	4	110
S	5	101
Z	1	111

Table III.1: Huffman Coding Example

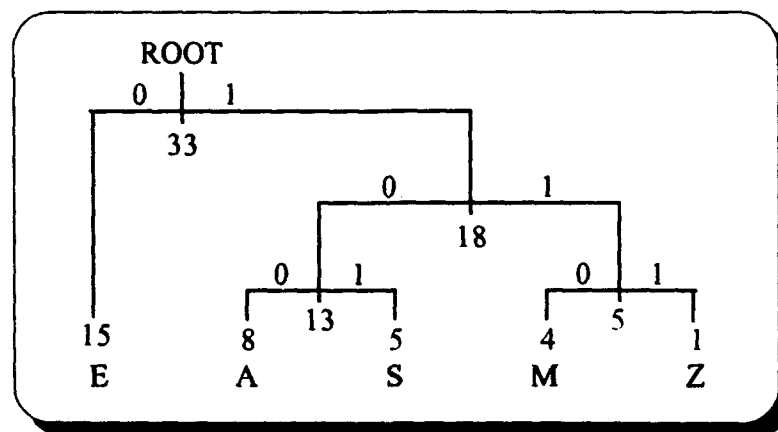


Figure III.1: The Huffman Binary Tree.

Huffman coding uses an integral number of bits for each code, which is usually slightly less than optimal. Additionally, the compression program has to pass a complete copy of the Huffman coding statistics with the compressed data. This effectively reduces

the amount of compression achieved. Huffman coding is not an optimal coding method, but it is the best approximation that uses fixed codes with an integral number of bits.

C. ARITHMETIC CODING

Arithmetic coding is a lossless compression method that produces a single output code for an entire message. Unlike Huffman coding, it does not produce a single code for each symbol. Instead, arithmetic coding encodes a stream of input symbols with a single floating-point output number in the range from 0 to 1. Each symbol added to the message incrementally modifies the output code. As in Huffman coding, each symbol's probability of occurrence in the file is first determined. Next, each symbol is assigned a range, corresponding to its probability of occurrence, in the interval from 0 to 1. Table III.2 contains a file with five different symbols, their probability of occurrence, and the range they occupy in the 0 to 1 interval. If the first symbol in the file is 'M', then the encoded floating-point output number will be a number between 0.60 and 0.70. Each new symbol to be encoded further restricts the range of the output number. If the next symbol to be encoded is 'A', then the encoded output number will be a number between 0.60 and 0.62 since 'A' is assigned the range 0.00 to 0.20 in the 0.60 to 0.70 subrange established by the symbol 'M'. The higher the probability of a symbol, the less it will reduce the range and, therefore, add fewer bits to the code. The net effect of each input symbol on the output code can be a fractional number of bits instead of an integral number since Arithmetic coding uses a fractional number of bits per code allowing it to incrementally improve compression performance. Table III.3 contains an example of the

Arithmetic encoding process resulting in the final low value, 0.61896, which will uniquely encode the message 'MAZES'. The symbol probabilities are taken from Table III.2. The number of bits required to represent the number 0.61896 can be determined from:

$$0.61896 = \sum_{i=1}^x A_i \frac{1}{2^i} \quad (\text{III.1})$$

where A_i is the i th bit of the binary representation of 0.61896, i is the index of the n th bit, and x is the minimum number of iterations necessary to represent the number in binary. Selecting x to be a value of 20 ensures that the left side of Equation III.1 will have sufficient resolution in order to represent 0.61896. Therefore, 0.61896 can be represented in as few as 20 bits compared to the 40 bits required to represent the message 'MAZES' using eight bits per character. This results in a CR of 50%. A simple algorithm can be applied to Equation III.1 in order to produce the sequence A_1, A_2, \dots, A_{20} . Simply multiply the left side of Equation III.1 by 2 repeatedly until an integer is produced as a leading digit. Then subtract one and continue. For each 1 produced record a one, otherwise record a zero [12, p. 9].

SYMBOL	PROBABILITY	RANGE
A	2/10	$0.00 \leq R < 0.20$
E	4/10	$0.20 \leq R < 0.60$
M	1/10	$0.60 \leq R < 0.70$
S	2/10	$0.70 \leq R < 0.90$
Z	1/10	$0.90 \leq R < 1.00$

Table III.2: Arithmetic Coding Symbol Range

Symbol	Low Value	High Value
	0.0	1.0
M	0.6	0.7
A	0.60	0.62
Z	0.618	0.620
E	0.6184	0.6192
S	0.61896	0.61912

Table III.3: Arithmetic Encoding Example.

Decoding consists of determining which symbol falls within the range of the encoded message. In the example in Table III.3, the encoded message falls in the interval between 0.6 and 0.7. Therefore, the first character in the message must be 'M'. The next character is decoded by subtracting the low value of 'M' from the encoded value, dividing by the width of the range of 'M' (0.1), and determining which character falls within the new interval. Table III.4 contains an example of the decoding process.

Encoded Number	Output Symbol	Low	High	Range
0.61896	M	0.6	0.7	0.1
0.1896	A	0.0	0.2	0.2
0.948	Z	0.9	1.0	0.1
0.48	E	0.2	0.6	0.4
0.7	S	0.7	0.9	0.2
0.0				

Table III.4: Arithmetic Decoding Example.

D. LIMPEL-ZIV (LZ) COMPRESSION

LZ compression is a lossless compression method based on the work of Jacob Ziv and Abraham Lempel in 1977-1978 [2, p. 23]. It is a dictionary-based method using an adaptive dictionary to achieve compression. LZ compression is based on strings of symbols instead of individual symbols thereby exploiting the interdependency between symbols in a string. A table of strings is created from the input data and placed into a string dictionary. As each new string is input from the input data, the string dictionary is searched for a string match. If a match is found, a code is output which represents the string in the dictionary.

Ziv and Lempel's work resulted in two LZ compression methods, LZ77 and LZ78. LZ77 uses a sliding-window approach in constructing its dictionary. The dictionary consists of all the strings in a window of the input data stream. For example, if a 4K byte window is used as the dictionary, the LZ77 algorithm looks for matches with strings found in the previous 4K bytes of data already read in. As new symbols of the input data stream are read in, the 4K byte window slides so that the last 4K bytes of input data is in the window, hence the term sliding-window. All string matches are encoded as pointers to the string in the dictionary. The amount of compression depends on how long the dictionary strings are and how large the sliding-window is. Figure III.2 shows a simple flowchart of the LZ77 compression process.

LZ78 differs from LZ77 in the way that it builds and maintains its dictionary. Instead of having a limited-size window of the preceding input data, LZ78 builds its dictionary out of all of the previously input symbols in the input data stream. The

dictionary of strings is built a single symbol at a time. The first symbol input from the input data stream is stored in the dictionary and becomes the current prefix. Each subsequent symbol from the input is added to the current prefix before a search for a match is made in the string dictionary. If a string match is found, a pointer code is output which represents an offset into the string dictionary. If no match is found, the string is added to the dictionary. Once a string is added to the dictionary, it is available to the encoder at all times, not just for the next few thousand characters as in LZ77. This incremental procedure works very well at isolating frequently used strings and adding them to the dictionary. Consequently, strings in LZ78 can be very long, resulting in high compression ratios.

Another variation of LZ compression is the LZW compression method, developed by Terry Welch in 1984 [2, p. 285]. LZW is an extension of LZ78. LZW differs from LZ78 in the way that it initially builds the dictionary. The dictionary is initialized with single-symbol strings equal to the number of ASCII characters. In other words, the first 256 entries in the dictionary are initialized with the byte values 0 to 255. Thus, there is no symbol that cannot be immediately encoded even if it has not already appeared in the input data stream. LZW uses a current prefix buffer and a current string buffer like the LZ78 algorithm. The current string is defined as the current prefix plus the next symbol input from the input data. A match is found for the first symbol. A code is output, and the new string is added to the string table. The current string is added to the current prefix. This process continues until the input data stream ends.

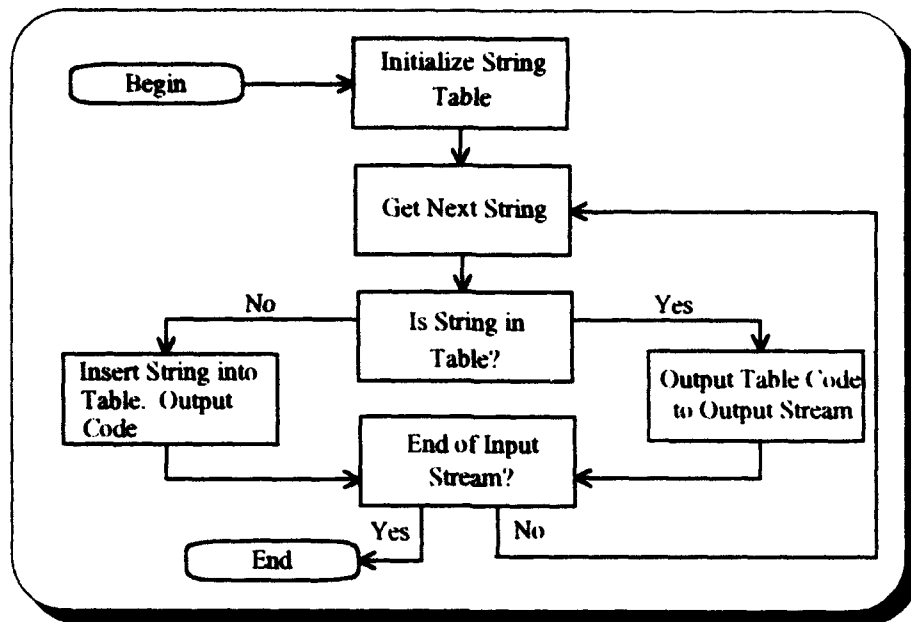


Figure III.2: Flowchart of LZ77 Compression Process.

E. RUN LENGTH ENCODING

Run length encoding (RLE) is arguably the least complex and easiest to implement lossless compression method. RLE capitalizes on the successive repetition of characters in a binary bit stream or image. It is effective only in applications involving many repeated characters. Instead of repeating each character, run length encoding uses a code which specifies how many consecutive characters are in the particular run. In the case of images, many consecutive grey-scale pixels having the same value are an example in which run length encoding would produce some degree of compression. Run length encoding may actually expand a file if the average length of consecutive characters is less than the code used to specify them. Run length encoding can be

performed at the byte or the bit level depending on the application. It is used most often as a preprocessor for other compression algorithms [6, p. 37].

F. BIT PLANE ENCODING

Bit plane encoding is the process of grouping single bits from the same position in a binary representation together to form a binary array. For example, an image containing $N \times N$ pixels, each pixel represented by k bits, can be broken up into k different $N \times N$ bit planes. The most significant bit (MSB) of each pixel binary representation is grouped together with the MSB of the remaining pixels to form a bit plane. Repeating this process for the other $k-1$ bits in each pixel results in k bit planes. Hence, the original image is now represented by k , $N \times N$ bit planes. The advantages of bit plane encoding are twofold. First, each individual bit plane can be encoded efficiently using a lossless compression routine. Secondly, bit plane encoding permits a technique called progressive transmission to be implemented. In progressive transmission, bit planes are transmitted in a sequence starting with the MSB bit plane and ending with the LSB bit plane. The transmitted bit planes are progressively reconstructed at the terminal end. The user may view an image as it is being reconstructed and elect to terminate the transmission or proceed depending on the level of quality desired [1, p. 194].

The most significant bit planes tend to contain a lot of redundancy and are highly compressible. The least significant bit planes contain less redundancy and exhibit the

behavior of random noise. As a result, the least significant bit planes are less compressible than the more significant bit planes [1, p. 54].

Bit planes may be combined together into subsets [5, p. 35]. Each subset may then be compressed with a lossless compression method. Grouping the bit planes into subsets may achieve higher compression ratios than performing lossless compression on each individual bit plane. The distributions of each bit plane or subset with respect to bit values of one and zero determines which optimum lossless compression technique to utilize for compression.

G. PREDICTIVE ENCODING

Predictive encoding may be either a lossless or a lossy compression method. The lossless predictive encoding method is discussed here. Images are typically highly correlated from pixel to pixel, especially between adjacent pixels. This correlation between pixel values can be exploited to achieve compression of the image by using predictive encoding. Predictive encoding predicts the value of a given pixel based on the values of the pixels surrounding it. Numerous combinations of pixels exist. After predicting the value of the pixel, the predicted pixel value is subtracted from the actual pixel value to form an error value. This process is continued for all of the pixels in the image. The resulting error image will have a significantly different distribution or histogram than the original image. If the predictor accurately predicts the pixel values, the error will be small and the error image histogram will be narrow and Laplacian in nature [1, p. 62]. The error is encoded using a lossless compression method such as

Huffman or Arithmetic. The better the predictor is at predicting the pixel values, the smaller the resulting error. The smaller error can be encoded more efficiently, resulting in better overall compression of the image. The order of the predictor is determined by the number of surrounding pixel values used to make the prediction. Generally, a higher order predictor will outperform a lower order one [1, pp. 58-60].

H. JPEG

JPEG is a compression standard created by the Joint Photographic Experts Group (JPEG). The JPEG compression standard has not yet been finalized but is currently in the final stages of the standardizations process. The JPEG standard includes a specification for both lossy and lossless compression of images. The Discrete Cosine Transform (DCT) algorithm with quantization is used for lossy compression and a predictive method is used for lossless compression. The JPEG encoder consists of three stages: a transformation stage, a lossy quantization stage, and a lossless coding stage. The advantages of the DCT over the Discrete Fourier Transform (DFT) lie in the differences in their periodicities [1, pp. 108-111]. The DCT transformation stage converts the image to the frequency domain and concentrates the information energy into the first few transform coefficients, the quantization stage causes a controlled loss of information, and the lossless stage further compresses the image data. Figure III.3 displays a block diagram of the JPEG encoder. The DCT equation for an $N \times N$ pixel block is:

$$X(i,j) = \frac{1}{\sqrt{2N}} C(i) C(j) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m,n) \cos \left[\frac{(2m+1)i\pi}{2N} \right] \cos \left[\frac{(2n+1)j\pi}{2N} \right] \quad (\text{III.2})$$

where: $C(i), C(j) = \frac{1}{\sqrt{2}}$ for $i, j = 0$, else $C(i), C(j) = 1$.

Decoding consists of reversing the process and using the Inverse Discrete Cosine Transform (IDCT) in place of the DCT. The IDCT equation is:

$$x(m,n) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)X(i,j) \cos\left[\frac{(2m+1)i\pi}{2N}\right] \cos\left[\frac{(2n+1)j\pi}{2N}\right] \quad (III.3)$$

The image is a three dimensional signal (graphical image) where the x and y axes are the two dimensions of the screen, and the z axis is the amplitude or value of a pixel. This is the spatial representation of a signal. The two dimensional DCT is obtained by performing a one dimensional DCT on the columns followed by a one dimensional DCT on the rows [2, pp. 356-357].

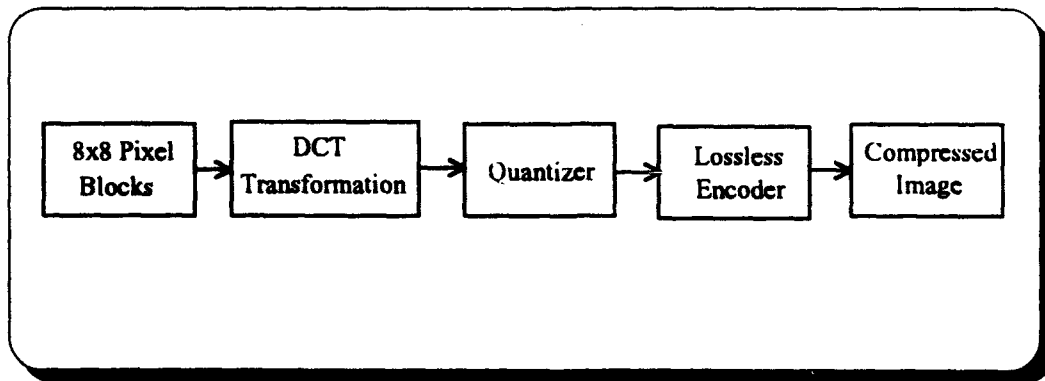


Figure III.3: Block Diagram of the JPEG Encoder.

The original image is first partitioned into 8x8 pixel blocks. Each block is independently transformed using the DCT. Each 8x8 pixel block has video energy distributed amongst its pixel elements. This video energy may be of low spatial frequency (slowly varying) or of high spatial frequency (quickly varying) [9, p. 5]. The DCT converts the spatial information into frequency or spectral information, with the x and y axes representing frequencies of the signal in two different dimensions. The transformed output of the 2-D DCT is an 8x8 array of 63 AC coefficients and 1 DC coefficient. The DC coefficient is the mean value of the array and is located in the upper left corner. The AC coefficients are ordered such that the lower frequency coefficients are located near the DC coefficient with the higher frequency coefficients located away from the DC coefficient. The DC coefficient always has the highest value of all the coefficients. Most images are composed of low frequency information. This suggests that the DC and lower frequency coefficients carry more useful information about the image than the higher frequency coefficients. As a result, the ordering of the coefficients in the array is significant. As we move farther away from the DC coefficient in the array, we find that the coefficients have lower values and become far less important for describing the image [2, p. 359]. An example of the effects of DCT processing on an 8x8 pixel block is shown in Figure III.4.

The quantization stage of the JPEG encoder quantizes the coefficients of the DCT transform array to reduce their magnitude and to increase the number of zero value coefficients. Quantization is the lossy stage in the JPEG encoder. The degree of quantization is controlled by a variable called the quality factor. The quality factor is a

number which changes the default quantization matrix by an effective multiplicative factor of the quality factor. Each of the DCT coefficients is divided by the corresponding quantizing value in the quantization matrix and rounded to the nearest integer. The greater the number of high frequency (lower information content) AC coefficients converted into zeros, the greater the compression achieved by the subsequent lossless encoding stage. Consequently, a higher quality factor results in better compression while a lower quality factor results in a better quality image upon decompression. A sample quantization matrix is shown in Figure III.5. Figure III.6 displays a sample DCT transformed image before and after quantization.

Prior to the final lossless encoding stage, the quantized DCT coefficients are arranged in a zig-zag pattern (see Figure III.7) with the lowest frequencies first and the highest frequencies last. The numbers 1-64 in Figure III.7 represent the sequence that the pixels are placed in the output sequential bit stream. This type of pattern is used to increase the number of consecutive zero coefficients in the 8x8 block. This allows for further compression using a lossless method such as run length encoding, Huffman or Arithmetic [8, p. xxiii].

The lossless encoder encodes the 8x8 pixel block DC coefficients using Differential Pulse Code Modulation (DPCM). DPCM encodes the difference between the quantized DC coefficient of the current block and the quantized DC coefficient of the previous block. The AC coefficients are coded using a combination of run length encoding and Huffman.

Input Pixel Values:

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

Output Pixel Values:

186	-18	15	-9	23	-9	-14	19
21	-34	26	-9	-11	11	14	7
-10	-24	-2	6	-18	3	-20	-1
-8	-5	14	-15	-8	-3	-3	8
-3	10	8	1	-11	18	18	15
4	-2	-18	8	8	-4	1	-7
9	1	-3	4	-1	-7	-1	-2
0	-8	-2	2	1	4	-6	0

Figure III.4: Sample Image Data before and after DCT Processing [2, p. 363].

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

Figure III.5: Sample Quantization Matrix [2, p. 367]

The JPEG compression standard also contains a lossless compression specification based on predictive encoding and Huffman. This lossless mode of operation is wholly independent of the DCT processing previously discussed. The lossless JPEG predictive encoder has seven different predictors to choose from. The seven different predictor models combine the values of up to three neighboring pixels (A, B, and C) to predict the current pixel value (X) in Figure III.8. This prediction is then subtracted from the actual pixel value, and the difference is encoded losslessly using Huffman. Any one of the seven predictors ($K = 1 - 7$) listed in Table III.5 can be used. The $K = 1, 2$, and 3 predictors are one dimensional predictors while the $K = 4, 5, 6$, and 7 predictors are two dimensional predictors.

DCT Matrix before Quantization:

92	3	-9	-7	3	-1	0	2
-39	-58	12	17	-2	2	4	2
-84	62	1	-18	3	4	-5	5
-52	-36	-10	14	-10	4	-2	0
-86	-40	49	-7	17	-6	-2	5
-62	65	-12	-2	3	-8	-2	0
-17	14	-36	17	-11	3	3	-1
-54	32	-9	-9	22	0	1	3

DCT Matrix after Quantization:

90	0	-7	0	0	0	0	0
-35	-56	9	11	0	0	0	0
-84	54	0	-13	0	0	0	0
-45	-33	0	0	0	0	0	0
-77	-39	45	0	0	0	0	0
-52	60	0	0	0	0	0	0
-15	0	-19	0	0	0	0	0
-51	19	0	0	0	0	0	0

Figure III.6: DCT Transformed Image before and after Quantization [2, p. 368].

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Figure III.7: Zig-zag Pattern.

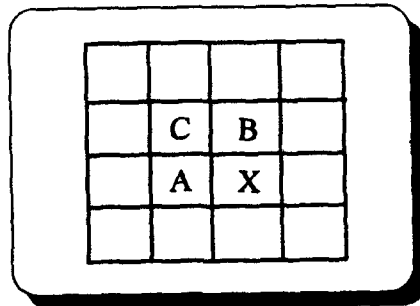


Figure III.8: Sample Prediction Pixel Neighbors.

Selection Value (K)	Predictor
1	A
2	B
3	C
4	$A + B - C$
5	$A + ((B - C) / 2)$
6	$B + ((A - C) / 2)$
7	$(A + B) / 2$

Table III.5: Lossless JPEG Predictors.

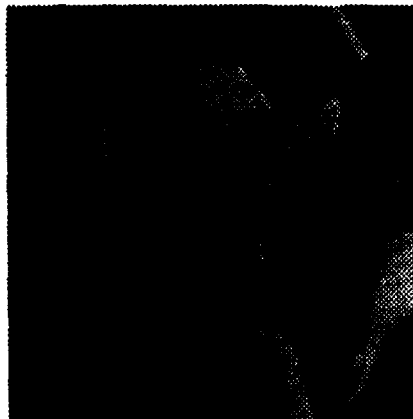
IV COMPARISON OF COMPRESSION METHODS

A. OVERVIEW

A comparison of the compression ratios achieved by direct compression of three test images using the standard lossless compression methods is performed. A comparison of compression ratios is also performed when the standard lossless compression methods are tested in the hybrid lossless compression model. Additionally, the hybrid model compression results are compared to the direct compressions achieved by the standard lossless methods.

B. TEST IMAGES

Three different 8-bit, 256x256 (65536 bytes) pixel grey-scale images in raw pixel grey map format were used to evaluate the hybrid lossless compression model. The three test images are displayed in Figure IV.1. The first image, LENA, is an image whose pixel values range over most of the 256 possible grey-scale levels. The image contains sharp contrasts and edges. The second image, SHUTTLE, has a range of pixel values that is less than that of LENA. A small range of pixel values dominate the image. The image contains large areas where the pixel values do not change significantly, such as the plume from the rocket motors and the sky background. The third image, FINGERPRINT, is dominated by a more narrow range of pixel values. The image contains large areas of whitespace. Pixel values that are contained in an image and their frequency of occurrence are plotted in a histogram. Histograms of each of the three test images is displayed in Figure IV.2. As expected, LENA contains a wide range of pixel



(a)

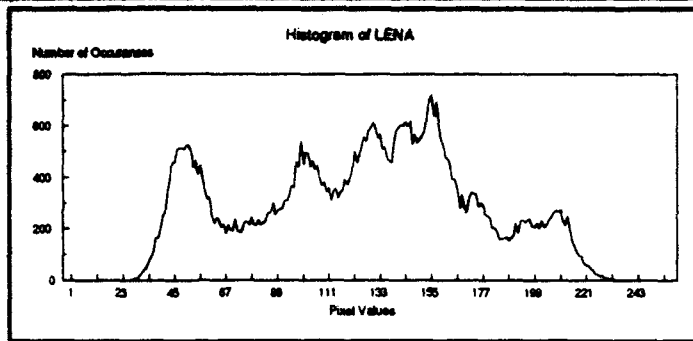


(b)

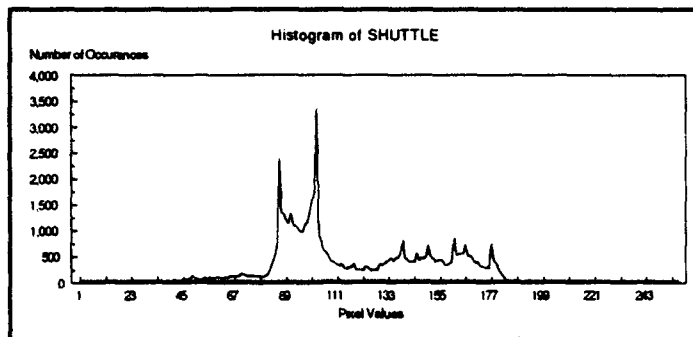


(c)

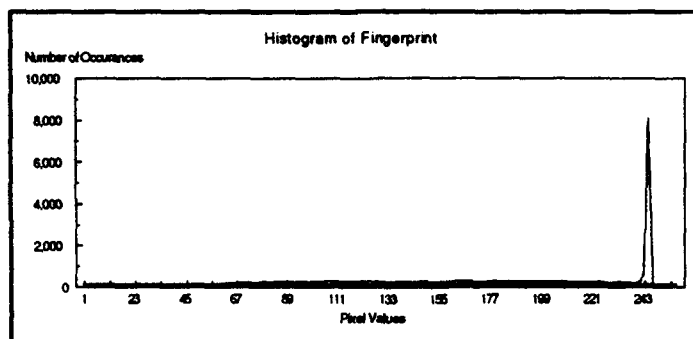
**Figure IV.1: Three Test Images (a) LENA, (b) SHUTTLE,
(c) FINGERPRINT.**



(a)



(b)



(c)

Figure IV.2: Histograms of the Three Test Images (a) LENA, (b) SHUTTLE, (c) FINGERPRINT.

values; SHUTTLE is dominated by a smaller range of pixel values; and FINGERPRINT contains a very narrow range of dominant pixel values.

C. LOSSY JPEG

The lossy JPEG algorithm used in the model was developed by Andy C. Hung at the Portable Video Research Group (PVRG), Stanford University [9]. The quality factor used when compressing an image determines the amount of compression achieved and the resolution of the image when it is decompressed. The higher the quality factor, the greater the compression and the less the resolution upon decompression. Figure IV.3 graphically displays the quality factor versus compression ratio achieved for the three test images. The graph data is tabulated in Table A.1 in Appendix A. The measure of the resolution of the decompressed image as compared to the original image is termed the root mean square error (e_{rms}) and is a measure of the error between the two images [3, pp. 256-257]:

$$e_{rms} = \frac{1}{N} \left[\sum_{x=0}^{N-1} \sum_{y=0}^{N-1} |g(x,y) - f(x,y)|^2 \right]^{0.5} \quad (IV.1)$$

where, for NxN pixel images, $f(x,y)$ is the array of pixel values for the original image while $g(x,y)$ is the array of pixel values for the decompressed image. Figure IV.4 graphically displays a plot of quality factor versus e_{rms} for each of the three test images. The graph data is tabulated in Table A.2 in Appendix A. As the quality factor is increased, the e_{rms} of the decompressed image decreases as expected. The decompressed

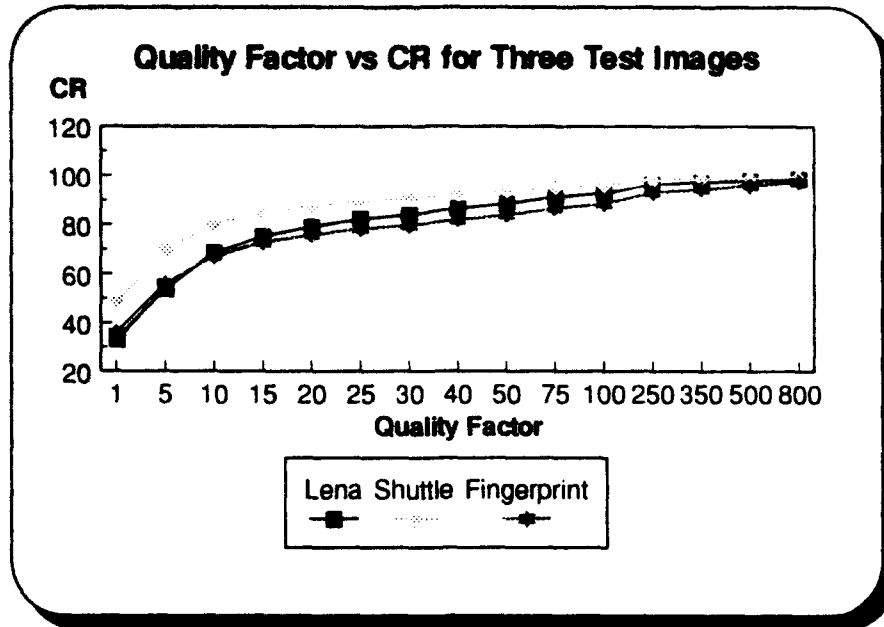


Figure IV.3: Comparison of Quality Factor vs CR for the Three Test Images.

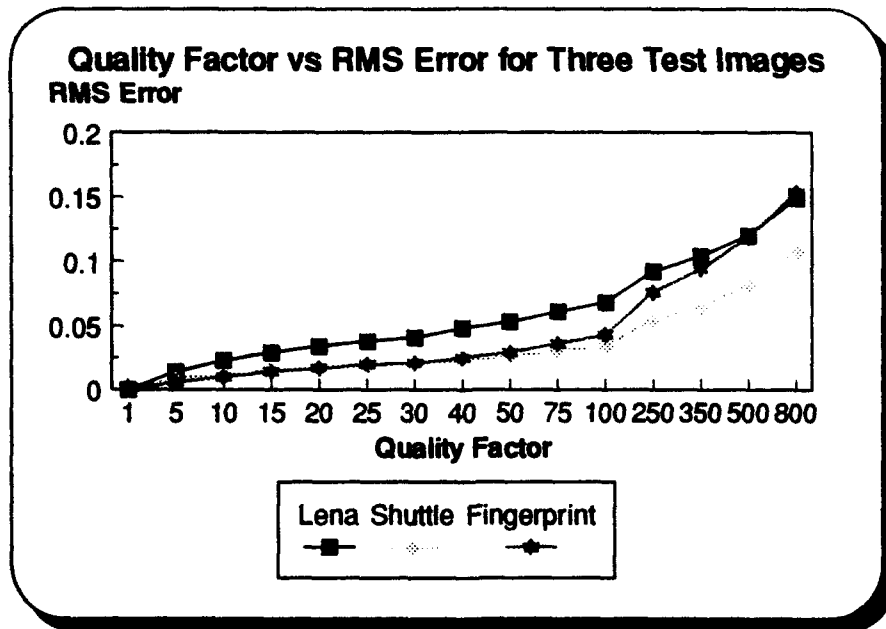


Figure IV.4: Comparison of Quality Factor vs e_{rms} for the Three Test Images.

test image LENA is displayed in Figure IV.5 after compression at various quality factors. Note that as the quality factor increases, the resolution of the decompressed image decreases. At quality factors greater than 100, the decompressed image begins to exhibit distinct blockiness due to the processing of 8x8 pixel blocks by the JPEG algorithm.

D. SECONDARY COMPRESSION

The hybrid lossless compression model was first evaluated by assessing if it is feasible, in terms of compression overhead, to use secondary compression to achieve a lossless process. In order for secondary compression to be feasible, it would have to contribute some measureable increase in the compression achieved after compressing an image using lossy JPEG. The lossless compression methods used for secondary compression are Huffman, Arithmetic, and LZW and the code is taken from Nelson [2]. Table IV.1 contains the results of secondary compression on the three test images first compressed using lossy JPEG at different quality factors. The results are expressed as the percent compression ratio (CR) achieved. The results show that secondary compression does not significantly increase the compression of the three test images used. In fact, in all but a few cases, secondary compression of the lossy JPEG compressed image resulted in an expansion (i.e., negative CR) of the compressed image file size instead of compression. Since secondary compression does not provide a significant reduction in the compressed image file size, the hybrid lossless compression model was modified accordingly. The modified hybrid lossless compression model is displayed in Figure IV.6.

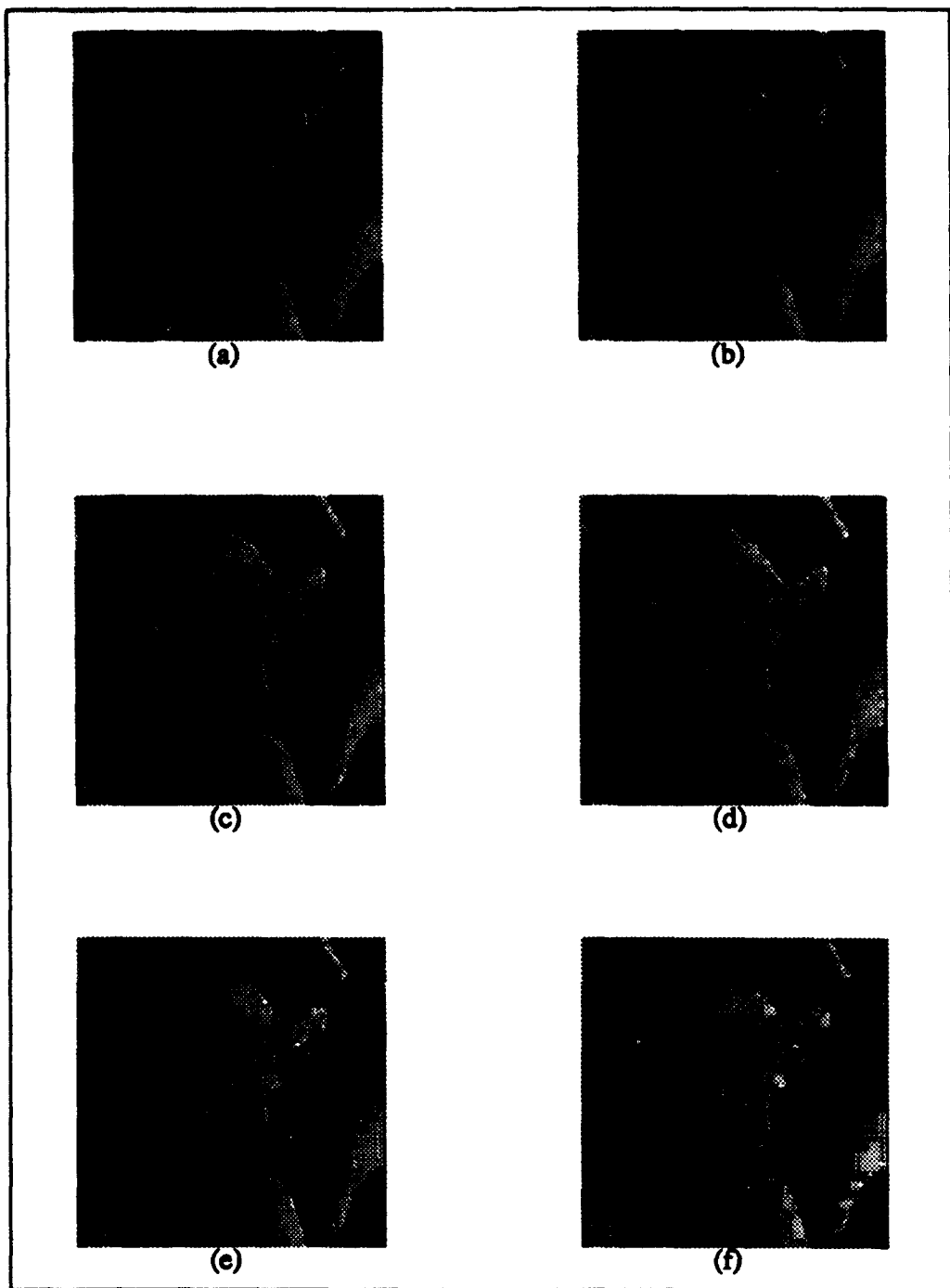


Figure IV.5: Decompressed LENA at Various Quality Factors (a)Original Image, (b) Q=100, (c) Q=250, (d) Q=350, (e) Q=500, (f) Q=800.

which achieved the best results is graphed in Figure IV.7. For all three test images, lossless JPEG achieved greater compression than the other three lossless compression methods used. Nonetheless, the lossless JPEG does not provide the convenience of a browse and residual decomposition.

The hybrid lossless compression model (Figure IV.6) was first evaluated using standard lossless compression techniques. Another lossless method, Diagonal coding, recently reported in the literature, will be discussed in the context of the hybrid model in Chapter V. Huffman, Arithmetic, LZW, and lossless JPEG were used to compress the residual image ((B) shown in Figure IV.6). The results achieved after compressing the three test images using the hybrid lossless compression model with Huffman, Arithmetic, LZW, and the lossless JPEG methods are graphically displayed in Figures IV.8, IV.9, and IV.10 respectively. The test images were compressed at various quality factors. The lossless JPEG predictor algorithm that achieved the greatest compression of each residual image is graphed. The second and third order predictor algorithms ($K=4, 5, 6, 7$) predominantly achieved the greatest CR on the residual images and are identified in Table A.3 in Appendix A for each image.

A comparison between the compression results achieved by the direct lossless compression methods and the hybrid lossless compression model is graphically displayed in Figures IV.11, IV.12, and IV.13 for each of the three test images at various quality factors. For ease of reading, it should be noted that the right-most 3-D bar in each column represents the compression achieved compressing the image with that particular direct lossless compression method (not using the hybrid lossless compression model).

F. CONCLUSIONS

The lossless JPEG algorithm achieves the greatest compression on each of the three test images when compared to the other three direct lossless compression methods. The lossless JPEG predictor algorithm which achieved the greatest compression was $K=2$, $K=6$, and $K=5$ for the test images LENA, SHUTTLE, and FINGERPRINT, respectively, and achieved compression ratios of 34%, 49%, and 27% (see Figure IV.7). The highest compression ratio achieved by the other three direct lossless compression methods for each of the three images was 8%, 18%, and 13% (see Figure IV.7).

The hybrid lossless compression model achieved its best compression results on the test image LENA when the arithmetic method was used to compress the residual image. The best overall compression was achieved using a quality factor of 100 to compress the original image with lossy JPEG. A hybrid compression ratio of 31% was achieved compared to direct Huffman (7%), Arithmetic (7%), LZW (-3%), and lossless JPEG (34%) (see Figure IV.8).

The hybrid model achieved its greatest compression on SHUTTLE when using a quality factor of 50 to compress the original image using lossy JPEG and the arithmetic method to compress the residual image. A hybrid compression ratio of 48% was achieved using this combination compared to direct Huffman (16%), Arithmetic (16%), LZW (18%), and lossless JPEG (49%) (see Figure IV.9).

The greatest compression was achieved on FINGERPRINT when using a quality factor of 50 in combination with the arithmetic method. A hybrid compression ratio of

31% was achieved compared to direct Huffman (13%), Arithmetic (13%), LZW (13%), and lossless JPEG (27%) (see Figure IV.10).

In all cases, the hybrid lossless compression model achieved greater compression ratios on all three test images than did the direct lossless compression methods with the exception of the direct lossless JPEG method. Due to the wide diversity of images compressed using the hybrid model, these results suggest that the hybrid model will achieve similar favorable compression results on any grey-scale image. The hybrid model achieved a lesser CR on LENA and SHUTTLE than did direct lossless JPEG; however, the model did achieve a greater CR than direct lossless JPEG on FINGERPRINT at quality factors of 50 and 100 (see Figure IV.10). The hybrid model enjoys the advantage of producing a compressed browse image which is significantly more compressed than the direct lossless JPEG compressed image. For instance, using a quality factor of 100 to compress LENA produces a lossy compressed browse image with a file size of 4823 bytes (compression ratio of 92%). The best lossless JPEG predictor algorithm produces a direct lossless compressed file size of 43322 bytes (compression ratio of 34%). Decompressing the lossy compressed LENA browse image produces an image that is visually lossless with no visual distortions (see Figure IV.5 (b)). If a lossless image is desired then the residual image of 40353 bytes can be transmitted and added to the browse image to produce an exact replica of the original image.

In the next chapter, a recently discovered lossless method known as Diagonal coding [7] is discussed and tested. Comparison to the results of this chapter will be made.

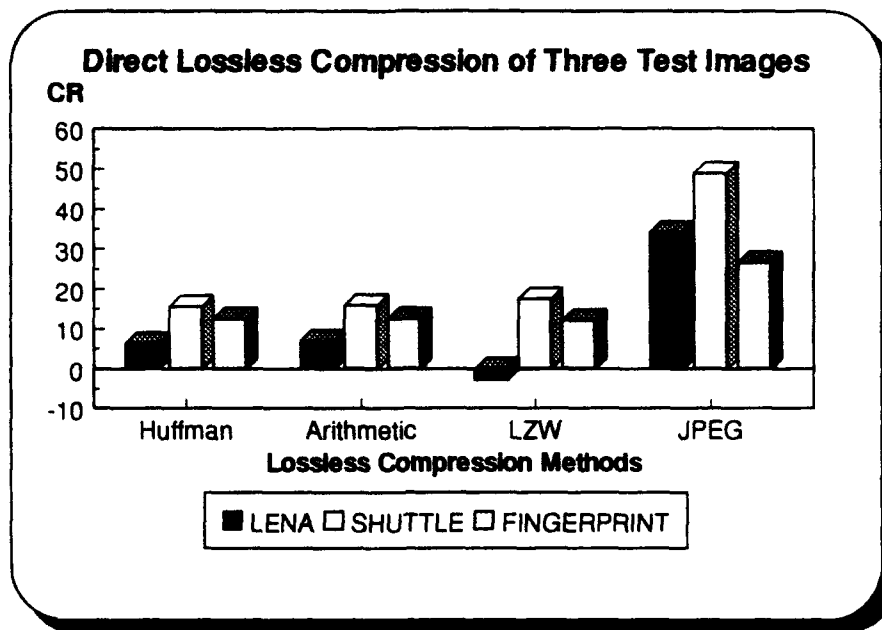


Figure IV.7: Comparison of Direct Lossless Compression on the Three Test Images.

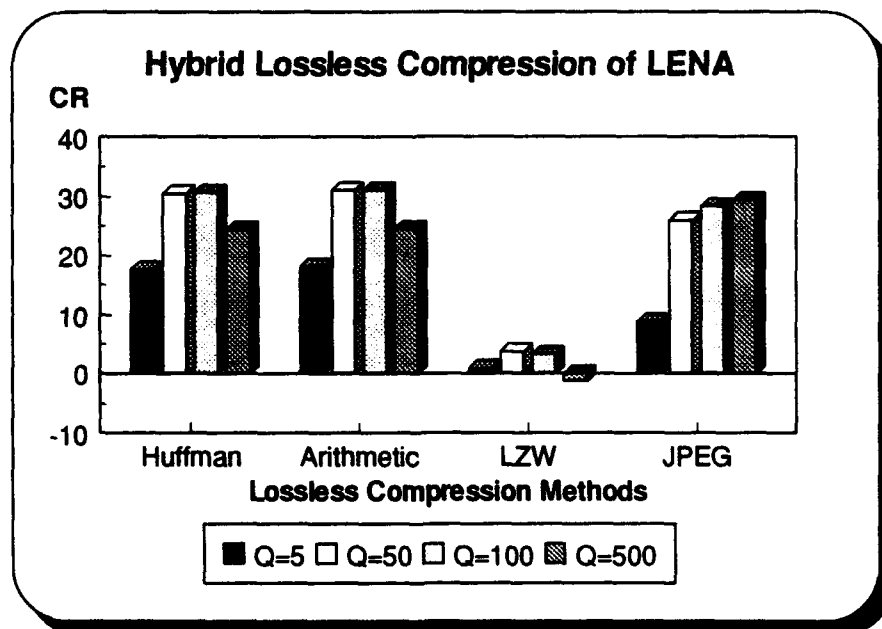


Figure IV.8: CR Achieved Using the Hybrid Lossless Compression Model on LENA at Various Quality Factors.

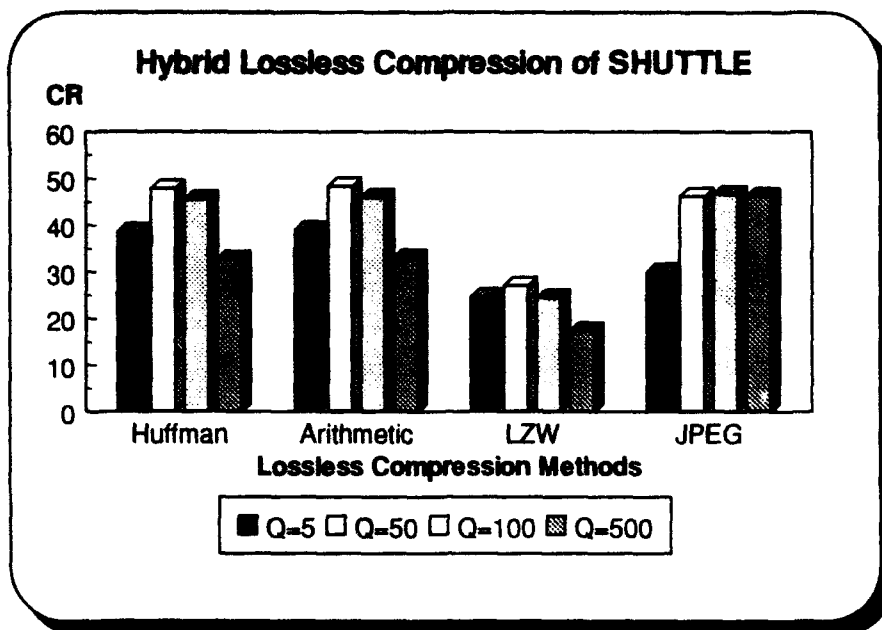


Figure IV.9: CR Achieved Using the Hybrid Lossless Compression Model on SHUTTLE at Various Quality Factors.

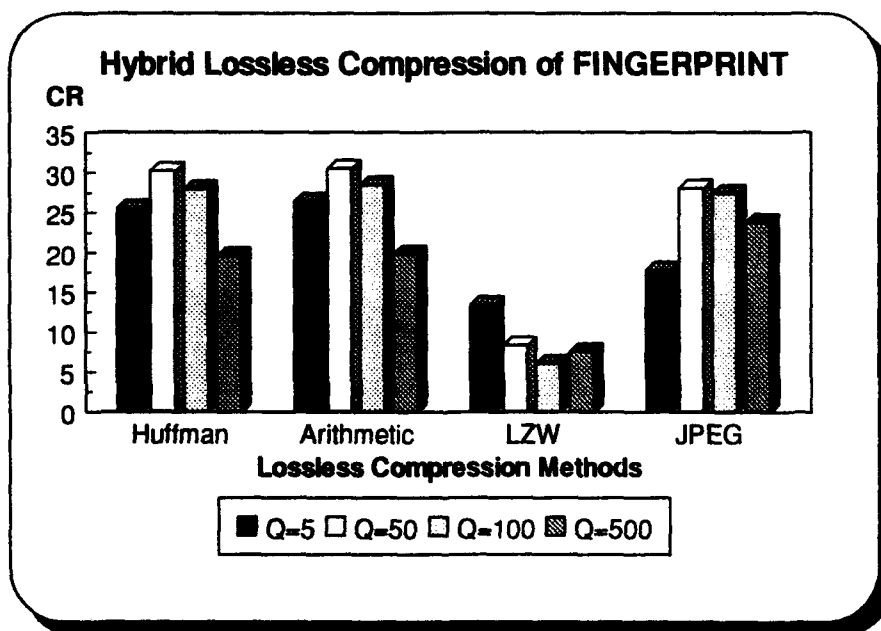


Figure IV.10: CR Achieved Using the Hybrid Lossless Compression Model on FINGERPRINT at Various Quality Factors.

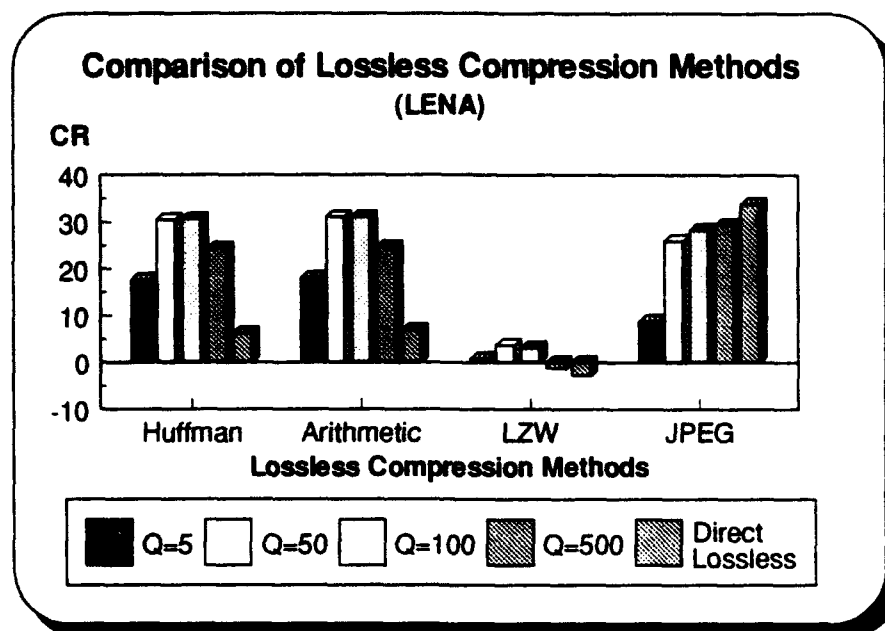


Figure IV.11: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for LENA at Various Quality Factors.

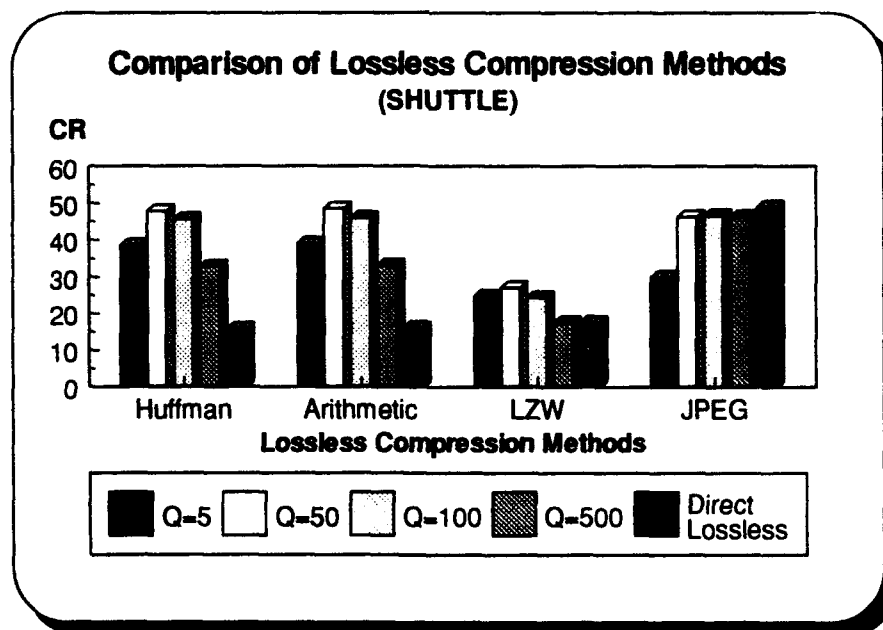


Figure IV.12: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for SHUTTLE at Various Quality Factors.

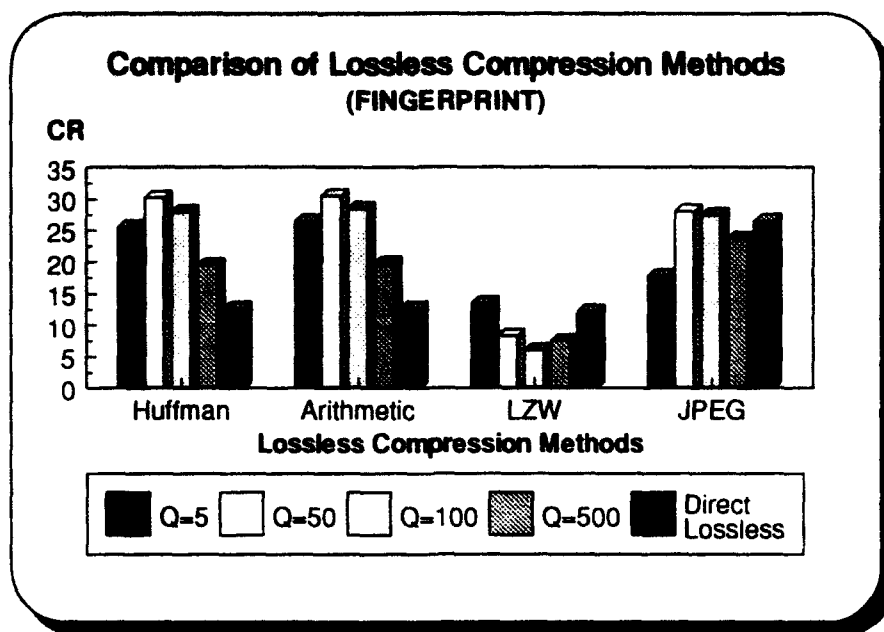


Figure IV.13: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for FINGERPRINT at Various Quality Factors.

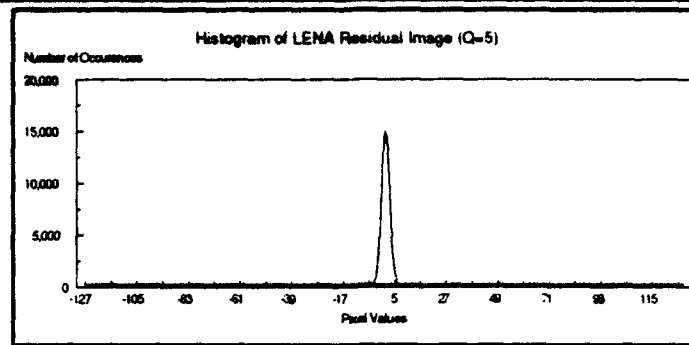
V. DIAGONAL CODING

A. INTRODUCTION

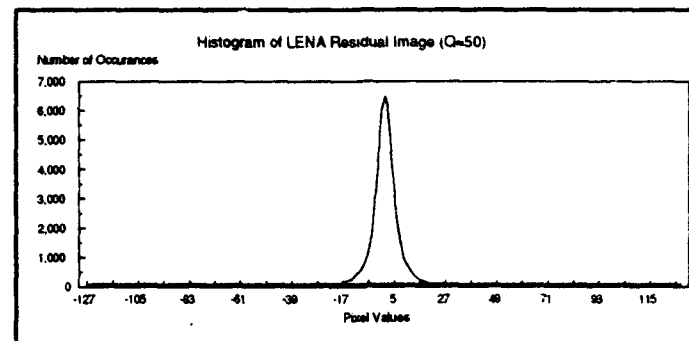
Another lossless compression method is Diagonal coding. Although not a standard compression method, it is nonetheless a simple, easy to implement compression method which achieves good compression ratios when used to compress residual images resulting from the compression of the original image at low quality factors. The compression ratios achieved by Diagonal coding are compared to those attained by the other standard lossless compression methods.

B. RESIDUAL IMAGE HISTOGRAM

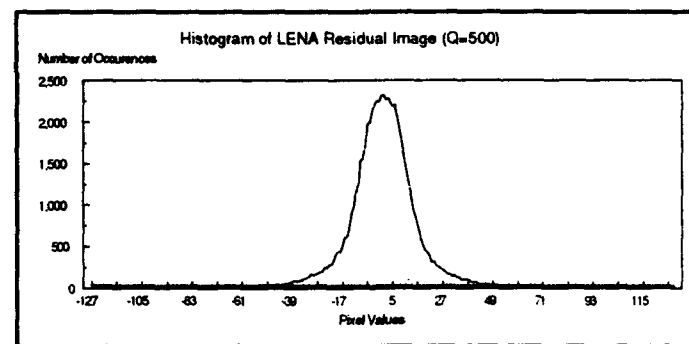
The residual image resulting from the pixel by pixel differences in the original image and the decompressed image exhibits a Laplacian distribution with a mean of zero. The residual image distribution, or histogram, has a reduced variance compared to the original image and is also significantly less correlated [1, p. 60]. The shape of the residual image histogram is dependent upon the quality factor used to compress the original image using lossy JPEG. As previously discussed in Chapter IV, the higher the quality factor used, the more compression achieved; however, the decompressed image will less resemble the original image. This results in a residual image containing a wider range of pixel values. As a result, the residual image histogram will exhibit a wider Laplacian distribution. Lossless compression routines which are designed to take advantage of this type of image distribution will achieve significant compression results. Figure V.1 displays residual image histograms of LENA for various quality factors. Note



(a)



(b)



(c)

Figure V.1: Residual Image Histograms of LENA (a) $Q=5$, (b) $Q=50$, (c) $Q=500$.

that as the quality factor used to compress the original image of LENA is increased, the distribution of the corresponding residual image widens.

C. DIAGONAL CODING

Due to the residual image exhibiting a Laplacian distribution with a smaller variance of pixel values than the original image, a lossless compression method that employs variable length encoding should achieve significant compression of the data [7, pp. 9-10]. Diagonal coding is a type of variable length encoding designed to take advantage of the Laplacian distribution of the residual image. In Diagonal coding, each pixel value is represented by the number of zeros corresponding to that value, terminated by a one. Since higher pixel values in the residual image data occur less frequently than lower pixel values, the coding is optimal [7, p. 10]. As with other lossless compression methods, there are variations to Diagonal coding. One variation is to group residual image data values together into sets and assign a diagonal code to each set. For example, a set may consist of the four values -1, 0, 1, and 2. This set may be called set 0 and assigned the diagonal code of 1. An example of Diagonal coding using sets is displayed in Table V.1. During encoding, the diagonal code representing each set is followed by two bits used to identify which value in the set is being encoded. For example, the combination of two bits of 00, 01, 10, and 11 is used to identify the residual image data values of -1, 0, 1, or 2 in set 0. Using Table V.1 as a reference, encoding the residual image data values of -1, 3, -4, and 8 would result in the code of 100011000101000111. Note that the length of a bit sequence associated with a particular residual data value (one

byte) will depend on its location in Table V.1. The C high-level language reads and writes bytes at a time. For efficient compacting of the coded bit stream, a special C source code program was written that operates at the bit level. Operating at the byte level would destroy any advantages of this coding method. The source code for the Diagonal coding (encoding and decoding) used in the thesis was written by the author and is enclosed as Appendix B. A flowchart of the source code for encoding and decoding is shown in Figures V.2 and V.3 respectively.

Set	Range	Diagonal Code
0	(-1,0,1,2)	1
1	(-3,-2,3,4)	01
2	(-5,-4,5,6)	001
3	(-7,-6,7,8)	0001
4	(-9,-8,9,10)	00001
5	(-11,-10,11,12)	000001
6	(-13,-12,13,14)	0000001
7	(-15,-14,15,16)	00000001
8	(-17,-16,17,18)	000000001
9	(-19,-18,19,20)	0000000001

Table V.1: Diagonal Coding Example.

Diagonal coding was first used in a direct compression role to compress the three test images. A comparison of diagonal coding with the other four lossless compression methods was performed. The results are graphically displayed in Figure V.4. The graph data is tabulated in Table A.4 in Appendix A. Figure V.4 is the same as Figure IV.7 in

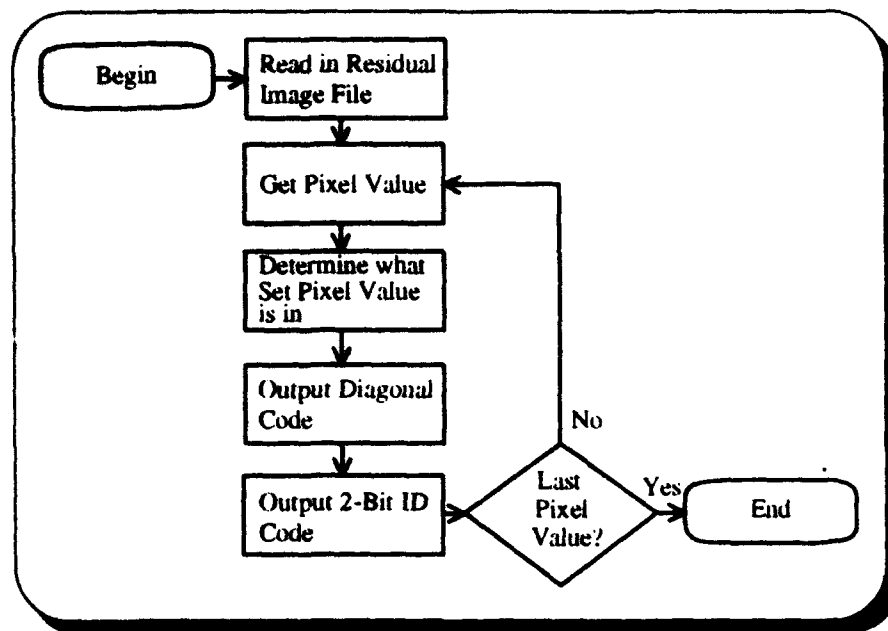


Figure V.2: Flowchart for Diagonal Encoding.

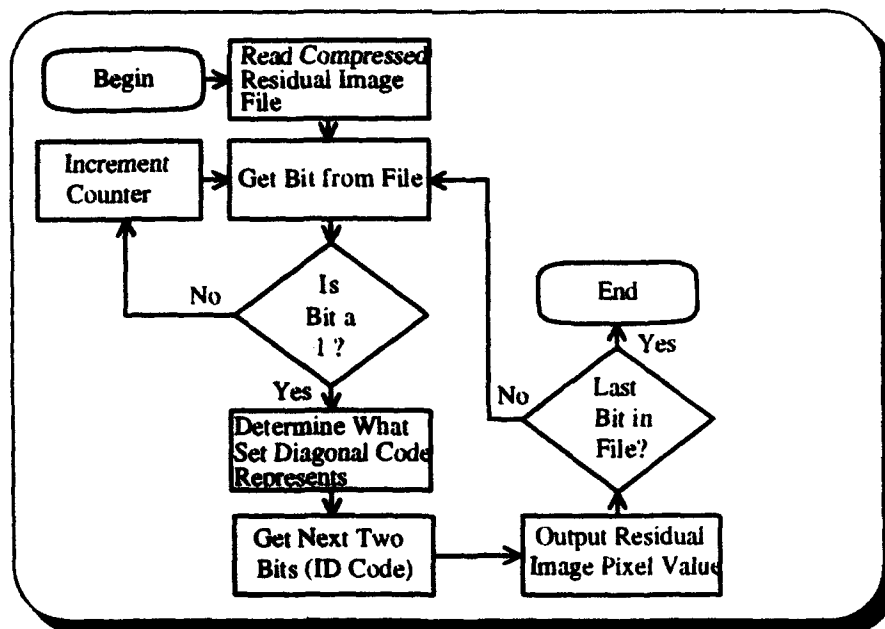


Figure V.3: Flowchart for Diagonal Decoding.

Chapter IV with the addition of the Diagonal coding results. Diagonal coding produced an expansion in the image file size after compression for all three of the test images. It is clearly not a viable lossless compression method for images which do not exhibit a narrow Laplacian distribution (histogram) with a mean of zero.

Next, Diagonal coding was used in the hybrid lossless compression model to compress the residual image. Each of the three test images were used and were compressed at various quality factors. A comparison of Diagonal coding with the other four lossless compression methods was performed. The results are graphically displayed in Figures V.5, V.6, and V.7. It is observed that at low quality factors (i.e., low c_{rms}) the standard entropy based methods, Huffman and Arithmetic, are very competitive in the hybrid model. At high quality factors (i.e., high c_{rms}), the lossless JPEG tends to be the most competitive. It is noted that Diagonal coding is very inefficient at a quality factor of 500. The graph data is tabulated in Tables A.5, A.6, and A.7 in Appendix A. These three figures are the same as Figures IV.8, IV.9, and IV.10 in Chapter IV with the addition of the Diagonal coding results.

A comparison between the compression results achieved using direct lossless compression and the hybrid lossless compression model using Huffman, Arithmetic, LZW, lossless JPEG, and Diagonal coding to compress the residual image is graphically displayed in Figures V.8, V.9, and V.10. It is observed that, with the exception of lossless JPEG, generally one or more of the hybrid compression schemes will achieve a higher CR than its direct counterpart (see the fifth column for each method in Figures

V.8, V.9, and V.10). In the lossless JPEG case, the hybrid methods are fairly competitive to the direct, lossless JPEG and even demonstrate a slight marginal CR advantage in the case of FINGERPRINT (see Figure V.7). In most cases, LZW is not competitive with the other lossless compression methods. The graph data is tabulated in Table A.8, A.9, and A.10 in Appendix A. Once again, these three figures are the same as Figures IV.11, IV.12, and IV.13 in Chapter IV with the addition of the Diagonal coding results.

Other variations of Diagonal coding were tested and evaluated in an attempt to achieve higher compression results when compressing the residual images. One variation consisted of altering the number of range values in each set and performing Run-length encoding on the longer diagonal codes. This variation achieved minimal compression improvements and in most instances resulted in less compression than did the baseline Diagonal coding method. Another variation consisted of breaking up the residual image data into bit planes and performing Diagonal coding on them. For example, the six most significant bit (MSB) bit planes were combined together, and the two least significant bit (LSB) bit planes were combined together to form two separate data sets. Diagonal coding was performed on each data set and the resulting compressed files were added together to form an 8-bit compressed file. Different combinations of bit-planes were tested and evaluated; however, none achieved the compression results attained by performing Diagonal coding on the original 8-bit residual image file. The two LSB's in the residual image are primarily noise and contribute little to the quality of the original image. If they are removed from the original image, no significant visual degradation

occurs to the image. Indeed, high compression ratios were achieved using Diagonal coding to compress the six MSB's; however, the process is not a truly lossless one and was therefore not included in the compression ratio comparisons. A representative sample of data produced from each of the Diagonal coding variations is tabulated in Tables A.11 and A.12 in Appendix A.

D. CONCLUSIONS

Diagonal coding is not as effective as Huffman, Arithmetic, and lossless JPEG in compressing the residual image; however, Diagonal coding does achieve higher compression of the residual image than does LZW in most cases. Diagonal coding achieves close to the same compression results as Huffman, Arithmetic, and lossless JPEG at some quality factors. As the quality factor used to compress the original image is increased, the compression achieved using Diagonal coding decreases. This is due to the residual image distribution widening, thereby resulting in longer diagonal codes. At some point, Diagonal coding will result in the expansion of the residual image file size. Diagonal coding resulted in an expansion of the residual image size when used to compress FINGERPRINT at a quality factor of 500 (see Figure V.7). The benefits of using Diagonal coding is its ease of implementation and non-complex nature. It is a non-CPU intensive algorithm with minimal execution times as compared to Huffman and Arithmetic. Additionally, it achieves comparable compression results at some quality factors.

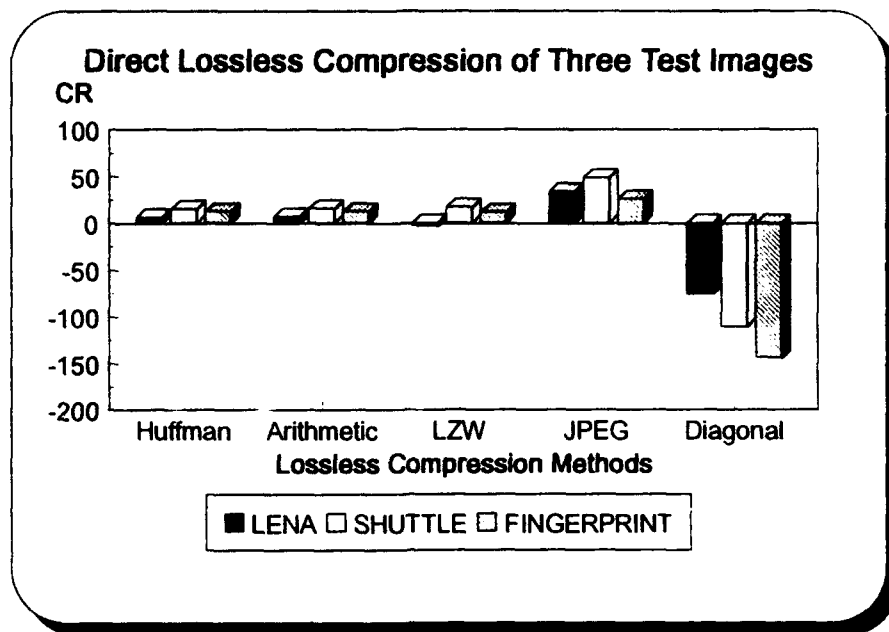


Figure V.4: Direct Lossless Compression of Three Test Images.

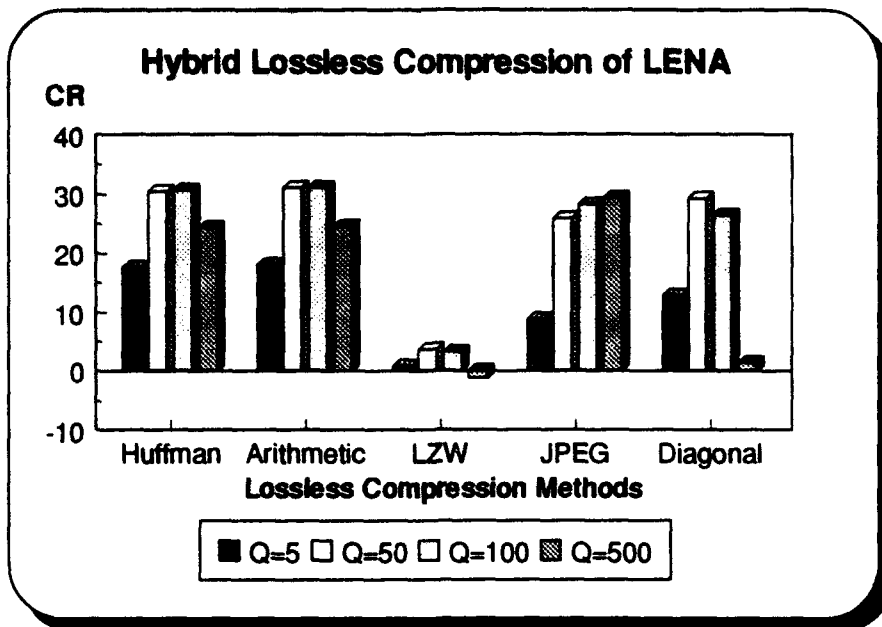


Figure V.5: CR Achieved Using the Hybrid Lossless Compression Model on LENA at Various Quality Factors.

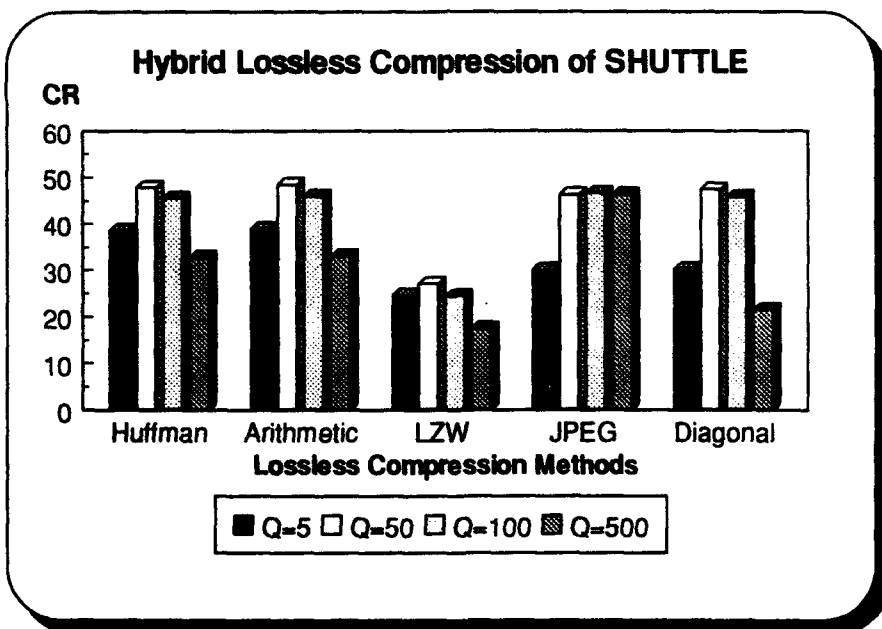


Figure V.6: CR Achieved Using the Hybrid Lossless Compression Model on SHUTTLE at Various Quality Factors.

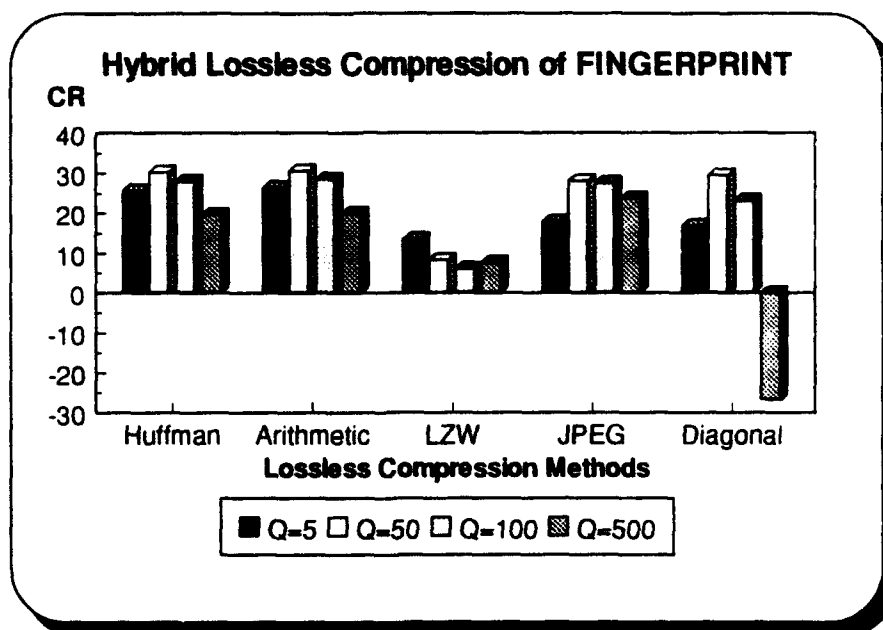


Figure V.7: CR Achieved Using the Hybrid Lossless Compression Model on FINGERPRINT at Various Quality Factors.

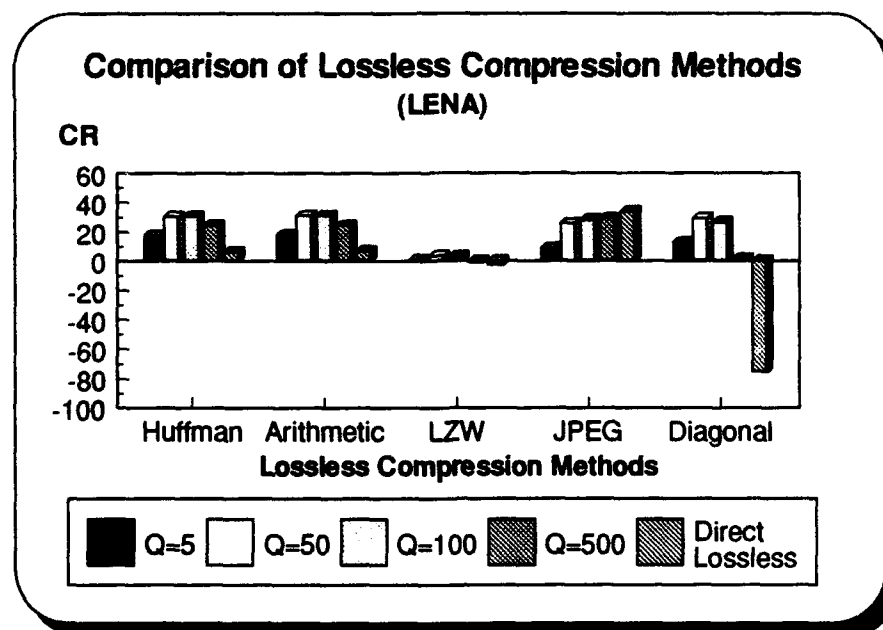


Figure V.8: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for LENA at Various Quality Factors.

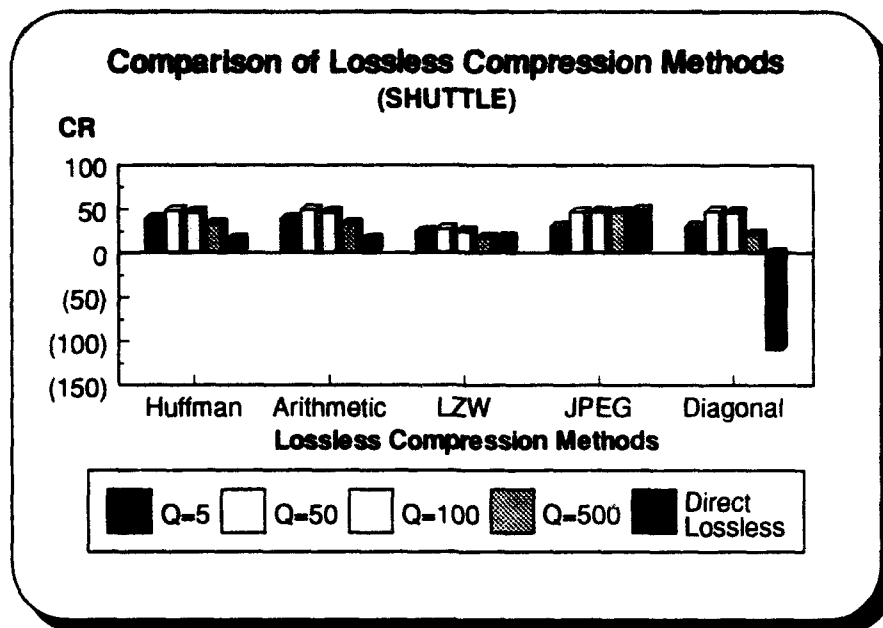


Figure V.9: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for SHUTTLE at Various Quality Factors.

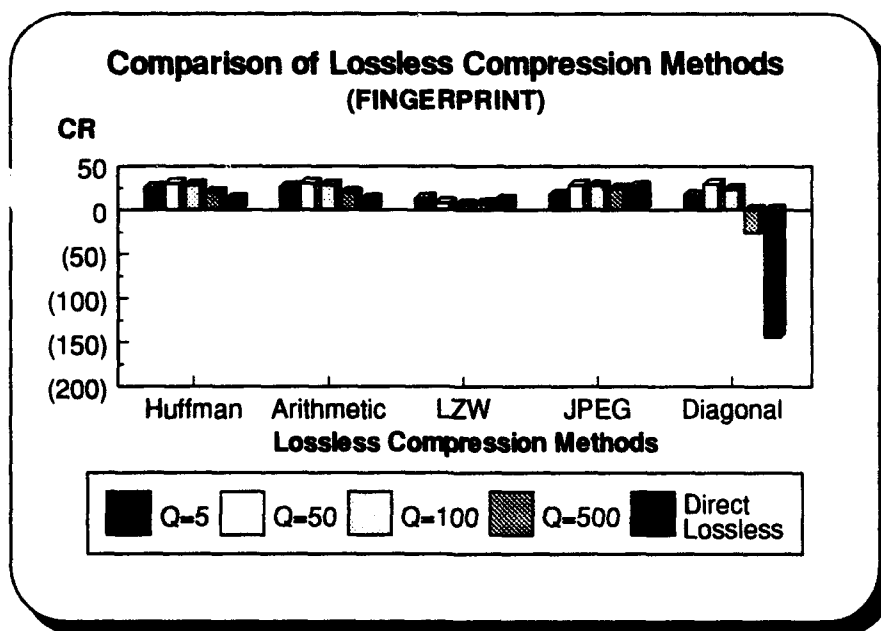


Figure V.10: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for FINGERPRINT at Various Quality Factors.

VI. HYBRID MODEL OPTIMIZATION

As discussed in section V.B, the quality factor will impact the Laplacian distribution of the residual image. The results of this chapter will show that the compressibility of both the browse and residual images depend on the quality factor. At low quality factors, minimal compression is achieved on the browse image; however, the residual image becomes highly compressible. As the quality factor is increased, the browse image is more compressible, but the residual image compresses less. Since the overall lossless image is the sum of the compressed browse and residual image data (see Equation II.2), achieving maximum overall compression would ostensibly depend on finding some optimal quality factor. In this chapter, we will examine this issue as well as the sensitivity of the overall CR to the quality factor for the images chosen.

Figures VI.1, VI.2, and VI.3 display the overall CR achieved using the hybrid lossless compression model with the three test images. These three figures are very similar to Figures V.5, V.6, and V.7. The difference is in the way the data is displayed and the number of quality factors used. The graphical data is tabulated in Tables A.13, A.14, and A.15 in Appendix A. Note that the graphical results of using Diagonal coding to compress FINGERPRINT in Figure VI.3 are limited to a quality factor of 350. This is due to the degree of expansion Diagonal coding produces at quality factors greater than 350 on FINGERPRINT. The quality factor used to compress the original image ranges from a value of 5 to 1000 so that a wide range of browse and residual images are produced and evaluated. The three figures show that, for high quality factors, lossless JPEG is the compression method which achieves the best CR on the test images. In most

instances, Huffman, Arithmetic, LZW, and Diagonal coding achieve decreasing compression on each of the images at the higher quality factors while the hybrid lossless JPEG achieves virtually the same CR at quality factors of 50 or higher. This signifies that at higher quality factors, the hybrid model is relatively insensitive to the quality factor provided that lossless JPEG is used to compress the residual image. In other words, the correct choice for the quality factor is essentially dictated by conditions such as browse image compression and browse image quality, not overall hybrid lossless CR. The browse image becomes visually distorted and lossy at the higher quality factors. It is left up to the user to determine what a good quality browse image is for the particular application the hybrid model is being used.

At the lower quality factors, Huffman, Arithmetic, Diagonal, and lossless JPEG achieve comparable compression ratios at different quality factors for the three images. The choice of which lossless compression method to use depends on the user's requirements for complexity and compression/decompression time. LZW does not appear to be a wise choice for lossless compression in almost any case.

The advantage of decomposing the original image into a browse and residual image is the reduced compressed browse image file size compared to the direct lossless compressed image file size. Figures VI.4, VI.5, and VI.7 display the browse and residual image compression ratios, and corresponding overall hybrid compression ratios, for the three test images at various quality factors. The lossless compression method (Huffman, Arithmetic, LZW, lossless JPEG, or Diagonal coding) that produces the highest overall hybrid CR is the one that is graphed. The best direct lossless JPEG compression ratio is

graphed for comparison. The graphical data is tabulated in Tables A.16, A.17, and A.18 in Appendix A. As the quality factor increases, the browse CR decreases and the residual CR increases as expected. In all cases, the browse CR is significantly greater than the direct lossless CR. At quality factors of 100 or less, all three test images are visually lossless. A comparison of the browse CR with the direct lossless JPEG CR for quality factors of 100 or less (see Figures VI.4, VI.5, and VI.6) demonstrates the advantage of decomposing the original image into a browse and residual image (i.e., a visually lossless browse image is produced which has a significantly higher CR than the direct lossless JPEG).

The highest overall hybrid CR was achieved using Arithmetic coding at quality factors of 5, 50, and 100 for LENA in Figure VI.4. At quality factors of 500 and 800, lossless JPEG was used to compress the residual image.

The highest overall hybrid CR was achieved using Arithmetic coding at quality factors of 5 and 50 for SHUTTLE in Figure VI.5. Lossless JPEG was used to compress the residual image at quality factors of 100, 500, and 800.

The highest overall hybrid CR was achieved using Arithmetic coding at quality factors of 5, 50, and 100 for FINGERPRINT in Figure VI.6. At quality factors of 500 and 800, lossless JPEG was used to compress the residual image.

The results indicate that for low quality factors (≤ 50) Arithmetic coding is the best choice for lossless compression of the residual images while at higher quality factors (> 50), lossless JPEG is the best choice.

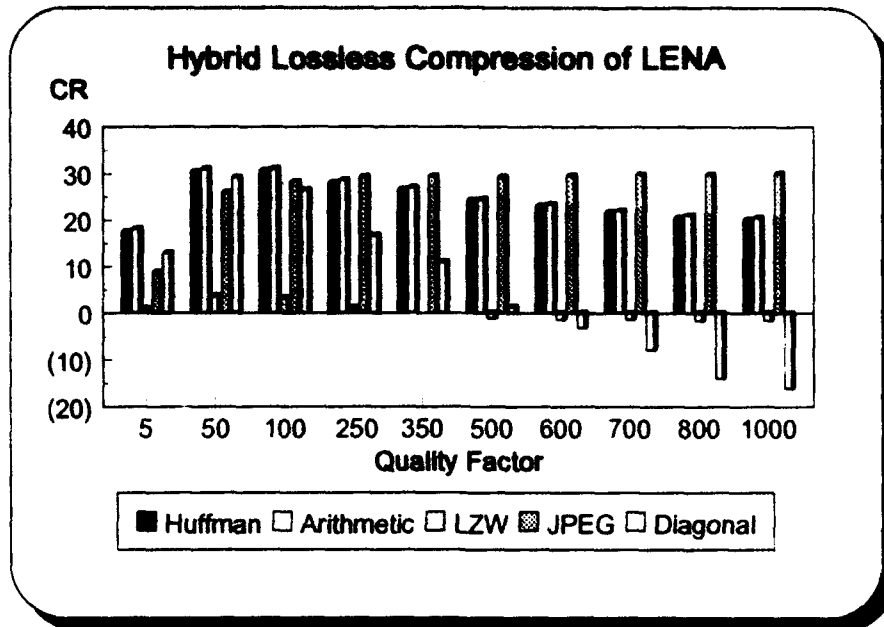


Figure VI.1: Hybrid Lossless Compression of LENA at Various Quality Factors.

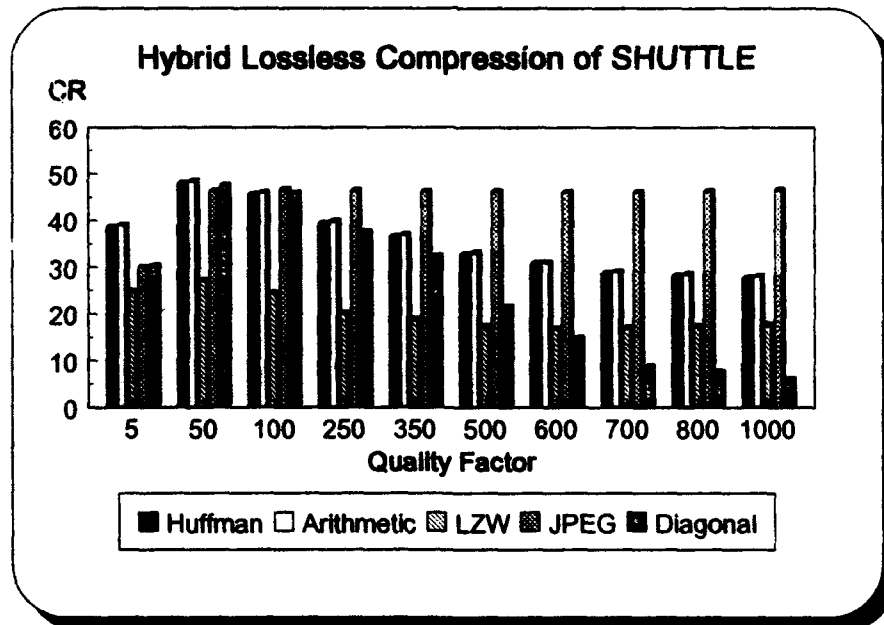


Figure VI.2: Hybrid Lossless Compression of SHUTTLE at Various Quality Factors.

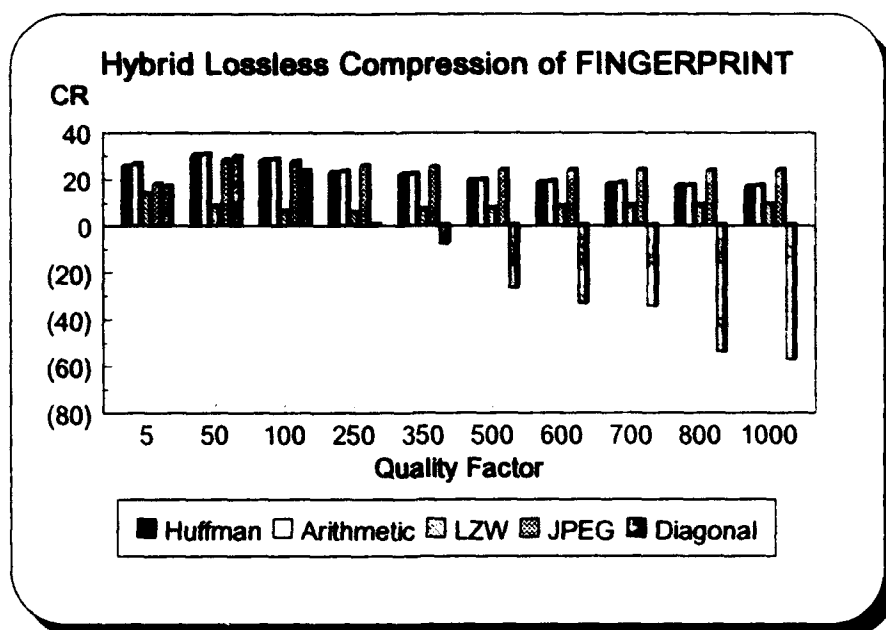


Figure VI.3: Hybrid Lossless Compression of FINGERPRINT at Various Quality Factors.

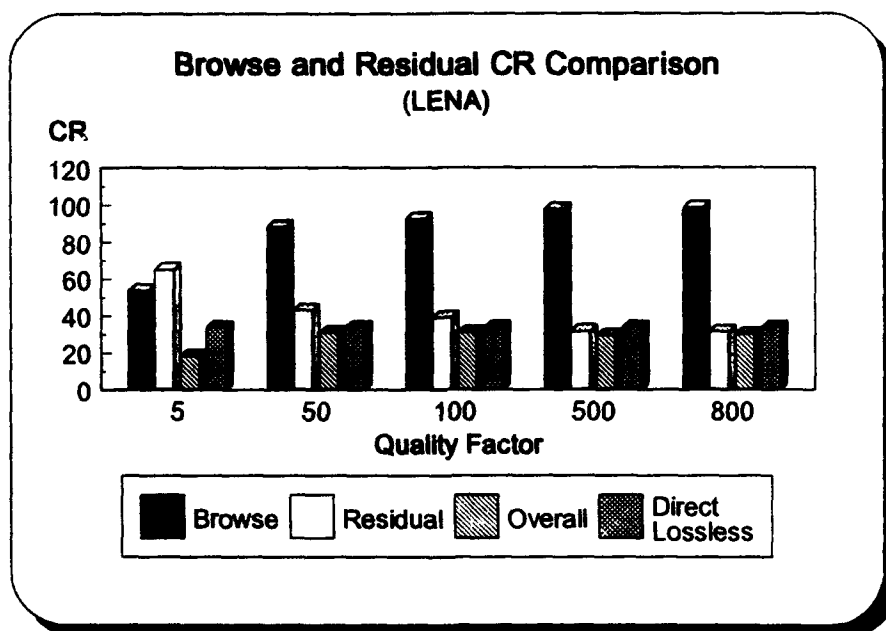


Figure VI.4: Browse and Residual CR Comparison with Direct Lossless Compression for LENA.

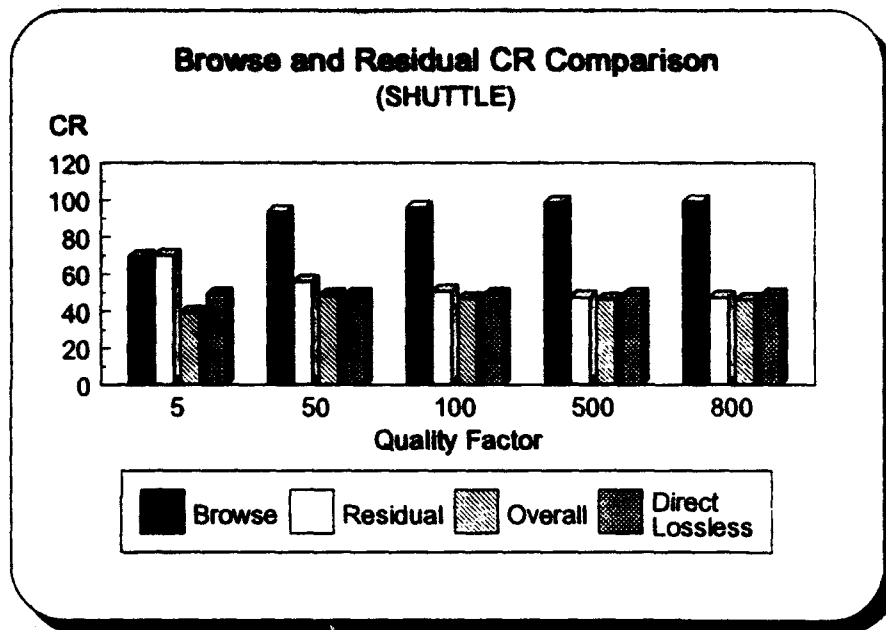


Figure VI.5: Browse and Residual CR Comparison with Direct Lossless Compression for SHUTTLE.

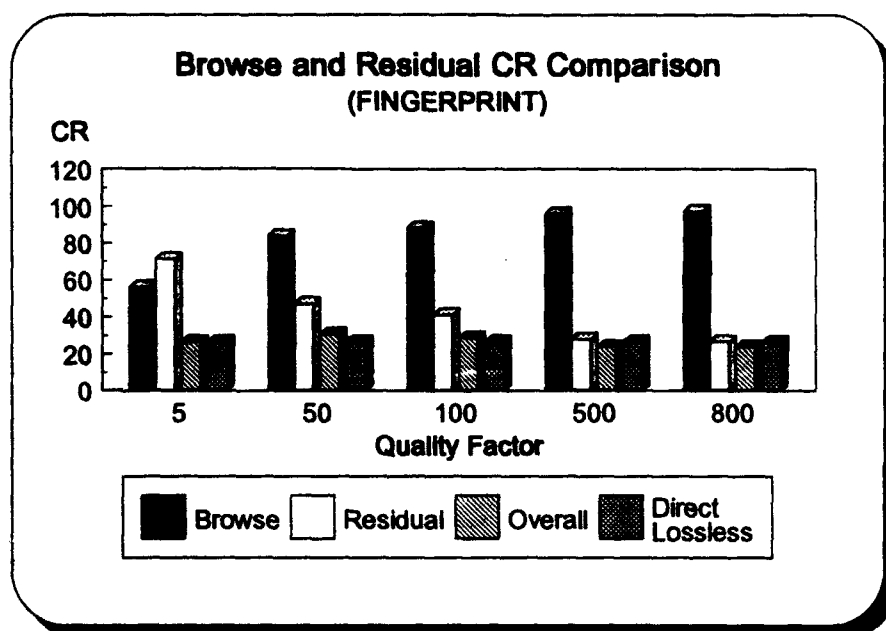


Figure VI.6: Browse and Residual CR Comparison with Direct Lossless Compression for FINGERPRINT.

VII. CONCLUSIONS

An analysis of the results of Chapters IV, V, and VI indicates that the proposed hybrid lossless compression model has merit as a lossless image compression method. With the exception of lossless JPEG, the substitution of the other lossless compression methods (Huffman, Arithmetic, LZW, and Diagonal coding) into the hybrid model produce compression results that generally outperform their direct compression counterparts. The decomposition of the original image into browse and residual images gives an end-user the ability to browse an image and determine whether the residual image should be transmitted and added to the browse image to reproduce the original image. This feature is not available with any direct lossless compression method. The quality of the browse image and the overall compression achieved are determined by the quality factor used to compress the original image using lossy JPEG and is a user controlled variable. The better the browse approximates the original data, the more compressible is the residual image data. Thus, a better quality browse results in a residual that can be compressed better in lossless mode. However, a better quality browse results in a larger browse image file size. The key factors are to select a quality factor which produces a visually acceptable browse image and a lossless compression method that achieve the best overall compression.

The results show that LZW is not a lossless compression method which should be used to compress the residual image. The residual images do not contain long repetitive strings of pixel values which are necessary for LZW to achieve high compression results. This is not surprising since the LZW method is designed primarily for compressing text,

not visual graphics [2, pp. 23-24]. Diagonal coding is a viable candidate for lossless hybrid compression at lower quality factors. As the quality factor increases though, Diagonal coding results in poor compression and eventually even expansion of the residual image file size. Huffman and Arithmetic achieve comparable compression results at all quality factors. At the lower quality factors, Huffman and Arithmetic do as well as or better than lossless JPEG in most cases; however, lossless JPEG is the prime choice for lossless compression of the residual image at higher quality factors (i.e., high e_{rms}). Under these conditions, the JPEG predictor is better able to accurately predict pixel values for all residual image distributions resulting in higher compression ratios. This ostensibly is a result of a higher 2-D correlation of pixel values within the corresponding residual images and, consequently, facilitates compression in the JPEG lossless method.

Future areas of research include the classification of image types so that the optimum or nearly optimum combination of quality factor and lossless compression method may be selected which produces a visually acceptable browse image and the greatest overall compression ratio. Unfortunately, quality factors or rms error parameters are not perfect indicators of subjectively evaluated image quality. Until such an indicator exists, it appears that producing a general guideline for selecting a lossless compression method applicable to all images (in general) may not be possible.

Another option is to evaluate the hybrid lossless compression model using fractal image compression as the lossy compression algorithm. This has been pursued using a combination of lossy fractal compression and lossless LZW; however, other lossless

compression methods were not reported to have been tested in their hybrid model [4]. For the data presented, it appears that although the combination of lossy fractal and lossless LZW compression produced a lossless replica of the original image, the overall CR achieved using the hybrid technique resulted in an expansion of the image file size. Compression was achieved only when the number of grey-scale values was limited (i.e., representing pixels using less than 8 bits).

APPENDIX A

Quality Factor	LENA	SHUTTLE	FINGERPRINT
1	33.0	48.5	35.7
5	53.9	69.4	55.9
10	68.4	80.2	67.1
15	75.0	84.8	72.7
20	78.9	87.3	75.7
25	82.1	89.4	78.1
30	83.8	90.5	79.5
40	86.4	92.1	81.9
50	88.4	93.4	83.7
75	91.1	94.9	86.4
100	92.6	95.8	88.2
250	96.1	97.8	93.0
350	97.0	98.3	94.4
500	97.7	98.7	95.8
800	98.5	99.0	97.2

Table A.1: Comparison of Quality Factor vs CR for the Three Test Images.

Quality Factor	LENA	SHUTTLE	FINGERPRINT
1	0.000	0.000	0.000
5	0.014	0.009	0.006
10	0.023	0.012	0.010
15	0.029	0.015	0.014
20	0.034	0.017	0.017
25	0.038	0.019	0.020
30	0.041	0.021	0.021
40	0.048	0.023	0.025
50	0.053	0.026	0.029
75	0.061	0.030	0.036
100	0.068	0.034	0.043
250	0.092	0.055	0.076
350	0.104	0.065	0.094
500	0.120	0.082	0.119
800	0.149	0.108	0.154

Table A.2: Comparison of Quality Factor vs e_{rms} for the Three Test Images.

Test Image	Q=5	Q=50	Q=100	Q=500
LENA	3	7	7	6
SHUTTLE	7	7	7	6
FINGERPRINT	7	7	7	6

Table A.3: Best JPEG Algorithm (K) for Test Images at Various Quality Factors.

Compression Type	LENA	SHUTTLE	FINGERPRINT
Huffman	6.3	15.6	12.6
Arithmetic	7.0	15.9	12.6
LZW	-2.9	17.5	12.1
JPEG	34.0	48.9	26.5
Diagonal	-75.9	-110.7	-144.5

Table A.4: Comparison of Lossless Compression Methods on the Three Test Images.

Compression Type	Q=5	Q=50	Q=100	Q=500
Huffman	17.6	30.4	30.6	24.3
Arithmetic	18.1	31.0	31.0	24.5
LZW	0.9	3.7	3.2	-1.3
JPEG	8.8	25.9	28.2	29.3
Diagonal	12.9	29.2	26.4	1.4

Table A.5: CR Achieved Using Hybrid Lossless Compression Model on LENA at Various Quality Factors.

Compression Type	Q=5	Q=50	Q=100	Q=500
Huffman	38.5	47.8	45.5	32.7
Arithmetic	38.9	48.4	45.9	33.0
LZW	24.6	27.1	24.3	17.3
JPEG	30.0	46.2	46.6	46.2
Diagonal	30.2	47.4	45.7	21.5

Table A.6: CR Achieved Using Hybrid Lossless Compression Model on SHUTTLE at Various Quality Factors.

Compression Type	Q=5	Q=50	Q=100	Q=500
Huffman	25.5	30.2	27.9	19.5
Arithmetic	26.4	30.5	28.4	19.8
LZW	13.4	8.3	6.0	7.5
JPEG	17.8	28.0	27.4	23.7
Diagonal	16.8	29.4	23.1	-26.9

Table A.7: CR Achieved Using Hybrid Lossless Compression Model on FINGERPRINT at Various Quality Factors.

Compression Type	Q=5	Q=50	Q=100	Q=500	Direct Lossless
Huffman	17.6	30.4	30.6	24.3	6.3
Arithmetic	18.1	31.0	31.0	24.5	7.0
LZW	0.9	3.7	3.2	-1.3	-2.9
JPEG	8.8	25.9	28.2	29.3	33.8
Diagonal	12.9	29.2	26.4	1.4	-75.9

Table A.8: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for LENA at Various Quality Factors.

Compression Type	Q=5	Q=50	Q=100	Q=500	Direct Lossless
Huffman	38.5	47.8	45.5	32.7	15.6
Arithmetic	38.9	48.4	45.9	33.0	15.9
LZW	24.6	27.1	24.3	17.3	17.5
JPEG	30.0	46.2	46.6	46.2	48.9
Diagonal	30.2	47.4	45.7	21.5	-110.7

Table A.9: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Models for SHUTTLE at Various Quality Factors.

Compression Type	Q=5	Q=50	Q=100	Q=500	Direct Lossless
Huffman	25.5	30.2	27.9	19.5	12.6
Arithmetic	26.4	30.5	28.4	19.8	12.6
LZW	13.4	8.3	6.0	7.5	12.1
JPEG	17.8	28.0	27.4	23.7	26.5
Diagonal	16.8	29.4	23.1	-26.9	-144.5

Table A.10: Comparison of Hybrid Lossless Compression Model with Standard Lossless Compression Methods for FINGERPRINT at Various Quality Factors.

Quality Factor	LENA	SHUTTLE	FINGERPRINT
5	10.4	28.6	15.4
50	23.2	43.2	22.6
100	21.1	40.1	15.6
500	-72.0	13.4	-11.3

Table A.11: CR Achieved Using Diagonal Coding and RLE Variation in Hybrid Model.

Quality Factor	LENA	SHUTTLE	FINGERPRINT
5	-8.5	6.9	-6.5
50	23.4	30.6	19.7
100	26.2	32.5	22.0
500	23.7	30.3	16.3

Table A.12: CR Achieved Using Diagonal Bit Plane Coding Variation in Hybrid Model.

Quality Factor	Huffman	Arithmetic	LZW	Lossless JPEG	Diagonal
5	17.6	18.1	0.9	8.8	12.9
50	30.4	31.0	3.7	25.9	29.2
100	30.6	31.0	3.2	28.2	26.4
250	28.1	28.6	1.1	29.3	16.7
350	26.5	27.0	0.0	29.4	11.0
500	24.2	24.5	-1.3	29.3	1.4
600	23.1	23.4	-1.5	29.6	-3.3
700	21.8	22.1	-1.6	29.8	-8.1
800	20.6	21.1	-1.8	29.9	14.2
1,000	20.1	20.6	-1.8	30.1	-16.3

Table A.13: Hybrid Lossless Compression of LENA at Various Quality Factors.

Quality Factor	Huffman	Arithmetic	LZW	Lossless JPEG	Diagonal
5	38.5	38.9	24.6	30.0	30.2
50	47.8	48.4	27.1	46.2	47.4
100	45.5	45.9	24.3	46.6	45.7
250	39.3	39.8	20.0	46.3	37.4
350	36.3	36.8	18.7	46.2	32.2
500	32.7	33.0	17.3	46.2	21.5
600	30.7	30.9	16.7	46.0	14.7
700	28.7	29.0	16.8	46.0	8.7
800	28.2	28.5	17.3	46.2	7.5
1,000	27.7	28.0	17.6	46.5	5.9

Table A.14: Hybrid Lossless Compression of SHUTTLE at Various Quality Factors.

Quality Factor	Huffman	Arithmetic	LZW	Lossless JPEG	Diagonal
5	25.5	26.4	13.4	17.8	16.8
50	30.2	30.5	8.3	28.0	29.4
100	27.9	28.4	6.0	27.4	23.1
250	22.7	23.2	5.6	25.6	0.2
350	21.9	22.1	6.9	24.8	-7.9
500	19.5	19.8	7.5	23.7	-26.9
600	18.5	18.9	8.1	23.7	-33.6
700	17.9	18.2	8.4	23.7	-34.5
800	17.0	17.3	8.7	23.6	-54.2
1,000	16.7	17.0	9.1	23.9	-57.5

Table A.15: Hybrid Lossless Compression of FINGERPRINT at Various Quality Factors.

Quality Factor	Browse	Residual	Hybrid	Direct Lossless
5	53.9	65.0	18.1	33.8
50	88.4	43.0	31.0	33.8
100	92.6	39.0	31.0	33.8
500	97.7	31.5	29.3	33.8
800	98.5	31.4	29.9	33.8

Table A.16: Browse and Residual CR Comparison for LENA.

Quality Factor	Browse	Residual	Hybrid	Direct Lossless
5	69.4	70.0	38.9	48.9
50	93.4	56.0	48.4	48.9
100	95.8	50.8	46.6	48.9
500	98.7	47.4	46.2	48.9
800	99.0	47.2	46.2	48.9

Table A.17: Browse and Residual CR Comparison for SHUTTLE.

Quality Factor	Browse	Residual	Hybrid	Direct Lossless
5	55.9	71.0	26.4	26.5
50	83.7	47.0	30.5	26.5
100	88.2	41.0	28.4	26.5
500	95.8	27.8	23.7	26.5
800	97.2	26.4	23.6	26.5

Table A.18: Browse and Residual CR Comparison for FINGERPRINT.

APPENDIX B

This appendix contains the source code for the Diagonal coding lossless compression method. The files Bitio.c, Bitio.h, Errhand.c, and Errhand.h are adapted from Nelson's text source code. The files Browse-c.c and Browse-c.c are written by the author. The programs are written in C and were compiled on a Sun Workstation with the GNU C compiler. The command to compile and link the programs is:

```
gcc Browse-c.c Bitio.c Errhand.c -lm -o Browse-c
```

This will result in a program called Browse-c which will encode an 8-bit 256x256 grayscale image using Diagonal coding. Substitute Browse-c.c into the compile command to produce the program which will decode a Diagonal coded compressed image file.

```
/*
 * browse-c.c: Program that performs Diagonal coding (encoding) on a 256x256
 * To compile: gcc browse-c.c bitio.c errhand.c -lm -o browse-c
 * To run:      browse-c [input image file] [output image file]
 * Author:      Doug Abbott
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "bitio.h"
#include "errhand.h"
```

```
#define ROWS    256
#define COLS    256
```

```
#ifdef __STDC__
```

```
void CompressFile( FILE *input, BIT_FILE *output );
void ReadInputFile( FILE *input, BIT_FILE *output_file );
int OutputCode( BIT_FILE *output_file, int code );
void print_ratios( char *input, char *output );
long file_size( char *name );
```

```
#else
```

```
void CompressFile();
void ReadInputFile();
int OutputCode();
void print_ratios();
long file_size();
```

```
#endif
```

```
main( argc, argv )
```

```
int argc;
char *argv[];
{
```

```
FILE *input;
BIT_FILE *output;
```

```
input = fopen( argv[1], "rb" );
if ( input == NULL )
    fatal_error( "Error opening %s for input\n", argv[1] );
output = OpenOutputBitFile( argv[2] );
if ( output == NULL )
    fatal_error( "Error opening %s for output\n", argv[2] );
CompressFile( input, output );
CloseOutputBitFile( output );
fclose( input );
print_ratios( argv[1], argv[2] );
return( 0 );
```

```

}

void CompressFile( input, output )
FILE *input;
BIT_FILE *output;

{
ReadInputFile( input, output );
}

void ReadInputFile( input, output_file )
FILE *input;
BIT_FILE *output_file;
{

int row;
int col;
int c, y;

/* Read in the input file to be compressed. */

for ( row = 0; row < ROWS; row++ ) {
    for ( col = 0; col < COLS; col++ ) {
        c = getc( input );
        if ( c == EOF )
            fatal_error( "Error reading input grey scale file\n" );
        y = OutputCode( output_file, c );
    }
}

/* Purge the mask of any remaining bits. */

OutputBits( output_file, 257L, 1 );
}

/* Function which determines which range set the pixel value is located in
 * and its location within the range set. The appropriate diagonal code
 * and identification value are output.
 */

int OutputCode( output_file, code )
BIT_FILE *output_file;
int code;
{
    int top_of_range, bottom_of_range;
    int bit_count, count;

    top_of_range = 127;
    bottom_of_range = 128;
    bit_count = 0;
    count = 0;

/* Determine which range set the pixel value is located in. */

    if ( code > 127 ) {
        while ( code > top_of_range ) {

```

```
        bit_count++;
        count += 1 ;
        top_of_range = top_of_range + 2;
    }
}

if ( code <= 127 ) {
    while ( code < bottom_of_range ) {
        bit_count++;
        count += 1 ;
        bottom_of_range = bottom_of_range - 2;
    }
}

/* Output diagonal code. */

OutputBits( output_file, 1L, bit_count );

/* Determine the location of the pixel value within the range set and */
/* output the two identification bits. */

if ( code <= 127 ) {
    if ( code == bottom_of_range )
        OutputBits( output_file, 0L, 2 );
    else
        OutputBits( output_file, 1L, 2 );
}
if ( code > 127 ) {
    if ( code == top_of_range )
        OutputBits( output_file, 3L, 2 );
    else
        OutputBits( output_file, 2L, 2 );
}
return ( count );
}

/* Determine the size of the input and output files (in bytes). */

#ifdef SEEK_END
#define SEEK_END 2
#endif

long file_size( name )
char *name;
{
    long eof_ftell;
    FILE *file;

    file = fopen( name, "r" );
    if ( file == NULL )
        return( 0L );
    fseek( file, 0L, SEEK_END );
    eof_ftell = ftell( file );
    fclose( file );
    return( eof_ftell );
}
```



```
/* Compute the compression ratio achieved. */

void print_ratios( input, output )
char *input;
char *output;
{
    long input_size;
    long output_size;
    int ratio;

    input_size = file_size( input );
    if ( input_size == 0 )
        input_size = 1;
    output_size = file_size( output );
    if ( output_size == 0 )
        output_size = 1;
    ratio = 100 - (int) ( output_size * 100L / input_size );
    printf( "\nInput bytes:      %ld\n", input_size );
    printf( "Output bytes:          %ld\n", output_size );
    printf( "Compression ratio:  %d%%\n", ratio );
}
```

```
/*
 * browse-e.c: Program that performs decoding of a compressed image file
 *             that has been encoded with Diagonal coding.
 * To compile: gcc browse-e.c bitio.c errhand.c -lm -o browse-e
 * To run:    browse-e [input image file] [output image file]
 * Author:    Doug Abbott
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "bitio.h"
#include "errhand.h"
```

```
#define ROWS    256
#define COLS    256
```

```
#ifdef __STDC__
```

```
void ExpandFile( BIT_FILE *input, FILE *output );
void ReadInputFile( BIT_FILE *input, FILE *output_file );
int InputCode( BIT_FILE *input_file );
long file_size( char *name );
void print_filesize( char *input, char *output );
```

```
#else
```

```
void ExpandFile();
void ReadInputFile();
int InputCode();
long file_size();
void print_filesize();
```

```
#endif
```

```
main( argc, argv )
int argc;
char *argv[];
{
```

```
FILE *output;
BIT_FILE *input;
```

```
input = OpenInputBitFile( argv[1] );
if ( input == NULL )
    fatal_error( "Error opening %s for input\n", argv[1] );
output = fopen( argv[2], "wb" );
if ( output == NULL )
    fatal_error( "Error opening %s for output\n", argv[2] );
ExpandFile( input, output );
CloseInputBitFile( input );
fclose( output );
print_filesize( argv[1], argv[2] );
return( 0 );
}
```

```

void ExpandFile( input, output )
BIT_FILE *input;
FILE *output;
{
    ReadInputFile( input, output );
}

void ReadInputFile( input, output_file )
BIT_FILE *input;
FILE *output_file;
{
    int row;
    int col;
    unsigned char amplitude;

    /* Read in the input file to be decompressed. */

    for ( row = 0; row < ROWS; row++ )
        for ( col = 0; col < COLS; col++ ) {
            amplitude = InputCode( input );
            putc( amplitude, output_file );
        }

    /* Function which decodes the compressed image file. */

    int InputCode( input_file )
    BIT_FILE *input_file;
    {
        int bit_count;
        int result;
        unsigned char amp;
        int top_of_range;
        int bottom_of_range;
        int count;

        top_of_range = 0;
        count = 1;

        /* Get bits from input image file. */

        bit_count = (int) InputBits( input_file, 1 );

        /* Looking for a bit '1'. */

        while ( bit_count == 0 ) {
            count++;
            bit_count = (int) InputBits( input_file, 1 );
        }

        /* Determine what range set the decoded code belongs to. */

        top_of_range = 130 + (( count - 1 ) * 2 );
        bottom_of_range = 127 - (( count - 1 ) * 2 );
    }
}

```

```
/* Get next two bits from the input data. These are the two identification */
/* bits. */

result = (int) InputBits( input_file, 2 );

/* Determine what the decoded pixel value is. */

if ( result > 1 )
    amp = ( top_of_range - 3 ) + result ;
else
    amp = bottom_of_range + result;
return( amp );
}

/* Determine the size of the input and output files (in bytes). */

#ifdef SEEK_END
#define SEEK_END 2
#endif

long file_size( name )
char *name;
{
    long eof_ftell;
    FILE *file;

    file = fopen( name, "r" );
    if ( file == NULL )
        return( 0L );
    fseek( file, 0L, SEEK_END );
    eof_ftell = ftell( file );
    fclose( file );
    return( eof_ftell );
}

void print_filesize( input, output )
char *input;
char *output;
{
    long input_size;
    long output_size;

    input_size = file_size( input );
    if ( input_size == 0 )
        input_size = 1;
    output_size = file_size( output );
    if ( output_size == 0 )
        output_size = 1;
    printf( "\nInput bytes:    %ld\n", input_size );
    printf( "Output bytes:    %ld\n", output_size );
}
```

```

/***** Start of BITIO.C *****/
*
* This utility file contains all of the routines needed to implement
* bit oriented routines under either ANSI or K&R C. It needs to be
* linked with every program used in the entire book.
*
*/
#include <stdio.h>
#include <stdlib.h>
#include "bitio.h"
#include "errhand.h"

#define PACIFIER_COUNT 2047

BIT_FILE *OpenOutputBitFile( name )
char *name;
{
    BIT_FILE *bit_file;

    bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
    if ( bit_file == NULL )
        return( bit_file );
    bit_file->file = fopen( name, "wb" );
    bit_file->rack = 0;
    bit_file->mask = 0x80;
    bit_file->pacifier_counter = 0;
    return( bit_file );
}

BIT_FILE *OpenInputBitFile( name )
char *name;
{
    BIT_FILE *bit_file;

    bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
    if ( bit_file == NULL )
        return( bit_file );
    bit_file->file = fopen( name, "rb" );
    bit_file->rack = 0;
    bit_file->mask = 0x80;
    bit_file->pacifier_counter = 0;
    return( bit_file );
}

void CloseOutputBitFile( bit_file )
BIT_FILE *bit_file;
{
    if ( bit_file->mask != 0x80 )
        if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
            fatal_error( "Fatal error in CloseBitFile!\n" );
    fclose( bit_file->file );
    free( (char *) bit_file );
}

void CloseInputBitFile( bit_file )
BIT_FILE *bit_file;

```

```

{
    fclose( bit_file->file );
    free( (char *) bit_file );
}

void OutputBit( bit_file, bit )
BIT_FILE *bit_file;
int bit;
{
    if ( bit )
        bit_file->rack |= bit_file->mask;
    bit_file->mask >>= 1;
    if ( bit_file->mask == 0 ) {
        if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
            fatal_error( "Fatal error in OutputBit!\n" );
        else
            if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                putc( '.', stdout );
        bit_file->rack = 0;
        bit_file->mask = 0x80;
    }
}

void OutputBits( bit_file, code, count )
BIT_FILE *bit_file;
unsigned long code;
int count;
{
    unsigned long mask;

    mask = 1L << ( count - 1 );
    if ( code == 257 )
        putc( bit_file->rack, bit_file->file );
    else {
        while ( mask != 0 ) {
            if ( mask & code )
                bit_file->rack |= bit_file->mask;
            bit_file->mask >>= 1;
            if ( bit_file->mask == 0 ) {
                if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
                    fatal_error( "Fatal error in OutputBit!\n" );
                else if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                    putc( '.', stdout );
                bit_file->rack = 0;
                bit_file->mask = 0x80;
            }
            mask >>= 1;
        }
    }
}

int InputBit( bit_file )
BIT_FILE *bit_file;
{
    int value;

```

```

    if ( bit_file->mask == 0x80 ) {
        bit_file->rack = getc( bit_file->file );
        if ( bit_file->rack == EOF )
            fatal_error( "Fatal error in InputBit!\n" );
        if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
            putc( '.', stdout );
    }
    value = bit_file->rack & bit_file->mask;
    bit_file->mask >>= 1;
    if ( bit_file->mask == 0 )
        bit_file->mask = 0x80;
    return( value ? 1 : 0 );
}

unsigned long InputBits( bit_file, bit_count )
BIT_FILE *bit_file;
int bit_count;
{
    unsigned long mask;
    unsigned long return_value;

    /*
        if ( bit_count == 75 ) {
            if ( bit_file->mask == 0x80 )
                bit_file->mask = 0x02;
            else
                if ( bit_file->mask == 0x40 )
                    bit_file->mask = 0x01;
                else
                    if ( bit_file->mask <= 0x20 )
                        bit_file->mask <<= 2;
            if ( bit_file->mask == 0x80 )
                flag = 1;
        }
    */
    mask = 1L << ( bit_count - 1 );
    return_value = 0;
    while ( mask != 0 ) {
        if ( bit_file->mask == 0x80 ) {
            bit_file->rack = getc( bit_file->file );
            if ( bit_file->rack == EOF )
                fatal_error( "Fatal error in InputBit!\n" );
            if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                putc( '.', stdout );
        }
        if ( bit_file->rack & bit_file->mask )
            return_value |= mask;
        mask >>= 1;
        bit_file->mask >>= 1;
        if ( bit_file->mask == 0 )
            bit_file->mask = 0x80;
    }
    return( return_value );
}

```

```
void FilePrintBinary( file, code, bits )
FILE *file;
unsigned int code;
int bits;
{
    unsigned int mask;

    mask = 1 << ( bits - 1 );
    while ( mask != 0 ) {
        if ( code & mask )
            fputc( '1', file );
        else
            fputc( '0', file );
        mask >>= 1;
    }
}
```

```
/***** End of BITIO.C *****/
```



```

/***** Start of BITIO.H *****/

#ifndef _BITIO_H
#define _BITIO_H

#include <stdio.h>

typedef struct bit_file {
    FILE *file;
    unsigned char mask;
    int rack;
    int pacifier_counter;
} BIT_FILE;

#ifdef __STDC__

BIT_FILE *OpenInputBitFile( char *name );
BIT_FILE *OpenOutputBitFile( char *name );
void OutputBit( BIT_FILE *bit_file, int bit );
void OutputBits( BIT_FILE *bit_file,
                 unsigned long code, int count );
void OutTwoBits( BIT_FILE *bit_file,
                 unsigned long code, int count );
int InputBit( BIT_FILE *bit_file );
unsigned long InputBits( BIT_FILE *bit_file, int bit_count );
void CloseInputBitFile( BIT_FILE *bit_file );
void CloseOutputBitFile( BIT_FILE *bit_file );
void FilePrintBinary( FILE *file, unsigned int code, int bits );

#else /* __STDC__ */

BIT_FILE *OpenInputBitFile();
BIT_FILE *OpenOutputBitFile();
void OutputBit();
void OutputBits();
void OutTwoBits();
int InputBit();
unsigned long InputBits();
void CloseInputBitFile();
void CloseOutputBitFile();
void FilePrintBinary();

#endif /* __STDC__ */

#endif /* _BITIO_H */

/***** End of BITIO.H *****/

```

```
/****** Start of ERRHAND.C *****/
*
* This is a general purpose error handler used with every program in
* the book.
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "errhand.h"

#ifdef __STDC__
void fatal_error( char *fmt, ... )
#else
#ifdef __UNIX__
void fatal_error( fmt, va_alist )
char *fmt;
va_dcl
#else
void fatal_error( fmt )
char *fmt;
#endif
#endif
{
    va_list argptr;

    va_start( argptr, fmt );
    printf( "Fatal error: " );
    vprintf( fmt, argptr );
    va_end( argptr );
    exit( -1 );
}

/****** End of ERRHAND.C *****/
```

```
/****** Start of ERRHAND.H *****/

#ifndef _ERRHAND_H
#define _ERRHAND_H

#ifdef __STDC__
void fatal_error( char *fmt, ... );
#else /* __STDC__ */
void fatal_error();
#endif /* __STDC__ */
#endif /* _ERRHAND_H */

/****** End of ERRHAND.H *****/
```

LIST OF REFERENCES

- [1] Rabbani, M. and Jones, P.W., *Digital Image Compression Techniques*, SPIE Optical Engineering Press, Bellingham, WA, 1991.
- [2] Nelson, M., *The Data Compression Book*, M&T Publishing, Inc., San Mateo, CA, 1992.
- [3] Gonzalez, R.C. and Wintz, P., *Digital Image Processing*, 2nd ed., Addison-Wesley Publishing Co., Inc., Reading, MA, 1987.
- [4] Hannah, S.J., *A Hybrid Encoding Technique for Lossless Fractal Compression*, Master's Thesis, University of Alabama, September, 1993.
- [5] Takahashi, K. and Ohta, M., "Data Compression Coding of Gray-Scale Images using Bit Planes," *Proceedings of ICC*, v 2.3.1, pp. 34-41, 1985.
- [6] Weiss, J. and Schremp, D., "Putting Data on a Diet," *IEEE Spectrum*, pp. 36-39, August, 1993.
- [7] Novik, D.A., Tilton, J.C., and Manohar, M., "Compression through Decomposition into Browse and Residual Images," *The 1993 Space and Earth Science Data Compression Workshop*, pp. 7-12, April, 1993.
- [8] Wallace, G. K., "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, v. 38, pp. xviii-xxxiv, February, 1992.
- [9] Hung, A.C., PVRG-JPEG Codec 1.1, (Computer Program Documentation), Portable Video Research Group, Stanford University, August, 1993.
- [10] Jackson, J.J. and Hannah, S.J., "Comparative Analysis of Image Compression Techniques," *The 25th Southeastern Symposium on System Theory*, pp. 513-517, 1993.
- [11] Kay, R.T., *A Comparison of some of the Most Current Methods of Image Compression*, Master's Thesis, Naval Postgraduate School, Monterey, CA June, 1993.
- [12] Mowle, F.J., *A Systematic Approach to Digital Logic Design*, p. 9, Addison-Wesley Publishing Co., Inc., Reading, MA, 1976.

INITIAL DISTRIBUTION LIST

- | | | |
|----|---|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 52
Naval Postgraduate School
Monterey, CA 93943-5002 | 2 |
| 3. | Chairman, Code EC
Department of Electrical and Computer
Engineering
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 4. | Professor Ron J. Pieper, Code EC/Pr
Department of Electrical and Computer
Engineering
Naval Postgraduate School
Monterey, CA 93943 | 2 |
| 5. | Professor Murali Tummala, Code EC/Tu
Department of Electrical and Computer
Engineering
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 6. | Dan Jensen, Code EC/E1
Department of Electrical and Computer
Engineering
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 7. | Walter D. Abbott III, LT, USN
805 Radcliff Drive
Lynn Haven, FL 32444 | 2 |