Model reduction and predictor selection in image compression
by Joel Eugene Henry

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University

Abstract:
Image compression systems rely heavily on prediction and accurate model generation. Previous research has largely ignored the effect of prediction upon the amount of available compression. Ten increasingly sophisticated prediction techniques were implemented and tested. The results indicated that increased compression could be obtained from two of these methods. The testing illustrated the importance of effective prediction in a compression system. Poor prediction technqiues reduce available compression, sometimes dramatically.

An ideal model which captures the frequency of the events to be encoded is excessively large. An effective, manageable model can be obtained through the use of equivalence classes or buckets. The entire set of possible conditioning states are grouped into buckets. These buckets form the new set of conditioning states. Bucketing is also performed on the error range of each conditioning class. In this way the number of contexts and the number of error parameters can be reduced. Bucketing produced effective compression while controlling the size of the model. Increasing the number of error buckets increased compression. The increased model file overhead to obtain better compression was significant.

\

MODEL REDUCTION AND PREDICTOR SELECTION IN IMAGE COMPRESSION

by

Joel Eugene Henry

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

August 1986

APPROVAL

of a thesis submitted by

Joel Eugene Henry

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

August 1, 1986
Date

Chairperson, Graduate Committee

Approved for the Major Department

August 1st, 1986
Date

J. Dunbigh Stanley
Head, Major Department

Approved for the College of Graduate Studies

August 7, 1986
Date

Henry L. Parsons
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his/her absence, by the Director of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Joel E. Henry

Date July 31, 1986

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Image compression systems rely heavily on prediction and accurate model generation. Previous research has largely ignored the effect of prediction upon the amount of available compression. Ten increasingly sophisticated prediction techniques were implemented and tested. The results indicated that increased compression could be obtained from two of these methods. The testing illustrated the importance of effective prediction in a compression system. Poor prediction technqiues reduce available compression, sometimes dramatically.

An ideal model which captures the frequency of the events to be encoded is excessively large. An effective, manageable model can be obtained through the use of equivalence classes or buckets. The entire set of possible conditioning states are grouped into buckets. These buckets form the new set of conditioning states. Bucketing is also performed on the error range of each conditioning class. In this way the number of contexts and the number of error parameters can be reduced. Bucketing produced effective compression while controlling the size of the model. Increasing the number of error buckets increased compression. The increased model file overhead to obtain better compression was significant.

Chapter 1

INTRODUCTION TO DATA COMPRESSION

Data Compression - A Definition

Data compression can be defined as the process of converting digital information into a form of reduced size. Compression is commonly achieved through the use of an encoding scheme which capitalizes upon the redundancies in the data. Data compression systems are often used to transmit encoded data across communication links where decoding takes place [1].

A Theoretical Basis for Data Compression

The encoding algorithms used in data compression systems rely upon accurate frequency measures of the events they encode. An event is a value, symbol or a group of symbols to be encoded. The relation of events to their probabilities is the fundamental concept upon which data compression is built. A simple coding technique which demonstrates this concept is one which assigns short code words to those source symbols with high probabilities and long code words to those symbols with low probabilities [2].

Consider Table 1 which demonstrates a simple coding scheme for conveying weather information. The average length of a code word in Table 1 is two bits if the information is conveyed using binary digital signals.

Table 1. Weather code.

Weather	Code
sunny	00
cloudy	01
rainy	10
snowy	11

Suppose the probability of each weather event is know, as shown in Table 2.

Table 2. Compressed weather code.

Weather	Probability	New Code
sunny	.50	0
cloudy	.25	10
rainy	.125	110
snowy	.125	1110

The new code produces an average code word length of 1.875 bits, calculated as:

$$L = 1 \times \text{probability(sunny)} + 2 \times \text{probability(cloudy)} + 3 \times \text{probability(rainy)} + 3 \times \text{probability(snowy)}.$$

The new coding method is able to communicate information using fewer bits per message, on the average.

Events and their corresponding probabilities can be related more formally as follows:

Let E be some event which has a probability P(E). Each time event E occurs we can say we have received:

$$I(E) = \log (1/P(E)) \qquad (1.1)$$

bits of information [2].

The quantity, I(E), is significant in that a measure of the information provided by an event(source string) can now be calculated. I(E) can be used to define an important measure in the field of data compression.

Let a source alphabet be defined by,

$$S = \{ s_1, s_2, \ldots, s_n \}.$$

Suppose the probabilities of the source symbols are given by (s_i). The average amount of information per source symbol from source alphabet S, denoted H(S), can be calculated by equation 1.2.

$$H(S) = P(s_i) \times I(s_i) \qquad (1.2)$$

This quantity is called the entropy, and is calculated for a source alphabet or any alphabet to be encoded [2].

The interpretation of entropy, which is most important to data compression, is provided by Shannon's first theorem. This fundamental theorem states that we can make the average number of binary code symbols per source symbol as small as, but no smaller than, the entropy of the source alphabet measured in bits. Entropy can, therefore, be viewed as a upper bound on the amount of available compression. H(S) is the best compression that can be achieved, on the average, when compressing a string of symbols from source alphabet S.

Shannon's first theorem is based on two major premises, the Kraft inequality and code word lengths chosen in a specific manner. The Kraft inequality can be stated as:

A necessary and sufficient condition for the existence of an instantaneous code with code words lengths 1_1, 1_2, ...,1_n is that $2^{-1} <= 1$ for a binary code alphabet [2].

An instantaneous code is a code which allows the decoding of source strings without the need to inspect succeeding code words. The importance of this type of code will be shown later. The Kraft inequality is a critical assumption in the proof of Shannon's first theorem and establishes a necessary and sufficient condition for the existence of an instantaneous coding scheme.

Shannon's first theorem assumes that code word lengths are chosen in a specific way. Let a code word i, with length 1_i, be selected by,

$$\log_2 (1/P(s_i)) <= \log_2 (1/P(s_i)) + 1 \qquad (1.3)$$

In effect 1_i equals $\log_2 (1/P(s_i))$ if this value is an integer, otherwise select the next greatest integer.

Choosing code word lengths in this manner produces a code which satisfies the Kraft inequality. This can be shown by considering the left inequality of equation 1.3.

$$\log_2 (1/P(s_i) <= 1_i \qquad (1.4)$$

Taking exponentials of both sides of the equation, then inverting.

$$P(s_i) >= 2^{-1i} \qquad (1.5)$$

Summing equation 1.5 over all possible values of i,

$$1 >= 2^{-1i} \qquad (1.6)$$

Equation 1.3 therefore defines an acceptable set of code lengths for an instantaneous code.

If equation 1.3 is multiplied by $P(s_i)$ and summed over all i, the following inequality relationship results:

$$P(s_i) \times \log^2 (1/P(s_i)) <= P(s_i)l_i <= P(s_i) \times \log_2 (1/P(s_i)) + P(s_i) \qquad (1.7)$$

Equation 1.7 can be rewritten as,

$$H(S) <= L <= H(S) + 1. \qquad (1.8)$$

Equation 1.8 is a version of Shannon's first theorem [2]. This theorem proves that we can make the average number of binary code symbols per source symbol as small as, but no smaller than, the entropy of the source alphabet. Entropy defines the best possible compression, on the average, attainable by a coding scheme relying on the set of probabilities defined for the source symbols.

Entropy is used to evaluate the set of probabilities used by an encoding algorithm. This measure establishes a upper bound for the best average compression possible. Entropy calculations are commonly used to evaluate and compare the compression attainable by a differing groups of source string probabilities [3].

The Goals of a Compression System

The goal of a compression system is to transform data into an encoded form of reduced size. This encoded data is then typically transmitted to a destination and decoded into its original form. Most

compression techniques use a model which makes assumptions about the data and the events to be encoded.

A compression system may encode source strings or predict the succeeding value and encode the error. Typically the data to be compressed drives a model builder which determines the relative frequency of the events to be encoded. The encoder accepts these events and their probability measures in order to generate an encoded file. A compression system should be capable of allowing changes in the modeling technique without affecting the encoding and decoding methods. The encoding and decoding schemes should not rely on any specific modeling method [1].

Coding techniques require accurate event probabilities to produce compressed data encodings. The primary concern of any compression system is accurate, concise model formation [4]. The probabilities obtained in the modeling process must reflect the redundancies in the data in order for compression to take place.

Components of an Image Compression System

A data compression system, which uses digital grey-scale images as input, is referred to as an image compression system. These systems, which use a prediction process and encode the errors, commonly have the following components:

1. Predictor
2. Modeler
3. Encoder
4. Decoder
5. Unpredictor

Each component accomplishes a specific task and passes critical information to other components as shown in Figure 1 [5].

Components can be combined where needed for optimization or to implement a specific type of system. The modeler should place no restrictions on or be restricted by the encoder or decoder routines. The encoder is concerned only with events and their probabilities, not with how the frequency measures were obtained. The modeler need only generate probabilities and should be oblivious to the encoding method.

Predictor

The prediction process can be described as the technique which attempts to approximate a succeeding value from some combination of previously scanned values. This process is used to produce a file of errors with a greater amount of redundancy than the original image file. The error file, if predicted properly, will have a large number of values clustered around zero.

An image file is input to the prediction component. This image file is considered to be made up of picture element(pixel) values. The output of a prediction component is a file of errors and a file of statistics describing the error file.

The gathering of pertinent statistical information is the second task performed by the prediction component. These statistics may be frequency counts, mean, variance, covariance, or any other measure which may be needed by the modeling process.

The primary goal of prediction is to produce a collection of errors with small absolute values that cluster around zero. An error

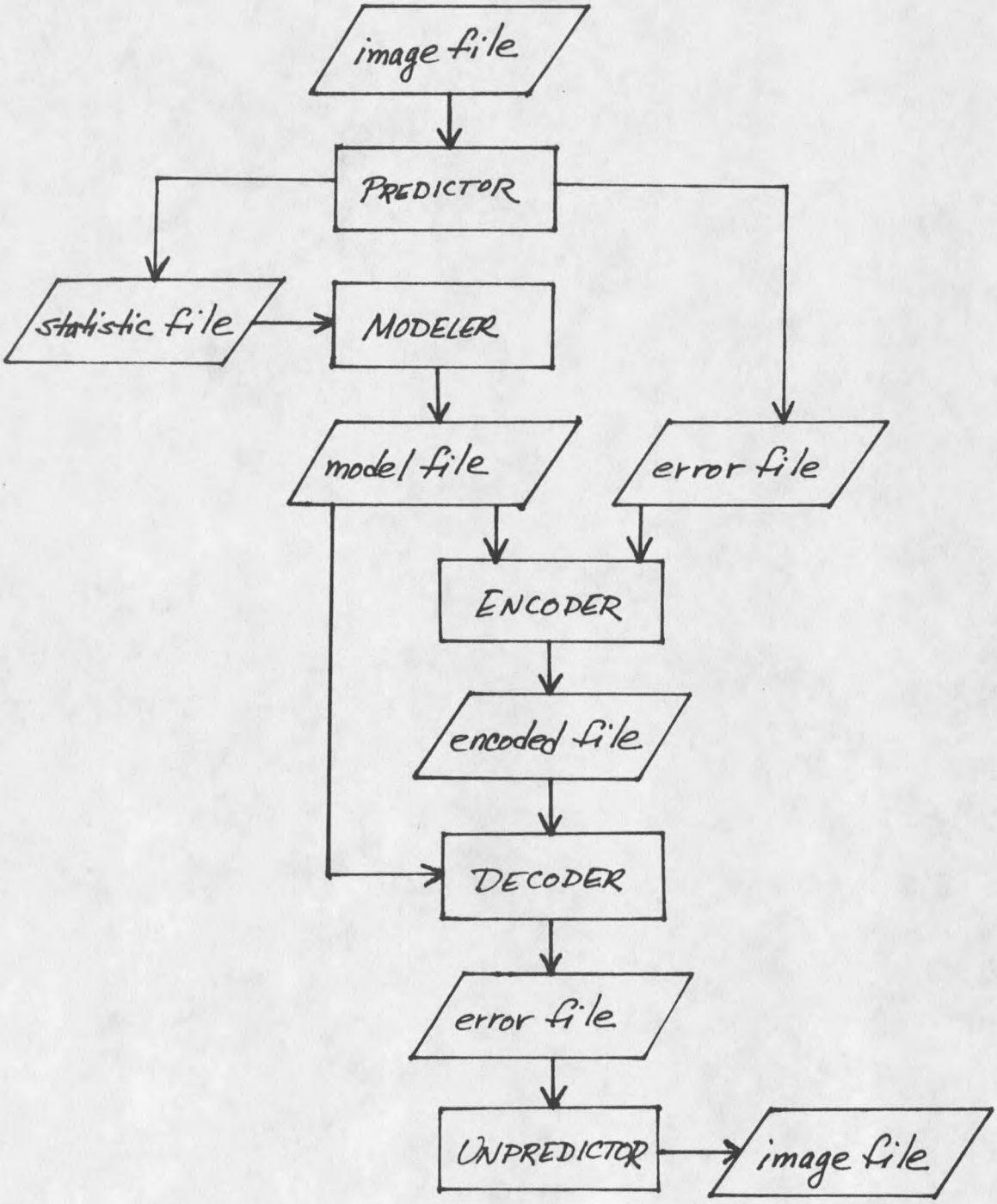

Figure 1. Image compression system components.

file with this property has a great deal of redundancy and therefore can be compressed effectively.

Modeler

The modeling process is the most important component of a data compression system [4]. It is this component which establishes the frequency measures used in the encoding process. The compression achieved by a system depends directly upon accurate event probabilities.

The modeling component uses an input file of error file statistics to produce a model file of error frequency measures. The model must provide a probability for each error value to be encoded. The model should provide probabilities for each error, regardless of the distribution of the errors.

A model consists of two parts: 1) the structure which is the set of source strings(errors) and their conditioning state or context, and 2) the parameters or coding probabilities assigned to the source strings [4]. The structure is used to capture the redundant characteristics of an error file. The parameters specify the individual probabilities of each error value given its context.

The model file, output by the modeling component, contains the probabilities of each error value or the necessary values which can be combined to form the error probability. It is accessed by the encoding and decoding components and is necessary for the execution of both components.

Encoder

The encoding process encodes an error using it's individual probability. The encoding technique produces a compressed file when provided accurate frequency measures.

The encoding component uses a model file of error probabilities and a file of errors as input. The output of this component is an encoded file which is smaller in size than both the original data file and the error file. The encoded file must be uniquely decodable in order for the compression system to provide lossless compression. The encoding process is of the first-in first-out type, meaning the first error encoded is the first error decoded. The encoding method should be instantaneous in order for the decoder to decode an error without inspection of succeeding symbols.

Decoder

The decoding process decodes a single error from the encoded file through the use of error probabilities. The errors are decoded sequentially as they were input. The decoder must determine the context of the encoded error in order to decode correctly. If the decoder must inspect succeeding symbols whose contexts are not decoded, it will be unable to uniquely decode the encoded file. This is why an instantaneous coding scheme is required.

The decoding component accepts an encoded file and a model file as input. Decoding is accomplished through the use of the probabilities in the model file, producing an output file of errors.

Unpredictor

The unprediction process sums the error and the combination of preceeding values, which form the predicted value, in order to obtain the original pixel value. This process is done sequentially and should produce the original image file.

The unprediction component uses an input file of errors and produces an output file which must be identical to the original image. This is the requirement of a lossless compression system.

Chapter 2

PREDICTOR TESTING AND MODEL REDUCTION

Prediction Effects on Image Compression - An Unanswered Question

Prediction is the process which produces an error file for encoding. The contents and structure of this file depend completely on the prediction technique used. This technique combines previous values to predict the next value in the image file while gathering context and error statistics.

Few prediction methods appear in the literature, in reference to image compression. Commonly, modeling and encoding algorithms have received the majority of research attention. These compression systems have made use of simple prediction techniques while concentrating on model formation or encoding/decoding schemes [3].

Modeling and encoding/decoding procedures are critical parts of a data compression system. These two components establish error probabilities and perform the actual compression, respectively. The prediction process is also important. It is this component which produces an error file with some amount of redundancy. The error distribution affects the amount of compression obtainable.

Specific Prediction Questions

The questions concerning prediction, which this thesis seeks to answer, can be stated as follows:

1. How much compression can be obtained through the use of more sophisticated prediction techniques than those appearing in the literature?
2. To what extent is compression affected by the prediction process?
3. How important is the prediction step in a image compression system?

These questions are important to the study of image compression. Question number three can be answered by the solutions to questions one and two. Question three will answer the question of how much emphasis should be placed on the prediction component of a compression system. The significance of the prediction method and its performance will also be known.

Prediction Answers Through Predictor Testing

Answers to the previously stated questions can be found by investigating the compression results of a number of different, increasingly sophisticated predictors. In order to compare prediction methods, a measure must be used which is calculated without regard to the predictor used. The entropy H(S) developed in Chapter 1 is such a measure. Entropy is a measurement which is used with probability establishing models. Predictors, therefore, will be compared by calculating the entropy of the error files they produce.

Each predictor must be used with the same modeling routine, to prevent model influence on the results. A standard modeling routine

will be used by each prediction method. This modeling process will be described later in this chapter.

The predictors implemented and tested are of five types: linear, planar, Taylor series, least mean square, and differential pulse code modulation. These types of predictors are increasingly complex and computationally expensive to implement. Fourteen distinct predictors will be used for comparison.

These predictors are as follows:

1. horizontal
2. vertical
3. planar type one
4. planar type two
5. Taylor series
    a) first order
        (1) using zeroth derivative
        (2) using first derivative
        (3) using second derivative
        (4) using third derivative
        (5) using fourth derivative
    b) second order
        (1) using first derivative
        (2) using second derivative
6. least mean square
7. differential pulse code modulation
    a) first order
    b) second order

The results of these prediction techniques will indicate the effect of prediction on compression results. Each of the five types is significantly different enough to allow technique comparison. The seven Taylor series predictors will be used to study the effects of using an increasing number of surrounding pixels.

The prediction techniques will be judged as to their compression results in search of an optimal method. The overall results of all predictors will be used to determine the effects of prediction on compression results. These two comparisons will indicate the importance of the prediction component in a data compression system.

The Modeling Problem

Modeling is the process of defining error probabilities. These error probabilities make up a model file which is used by both the encoder and decoder. Defining the conditioning state and calculating the coding parameters are the tasks which are performed in the modeling process.

The conditioning state or context determines the probability of an error. The context's role in error frequency determination is to condition the error value. The coding parameters produced by the modeling process are the probabilities of an error given a context, represented by P(e/w).

A context is often defined as the state of the surrounding values [6]. This type of context definition assumes that the current error is dependent on the magnitude of some of the surrounding pixel values. An assumption of this kind is valid for some types of input data -

grey-scales images for example. Each of the possible contexts defines an error distribution. These error distributions cover the entire extent of possible errors.

The modeling problem can now be described as the problem of defining an accurate model whose size is not an excessive overhead to the compression system. Stated more simply, a model file is needed which contains coding parameters very close to the actual probabilities and is of reasonable, manageable size.

Consider an example of the problem as follows: 1) Suppose three surrounding values are used as contexts, each value is between 0 and 255 and 2) the error range is -255 to +255. This example has 256^3, or 16,777,216 contexts. Each of these contexts defines an error range of -255 to +255 or 511 possible errors. If frequency counts are kept throughout the prediction process, 8,573,157,376 such counts may be needed.

The model described in the previous example is the ideal, complete model. Such a model would provide the true conditional probabilities needed by the encoder. The size of this model is prohibitive. In fact, if the original data file contained 500,000 values, the model file would have 17,146 times more probabilities. This is far too many coding parameters to be stored and/or transmitted effectively. The problem is to reduce the size of the model and therefore, the number of coding parameters, while continuing to achieve effective compression.

Model Reduction

The example in the previous section could be considered to be that of an image compression system. The input data would be an image file of eight bit pixel values (0-255). The error file would contain exactly the same number of values, with value range -255 to +255.

There are actually two problems in this example, too many contexts and too many parameters [3]. In order to reduce the number of coding parameters to be passed to the encoder and decoder routines, the number of contexts and the number of parameters must be reduced.

The number of contexts can be reduced by defining equivalence classes or buckets [3]. These buckets subdivide the total number of contexts into a new set of conditioning states or contexts. These new contexts are made up of a bucket of original contexts, which can be used to condition the error. Figure 2 is a conceptual representation of the bucketing technique using a tree type structure.

Consider the three surrounding pixels used to determine the context of an error to be point1, point2, and point3. Level one represents all possible values of point1. Levels two and three represent all possible combinations of values of two points and three points respectively.

The bucketing technique can reduce the total number of contexts to an arbitrary number. If frequency counts are used to establish coding parameters, the context bucket count is equal to the sum of the frequency counts of each original context. A frequency count only need be kept for the new bucketed context if equation 2.1 is used.

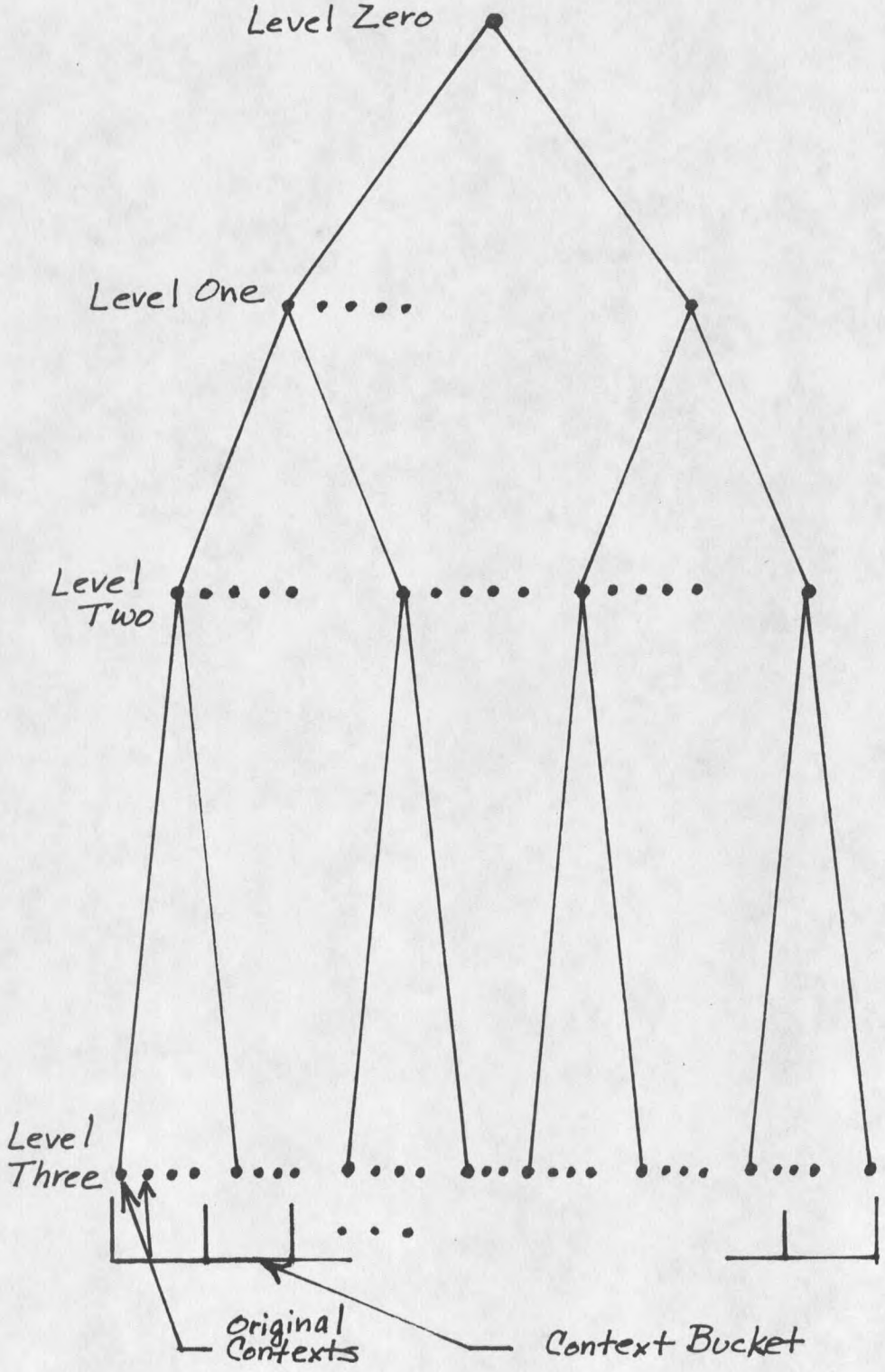

Figure 2. Conceptual context bucketing.

$$\text{new context number} = \frac{(\text{point1} \times 255^2 + \text{point2} \times 255 + \text{point3})}{\text{number of old contexts per new context}} \quad (2.1)$$

Equation 2.1 allows direct access to the correct context bucket at the leaf level(level three) of Figure 2.

Reducing the number of contexts greatly reduces the model size but an error range still exists for each new context. The model is still very large if frequency counts are used for each of the 511 possible errors for each of the new contexts. This means that if 10 context buckets are used per context pixel, resulting in only 1000 new contexts, there still exist 511,000 coding parameters.

This number can be further reduced by using the bucketing technique and making a single simplifying assumption [3]. The bucketing technique is used in the same manner as in the context reduction step. Define some number of error buckets as equivalence classes of the error range. Let each bucket contain roughly the same number of errors. Instead of predicting on each individual error, predict only the bucket in which the error falls. This reduces the 511,000 coding parameters to 11,000 if 11 error buckets are used. The individual error probabilities are now lost, bringing about the need for a critical simplifying assumption.

Suppose it is assumed that the error value within the bucket is independent of the context. This assumption allows us to keep frequency counts on a single error distribution. This distribution is divided by the bucket technique, and the distributions of interest are actually within each error bucket. The total number of coding

parameters has now been reduced from 8,573,157,376 in the original, ideal model to 11,511 in the bucketed model.

The conditional error probabilities needed by the encoding/decoding routines can be obtained by using equation 2.2, where b(e) represents the error bucket in which the error falls.

$$P(e/w) = P(b(e)/w) \times P(e/b(e)) \times P(w) \qquad (2.2)$$

The evaluation of this model reduction technique will be done using the entropy measure. The per pixel entropy is calculated by equation (2.3), where N is the total number of data values.

$$H(S) = (1/N) \times [I(e/b) + I(b/w)] \qquad (2.3)$$

The quantity I(b/w) is the error bucket contribution to the N-value data file entropy while I(e/b) is the contribution from the error within the bucket. A complete derivation of equation (2.3) from equation (1.2) is contained in the appendix.

Model Parameter Effects

The model parameters, the number of error buckets, and the number of context buckets per context pixel, have an effect on the available compression. The available compression produced by increasing the number of error buckets will be tested. Increased compression will be considered with respect to the increased size of the model file. The additional overhead of a larger model is the cost of achieving better compression.

Increasing the number of context buckets should provide better compression, since the model expands rapidly, approaching the ideal model. Consider five context buckets for each of three context pixels. The total number of contexts is then 125. If the number of context buckets is increased to six, the new total number of contexts becomes 216. Furthermore, each additional context requires a full set of error bucket counts be kept.

This rapid expansion of model size prevents the number of context buckets per context pixel from increasing above approximately 11. A model with 11 context buckets per context pixel has 1331 total contexts. If only five error buckets are used, the model will have 6655 values. Increasing the number of context buckets to 14 will result in 13720 values, more than double the 11 context bucket size. For this reason the number of contexts per context pixel will be used to retain a reasonably sized model, where possible.

Chapter 3

PREDICTION AND MODELING IMPLEMENTATION

Predictor/Modeler Implementation

The prediction process attempts to approximate the next pixel value using a combination of previous pixel values while gathering the statistics needed to produce a model file. An error file is produced by the predictor containing an error for each value in the image file. A software implementation of this process must accomplish all these tasks.

A prediction program should be efficient, memory conservative where possible, and execute as quickly as possible. The implementation language must satisfy these needs and solve other software-specific prediction tasks. Before choosing the implementation language, the design of the predictor was completed. The design outlined the following characteristics of the prediction process.

1. Prediction is highly input/output intensive, for each predicted value, the nearby points must be obtained for use in the prediction formula.
2. Bit manipulations are needed to output only as many bits as the error requires (prediction on eight bit quantities, 0-255, produces nine bit errors, -255 - +255).

3. Dynamic data structures need to be used to keep the frequency counts for a user-specified model size.

The modeler uses frequency counts to determine error probabilities. These counts are provided by the predictor using the model structure. The error probabilities are calculated and output in their final form by the modeler. These probabilities need to be calculated accurately and efficiently, at the minimal computation cost. The output model file should be accurate and concise to produce a reasonably sized file of coding parameters.

The modeler requires an implementation that will allow fast access to the frequency count and that will allow the probabilities to be output in their binary form.

The designed program and the previously stated characteristics led to the choice of 'C' as the implementation language. The target computer for the predictor/modeler routine was a Digital Equipment Corporation Microvax II with VMS operating system.

Predictor/Modeler Design

The overall predictor/modeler design required two passes through an image file. The first pass must be made to define the error bucket extents such that each error bucket contained roughly the same number of errors. The second pass produced frequency counts, calculated and output the coding parameters, and calculated the entropy of the error file.

The inputs to the predictor/modeler program are:

1. an image file

2. the horizontal and vertical resolution sizes of the image
3. the total number of error buckets to be used
4. the number of context buckets per context point
5. the prediction selector

The outputs of the predictor/modeler are:

1. an error file of nine bit prediction errors
2. a model file of coding parameters

A study of the input, output and model structure produced a high level list of tasks to be accomplished and a sequence of task execution. This list is as follows:

1. Create the data structures for the frequency counts needed.
2. Perform the first pass prediction, defining the error bucket extents.
3. Perform the second pass prediction, which accomplishes the following:
   a) determine each error's context, and the error bucket to which it belongs
   b) update the frequency counts
   c) output a nine bit error quantity
4. Generate the model file by using the frequency counts to establish error probabilities.
5. Calculate the entropy of the error file.

This list is the top-level algorithm of the predictor/modeler software. Implementation of each of the five steps can now be discussed.

Data Structure Implementation

There are a number of data structures needed to speed up execution or keep the necessary frequency counts. The data structure which allows faster execution holds an entire image, taking advantage of the virtual memory management system of the operating system.

The vertical and horizontal resolutions specify the size of the input image in bytes. Once this size is determined it is possible to request a block of memory from the operating system at execution time which will contain the entire image. The image is then read into this block of memory by a single read statement. The operating system fills the block of memory with large blocks of the image much faster than any other way available to the programmer. The virtual memory management system of the machine is then used to access the image, giving the appearance of having the entire image in main memory. The advantage of this approach over a seek and read for each pixel is obvious; only block accesses are made to secondary storage.

A data structure is created at run time to maintain frequency counts for each context and each error bucket within a context. The size of this structure is defined by the error bucket and context bucket parameters specified by the user. Figure 3 illustrates a conceptual model of this data structure.

This data structure, referred to as the context array, keeps frequency counts for the number of occurrences of each new context and each bucket given a context. Conceptual column zero maintains a frequency count of the number of times each context occurs. Columns

not through 'number of error buckets' are frequency counts of the number of occurrences of each bucket, given a particular context.

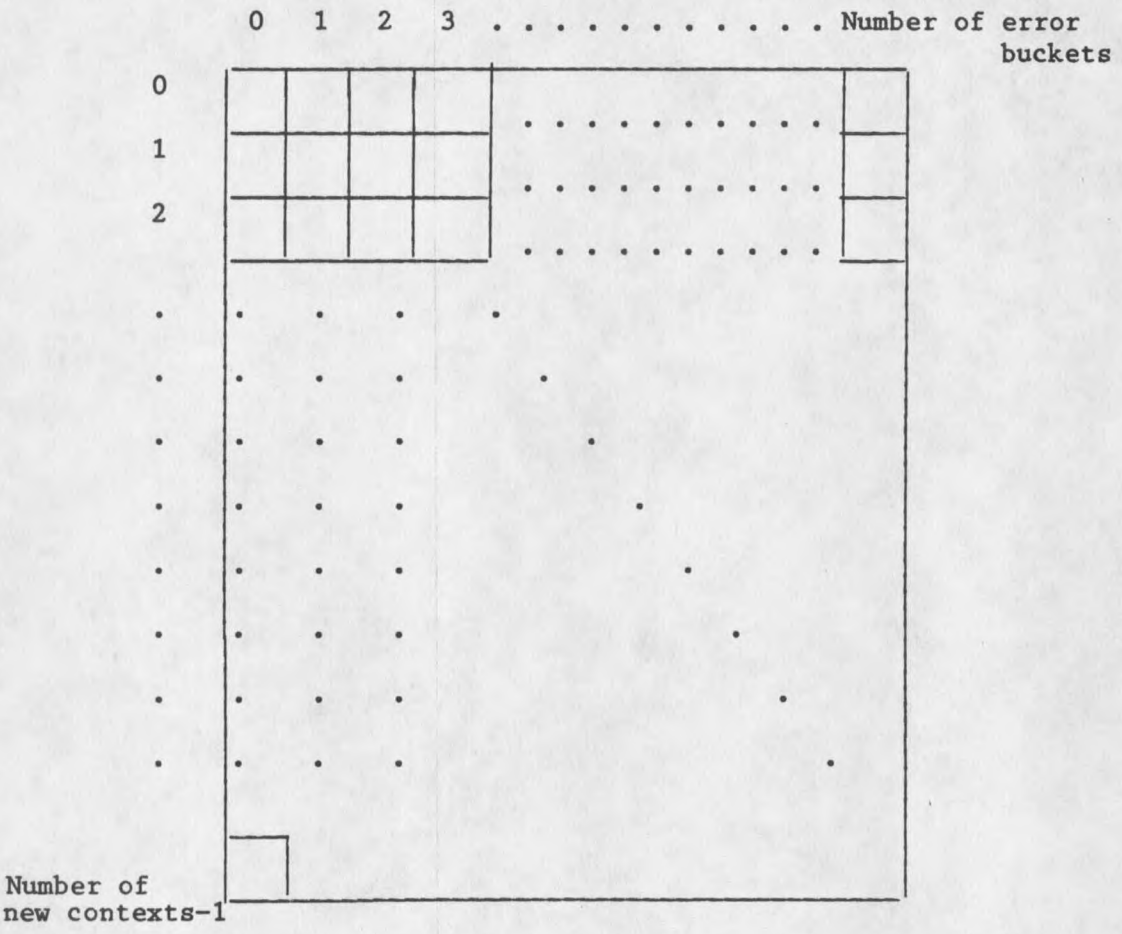

Figure 3. Context array.

Error array is the third data structure created dynamically. This structure contains the error extents of, and the total number of errors within, each error bucket. Error array can be thought of conceptually as appearing as in Figure 4. Rows zero through 'number of error buckets^{-1}' refer to each error bucket. Column zero contains

the number of errors within each bucket. Column one holds the lowest valued error within each bucket. Column two contains the highest valued error within each bucket.

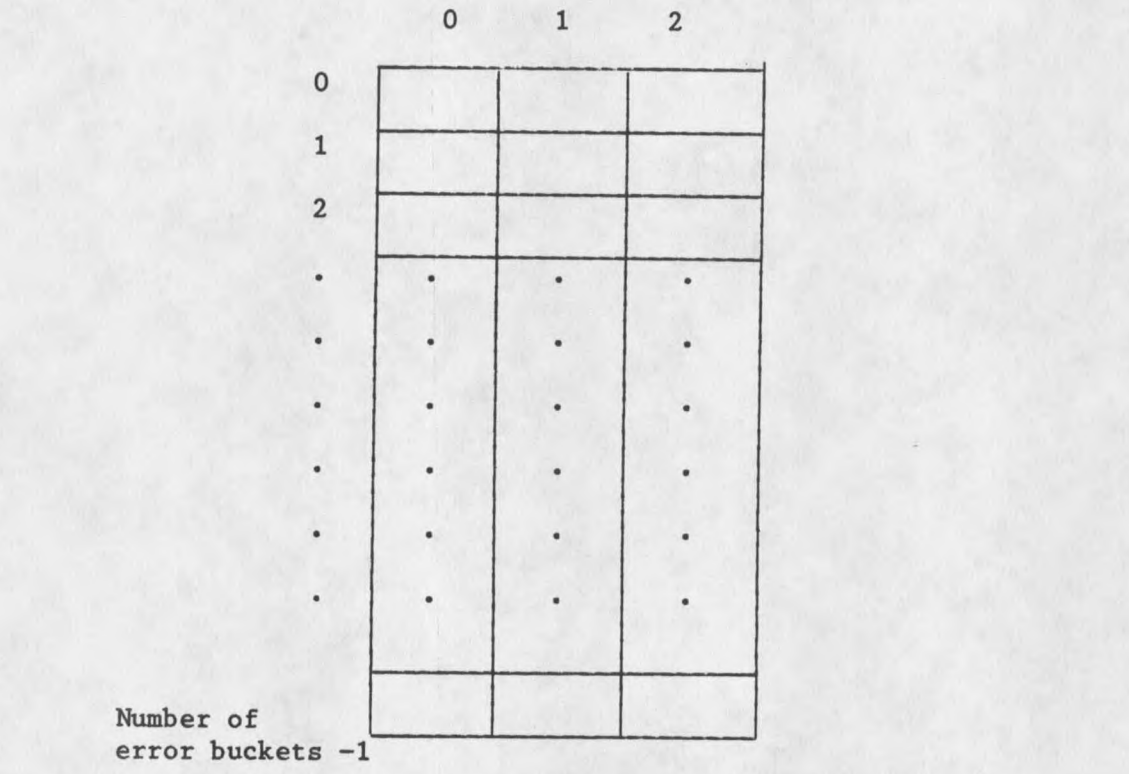

Figure 4. Error array.

A static data structure, called error counts, is used to hold the frequency counts of each error value. This structure can be static because the error range produced by eight bit pixel values in prediction will always be -255 to +255.

These data structures are required by the choice of model structure. The values must be provided by the predictor as they are only known at the time an error occurs.

First Pass Implementation

The first pass prediction through the image file is needed to define error bucket extents. This pass performs a prediction of each pixel value in the image file. The prediction technique is chosen by the user, specified by a number between one and fourteen. The number, referred to as prediction selector, specifies a prediction method as shown in Table 3. The implementation of each prediction method will now be explained.

Table 3. Prediction selector description.

Prediction Selector	Prediction Method
1	planer one
2	planer two
3	horizontal
4	vertical
5	Taylor series, first order, zero derivatives
6	Taylor series, first order, one derivative
7	Taylor series, first order, two derivatives
8	Taylor series, first order, three derivatives
9	Taylor series, first order, four derivatives
10	Taylor series, second order, one derivative
11	Taylor series, second order, two derivatives
12	Least mean square
13	DPCM, first order
14	DPCM, second order

Planar Predictors

Prediction selectors one and two select planar type predictors. Predictions one and two consider the intensity values of the surrounding pixels as describing a unique plane. The prediction of the next

pixel is the intensity value which would lie on this plane [3]. The surrounding points used are shown in the two dimensional Figure 5.

		pixel column j-1	j	j+1
Pixel Row	i-1	(i-1,j-1)	(i-1,j)	(i-1,j+1)
	i	(i,j-1)	(i,j)	

Figure 5. Local image region.

Let the pixel to be predicted be pixel(i,j). Consider the values of the surrounding pixels to be a third dimension. The height of any three surrounding pixels describes a unique plane. Pixel(i,j) can then be projected onto the plane. Prediction one is described by,

$$\text{predicted value} = (1,j-1) + ((i-1,j-1) - (i-i,j)) \tag{3.1}$$

Prediction two is defined by,

$$\text{predicted value} = (1,j-1) + ((i-1,j+2) - (i-1,j-1)) / 2 \tag{3.2}$$

All arithmetic is done in integers. If a value needed is off the edge of the image, zero is used as the value. The predicted value is restricted to the range of 0-255 by making negative prediction values equal to 0. Predicted values greater than 255 are set equal to 255.

Linear Predictors

Linear predictors were the simplest type implemented. These predictors assume that a line runs horizontally or vertically through the point being predicted [3].

Referring to Figure 5, prediction three is described by,

$$\text{predicted value} = (i,j-1) \tag{3.3}$$

Equation 3.3 is called a horizontal prediction.

Equation 3.4 defines prediction four, which is a vertical predictor.

$$\text{predicted value} = (i-1,j) \tag{3.4}$$

The predicted value is kept between 0 and 255 as shown previously.

Taylor Series Predictors

The Taylor series is the foundation of many of the numerical methods used by computers. A brief review of the Taylor series will now be presented.

If the value of a function f(x) can be expressed in a region of x, close to x=a, by the infinite power series of equation 3.5, then the series is called the Taylor series expansion of f(x) in the region of x=a.

$$f(x) = f(a) + (x-a) \times f'(a) + ((x-a)^2/2! \times f''(a) + \ldots. + ((x-a)^n/n!) \times f^n(a) \ldots. \tag{3.5}$$

Equation 3.5 allows an approximation of f(x), providing the value of f(a) and all derivatives of f(x) at x=a exist and are finite. It is also necessary that x be sufficiently close to a [7].

The series of equation 3.5 can be truncated at some desired level of accuracy to obtain an estimation. The truncation error is referred to as O(h).

The Taylor series lends itself well to prediction, providing numerical methods can be used to obtain the needed derivatives. Finite difference calculus can produce the quantities required through the use of backward differences [7].

Consider the function f(x) which is analytic in the neighborhood of a point x. The value of f(x+h) can be found by expanding the Taylor series about x as in equation 3.6 [7].

$$f(x+h) = f(x) + h \times f'(x) + (h^2/2!)f''(x) + ... \qquad (3.6)$$

It is possible to use equation 3.6 as a prediction formula, providing successively higher order derivative quantities can be obtained

The derivative quantities can be calculated using backward differences. A backward difference refers to the difference in value between two previously defined points. These differences are derived from the Taylor series expansion of f(x-h) as shown in equation 3.7 [7].

$$f(x-h) = f(x) - h \times f'(x) + (h^2/2!) \times f''(x) - .... \qquad (3.7)$$

Solving equation 3.7 for f'(x) results in equation 3.8.

$$f'(x) = (f(x) - f(x-h))/h + (h/2) \times f''(x) - (h^2/3!) \times f''(x) + \ldots . \quad (3.8)$$

If the higher order terms of equation 3.8 are dropped and 0(h) replaces their values, equation 3.9 remains.

$$f'(x) = (f(x) - f(x-h))/h + 0(h) \quad (3.9)$$

Similarly, equation 3.6 can be reduced to equation 3.10.

$$f(x+h) = f(x) + h \times f'(x) + 0(h) \quad (3.10)$$

Substitution of equation 3.9 into equation 3.10 for the f'(x) term results in a prediction formula, namely equation 3.11.

$$f(x+h) = f(x) + (f(x) - f(x-h)) + 0(h) \quad (3.11)$$

The f(x) - f(x-h) term is referred to as the first backward difference, or as the first order approximation using one derivative [7]. The accuracy of equation 3.11 can be improved by including higher order derivatives. A backward difference approximation for f"(x) can be obtained by expanding the Taylor series about x to calculate f(x-(2h)). The result is shown in equation 3.12 [7].

$$f''(x) = (f(x) - 2 \times f(x-h) + f(x-2h))/h^2 \quad (3.12)$$

Increasingly accurate approximations result from using higher order derivatives. These derivatives can be approximated using backward differences. Equations 3.9 and 3.12 are considered first

order derivative approximations because a single term is used to obtain the derivative value. Second order approximations use two terms in each estimation of the derivative.

Taylor series predictors for grey-scale images are obtained by letting f(x) be the intensity value of pixel x, and h be one. The prediction for f(x+h) then refers to the succeeding pixel.

Consider Figure 6 as an image region. The following Taylor series predictors were implemented using backward differences in two dimensions:

1. First order using zero derivatives is described by, predicted

   value = (i-1,j-1) (3.13)

2. First order using one derivative is described by predicted

   value = 3(i-1,j-1) - (i-1,j-2) - (i-2,j-1) (3.14)

3. First order using two derivatives is described by, predicted

   value = 5(i-1,j-1) - 3(i-1,j-2) + 0.5 (i-1,j-3) -
   3(i-2,j-1) + 1(i-2,i-2) + 0.5(i-3,j-1) (3.15)

4. First order using three derivatives is described by,

   predicted value = 6.33(i-1,j-1) - 5.00(i-1,j-2) +
   1.50(i-1,j-3) - 0.17(i-1,j-4) -
   5.00(i-2,j-1) + 3.00(i-2,j-2) -
   0.50(i-2,j-3) + 1.50(i-3,j-1) -
   0.50(i-3,j-2) - 0.17(i-4,j-1) (3.16)

5. First order using four derivatives is described by, predicted

   value = 6.99(i-1,j-1) - 6.33(i-1,j-2) +
   2.50(i-1,j-3) - 0.50(i-1,j-4) +
   0.04(i-1,j-5) - 6.33(i-2,j-1) +
   5.00(i-2,j-2) - 1.50(i-2,j-3) +
   0.17(i-2,j-4) + 2.50(i-3,j-1) -
   1.50(i-3,j-2) + 0.25(i-3,j-3) -
   0.50(i-4,j-1) + 0.17(i-4,j-2) +
   0.04(i-5,j-1) (3.17)

6. Second order using one derivative is described by, predicted

$$\begin{aligned} \text{value} = {} & 4.00(i-1,j-1) - 2.00(i-1,j-2) + \\ & 0.50(i-1,j-3) - 2.00(i-3,j-1) + \\ & 0.50(i-3,j-1) \end{aligned} \tag{3.18}$$

7. Second order using two derivatives is described by, predicted

$$\begin{aligned} \text{value} = {} & 8.50(i-1,j-1) - 8.00(i-1,j-2) + \\ & 4.00(i-1,j-3) - 1.00(i-1,j-4) + \\ & 0.25(i-1,j-5) - 8.00(i-2,j-1) + \\ & 4.00(i-2,j-2) - 1.00(i-2,j-3) + \\ & 4.00(i-3,j-1) - 1.00(i-3,j-2) + \\ & 0.25(i-3,j-3) - 1.00(i-4,j-1) + \\ & 0.125(i-5,j-1) \end{aligned} \tag{3.19}$$

Pixel Row	Pixel Column: j-5	j-4	j-3	j-2	j-1	j
i-5				i-5,j-2	i-5,j-1	
i-4				i-4,j-2	i-4,j-1	
i-3			i-3,j-3	i-3,j-2	i-3,j-1	
i-2	i-2,j-5	i-2,j-4	i-2,j-3	i-2,i-2	i-2,j-1	
i-1	i-1,j-5	i-1,j-4	i-1,j-3	i-1,j-2	i-1,j-1	
i						i,j

Figure 6. Taylor series image region.

These are selected by prediction numbers 7 through 11, respectively. The linear combination of points in the Taylor series predictions result from combining terms within the approximation formula.

Least Mean Square Predictor

Values of adjacent pixels are often highly correlated [8]. This fact leads to the assumption that, if a pixel, say (i,j-1), is a certain intensity level, then the adjacent pixel, (i,j), is likely to have a similar value.

In order to take advantage of this fact, a linear estimator that results in the least mean square estimation error(minimizing the variance of the error distribution) is given by,

$$\text{predicted value} = p(i,j-1) + (1-p)m \qquad (3.20)$$

The average of all intensity values in the image is m, and p is the normalized correlation coefficient. The mean m is given by equation 3.21 while the normalized correlation coefficient is found by using equation 3.22 [8].

$$m = E[(i,j)] \text{ (the expected value of pixel } (i,j)) \qquad (3.21)$$

$$p = E[(i,j) \times (i,j-1)]/E[(i,j)^2] \qquad (3.22)$$

Equation 3.20 can be interpreted as a weighted average of the preceding pixel, (i,j-1), and the mean pixel value. The weights depend on the correlation coefficient p. When the pixel values are highly correlated, p approaches 1.0 and the prediction is based primarily on (i,j-1). If the pixels are not highly correlated the prediction is based largely on the mean.

This prediction method is able to produce a predicted value which reflects both a local feature(the previous pixel) and an image comprehensive feature(the average pixel value). The importance of the

mean depends upon the value of p. In the event that very little correlation exists between adjacent pixels, the mean value provides a prediction value of some accuracy.

Differential Pulse Code Modulation Predictors

Differential pulse code modulation (DPCM) is a technique where waveform redundancy is utilized in time-domain operations in order to obtain reductions in bit rate [9]. In other words, DPCM attempts to produce an error stream requiring fewer bits than the original data source. Linear predictors are used in DPCM systems which are based on the recent history and time-invariant predictor coefficients.

A DPCM first order predictor is described by equation 3.23, where h_1 is the value which will produce the minimum error range variance.

$$\text{predicted value} = h_1(i,j-1) \qquad (3.23)$$

DPCM prediction attempts to minimize the variance of the error distribution by obtaining an optimal h_1 value. The error is found by equation 3.24.

$$\text{error} = (i,j) - h_1(i,j-1) \qquad (3.24)$$

The variance of the error distribution, found by taking squares and expectations is represented by equation 3.25.

$$Vd = (1 + h^2 - 2ph_1)Vi \qquad (3.25)$$

The normalized correlation coefficient is represented by p, and V_i is the variance of the image pixel values. Setting the partial

derivative of V_d with respect to h_1 yields the optimum h_1 and the corresponding minimum prediction error variance is achieved by letting h_1 equal p. The first order DPCM predictor given in equation 3.23 can be considered as equation 3.26, when h_1 is replaced by p [9].

$$\text{predicted value} = p(i,j-1) \qquad (3.26)$$

The second order DPCM predictor is given by equation 3.27 [9].

$$\text{predicted value} = h_1(i,j-1) + h_2(i,j-2) \qquad (3.27)$$

The error is obtained as in the first order case.

$$\text{error} = (i,j) - h_1(i,j-1) - h_2(i,j-2) \qquad (3.28)$$

The optimal h_1 and h_2 are found by setting partial derivatives of equation 3.28 equal to zero. The optimal h_1 and h_2 values are given in equations 3.29 and 3.30 [9].

$$h_1 \text{ optimal} = p_1(1-p_2)/(1-p_1^2) \qquad (3.29)$$

$$h_2 \text{ optimal} = (p_2-p_1^2)/(1-p_1^2) \qquad (3.30)$$

The normalized correlation coefficient of (i,j) and (i,j-1) is p_1 and p_2 is the normalized correlation coefficient of (i,j-1) and (i,j-2). The optimal h_1 and h_2 values produce a minimized V_d value as given in equation 3.31 [9].

$$\text{minimum}(V_d) = [1-p_1^2-((p_1^2-p_2)^2/(1-p_1^2))] \qquad (3.31)$$

The error variance of equation 3.31 is less than that of the first

order DPCM case in every case except one. If p_2 equals p_1^2, no gain is obtained by using a second order DPCM predictor.

Second Pass Implementation

The second pass prediction fills in the context array and outputs the nine bit error values. The prediction method is identical to the method used in the first pass. The error bucket extents are used in this pass and, in fact, the context array could not be filled in without these values.

As each prediction is performed on a pixel, the context and the error value are used to update the frequency counts kept in context array. The error bucket which the error falls into can easily be found through comparisons with the error extents.

Three points are used to determine the context. The points are (i,j-1), (i-1,j+1) and (i-1,j-1) as shown in Figure 5. The intensity values of these pixels are labeled point1, point2, and point3. They are used in equation 2.1 to determine the unique conditioning state of each error.

The context and error bucket values are now used as row and column indices for incrementing the proper frequency counts in context array. Specifically, context array row 'new context', column zero and context array row 'new context', column 'error bucket' are incremented by one.

The error is output as a nine bit quantity. The form of this output error is a sign bit followed by the eight bit absolute value of

the error. This form is storage efficient and logical in nature, while removing the need for two's complement representation.

Model File Generation

The frequency counts must be used to produce the error probabilities or probabilities which can be easily combined to obtain individual error probabilities. The structure and content of the model file is as follows:

1. The model file dimensions.
2. The error bucket extents.
3. The context probabilities.
4. The conditional probabilities of the error bucket given the context, P(b/w).
5. The conditional probabilities of the error given the bucket within which it falls, P(e/b).

The ability to vary the number of error buckets and the number of context buckets make it necessary to output these parameters with the model file. The model access routine must have the error and context bucket numbers in order to retrieve the probabilities needed. These parameters are output as four byte integers.

The error bucket extents from error array are written to the model file next. These values are needed to determine the error bucket of each error to be encoded. These extents are output as four byte integers.

The context probabilities are then output as four byte floating point probabilities. These values are obtained from the first column

of context array and are calculated by equation 3.32, where i is some new context number and N is the total number of pixels.

$$P(w) = \text{context array}[i,0]/N \qquad (3.32)$$

The conditional error probability of each error, P(e/w), is given by,

$$P(e/w) = P(b(e)/w) \times P(e/b(e)) \times P(w) \qquad (3.33)$$

The two conditional probabilities which are needed to obtain P(e/w), P(b(e)/w) and P(e/b(e)), can be derived from the frequency counts in context array, error array and error counts.

The probability quantity P(b(e)/w)(the probability of a bucket given a context) is obtained from context array. Given context i and error bucket j, $P(b_j/w_i)$ is found by,

$$P(b_j/w_i) = \text{context array}[i,j]/\text{context array}[i,0] \qquad (3.34)$$

Recalling Figure 3, context array[i,j] is the number of occurrences of error bucket j, given that context i was the conditioning state of the error. Context[i,0] is the number of times context i occurred.

P(e/b(e)) represents the probability of an error given the bucket within which the error falls. This quantity can be obtained from the frequency counts in error counts and the error bucket totals in error array. Error counts contains the frequency counts of each distinct error. Error array holds the total number of errors within each bucket as well as the error bucket extents. P(e/b(e)) is given by

equation 3.35, where b(e) represents the unique error bucket containing error e.

$$P(e/b(e)) = \text{error counts}[e]/\text{error array}[b(e),0] \tag{3.35}$$

The P(b/w) fractions are output as four byte quantities. The P(e/b(e)) values are then output using four bytes. Both quantities are floating point numbers.

This type of model formation effectively creates a file comprised of three main data structures; the P(w), P(b(e)) and P(e/(b(e)) probabilities. The number of coding parameters passed to the encoder/decoder routines is kept to a minimum while providing all the needed quantities to obtain each individual error probability. If any probability was zero, a flag value of negative one was output to indicate an error. The number of values in the model file is given by equation 3.36.

$$\text{total number of values} = \text{total number of contexts} \times (\text{number of error buckets}+1) + 511 + 2 + (2 \times \text{number of error buckets} \tag{3.36}$$

The total overall model size is four times the total number of values, as each value requires four bytes.

Entropy Calculation

It is desirable to calculate the entropy of the error file using the frequency counts kept. This can be done by calculating the amount of information provided by the error within the error bucket and the information content from determining the error bucket.

The contribution to the N-pixel image entropy from the error within the bucket, denoted I(e/b(e)), is given by equation 3.37 [3].

$$I(e/b(e)) = \sum_{b} n(b) \times \log_2(n(b)) - \sum_{b,e} n(e/b(e)) \times \log_2 n(e/b(e)) \quad (3.37)$$

The term n(e/b(e)) equals $\sum_{w} n(e/(b(e),w)$. The n(b) term represents $e \sum_{in} {}_{b} n(e/b(e))$.

The contribution to the entropy from determining the error bucket, I(b/w), is calculated by equation 3.38.

$$I(b/w) = \sum_{w} n(w) \times \log_2(n(w)) - \sum_{b,w} n(b/w) \times \log_2(n(b/w)) \quad (3.38)$$

the per pixel entropy is given by equation 3.39 [3].

$$H(S) = (1/N) \times [I(e/b(e)) + I\ (b/w)] \quad (3.39)$$

A derivation of equation 3.39 from equation 1.2 is contained in the appendix.

The summations needed in the calculation of equations 3.38 and 3.39 are readily available from the frequency counts in context array, error array and error counts. The quantity n(e/b) refers to the frequency of each individual error, while n(b) is the number of errors within an error bucket. The sum n(b/w) is the total of all errors within a bucket given a context, and n(w) is the number of occurrences of an individual context. The entropy is the evaluation calculation for the modeler and each prediction technique.

Chapter 4

COMPRESSION RESULTS

Test Data and Testing Method

The test data for the predictor/modeler software consisted of six grey-scale images. Five of these images were x-rays of various parts of the human body. These images have a considerable amount of information and should, therefore, be difficult to compress. The remaining image was an image of a student's face. All test data contained eight bit pixel values, value range 0 - 255, and were placed in computer storage in a row by row fashion. The x-ray images had vertical resolution of 484 and horizontal resolution of 756. The face was slightly larger, having 768 pixels horizontally and 512 pixels vertically.

The software was throughly tested for correct execution before entropy results were accumulated. Unprediction routines were also produced to insure all predictors were unpredictable in a first-in first-out manner.

Results were obtained by executing the predictor/modeler on the test images. The modeling function used user-supplied context and error bucket parameters. Results were gathered in order to compare the effect of increasing the number of error buckets. An upper limit on these parameters was the resulting size of the model file.

Prediction method results must be compared using identical models. The prediction techniques were also compared using all images, but only comparing within the specific image.

Entropies were calculated automatically at the conclusion of each individual execution by the software. Entropies indicate the optimal average compression obtainable for the probabilities established. This type of measurement allows model and prediction evaluation independent of any encoding scheme. The size of the encoded file depends upon the ability of the encoder to take advantage of the frequency measures provided by the modeling routine. The more available compression provided by the modeler can only result in more compression by any encoding scheme. Arithmetic, Huffman, or any other types of coding techniques are able to compress within some percentage or fraction of a percent of the entropy. Entropy calculation allows model and prediction evaluation independent of any encoding method.

Prediction Results

The results of each prediction technique for two different models are presented in Tables 4 and 5. The tables contain results of each prediction method for each image, represented in bits. The original images contained eight bit pixel values. Therefore, the average code lengths(entropies) in the tables are in bits out of eight. In other words, if one of the tables contained an entropy of 4.00, the best average length of the code words produced by a lossless encoding routine would be four bits. This four bit value must be compared with

a source string average length of eight bits in order to judge the effective compression.

Table 4 predictions were done using a model with 11 error buckets and 11 context buckets per context pixel. The size of the model file in this case was 66,072 bytes and contained 16,518 four byte values. This model represented an overhead of 18.06% on the x-ray images and 16.80% for the face.

Table 4. Prediction results with small model.

Prediction Selector	Face	Foot	Leg	Pelvis	Shoulder	Skull	Average
1	4.290	5.131	5.261	5.320	5.275	5.085	5.060
2	4.084	4.774	4.906	5.006	4.928	4.808	4.751
3	3.631	4.531	4.650	4.706	4.687	4.685	4.482
4	4.870	4.691	4.816	4.913	4.707	4.515	4.752
5	4.844	4.566	4.795	4.807	4.655	4.654	4.720
6	5.421	5.807	5.995	6.050	5.985	5.841	5.850
7	6.095	6.561	6.790	6.892	6.839	6.697	6.646
8	6.296	6.949	7.166	7.300	7.251	7.141	7.017
9	6.410	7.072	7.342	7.483	7.446	7.358	7.185
10	5.993	6.253	6.467	6.550	6.493	6.333	6.348
11	6.423	7.191	7.460	7.601	7.568	7.505	7.291
12	3.320	4.483	4.598	4.653	4.635	4.664	4.392
13	3.404	4.516	4.633	4.684	4.668	4.701	4.434
14	3.797	4.966	4.798	5.037	4.802	4.982	4.730

Table 5 made use of a model file with 22 error buckets and 22 context buckets per context pixel. The model file in this case contained 234,835 values and was an exceedingly large 939,340 bytes. The overhead in this case, 256.72% on the x-rays and 238.89% for the face, is much more than could be tolerated in an actual compression system. The results are of interest for prediction comparison.

Table 5. Prediction results with large model.

Prediction Selector	Face	Foot	Leg	Pelvis	Shoulder	Skull	Average
1	3.893	4.909	5.025	5.086	5.052	4.906	4.812
2	3.632	4.552	4.703	4.747	4.745	4.672	4.509
3	3.310	4.253	4.356	4.400	4.386	4.376	4.180
4	4.163	4.446	4.582	4.664	4.530	4.382	4.461
5	4.168	4.355	4.571	4.583	4.485	4.995	4.443
6	4.900	5.524	5.776	5.775	5.771	5.679	5.571
7	5.603	6.281	6.582	6.655	6.642	6.534	6.383
8	5.854	6.651	6.915	7.022	7.020	6.936	6.733
9	5.945	6.832	7.082	7.120	7.210	7.140	6.888
10	5.501	5.992	6.260	6.307	6.297	6.156	6.086
11	5.958	6.880	7.197	7.314	7.320	7.274	6.991
12	3.019	4.164	4.316	4.335	4.338	4.375	4.091
13	3.019	4.165	4.314	4.334	4.336	4.376	4.091
14	3.017	4.633	4.550	4.759	4.546	4.712	4.435

The tables show several points. The higher order Taylor series predictors produce poor compression results for each and every image. The entropies produced by the first order Taylor series predictors increased as the number of derivatives used increased.

Comparing the horizontal, least mean square, and order one DPCM predictors shows very small differences in the entropy measure. This means that two adjacent pixels on a scan line are highly correlated.

The prediction results show that prediction can greatly affect the amount of compression available. Well chosen prediction techniques can increase the available compression considerably, as shown in Tables 4 and 5.

These two tables indicate that compression results could be restricted by an ineffective prediction process. Despite an effective model structure, the higher order Taylor series predictors do not provide significant compression.

Modeling Results

The bucketing technique was able to produce effective compression while reducing model file size. The model file size could be effectively controlled through the use of the bucketing parameters. This type of control would be valuable in an information transmitting environment. A user could determine the tolerable amount of model file overhead and then specify bucket parameters which would restrict model file size accordingly.

Table 4 can be considered an example of the available compression provided by the bucketing model. Compression results of between 2.20:1 and 1.50:1 were produced. These results were obtained with less than 20% overhead. The effective compression provided by this overhead was 33.5% in the worst case and 54.7% in the best case. The ineffective higher order Taylor series prediction methods were not considered.

Table 5 contains better overall compression results. These results improved compression by at most 0.707 bits. This best case improvement represents and 8.8% compression increase, based on eight bit pixel values. The size of the model file needed to achieve this increase was more than 14 times larger than that needed in Table 4. This sort of trade off can hardly be considered worthwhile. The trade off being: to compress an image an additional 8.8% cost 14 times more overhead.

Table 6 illustrates the results of testing increasing numbers of error buckets for available compression. Tests were run on all images

but the results from the face are presented in Table 4.3. The face image results were representative of the results obtained from the entire test image set. The ineffective Taylor series predictors were omitted from Table 6. The number of context buckets per context pixel in all error bucket tests contained in Table 6 was 11.

Table 6. Error bucket results.

Prediction Selector	Error Buckets			
	5	11	65	255
1	3.786	4.290	4.010	3.963
2	3.642	4.084	3.850	3.804
3	3.563	3.631	3.429	3.398
4	3.530	4.870	4.397	4.269
5	3.775	4.844	4.383	4.254
12	3.477	3.320	3.212	3.188
13	3.764	3.404	3.210	3.186
14	4.230	3.797	3.465	3.374

The model file size must be considered when judging the increased amount of compression available. The model file size in the five error bucket, 11-context bucket case was 34,036 bytes. In the 11 error bucket and 11-context bucket case, the model file size was 66,072. The model file grew to 353,956 bytes when 65 error buckets were used. In the 255 error bucket test, a model file of 1,415,032 bytes was generated.

Increasing the number of error buckets provided better compression resultsfor predictors 11, 12, and 13. The overhead used to obtain these costs were once again prohibitive as the number of error buckets increased beyond 11. The greatest increase in compression was 0.473 bits or 5.9%. The increased compression does not appear to be,

and in most practical situations would not be, worth the significant increase in overhead. Increasing from five to 11 error buckets is not worthwhile, in most cases because compression was decreased.

Chapter 5

ANALYSIS OF RESULTS

Predictor Analysis

The 14 increasingly sophisticated prediction techniques provided results which can be used to answer the previously stated questions. The planar and linear techniques are suggested in compression literature while the ten remaining methods are absent. These more sophisticated prediction techniques provide increased compression.

The Taylor series predictors produced poor compression results. The use of an increasingly larger number of previous pixels resulted in entropy measures of increasing value. The Taylor series approximation attempts to fit a line, determined by previous pixel differences, to the succeeding pixel. The poor results of the Taylor series could be attributed to little correlation between previous pixels further than one step from the pixel being predicted. In other words, the more distant pixels have little relation to an approximated line through the predicted pixel. The Taylor series predictors do not provide increased compression.

The least mean square predictor provided as much or more compression than the planer and linear techniques in nearly every test on the face image. This results from the fact that in the face image there exists a high correlation between adjacent pixels. Tests of the

remaining five images produced similar relationships. The pixel correlation in most cases was so high(close to a maximum of 1.0) that the least mean square predictor became equivalent to the first order DPCM. As the correlation coefficient approaches 1.0, which the test cases show to be true, both of the previously mentioned prediction methods deteriorate to the simple horizontal predictor. In these test cases the least mean square predictor achieved slightly better compression than the planar and linear predictors. The cost of this increase was the computation expended to calculate the correlation coefficient.

The first order DPCM predictor achieved compression as good as, or better than the planar and linear prediction methods. The second order DPCM predictor produced less available compression than the two planar methods in most cases. The vertical and horizontal prediction techniques had lower entropies than the second order DPCM method. This could be attributed once again to a low correlation between non-adjacent pixels. The second order DPCM predictor required the additional computations involved in calculating two correlation coefficients.

The results of the Taylor prediction methods indicate that the available compression can be greatly affected by the prediction process. Tables 4 and 5 show significant increases in the entropy measure as more sophisticated Taylor series predictors were used. These increases developed despite an effective modeling technique. The amount of compression produced by an actual system could be

limited by the choice of prediction method, irregardless of the modeling or encoding technique.

Prediction is an important process in the compression of grey-scale images. A poorly chosen prediction technique would establish a compression limit which no modeling technique could lower. Prediction attempts to produce a file with increased redundancies. Ineffective prediction will not produce as many redundancies as are possible from the input image. Poorly chosen prediction methods should be avoided in order to obtain as much available compression as possible.

Model Analysis

The bucketing techniques implemented were able to reduce model file size while providing effective compression results, when well chosen predictors were used. Model file overhead could be reduced or increased to any limit considered tolerable. A flexible approach to model file size specification has several advantages. Each individual user or installation could set tolerable overhead standards. The bucketing technique would allow conformity to virtually any overhead limit.

A user could individually judge whether an increase in model file size was worthwhile for any or every image to be compressed. In some cases an increase of just 5% in the amount of compression could make a significant overall difference and therefore be worth the additional overhead.

Bucketing parameters could be adjusted for optimal performance on subsets of specific types of grey-scale images. For instance, aerial images might be found to compress better with a large number of error buckets and very few context buckets. Allowing bucket parameter adjustment at execution time permits application specific bucket parameters.

Increasing the number of error buckets produced more available compression for most prediction methods. The cost of this increase was prohibitive. In order to gain a 5.9% increase in available compression, more than a five fold increase in model file overhead was required. Such a cost overhead for such a small return is hardly justifiable in most situations. An increase from five error buckets to 11 does not seem worthwhile because the larger model file does not provide increased compression.

Increasing the number of context buckets above 11 produces a very rapid growth in model size. This type of expansion results in excessively large model file overhead. Even large increases in error bucket numbers cannot offset this growth. This behavior is easy to recognize because the error bucketing parameter increases the model file linearly. The context bucket parameter increases the overhead cubically while requiring an additional set of error bucket probabilities for each new context.

Conclusions

Bucketing reduces model file size while obtaining effective compression when used with an effective prediction method. Prediction

is an important process in the compression of grey-scale images. More sophisticated prediction methods can provide more available compression. The increased compression is not dramatic and is not achieved in all cases. More importantly, poor prediction techniques can dramatically reduce the amount of available compression.

Future Research

Future research in this area might be done in order to discover a more efficient way to cluster contexts into buckets. More effective compression might be obtained by bucketing contexts such that each bucket contained roughly equal number of original contexts. Increased compression may result from determining context bucket extents such that large numbers of unused original contexts existed between context buckets. The effect of multiple error distributions may also provide increased compression.

A large static model file could be used for all images to be compressed. This model could be updated periodically from statistics or frequency counts, thus removing the need to transfer modeling data with each image across communication lines. This approach would remove the need to generate an individual model for each compressed image.

The prediction methods tested could also be used as filters. The absolute value of the error was output to a file and displayed on a video screen by the author. The prediction techniques were edge, plane or surface sensitive. This could be useful in image analysis research.

REFERENCES CITED

REFERENCES CITED

Langdon, Glen G. "An Introduction to Arithmetic Coding", I.B.M. J. Res. Develop., Vol. 28, No. 2, March 1984, p. 135-140.

Abramson, Norman. Information Theory and Coding, McGraw-Hill Book Company, New York, New York, 1963.

Todd, Stephen; Langdon, Glen G.; and Rissanen, Jorma. "Parameter Reduction and Context Selection for Compression of Grey-Scale Images", IBM J. Res. Develop, Vol. 29, No. 2, March 1985, p. 188-193.

Rissanen, Jorma and Langdon, Glen G. "Universal Modeling and Coding", IEEE Trans. on Info. Theory, Vol. 27, No. 1, January 1981, p. 12-23.

Weyland, Nicholas J. "Compression and Run-Time Results for CFD Solution Set Files", NAS Technical Memo, NAS TM NW-102, August 1985.

Landgon, Glen G. and Rissanen, Jorma. "Compression of Black-White Images with Arithmetic Coding", IEEE Trans. on Comm., Vol. 29, No. 6, June 1981, p. 858-867.

Hornbeck, Robert W. Numerical Methods, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

Gonzales, Rafael C. and Wintz, Paul. Digital Image Processing, Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.

Jayant, N.S. and Noll, Peter. Digital Coding of Waveforms -- Principles and Applications to Speech and Video, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

APPENDIX

APPENDIX

Derivation of H(E/W)

$$H(E/W) = \sum_w P(w)\, H(E;w)$$

$$= \sum_w P(w) \sum_e P(e/w) \log \frac{1}{P(e/w)}$$

$$P(w) = \frac{n(w)}{N}$$

$$P(e/w) = P(e/b)\, P(b/w)$$

$$= \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)}$$

$$\log(P(e/w))^{-1} = \log n(b) + \log n(w) - \log n(e/b) - \log n(b/w)$$

$$\sum_e P(e/w) \log (P(e/w))^{-1} = \sum_e \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \log n(b)$$

$$+ \sum_e \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \log n(w)$$

$$- \sum_e \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \log n(e/b)$$

$$- \sum_e \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \log n(b/w)$$

Bucket Contribution:

$$\sum_e \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \left[\log n(b) - \log n(e/b)\right]$$

$$= \sum_b \left[\sum_{e \text{ in } b} n(e/b)\right] \frac{n(b/w)}{n(b)n(w)} \log n(b)$$

$$- \sum_b \left[\sum_{e \text{ in } b} n(e/b) \log n(e/b)\right] \frac{n(b/w)}{n(b)n(w)}$$

$$= \sum_b \frac{n(b/w)}{n(w)} \log n(b) - \sum_{b,e} \frac{n(b/w)}{n(b)n(w)} n(e/b) \log n(e/b)$$

Context Contribution:

$$\sum_{e} \frac{n(e/b)}{n(b)} \frac{n(b/w)}{n(w)} \left[\log n(w) - \log n(b/w)\right]$$

$$= \sum_{b} \left[\sum_{e \text{ in } b} n(e/b)\right] \frac{n(b/w)}{n(b)n(w)} \left[\log n(w) - \log n(b/w)\right]$$

$$= \sum_{b} \frac{n(b/w)}{n(w)} \log n(w) - \sum_{b} \frac{n(b/w)}{n(w)} \log n(b/w)$$

$$\therefore \quad H9E/W) = \sum_{w} \frac{n(w)}{N} * \text{[Bucket Contr. + Context Contr.]}$$

$$= \frac{1}{N}\left[\sum_{w}\sum_{b} n(b/w) \log n(b) - \sum_{w}\sum_{b,e} \frac{n(b/w)}{n(b)} n(e/b) \log n(e/b)\right]$$

$$+ \frac{1}{N}\left[\sum_{w}\sum_{b} n(b/w) \log n(w) - \sum_{w}\sum_{b} n(b/w) \log n(b/w)\right]$$

$$= \frac{1}{N}\left[\sum_{b}\left(\sum_{w} n(b/w)\right) \log n(b) - \sum_{b,e}\sum_{w} n(b/w) \frac{n(e/b)}{n(b)} \log n(e/b)\right]$$

$$+ \frac{1}{N}\left[\sum_{w}\left(\sum_{b} n(b/w)\right) \log n(w) - \sum_{b,w} n(b/w) \log n(b/w)\right]$$

$$= \frac{1}{N}\left[\sum_{b} n(b) \log n(b) - \sum_{b,e} n(e/b) \log n(e/b)\right]$$

$$+ \frac{1}{N}\left[\sum_{w} n(w) \log n(w) - \sum_{b,w} n(b/w) \log n(b/w)\right]$$