1993

Efficient software implementation of the JBIG compression standard

Craig M. Smith

Efficient Software Implementation of the JBIG Compression Standard

by

Craig M. Smith

May 13, 1993

A Thesis, submitted in partial

fulfillment of the requirements for the degree of

Master of Science in Computer Engineering

Approved by:

Ronald G. Matteson, Ph.D. (Committee Chairman)
Department of Computer Engineering

Tony H. Chang, Ph.D. (Committee Member)
Department of Computer Engineering

James R. Sullivan (Committee Member)
Eastman Kodak Company

Roy S. Czernikowski, Ph.D. (Department Head)
Department of Computer Engineering

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York

Title of Thesis: Efficient Software Implementation of the JBIG Compression Standard

I _____ , prefer to be contacted each time a request for reproduction is made. I can be reached at the following address.

48 Harvest Drive

Rochester, New York 14626

Date 5-17-93

Table of Contents

Appendices

List of Figures

List of Tables

Abstract

JBIG is a new binary image compression standard that is designed to handle both text and halftoned documents. It significantly outperforms the CCITT Group 3 and Group 4 standards especially on halftoned documents. The JBIG standard is based on an arithmetic encoder, and it features adaptive probability estimation, adaptive templates, and three different types of prediction. It also allows either sequential or progressive compression of image data.

A brief discussion of information theory is given as it applies to image compression. The JBIG algorithm is described, and several techniques are developed to efficiently implement the algorithm in software. The most important techniques include an efficient scheme for building the context that are required, and taking advantage of large all-white or all-black regions of images by designing very efficient loops for processing those areas. Other techniques are also discussed, such as, efficient implementation of deterministic prediction, and an improved method for handling the conditional exchange condition. Timing information is given for the final implementation on several platforms. The JBIG algorithm is compared with the CCITT Group 3 and Group 4 algorithms, and it is tested in noisy environments.

1 Introduction of Compression Concepts

Within digital computer equipment, information is represented as a series of symbols that are encoded as binary numbers. Almost any information can be represented this way including character based data such as reports and computer programs, databases of numbers representing scientific information or accounting information, and digital images. Information of this type can be stored in several ways, but the storage space is often costly. In addition, the information can be transmitted from one place to another, but when large amounts of data are transferred, it can take a long time.

To overcome the problems of storage and transmission time, it is often useful to convert the information into a more compact representation. The resulting data requires less storage and can be transmitted much more quickly. The process of converting the data is often called "data compression." When the data represents a picture, the process is known as "image compression."

The purpose of this chapter is to explain the concepts of image compression that are necessary to have an understanding of the JBIG (Joint Bi-level Image experts Group) compression standard [1]. Before this can be accomplished, it is important to understand why compression of digital images is useful. Next, a few basic concepts of information theory will be introduced. This will be followed by a discussion of an impractical but instructional encoding technique known as Elias coding [2]. Lastly, arithmetic coding will be introduced. Arithmetic coding is an algorithm that solves the shortcomings of Elias coding, and it is the heart of the JBIG standard.

1.1 Compression of Binary Images

In a great many applications it is useful to use numbers in a computer to represent a picture or image. This can be done a number of different ways. In general, an original scene or document is sampled at discrete locations by an analog sensor that is capable of sensing the brightness of the image within a sample as illustrated in figure 1. Each sample, or pixel (short for picture element) is then converted to a number that represents the brightness of the image within the sample. The number of bits used to represent each pixel can range anywhere from 1 to 16. In many applications, the pixels are stored as an 8-bit quantity so that the numbers are in the range of 0 to 255. In many business

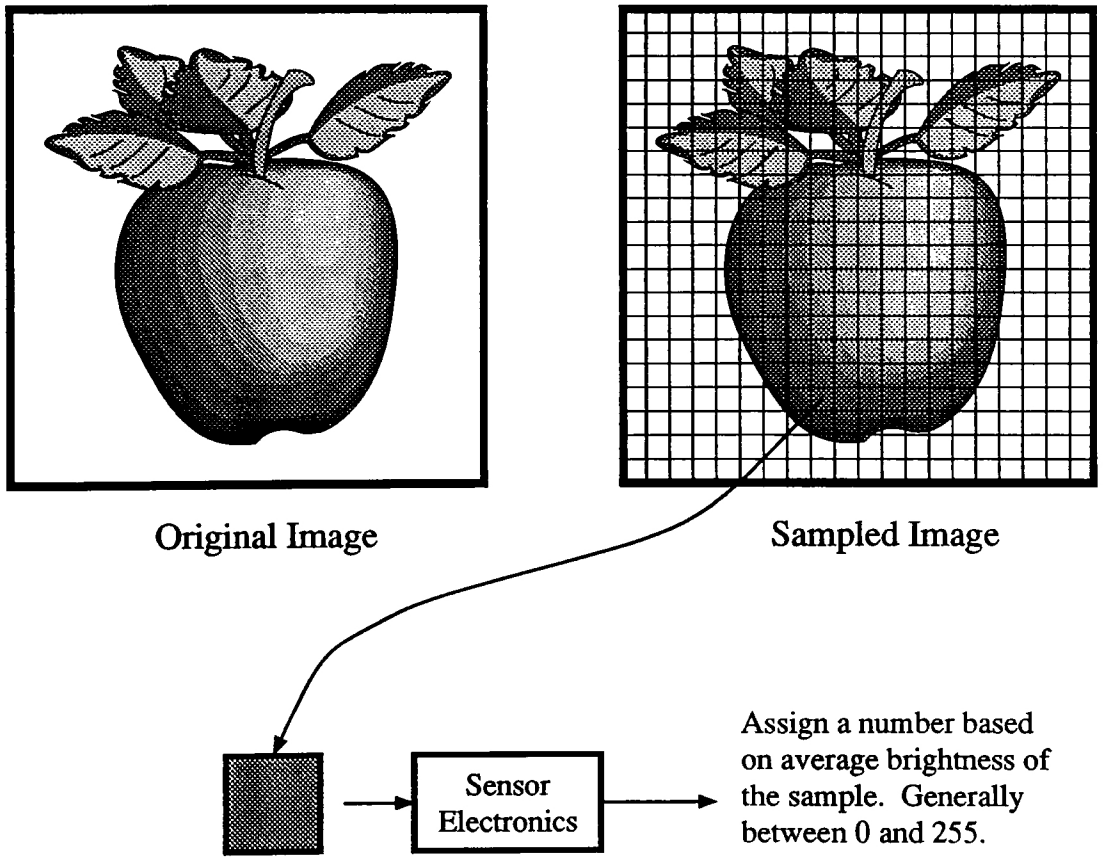


Figure 1 Formation of a digital Image

applications, where images are generally comprised of black and white text documents each pixel is stored as a 1-bit number. These 1-bit images are known as binary images.

An example of the use of binary images in a business application is a facsimile machine. A typical FAX machine scans an 8.5 x 11 inch sheet of paper and generates a 1728 x 1100 pixel image. The resulting image contains 1,900,800 pixels. Since most FAX machines transmit data through a phone line at 9600 bits/second, it would take about 3 minutes and 20 seconds to transmit the entire image. For most documents, however, it is possible to compress the image to 1/10th of its original size or less. This is accomplished through the use of standard FAX compression algorithms. As a result, most documents can be transmitted in 20 seconds or less.

Another application in which binary images are common is in binary printers such as laser printers. Since most laser printers print 300 dots per inch or more, a full page image requires 8,415,000 pixels. In other words, a standard high-density PC floppy could only hold one full page image, If the images are compressed, however, it is possible to store as many as 10-20 full-page images on a single floppy disk.

1.2 Basic Information Theory

To have any understanding of the underlying concepts of image compression in general, and the JBIG standard in particular, it is necessary to be aware of some of the fundamental concepts of information theory. In this section, a few of the basic concepts will be discussed briefly and informally. The foundation for this theory was developed by Shannon [3] in 1948. For a more complete introduction to information theory as it relates to image compression see Rabbani and Jones [4]. For a thorough discussion of background and uses of information theory see refs. [5-6].

A binary image is one example of a source of information. When it is sent to a laser printer, each binary pixel represents information for the printer. The pixel tells the printer whether to print a black dot at a specific location or not. Another example of a source of information is English text. In this case, the reader examines each character in the text and extracts information from it.

In general, it is possible to define an information source, S, that generates a series of symbols as shown in figure 2. Each symbol emitted by S is part of a fixed finite alphabet of size n, so $S = \{ s_1, s_2, ..., s_n \}$, and the probability of any symbol, s_i, in the alphabet is a known fixed value $p(s_i)$. If each symbol released by the source is statistically independent of all other symbols released by S, then S is known as a Discrete Memoryless Source (DMS) [4].

To understand the information carried by these symbols, consider a person receiving the symbols from a DMS. When receiving symbols, the receiver might guess that the incoming symbol will be the highest probability symbol. If he receives the symbol he expected, he is not surprised. In other words, the symbol carried little information. On the other hand, if the receiver receives a low probability symbol, he is very surprised. The low probability symbol represents an event that was not expected, so it carries much more information. In general, it is true that the amount of information carried by a symbol is inversely related to the probability of the symbol. If the information carried by a

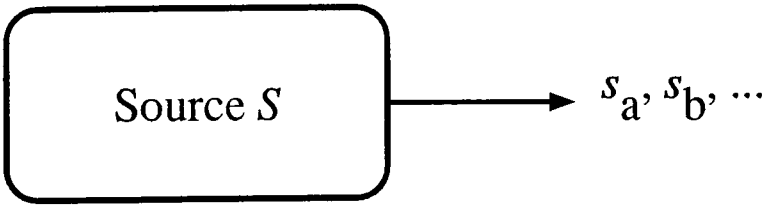


Figure 2 General Information Source

symbol s_i is encoded by a binary alphabet for use in a digital system, it is given by the equation

$$I(s_i) = -\log_2 p(s_i) \qquad (1)$$

Since the DMS generates symbols with a known probability, the average information per symbol can be calculated by summing the information generated by each symbol in the alphabet weighted by the probability of the symbol occurring as given by the equation

$$H(S) = \sum_{i=1}^{n} p(s_i) I(s_i) = -\sum_{i=1}^{n} p(s_i) \log_2 p(s_i) \qquad (2)$$

The quantity $H(S)$ is known as the entropy of source S. It represents the average amount of information that the source, S, can generate. It is a useful quantity because it represents the average theoretical limit of algorithms that encode the symbols using a binary alphabet. According to Shannon's noiseless source coding theorem, no uniquely decodable code can have an average length less that $H(S)$.

As an example, consider a DMS generating bits for a binary image, $S = \{ 0, 1 \}$. Any two symbols generated by the source are statistically independent, and the probabilities of the two output symbols are fixed. If the two symbols are equally probable, i.e. $p(0) = p(1) = 0.5$, then by using equation 2, the entropy of the source is calculated to be 1.0 bit per pixel. This means that trying to compress an image generated by the source would be fruitless because the entropy of the source is equal to the number of bits in the symbol sequence without compression. If the probabilities of the two symbols are unequal, however, the average amount of information per symbol would be lower and compression would be possible. Consider a binary DMS where $p(0) = 0.95$ and $p(1) = 0.05$. In this case, according to equation 2, the entropy of the source is 0.29 bits per pixel. This

means that compressing an image generated by the source could be a useful step because it should be possible to compress the image to about a third of its original size.

While this example is useful for illustrating the concept of entropy, it is limited because typical information sources are not statistically independent. Imagine a person reading English text one character at a time. If he sees the string 'exampl', he can easily guess with a high degree of accuracy that the next character is an 'e'. In this case, the 'e' carries very little information because the reader knows that the probability of an 'e' is unusually high based on knowledge of the previous symbols. In a similar manner, it can be shown that pixels in an image are statistically related to their neighbors. Therefore, the Discrete Memoryless Source is not an accurate model for most image data.

A better model for image information is a Markov Source [4]. An m^{th} order Markov Source is a source in which the probability of occurrence of a symbol is conditionally dependent on m previous symbols. The probability of a symbol, s_i is denoted as

$$p(s_i \mid \mathbf{S}) \text{ where } i = 1, 2, ..., n \text{ and } \mathbf{S} \text{ is a vector of } m \text{ previous symbols}$$

Since each of the m previous symbols in the $\mathbf{S}$ vector has n possible values, an m^{th} order Markov source has n^m possible states. Within each state, the Markov source is similar to a DMS in that each of the n possible symbols has a fixed probability. Therefore, the entropy of a single state of the Markov source, S_M, is given by

$$H(S_M) = -\sum_{i=1}^{n} p(s_i \mid \mathbf{S}) \log_2 p(s_i \mid \mathbf{S}) \qquad (3)$$

for $1 \le M \le n^m$. The overall entropy of the Markov source is the sum of the entropies of each state times the probability that the state can occur

$$H(S) = \sum_{j=1}^{n^m} p(S_j) H(S_j) \qquad (4)$$

It is important to note that the entropy of a source that is modeled as a Markov source is always less than or equal to the entropy of a source that is modeled as a DMS. This is because the Markov source takes advantage of the local information to make an improved prediction of the emitted symbols. As a result, on average, the symbols carry less information.

When a binary image is modeled as an m^{th} order Markov source, the probability of each pixel is dependent on m previously transmitted pixels. Typically, the pixels are selected to be the m pixels closest to the pixel of interest. These nearby pixels that are used to encode a pixel are known as the pixel's context or the pixel's neighborhood. Figure 3 shows a typical context for a 4^{th} order Markov model in which the values of pixels x_0, x_1, x_2, and x_3 form the vector $\boldsymbol{S}$.

While the Markov model is a significant improvement over the Discrete Memoryless Source, it still has two drawbacks. First, the complexity of the encoding algorithm grows quickly as m gets larger because the number of possible states grows exponentially. Since binary image data has only two possible symbols in its alphabet, the size of the context can be 10 or 12 pixels, as used by the JBIG algorithm, without unusually large memory requirements. Secondly, the improvement supplied by the Markov model is dependent on knowing the conditional probabilities to more accurately predict each pixel. It is not generally true, however, that the conditional probabilities used in a

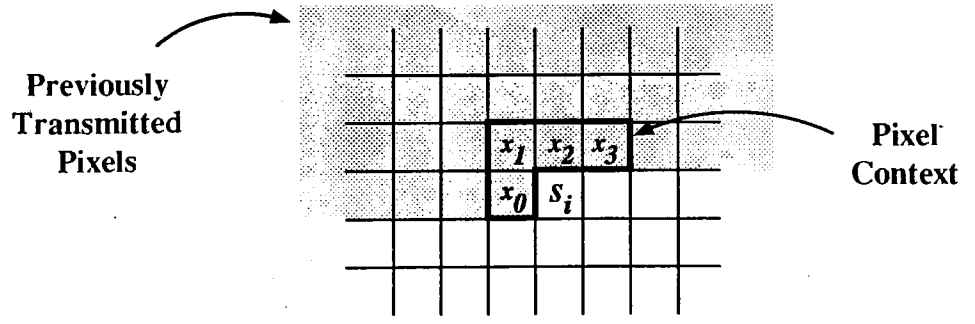


Figure 3 Example Pixel context (or neighborhood) for a 4th order Markov Source

Markov model are the same for all binary images. In fact, the conditional probabilities can change dramatically from image to image. To overcome this problem, the JBIG standard uses an adaptive probability estimate technique that allows the algorithm to "learn" the conditional probabilities for each image. This mechanism for this will be explained in section 2.5.

To summarize, the information content of a set of symbols has been described in terms of entropy. It has been asserted that the entropy of a source can be calculated based on a statistical model of the generated symbols. It has also been implied that a sequence of symbols generated by an information source can be encoded using a binary alphabet into a data stream that is approximately the same size as the entropy of the data. Therefore, if an information source is modeled to create an accurate and biased probability estimate for predicting symbols based on a local context of each symbol, the entropy of a source can be reduced, thereby reducing the size of the output binary data. Nothing has been said about how the symbols are encoded to generate this compressed output data stream.

In fact, there are several well known and simple algorithms for encoding symbols such as Huffman coding [7] and run-length coding [8] that are often used. These algorithms use variable length binary codes to represent one or more source symbols. While these techniques are simple to use, they are not included in the JBIG standard, and will not be discussed here. Within JBIG, the estimated probabilities of each symbol are encoded using a relatively new technique known as arithmetic coding. This is based on an older, but impractical encoding algorithm known as Elias coding.

1.3 Elias Coding

The technique known as Elias coding was introduced by Elias [2] as an algorithm for encoding a sequence of symbols very efficiently, i.e. the average number of bits re-

quired to encode a sequence of symbols is close to the entropy of the sequence. The drawback of the technique is that it is not implementable for image compression because it requires the use of extremely large registers. Regardless of its implementation limitations, it is instructional to understand of the basic concepts of Elias coding as a basis for understanding arithmetic coding which will be discussed in section 1.4.

In Elias coding, a string of symbols generated by a source, S, is encoded as a binary real number in the range of [0, 1) by recursively subdividing probability intervals. Figure 4 shows a graphical representation of this type of subdivision. The entire initial region is divided into two intervals proportional to the probabilities of 0 and 1. When a symbol is encoded, its subinterval is selected and proportionally subdivided for the next symbol. This continues until the entire string is encoded.

To be more precise, consider a series of binary symbols to be encoded in which 0 is the most probable symbol and has a probability of occurrence equal to p. 1 is the least probable symbol and has a probability of occurrence equal to q or 1 - p. To encode one

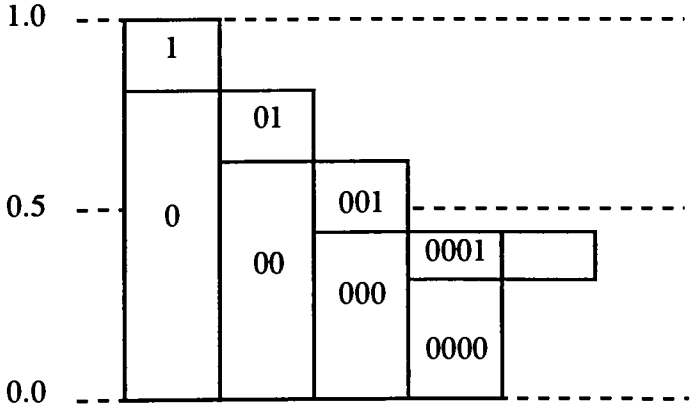


Figure 4 Example of recursive probability interval subdivision in Elias coding.

of these symbols, the real number range [0, 1) is subdivided into two intervals, [0, p) representing the symbol 0, and [p, 1) representing the symbol 1. If the symbol 0 is coded, then the interval [0, p) is subdivided again into regions that are proportional to p and q, namely [0, p^2) for 0, and [p^2, p) for 1. If instead of transmitting the first symbol as a 0, the first symbol had been a 1, then the new interval would have been [p, 1), and it would have been subdivided into regions proportional to p and q, [p, p+qp) for 0, and [p+qp, 1) for 1. This procedure of recursively dividing a subinterval into two new subintervals that are proportional to the probabilities of the symbols being transmitted is repeated for each symbol that is to be transmitted until the the entire message is sent. From the above example, the following characteristics can be seen:

- After transmitting two symbols, the interval [0.0, 1.0) could have been subdivided into one of four possible intervals. In general, after transmitting n symbols the initial interval could have been subdivided into one of 2^n possible intervals.
- None of the possible intervals overlaps any other interval at any point.
- The subintervals cover the entire original range of [0.0, 1.0).
- For n input symbols are being transmitted, the original interval of [0.0, 1.0) will be subdivided in such a way, that every point in the original interval will be associated with exactly one of the final possible subintervals.

After encoding n symbols, a final subinterval is selected. This subinterval can be described by the value of the first point in the interval and the width of the interval (the distance to the first point in the next interval). The width of this final interval, w, is given by the equation

$$w = \prod_{i=1}^{n} p(s_i) \qquad (5)$$

Where $p(s_i)$ is the probability of the i^{th} symbol in the string. The reason that this is important is that w is exactly equal to the joint probability that the entire string will occur. Based on the discussion in section 1.2, it should be clear that the entropy of the entire string generated by source S is

$$I_S = -\log_2 w \qquad (6)$$

Once the final subinterval is selected, it can be uniquely identified by encoding any point within the subinterval as a binary fraction. This is done by creating a binary number, $b_1 b_2 \ldots$ that is interpreted as $b_1 \times 2^{-1} + b_2 \times 2^{-2} + \ldots$. A decoder that receives this faction can identify the subinterval that contains this point, and therefore, completely decode the entire string.

To determine the best choice of points to encode, the value of the beginning of the interval is encoded and the first $\lceil I_S \rceil$ bits of the binary fraction are used, where $\lceil\ \rceil$ represents the ceiling function. The resulting fraction represents the shortest possible binary code to transmit the information of the string and must be within the final subinterval because

$$w \geq 2^{\lceil I_S \rceil} \qquad (7)$$

So far in this discussion of Elias coding, it has been assumed that the probabilities p and q are fixed and globally known by both the encoder and decoder. As seen in the example of a Markov source in section 1.2, however, the entropy of a set of data can be lowered by using a local context to make better predictions of each new symbol. If the information source is modeled as a Markov source, the probabilities that are used to subdivide each interval must change from symbol to symbol depending on the current Markov state. This added complication does not change anything in the previous discussion. In general, as long as the probabilities used for each symbol are causally

known by both the encoder and decoder, the Elias coding algorithm could optimally encode and uniquely decode any string.

The major drawback of the Elias algorithm is that encoding a string of arbitrary length requires infinite precision registers. When encoding a binary image consisting of millions of pixels, registers that are several million bits long could be required to store the location of the beginning of the interval and the current width of the interval. This problem and others related to it are overcome by a compression technique known as arithmetic coding.

1.4 Arithmetic Coding

Arithmetic coding [9-13] is a relatively new encoding technique that is the central coding algorithm for the JBIG compression standard. While it is similar in concept to Elias coding, it overcomes the infinite precision problems. This is accomplished by using a rescaling procedure that slides a fixed precision window on the infinite precision registers of Elias coding.

In this section, a brief description of the JBIG arithmetic encoding algorithm is given to introduce the basic concepts and properties of the algorithm. For a more precise definition of the algorithm see the JBIG specification [1]. For more background on the implementation of arithmetic coding algorithms, it is useful to study a well documented arithmetic encoding algorithm developed IBM known as the Q-coder [14-16].

For an arithmetic coder, a fixed number of bits is allocated for encoding the probability of a symbol. In JBIG, 16 bits are allocated. Two registers of this depth are required, one to keep track of the width of the current subinterval (A), and one to keep track of the bottom of the subinterval (C). These registers define the current subinterval exactly as in the Elias coding algorithm except that each time the width of the interval (the

A register) falls below one half, both registers are doubled repeatedly until the *A* register is above the one half level. This scaling process is known as renormalization and it effectively slides both registers down the arithmetic axis of the two Elias values as shown in figure 5 until the first 1 bit in the fractional representation of the Elias interval is the high order bit in the arithmetic *A* register. The bits shifted out of the *C* register are the bits that are transmitted as part of the encoded data stream with one exception which is known as the carry propagation problem.

The carry propagation problem is caused by the fact that the value of the beginning of the Elias subinterval will always stay the same or increase. When it increases, the value of the arithmetic *C* register can overflow. The result of the binary overflow will be that all 1's above the *C* register will be flipped to 0's and the first 0 before the *C* register will be flipped to a 1. In general, this carry problem can be handled a couple of different ways. The JBIG algorithm keeps the last output byte that contained a zero and a count of the number of "all 1" bytes (255's) that have occurred since. When an overflow occurs, the last byte containing a zero is incremented and transmitted and the 255 bytes are transmitted as 0 bytes.

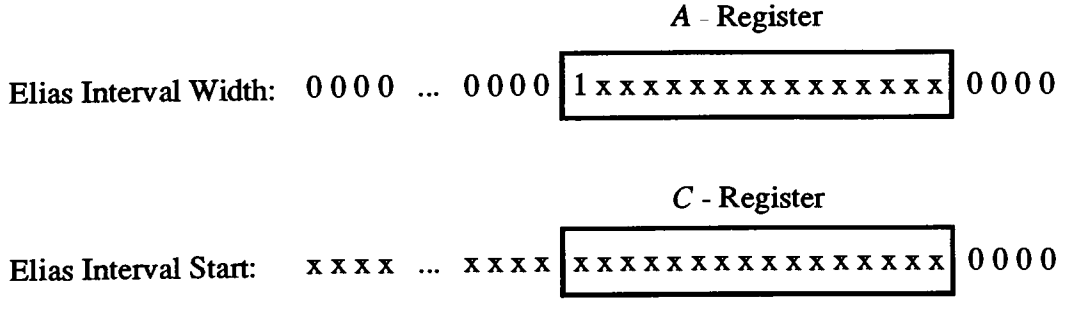


Figure 5 Alignment of the arithmetic coding registers

Another important concept to understand about the arithmetic coder is the fact that it is computationally efficient. When encoding a binary symbol, the most probable symbol will be encoded a high percentage of the time. When this occurs, the A register is reduced by the size of the least probable subinterval. The exact size of the least probable subinterval is the product of the current interval size, A, and the probability of the least probable symbol, q. Since the multiply is undesirable, it is avoided by replacing A with the statistical average of A, which is a constant. By using the statistical average of A, the following approximation is made to calculate an estimate of the least probable interval, for every possible value of q

$$q \times A \approx q \times \alpha = q_e \qquad (8)$$

where α is the average over the probability density of A, and q_e is the resulting estimate of the least probable subinterval size. Since the normalized value of A is bounded between 0.5 and 1, the value of A can be replaced by α without introducing a large error. Since q_e is known for every value of q, it can be stored in a table and found without the need for a multiply. As a result, the only required operation when encoding a most probable symbol is the update for the A register given by

$$A \leftarrow A - q_e \qquad (9)$$

When the least probable symbol is encoded the A and C registers are changed as follows

$$\begin{aligned} C &\leftarrow C + A - q_e \\ A &\leftarrow q_e \end{aligned} \qquad (10)$$

Even though the approximation of the q interval can simplify the arithmetic calculations without imposing a large coding inefficiency, it does cause one anomaly. Since the estimation of the q interval is independent of the current interval size, when the interval, A, is small only slightly larger than 0.5, the least probable subinterval can actually become

larger than the most probable subinterval. This can cause a fairly large inefficiency. Since this condition can be causally detected by a decoder, it can be corrected by both the encoder and decoder by using the least probable subinterval to encode the most probable symbol and using the most probable subinterval to encode the least probable symbol. This strategy is known as conditional exchange and it is used in the JBIG arithmetic encoder.

2 Description of the JBIG Image Compression Standard

In this chapter, a general description of the JBIG compression standard will be given. This is not intended to be a complete definition of the standard. Instead, each of the components of the algorithm will be discussed so that the reader has an understanding of the issues and complexities related to the algorithm. With this understanding, the reader should be aware of the need to develop efficient techniques for implementing the full algorithm.

This discussion is provided as an introduction to the optimization steps discussed in the next chapter. Since these optimizations only deal with issues in a JBIG compressor, the details of the decompressor are not discussed here. For information about the JBIG decompressor and a more detailed description of the JBIG compressor, the reader is referred to the JBIG specification [1].

2.1 Purpose of JBIG

Work on JBIG (Joint Bi-level Image Experts Group) began in 1988. Its purpose was to define a standard, CCITT T.82 and ISO 11544, for compressing binary image data for a wide range of applications. These applications include such areas as facsimile, audiographic teleconferencing, and image databases. To fulfill the requirements of these applications and many more, the algorithm needed to be flexible enough to perform well over a wide range of image types and imaging systems.

Prior to JBIG, the primary binary image compression standards in the world were the CCITT Group 3 [17] and Group 4 [18] standards. These standards are reviewed and compared with the JBIG standard by Urban [19]. They use a modified Huffman code to

one dimensionally encode runs of consecutive white and black pixels on every k^{th} line of the image. Between the one dimensionally encoded lines, the lines are processed by encoding differences between two consecutive lines using a technique known as Modified READ (Relative Element Address Designate). Since the Huffman codes used by these standards were designed specifically for text business documents, the algorithms perform well on that type of document. Halftoned images, however, have distinctly different characteristics from text documents. As a result, halftoned images are not compressed well by CCITT Group 3 and Group 4. In fact, the result of using Group 3 or Group 4 on a halftoned image is normally a larger file than the original, because these algorithms cannot adapt to the local statistics of a halftoned image. They often expand the document instead of compressing it.

JBIG was developed to be a significant improvement over Group 3 and Group 4. Specifically, it was desired that the JBIG algorithm provide the following improvements over the previous standards:

1. provide improved compression for text and line art documents (see section 4.3),

2. provide significant compression of halftoned documents (see section 4.3), and

3. provide the ability to transmit images progressively as well as sequentially (see section 2.2).

To accomplish these goals, it was necessary to develop an algorithm that was significantly more complex than Group 3 or Group 4. This extra complexity is possible because of advances in integrated circuit technology and microprocessor performance.

2.2 Sequential vs. Progressive Processing

In many image processing applications, an image is processed at full resolution from left to right and from top to bottom in a single pass. This type of processing is known as sequential processing. CCITT Group 3 and Group 4 are examples of sequential compression algorithms. Sequential applications are normally simpler and relatively inexpensive to implement because it is not necessary to store an entire image in a page buffer. The disadvantage of sequential compression algorithms is that image data must be processed at full resolution before a user can view the image. Since this can take a significant amount of time, it is not optimal for many applications.

In other applications, it can be useful to transmit a low resolution version of an image that can be displayed to the user almost immediately. Increasingly higher resolution versions of the image are transmitted so that the image displayed to the user is steadily improved. This type of processing is known as progressive processing. Progressive transmission of images is useful for two reasons:

1. **Improved user interaction:** Since a representation of the full image is presented to a user very quickly, it takes much less time for the user to recognize and the respond to the image. If the user is interested in the image, the user can wait for the full image to be transmitted. On the other hand, if the user is not interested in the image, it would be possible to terminate the transmission without waiting. This feature is very important in image browsing applications where a user is searching though a long list of available images.

2. **Supporting peripherals with differing resolutions:** Images can be compressed and stored in an image database and still used on a variety of peripherals at several different resolutions. For instance, an image could be extracted from a database for display on a 75 DPI CRT display very quickly by ignoring the higher resolution ver-

sion of the image. The same image could be printed at appropriate resolutions on a 300 DPI printer or a 600 DPI printer.

The disadvantage of progressive algorithms is that they require a page buffer to store the entire image. This adds to the cost and complexity of the system.

The algorithm defined by the JBIG standard can operate both sequentially and progressively. In addition, JBIG provides a mode known as progressive-compatible sequential mode. In this mode, an image is divided into horizontal sections called stripes. The stripe is decomposed into the required resolution layers and compressed as in progressive mode. The advantage of this mode is that it only requires enough memory to buffer a stripe instead of the entire image. The compressed data for progressive and progressive-compatible sequential modes is exactly the same; it is simply stored in a different order. As a result, images can be stored in an image database and transmitted in progressive mode to a CRT or in progressive-compatible sequential mode to a binary printer.

The choice of progressive or sequential processing is dependent on the needs of the application. It usually has very little effect on the overall compression rate of an image, although progressive processing does require more processing time because each of the resolution layers must be compressed.

When progressive processing is used, a new low resolution pixel is created for each 4 high resolution pixels. Figure 6 shows how this is normally represented notationally.

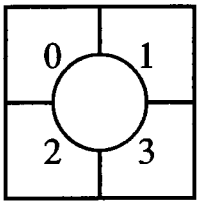


Figure 6 Four phases of high resolution pixels

The circle represents a low resolution pixel and the four squares represent the four high resolution pixels associated with it. The position of the high resolution pixel relative to the low resolution pixel is known as the spatial phase of the high resolution pixel.

2.3 Resolution Reduction

For the JBIG algorithm to compress progressively, it is first necessary to create each of the low resolution images. To create a low resolution image, a high resolution image is processed by an algorithm that creates a new image that is half the height and half the width of the high resolution image. This process is repeated for each new resolution that is needed. Figure 7 illustrates this process for two reductions. Each of these images is then compressed. A decompressor begins by decoding the low resolution image and builds up to the desired high resolution image.

The JBIG specification defines a default resolution reduction algorithm, although it is possible to use a custom algorithm to do the reduction..

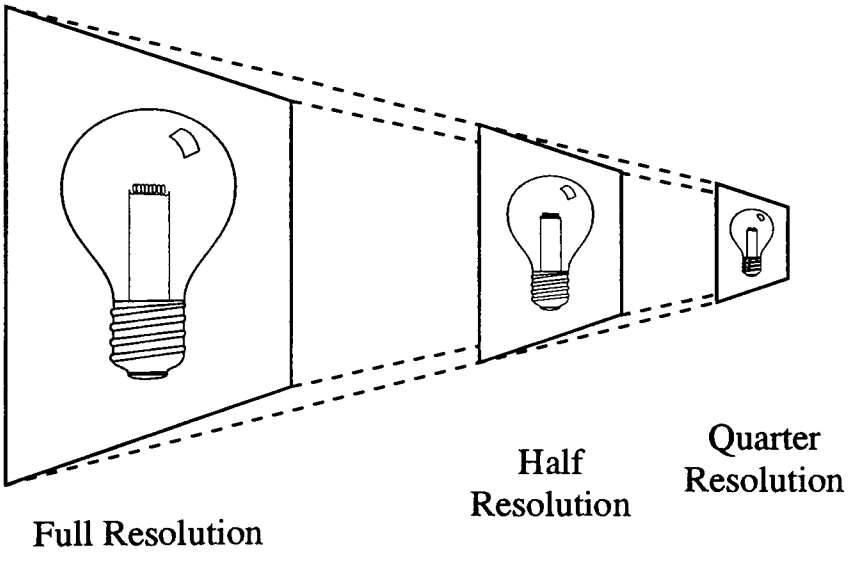


Figure 7 Resolution Reduction of an image

The default resolution reduction algorithm uses a table to generate new low resolution pixels so that the implementation is efficient. It is based on a filter that is designed to preserve the average density of any area. Some exceptions to the filter design are made to the table to improve the quality of the resulting low resolution images. The exceptions are made to improve the ability of the algorithm to preserves edges, lines, periodic patterns and dither patterns. Pixels in the neighborhood of the new low resolution pixel are used to index the resolution reduction table. The order of the pixels is shown in figure 8. Examples of low resolution images are shown in Appendix C.2.

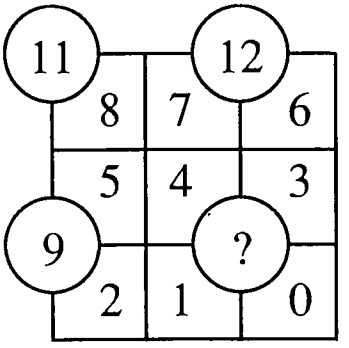


Figure 8 Context for Resolution Reduction

2.4 Prediction

The JBIG standard uses three different algorithms to predict pixels. One for the lowest resolution layer and two for differential layers (all layers but the lowest resolution). If one or more of these algorithms determines that a pixel is predictable, then it is not encoded because it can be predicted by the decoder. Each of these prediction algorithms can be enabled or disabled depending on the desired complexity of an implementation.

Two of these prediction algorithms, one for the lowest resolution layer, and one for all other layers, are known as typical predictors. They are called "typical" because they predict events that are typically true, but not always true. At the beginning of each line, a pseudo-pixel is encoded to tell the decoder whether or not the condition is true for the

entire line. If the condition is true, then the algorithm can predict pixels; otherwise, the algorithm is inactive for that line.

The third prediction algorithm is known as deterministic prediction. This algorithm uses a knowledge of the resolution reduction to determine when only one value is possible for a pixel.

2.4.1 Differential Layer Typical Prediction

The purpose of differential layer typical prediction (DTP) is to predict solid areas of an image. The algorithm works on the assumption that if all of the pixels in a three by three neighborhood of a low resolution image are the same color, then the four high resolution pixels associated with the center low resolution pixel are the same color as the neighborhood. Figure 9 illustrates the pixels used in this assumption. While this is not always true, it is valid most of the time. When differential layer typical prediction is enabled the assumption is tested for two lines at a time to determine if it is valid for that pair of lines (two lines are used because all four high resolution pixels are tested). If the assumption is true, then the lines are said to be typical. A pseudo-pixel is then encoded

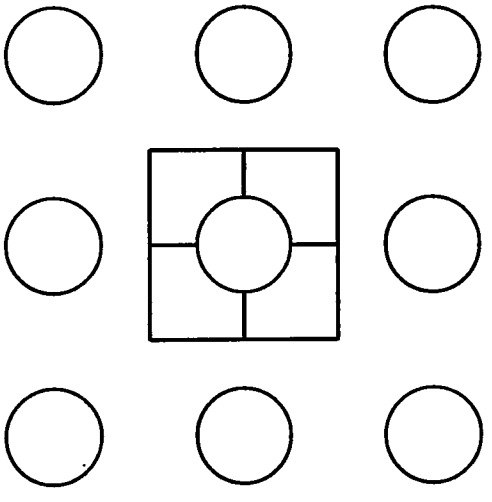


Figure 9 Context for differential layer typical prediction.

at the beginning of the first line to tell the decoder whether or not the assumption is valid.

When a pair of image lines is found to be typical, it means that this is a solid area, and all of the pixels can be skipped. The decoder will know how to generate the pixels just by reading the value of the pseudo-pixel. Since most documents contain a large amount of solid area (especially white background), many of the pixels in the document can be skipped. As a result, software implementations of the algorithm will run much faster. This is the entire purpose of the DTP algorithm.

In general, differential layer typical prediction has very little effect on the compression performance of the JBIG algorithm. In fact, enabling this feature often reduces the compression ratio slightly. The reason for this is that the pixels that are skipped carry almost no information (which is why the DTP assumption is normally valid). Removing these pixels has almost no effect on the overall entropy of the document. However, for each pair of lines a new pseudo pixel is encoded. The probabilities associated with this pseudo-pixel are much less biased than the skipped pixels, and therefore, carry more information. As a result, the total entropy of the document does not change significantly.

2.4.2 Lowest Resolution Typical Prediction

The algorithm used for typical prediction in the differential layer assumes the presence of a lower resolution image. When the lowest resolution layer is encoded (or the only resolution layer for sequential processing), a separate algorithm is needed. In this case, JBIG defines a lowest resolution layer typical prediction algorithm (also known as base layer typical prediction or BTP) that simply tests two consecutive lines to determine if they are exactly the same. If the lines are found to be the same, a pseudo-pixel is encoded to transmit the information, and the entire line is skipped with no further process-

ing. If the lines are not the same, then the pseudo-pixel is encoded differently, and the rest of the line is processed normally.

Typical prediction is most useful for processing blank lines within the image. For this reason a high percentage of the pixels in most simple business documents do not need to be processed. As with differential layer typical prediction, this greatly improves the speed of a software implementation, but is has little effect on the compression ratios of the algorithm.

2.4.3 Deterministic Prediction

The purpose of deterministic prediction (DP) is to remove the necessity of encoding redundant information between a low resolution image and a high resolution image. Since the low resolution image must be similar in appearance to the high resolution image, there are cases where high resolution pixels can be identified as having only one possible value. This can be done because a known algorithm is used to do the resolution reduction. As an example of why this is valid, consider a simple resolution reduction algorithm that will create a black low resolution dot whenever two or more high resolution dots are black. If in a given area, the low resolution pixel is white, and one black high resolution pixel is decoded, then all of the rest of the pixels must be white. If any of the other pixels were black, then the low resolution pixel would have been black. While this example assumes a very simple resolution reduction algorithm, the basic concept can be applied for any reduction algorithm. In fact, for the default resolution reduction algorithm, most images contain a significant number of pixels that are predictable by the deterministic prediction algorithm.

The deterministic prediction algorithm is table driven so that it can be implemented efficiently. Separate tables are used for each of the four high resolution pixels associated with a low resolution pixel. To use these tables, an index is built using the same pixels

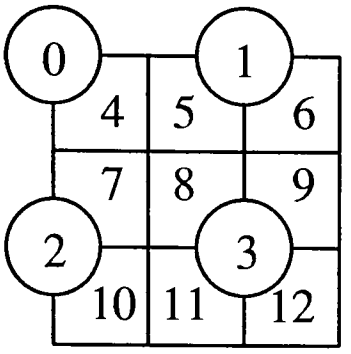


Figure 10 Context for Deterministic Prediction

that were used by the resolution algorithm. In figure 10, the pixels that are used to build the index are shown along with the order in which the bits are placed in the index. The pixels numbered 8, 9, 11, and 12 are the pixels being predicted. For each pixel, only the previously transmitted pixels are used. Specifically, for pixel 8, the index is constructed from pixels 0-7, for pixel 9, the index is constructed from pixels 0-8, for pixel 11, the index is constructed from pixels 0-10, and for pixel 12, the index is constructed from pixels 0-11. The JBIG standard defines the four deterministic prediction tables that will work with the default resolution reduction algorithm. In these tables, a large number of the entries are for predictable pixels. Table 1 defines the number of context states possible for each pixel and how many of these states represent predictable pixels.

Target Pixel	Number of Context States	Number of States with Predictable Pixels
8	256	20
9	512	108
11	2048	526
12	4096	1044

Table 1 Number of deterministically predictable states in the default DP tables

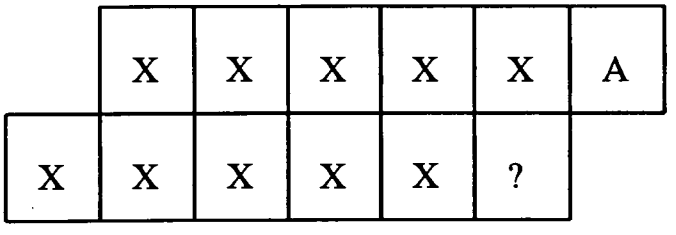
It is also possible to transmit a set of custom DP tables as a part of the image header defined by the standard. This is required if a non-standard resolution reduction algorithm is used.

The main purpose of deterministic prediction is to improve the compression performance of the algorithm. Even though DP predicts only a very few pixels relative to the large number predicted by typical prediction, it has a much larger effect on the overall compression ratio. To understand why this is the case, consider the pixels that are predicted by the deterministic prediction algorithm. Many of the pixels predicted by DP do not have highly skewed probabilities. In other words, the pixels predicted by DP carry more information than the TP pixels, so more bits would be required to encode them. Since these bits no longer need to be transmitted, and no extra information needs to be encoded as in the typical prediction algorithm, the compression ratio improves. The JBIG specification suggests that using DP will provide coding gains of about 7%.

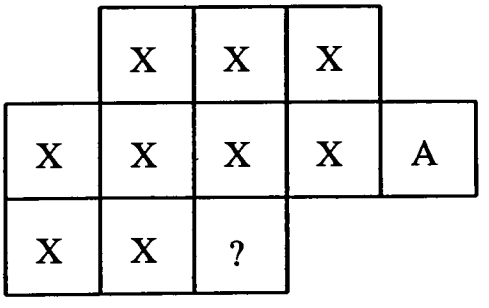
2.5 Probability Estimation

Within the JBIG standard probabilities are conditionally estimated based on a group of pixels in the neighborhood of the current pixel. This is similar to the Markov model discussed in section 1.2. One of the problems with a Markov model is that the conditional probabilities that are used are not consistent from image to image. To overcome this problem, the JBIG standard defines a method of adaptive probability estimation so that it can learn the correct probabilities as it processes an image

Within JBIG, a neighborhood of 10 pixels is defined around the pixel being encoded. These 10 pixels are designed to be the 10 pixels most correlated with the current pixel. There are six different encoding contexts defined in JBIG for different encoding conditions. These contexts are graphically represented in figures 11 and 12. In each context, there is a pixel mark A. The purpose of this pixel will be discussed in section 2.6.



Two Line Context



Three Line Context

Figure 11 Encoding contexts for the lowest resolution layer.

When encoding the base layer, there are two possible encoding contexts, a two line context, and a three line context. The JBIG standard suggests that the two line context will allow software to run somewhat faster while the three line context will improve the encoding by about 5%.

When encoding the differential resolution layers, there are four different contexts that are used. The choice of context is dependent on the position (or phase) of the high resolution pixel being encoded relative to its corresponding low resolution pixel. The values

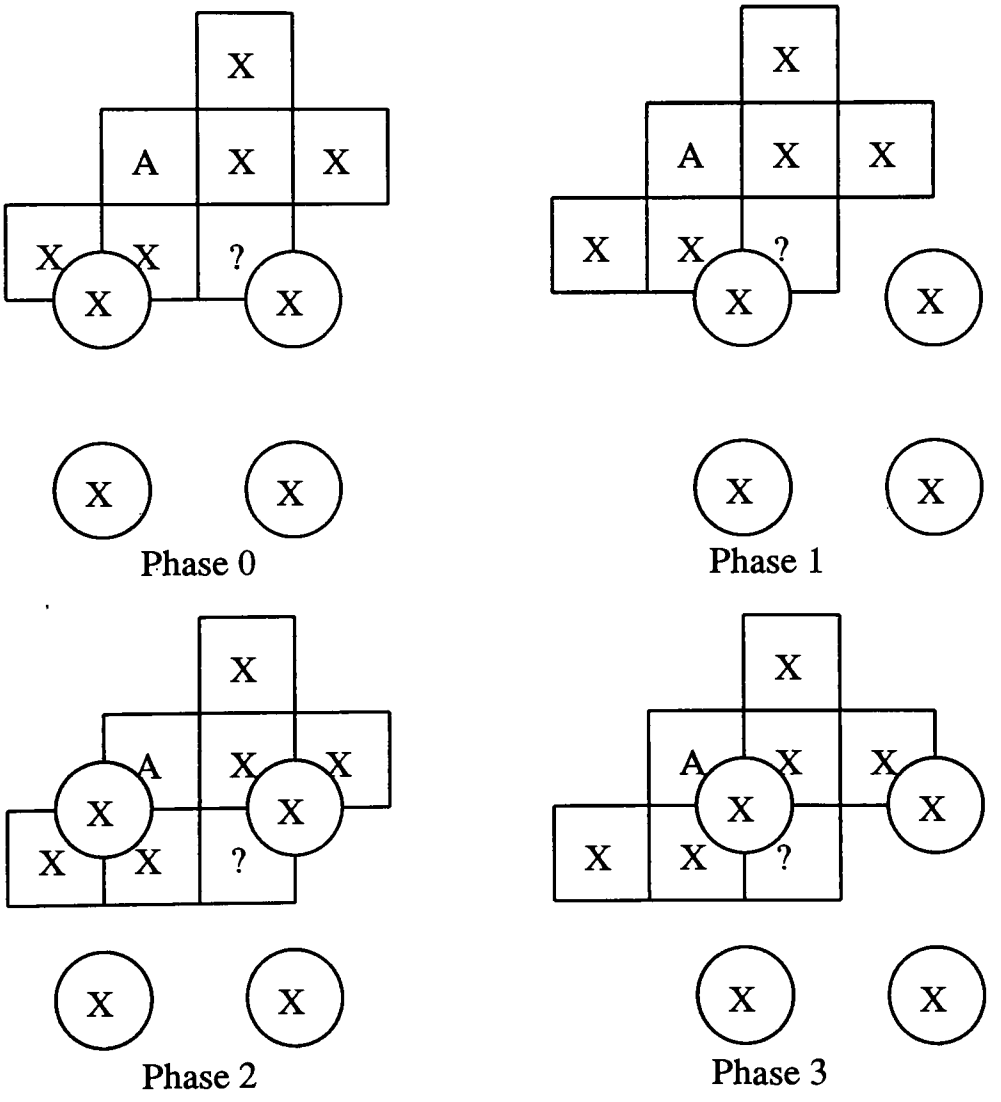


Figure 12 Encoding contexts for differential layers.

of the 10 pixels in the neighborhood are combined with the two bit value of the phase of the pixel to form a 12 bit context. Within each resolution layer (and also within each bit plane for multibit images), the algorithm keeps track of the current most probable symbol (MPS) and the current probability state (ST) for every context. The standard defines a probability estimation state table with 113 possible states. For each state in the table, an estimation of the least probable symbol subinterval width (LSZ) is stored. LSZ is a 16 bit representation of the value q_e defined in equation 8 on page 14. Also included in the state table is the next probability estimation state to be used for a context when the encoded value is the most probable symbol (NMPS), and the state to use next when the least probable symbol is encoded (NLPS). The probability estimation state changes each time a renormalization is performed. The last value in the state table also is the value, SWTCH that is used to indicate that the value of MPS should be inverted if NLPS occurs.

Figure 13 represents the flow of data in the probability estimation process for one resolution layer or bit plane. Prior to encoding an image, each context, for each layer and bit plane, is initialized so that the most probable symbol is 0, but the probabilities for the two possible symbols are nearly equal. When a pixel is encoded, the value of the pixel

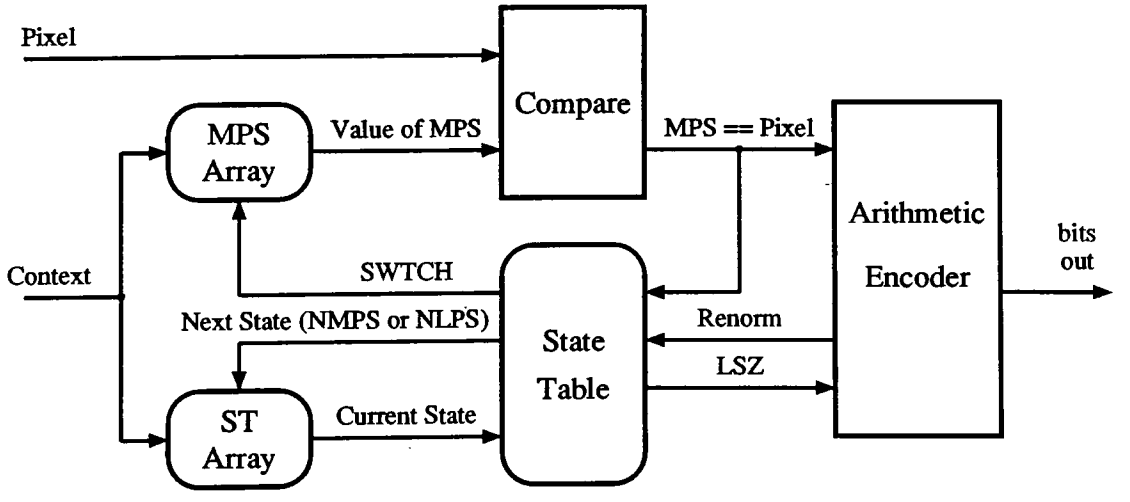


Figure 13 Structure of probability estimation

and it's context word (10 or 12 bits) are used to determine the probability interval to be encoded by an arithmetic encoder. The context is used as an index into arrays that hold the current state and most probable symbol for that context. The state is used as an index into the state table to find the least probable subinterval (LSZ), and the pixel value is compared with the MPS. These values are passed to an arithmetic encoder that generates an output stream of bits. Whenever a renormalization occurs in the arithmetic encoder, the probability state for that context is updated. The value of the current state indexes the state table to look up either NMPS or NLPS depending on whether an MPS was encoded or not. When the next state is NLPS, the value of SWTCH is also accessed. If SWTCH is 1 then the value of the MPS array for the current context is inverted. This process is repeated for each pixel in the image. In this way the algorithm learns the correct probabilities for each possible context.

2.6 Adaptive Templates

The JBIG standard allows for a small amount of adaptivity in the contexts used for probability estimation. This feature is known as adaptive templates (AT). The primary motivation for this feature is to improve the ability of the algorithm to encode halftoned images. Halftoned images are the result of using a binary printing process to represent gray scale images. They occur often in compression applications as a result of either scanning a previously halftoned image or from capturing a continuous tone image and thresholding it with a periodic pattern to simulate gray levels [20]. The important feature of halftoned images is that they usually involve a periodic pattern. In terms of a Markov context for compressing an image it means that the pixels most correlated with the current pixel can be several pixels away.

In each of the contexts used for probability estimation shown in figures 11 and 12, one pixel is marked with an A. The locations shown for A represent the default locations

only. The standard allows an encoder to move this pixel to another location if the encoder believes that doing so will improve the compression. It should be noted that this feature should be used with care, because each time the AT pixel is moved, the probability estimation arrays are reset to their initial values. They must relearn the probabilities for the new contexts.

When an AT movement is specified, it only applies to the current resolution layer (and bit plane). According to the standard, the AT pixel can be moved to any previously transmitted pixel position within ±127 pixels in the horizontal direction, and 255 lines vertically. However, in practice, it is suggested that encoders limit the AT pixel movement to the current line and the previous eight pixels.

An algorithm to determine when and where the AT pixel should move is suggested in the standard, although any algorithm can be used. It is also possible to use no algorithm at all.

2.7 Arithmetic Coding

The heart of the JBIG algorithm is an arithmetic entropy coder. It encodes the symbols and probabilities that were described in section 2.5 for all of the unpredictable pixels as well as the pseudo-pixel generated by the typical prediction algorithm. The output of the arithmetic encoder is a stream of bits that make up the compressed image that can be stored or transmitted depending on the application.

The algorithm used for arithmetic encoding is the same as the algorithm described in section 1.4.

3 Optimizing JBIG Software

The goals of the JBIG committee were to develop an algorithm that would provide improved compression over a wide range of documents as previously discussed. This could not be done without creating an algorithm that was significantly more complex than the previous CCITT standards, because these standards were relatively simple algorithms. However, the new JBIG standard would not be effective if it could not be implemented in an extremely efficient manner.

A considerable amount of work went into creating an algorithm that could be implemented in either software or hardware efficiently. The result is an elegant algorithm that uses a series of tables to perform complicated functions such as resolution reduction, deterministic prediction, and probability estimation. Since few calculations are required, the resulting algorithm should be able to process pixels at relatively high speeds.

While the JBIG algorithm is capable of high speeds, little or nothing has been written to document software-specific techniques for improving the processing speed of the algorithm. As a result, software developers who don't have experience with compression algorithms could easily miss significant opportunities for implementing extremely fast software.

The purpose of this chapter is to document several techniques that have been used in an implementation of the JBIG algorithm to significantly improve execution time of the software. Throughout this work the following assumptions have been made:

- All software is written in ANSI C. No assembly language is used because the results would not be portable or generally useful.

- All optimizations are algorithmic in nature. The goal, in general, is to reduce the number of operations needed to perform a task. While it is possible to write software that takes advantage of the features of one specific platform, the final algorithm would only be useful to a relatively small audience.
- All changes are compatible with the existing JBIG standard. Even though other incompatible changes might be interesting, they would not be extremely useful.
- Memory is cheap. This means that it is valid to use significantly more memory than might normally be required, if doing so will reduce the number of required operations. The limit to this strategy is that the techniques developed should be useful for applications that run in PC environments with a 640K memory limitation.
- Image data will enter the algorithm in a one pixel per byte format. This should be valid for general purpose software since some applications will already have data in this format. Applications with a packed binary format will have to unpack the image data, but this is not considered part of the compression algorithm.

3.1 An Efficient Algorithm for Building JBIG Contexts

The largest obstacle to an efficient implementation of JBIG is the need to gather many pixels together to build large contexts for each part of JBIG. From looking at figures 8, 9, 10, 11, and 12, it can be seen that there are many different context patterns that may be required for any pixel. In fact, for any specific pixel, it may be required to build contexts for resolution reduction, typical prediction, deterministic prediction, and probability estimation, each of which requires the use of different neighboring pixels. This is especially true when encoding differential layers, although the techniques discussed in this section are also useful when encoding the lowest resolution layer.

It is important to realize that even though each algorithm uses a different neighborhood, there are many pixels that are common to several neighborhoods. In fact, the total number of pixels needed to build all possible contexts is fairly small. This can be seen by forming the union of all possible context bits. When doing this the adaptive pixel should be ignored because it must be handled separately, and the two line base layer template should be ignored as will be explained later. It is easy to see that the resulting union is three pixels high for both the low resolution data and the high resolution data.

3.1.1 Building Columns of Three Pixels

To facilitate building a context word, it is useful to group the pixels into columns of three pixels. Figure 14 shows how the three pixel values are arranged within a byte. Part of the reason for doing this is because it can be done very quickly and easily. The main reason, however, is that once it is done these columns can be very useful in building context words for both the high and low resolution pixels as shown in the next section. The code fragment in figure 15 shows an example of how this can be done. In this example, it is assumed that `pixel_data` is originally a two dimensional array of pixel values stored as one pixel per byte holding all the pixel data for one stripe of one resolu-

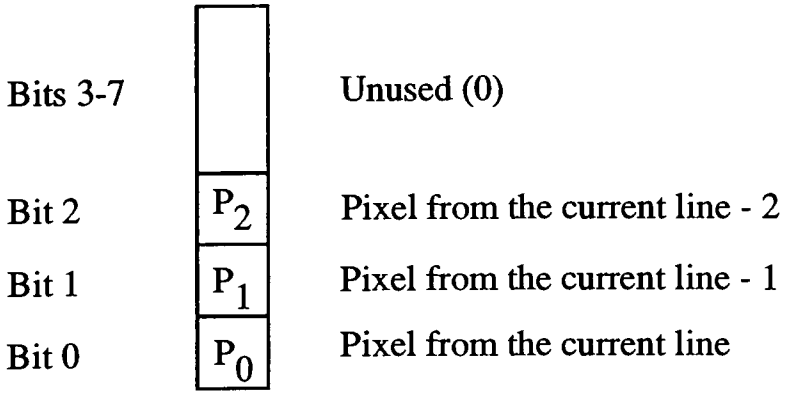


Figure 14 Pixels stored as a column of three pixels per byte

```
unsigned char pixels[STRIPE_HEIGHT][LINE_WIDTH];
int line;
register int i;
register long *current_line;
register long *previous_line;

previous_line = (long) (last line of the previous stripe)
for ( line = 0 ; line < STRIPE_HEIGHT ; line++ )
{
   current_line  = (long *)pixels[ line ];
   for ( i = (layer_width - 1)/4 ; i >= 0 ; i-- )
      current_line[i] |= (previous_line[i] << 1)
                         & 0x06060606;
   previous_line = current_line;
}
```

Figure 15 Code fragment for columnizing pixels

tion layer. After the loop is completed, each byte of the `pixels` array will hold three pixels in the format shown in figure 14.

Initially, `previous_line` points the the last line of the previous stripe which is already in the three pixel format (at the top of the image the previous line is all zeros). The pixels from the previous line are left shifted by one bit to position them correctly, and ANDed with the value 6 to remove the extra pixel. It is then ORed with the one bit pixel value for the current pixel to complete the calculation. In this example, pointers of type `long` are used to point to the pixel data. Using this technique, four pixels are processed each time through the loop. The operations required to process each group of four pixels are:

2 Array Reads
1 Array Write
1 Left Shift
1 AND
1 OR
6 Total Operations

Since four columns of three pixels are constructed at a cost of six operations, each column is built at a cost of about 1.5 operations. Once it is built, it can be used to build contexts as part of a high resolution layer and as part of a low resolution layer.

3.1.2 Building Common Context Words

As previously stated, when building the contexts for a pixel that is being encoded, it is important to realize that many of the pixels are used in more than one context. To take advantage of this, a union of all possible contexts (excluding the AT pixel and the two line base layer context) is made for both the high and low resolution layers. It can also be seen that most of the pixels used in the common contexts for one pixel are also present in the contexts for the next pixel.

```
unsigned char *lo_line, *hi_line;
int lo_context, hi_context0, hi_context1;
int lo_x, hi_x;

lo_line = array of columnized pixel data for low resolution;
hi_line = array of columnized pixel data for high resolution;
lo_context  = lo_line[0];
hi_context1 = hi_line[0];
for ( hi_x = 1, lo_x = 1 ;
      hi_x <= layer_width ;
      hi_x += 2, lo_x++  )
{
   lo_context  = (lo_context << 3)
               | lo_line[lo_x];
   hi_context0 = (hi_context1 << 3)
               | hi_line[hi_x];
   hi_context1 = (hi_context0 << 3)
               | hi_line[hi_x+1];

   encode first pixel using lo_context and hi_context0

   encode second pixel using lo_context and hi_context1
}
```

Figure 16 Code fragment for building common context words

These facts can be used to efficiently build common context words. Consider the code fragment for encoding differential layer data in figure 16. In this example, it is assumed that `lo_line` and `hi_line` both point to arrays of three pixel columns as discussed in the previous section. The common context bits for the high resolution layer are stored in `hi_context0` (phase 0) and `hi_context1` (phase 1) while the common context bits for the low resolution layer are stored in `lo_context`. Each time a new context word is to be encoded the previous context word is shifted left by three to make room for the next column of three pixels. The new column is then ORed into the context. The bit order for these common context words is shown in figure 17, where the pixel currently being encoded is pixel 3 in the high resolution word, and its associated low resolution pixel is pixel 4 in the low resolution word. It is worth noting that the pixel ordering of these words is significantly different from the pixel ordering in the for most of the contexts in the JBIG specification. These differences will be accounted for in the next section.

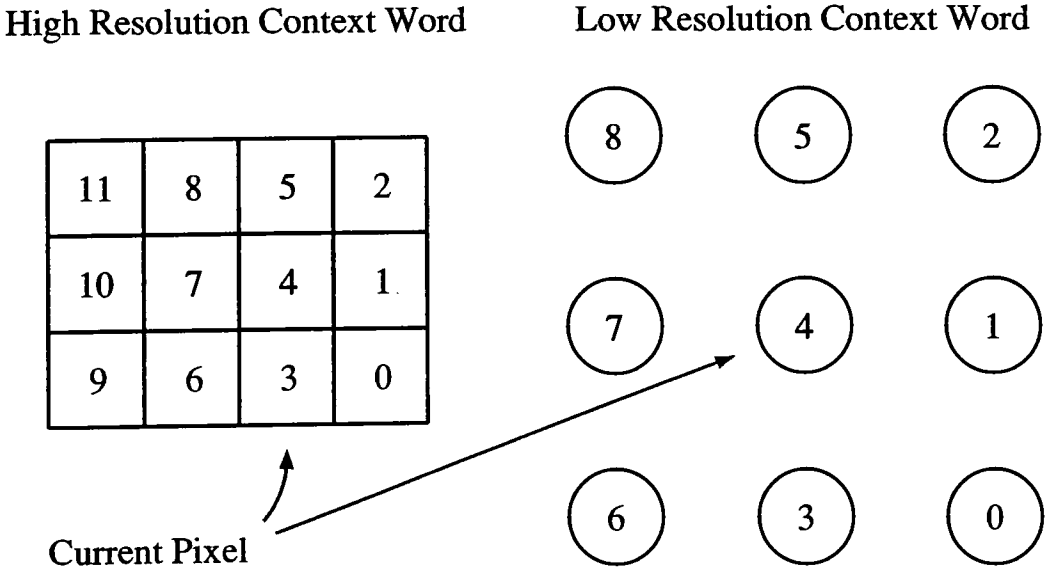


Figure 17 Pixel order for common context words.

In the example, two pixels are processed for each low resolution context; however, it is possible to encode two lines in a differential layer image in a single pass (see section 3.2). In this case, the low resolution context only needs to be moved once for every four high resolution pixels processed. The resulting software performs four high resolution context shifts and one low resolution context shift each time through the loop to encode four pixels. Therefore, the total number of operations required for building the context words for each group of four high resolution pixels is

5 Array Reads
5 Left Shifts
5 ORs
15 Total Operations

This is equivalent to 3.75 context building operations per pixel.

The two line base level context is an exception to the procedures defined in this section. In the JBIG specification, it is suggested that using the two line context will improve the software execution time at a cost of about 5% loss in compression. While this may be true in some cases, the third row of pixels is available for free when using the columnizing technique shown in the previous section. If these techniques are used, the two pixel context will actually slow down the software because each time through the loop, two pixels need to be ANDed out of the three pixel column. This could be solved for sequential mode operation by only storing columns of two pixel columns, but then the advantages of this technique are reduced. Storing only two pixel columns is not a valid option for the lowest resolution layer in a progressive transmission because when it is used to help encode the next higher resolution layer, the three pixel column will be required.

3.1.3 Using Tables to Build Reordered JBIG Contexts

Once the common context words have been built for the low and high resolution layers, it is necessary to group the required bits from each layer for the desired contexts. Figure 18 shows an example of how this can be done. In this fragment, unused bits in the high context word are masked off leaving only the required high resolution bits. A table is then used to rearrange the desired low resolution bits so that they can be ORed into the unused bits of the high resolution context.

The bit order of the resulting contexts can be non-obvious. Figure 19 shows an example of this for the probability estimation context of a phase 3 pixel. This is particularly important to note for the resolution reduction contexts and the deterministic prediction

```
#define DP_MASK       0x0ff0
#define ENCODE_MASK   0x027a

lo_context      &= 0x01ff;
DP_context       = (hi_context & DP_MASK) |
                   DP_lo_context[ lo_context ];
encode_context3 = (hi_context & ENCODE_MASK)  |
                   encode_lo_context3[ lo_context ] |
                   (AT_pixel[i] & 1);
```

Figure 18 Example code fragments for combining high and low resolution context words.

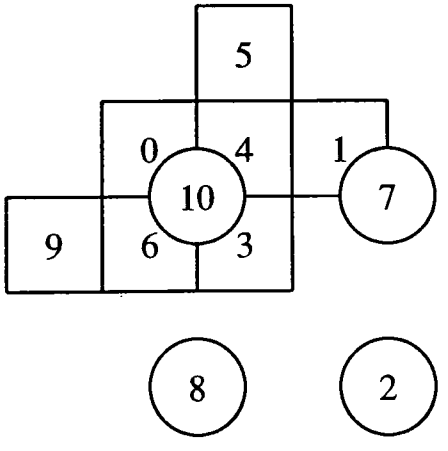


Figure 19 Bit order of probability estimation context for differential layer encoding (Phase 3).

context because these contexts are used to index predefined tables. Therefore, if this procedure is used, the predefined tables must be rearranged to use the new reordered JBIG contexts. Procedures to do this are discussed in appendix B.2.

For the probability estimation contexts, there are a few interesting things to note:

- Context bits do not need to be reordered because JBIG does not define a default order. The reason for this is that the probability is tracked for each context state regardless of its value.
- The AT pixel is handled separately from other pixels even though its default position is inside the common context block. This allows the AT pixel to move to any other position without having to changing the low context table. If it is known that the AT pixel will never move, the software could be designed to run faster.
- The pixel being encoded is included in the context. This will be explained in section 3.2.
- The probability estimation context is supposed to include two bits of pixel phase information. The low context table can be used to place them in bits 11 and 12 of the context word. This means that a separate table (512 words) is needed for each phase.

Each time a context is built, this procedure requires the following operations:

1 Array Read
1 AND
1 OR
3 Total Operations

3.2 Encoding Both Differential Layer Lines in One Pass

In JBIG, differential layer images are encoded in the same order in which sequential images are encoded, from top to bottom and from left to right. However, since each line in

a low resolution image corresponds to two lines in a high resolution image, there are many redundant operations if the two high resolution lines are processed separately. The reason for this is that the low resolution context is the same for both lines. If both lines are processed separately, then the low resolution context needs to slide along the line twice instead of just once. In the same way, the typical prediction pixels need to be found twice.

A more efficient method is to process both high resolution lines of the image in a single pass. Figure 20 shows how this can be done. The routines `process_lines_tp` and `process_lines_notp` are similar to the routine shown in figure 16 except that they are called to process two image lines with a single loop. It creates four high resolution contexts and one low resolution context in each pass through the loop. For each pixel, the probability estimation contexts are placed in a list to be encoded later. A separate routine, `arithmetic_encode`, is then called to encode each list of contexts in the proper sequential order.

```
short context_line1[MAX_WIDTH],
      context_line2[MAX_WIDTH];
int length1, length2, tp;

tp = LINE_NOT_TYPICAL;
if ( typical prediction is enabled )
{
  tp = process_lines_tp( context_line1, &length1,
                         context_line2, &length2 );
  arithmetic_encode( &tp_pixel_context[tp], 1 );
}
if ( tp == LINE_NOT_TYPICAL )
{
  process_lines_notp( context_line1, &length1,
                      context_line2, &length2 );
}
arithmetic_encode( context_line1, length1 );
arithmetic_encode( context_line2, length2 );
```

Figure 20 Software example for processing two lines in one pass

Since the routine `arithmetic_encode` also needs to know the value of the current pixel, it is left in the context word as shown in figure 19. Otherwise, the pixel would have to be extracted in the `process_lines_xx` routines and stored in a separate array.

In the example, it is worth noting that the function `process_lines_tp` is called to attempt to process lines with typical prediction. If, during processing, the typical prediction assumption is found to be false, no harm has been done because nothing has been encoded yet. The contexts in the arrays `context_line1` and `context_line2`. are simply discarded, and the line is reprocessed by the function `process_lines_notp`.

3.3 Moving Efficiently through Solid Areas

In most documents, a significant portion of document area is a solid color. For text documents, much of the document is simply white background. There can also be a significant amount of solid black area in large characters and logos. Even for halftoned images, most contain solid white or black regions, not to mention the white border that is usually around an image.

In general, it is possible to take advantage of solid areas in compression algorithms because the contexts do not change from one pixel to the next. In most cases, it is possible to create very efficient loops that simply check to see if all of the new context pixels are the same color as the previous solid area. If they are, then the current pixel can be encoded exactly the same as the previous pixel.

3.3.1 Resolution Reduction in Solid Areas

The default JBIG resolution reduction algorithm is used to generate new low resolution pixels from a high resolution image. It uses a context of the nine pixels from the high resolution plane and three pixels from the low resolution plane. If all of the pixels used in a context are zero, of course the new pixel that is generated is also a zero. Likewise, if all of the pixels are one, then the new pixel is also a one. If either of these cases occur, and all of the new context pixels for processing the next pixel from both planes are the same value, then the next context must be the same as the previous context, and the new pixel is the same as the previous pixel.

Figure 21 shows an example of how the resolution reduction algorithm can be optimized by the assumption that solid areas are common. In this example, the pointer `hi` points to a row of columnized high resolution data that corresponds to the context pixels for the high resolution row being generated, `prev_lo` points to the previous row of pixel data for the low resolution layer, and `lo` points the array of new pixels that will be generated. `RR_context` is the resolution reduction context word and is initialized to zero. The new line of image data is generated by looping through the line two pixels at a time until the end is reached.

Within the loop, `RR_context` is tested for the two solid area values, `SOLID_WHITE` and `SOLID_BLACK`. The solid white context is tested first since it is the most likely value for most documents. If the area is found to be solid, then all of the new context bits for the next two pixels are tested to see if they have the same value. If they agree, then the new low resolution pixels are known immediately. Since the context doesn't change, the next group of two pixels can be tested immediately, and the process continues as long as a solid area is detected or until the end of the line is reached. As soon as any of the new context pixels for either of the two new pixels does not agree with the

```
register unsigned char *hi, *lo, *prev_lo;
register int i;
int RR_context;

hi = array of columnized pixels for current high resolution line;
lo = array of columnized pixels for current low resolution line;
prev_lo = array of columnized pixels for previous low resolution line;
line_width = width of the high resolution line;
RR_context = 0;
for ( i = (line_width-1)/4 ; i >= 0 ; i-- )
{
  if ( RR_context == SOLID_WHITE )
  {
    while( *((long *)(hi)) == 0 &&
         (*((short *)(prev_lo)) & 0x0303) == 0) )
    {
      *((short *)(lo)) = 0;
      hi += 4;
      lo += 2;
      prev_lo += 2;
      if ( --i < 0 )
        goto end_loop;
    }
  }
  else if ( RR_context == SOLID_BLACK )
  {
    while( *((long *)(hi)) == 0x07070707 &&
         (*((short *)(prev_lo)) & 0x0303) == 0x0303) )
    {
      *((short *)(lo)) = 0x0707;
      hi += 4;
      lo += 2;
      prev_lo += 2;
      if ( --i < 0 )
        goto end_loop;
    }
  }

  generate two low resolution pixels

}
endloop:
```

Figure 21 Efficient resolution reduction in solid areas

previous context, the loop is exited and both new pixels are processed in the conventional manner described in the standard.

Since most of documents have large solid areas, a high percentage of the pixels in the document will be processed in the solid area loops. The goal, therefore, is to create loops that can run through large solid areas very quickly. For this reason, two pixels at a time are processed. By casting the `hi` pointer as a long pointer, it is possible to access and test four columns of high resolution pixels at once. Similarly, casting the `prev_lo` and `lo` pointers as short pointers, makes it possible to assess two columns of low resolution data at once. This allows the algorithm to take full advantage of processors with 16 or 32 bit data paths.

It should be noted that discretion should be used when casting unsigned character pointers as other data types. While this will work on some machines, it may not on others. When casting to a short pointer, care must be taken to make sure that the pointer value is on a valid 16 bit boundary in memory. Likewise, when casting to a long pointer, the pointer value must be on a valid 32 bit boundary. Since this example moves through the image line by steps of four in the high resolution layer, and by steps of two in the low resolution layer, it will work as long as each line begins on a valid 32 bit boundary. In this implementation of JBIG, memory for each line of the image within a stripe was allocated using the `malloc` function with 16 extra pixels at the beginning of the line to allow AT pixel movement of up to 16 pixels. Since `malloc` always returns a pointer that can be used by a long integer, and 16 is a multiple of 4, each line must start at a valid location.

3.3.2 Differential Layer Typical Prediction in Solid Areas

The purpose of typical prediction in the JBIG algorithm is to reduce the number of pixels that need to be encoded by allowing solid areas in differential images to be skipped.

While this greatly reduces the processing requirements for the algorithm, even more dramatic improvements can be made by designing the software to loop efficiently through these large solid areas.

Figure 22 shows an example of how the typical prediction algorithm can be implemented. In this example, two lines of the image are processed at once and four pixels (two from each line) are processed each time through the loop. At the beginning of the routine, the local variables are initialized as shown in figure 16 on page 36 except that two high resolution line pointers and contexts are initialized. Within the loop, the low resolution context for a pixel is built, and it is tested to see if it represents a solid area. If it does, the four high resolution pixels must be tested to see if they have the same value; if they do, they are skipped, and if not, TP assumption is invalid, so the routine is exited.

The important feature of typical prediction that is used in figure 22 is that when a block of typical prediction pixels are found, there is a high probability that the next block will also be a typical prediction block. Since the low resolution context word does not change after a typical prediction block, the next block of pixels can be tested immediately. The fragment shown in figure 22 includes a very tight loop that repeatedly tests a series of four pixel blocks until a non-solid area is found. Since the loop does not update the context words or the line pointers after each pixel, they must be updated at the end of the loop so that they will be ready to process the next pixels.

It is important to note that the typical prediction loop does not test for the end of the image line. This is handled by allocating extra pixels at the end of the line and forcing a non-typical condition at the end of the line that forces the loop to exit. At that point, the end of line condition is tested.

```
register int i;
register unsigned short *tp_ptr;
register unsigned char *lo_line;
unsigned char *hi_line1, *hi_line2;
int lo_context;
int hi_context0, hi_context1;
int hi_context2, hi_context3;

initialize local variables

for ( i = 0 ; i < line_width ; i++ )
{
  lo_context = ( (lo_context << 3) |
                 (int)(*lo_line++) ) & 0x01ff;

  if ( lo_context == 0 )           /* solid white */
  {
    tp_ptr = (unsigned short *) &hi_line2[-1];
    do
    {
      if ( *tp_ptr++ & 0x0303 )
      {
        if ( i >= width ) break;
        return LINE_NOT_TYPICAL;
      }
      i += 2;
    } while ( *lo_line++ == 0 );
    if ( i >= width ) break;

    set the high resolution line pointers to the current location;
    reset all the context words to prepare for encoding the next pixel;
  }

  else if ( lo_context == 0x01ff ) /* solid black */
  {
    similar to the solid white case;
  }

  build the probability contexts for the four pixels;
}
return LINE_TYPICAL;
```

Figure 22 Efficient typical prediction in solid areas

The example shown in this section corresponds to the function `process_lines_tp` discussed in section 3.2. When a line is found to not be typical, it returns the value LINE_NOT_TYPICAL. If the entire line is processed with the TP assumption valid, then the value LINE_TYPICAL is returned.

As in the example for efficient resolution reduction (figure 21 on page 44), the pointer to the high resolution line is cast to a short pointer so that two pixels at a time can be tested. It would be even more efficient to cast the high resolution pointer as a long pointer, and the low resolution pointer to a short pointer so that two pixels at a time could be tested. However, when a valid long pointer is used to point to four high resolution pixel columns, the pointer to the necessary two low resolution pixel columns is not on an even boundary. While this will work for some machines, it is completely invalid on others. Since it is not generally applicable, it is not used here.

3.3.3 Arithmetic Coding in Solid Areas (Sequential Mode Only)

It is also possible to use the characteristics of solid areas to optimize the algorithm for arithmetic coding. The basis for this optimization is the idea that when the same context is used for several consecutive pixels, several steps can be skipped in the basic arithmetic loop. The problem with this is that it is not useful when compressing differential resolution layers. When processing differential layers, the phase of the pixel is included in the context so that two consecutive pixels, in general, do not have the same context. Even if this were not the case, it still would not be particularly useful because it would be most effective in solid areas which are removed by the typical prediction algorithm in differential layers.

In sequential images, or in the lowest resolution layer of progressive images, it is possible to use a solid area optimization. This is because there is no phase information to be included in the context, and there are still large solid areas remaining after lowest layer

```
register int context_bits, context, old_context;
register unsigned char *line, *AT_pixel;
int i, lsz;
struct state_data **state_table, **state_ptr, *state;

for ( i = line_width-1 ; i >= 0 ; i-- )
{
  context_bits = (context_bits << 3) | (int)(*line++);
  context = (context_bits & 0x07fe) |
            (*AT_pixel++ & 1);
EXIT_LOOP:
  state_ptr = &state_table[ context & 0x7f7 ];
  state = *state_ptr;
  lsz = state->lsz;
  if ( ((context_bits & 8) >> 3) == state->mps )
  {
QUICK_LOOP:
    A_reg -= lsz;
    if ( A_reg < 0x8000 )
    {
      check for conditional exchange and renormalize;
    }
    else
    {
      if ( --i >= 0 )
      {
        context_bits = (context_bits << 3) |
                       (int)(*line++);
        old_context = context;
        context = (context_bits & 0x07fe) |
                  (*AT_pixel++ & 1);
        if ( old_context == context )
          goto QUICK_LOOP;
        goto EXIT_LOOP;
      }
    }
  }
  else
  {
    encode a least probable symbol;
  }
}
```

Figure 23 Efficient arithmetic coding in solid areas

typical prediction. As a result, a significant number of lowest resolution pixels are encoded with the same solid area context. If this is detected in the arithmetic encoding loop, then several steps can be saved.

Figure 23 shows an example of how this can be done. In this example, a line of lowest resolution image data is encoded using a three line context. For each pixel, a context is built for estimating probability of the pixel. This is used to find the estimate of the least probable interval, `lsz`. At this point, the JBIG specification supplies a simple flow chart of a well defined procedure for arithmetic encoding. While this procedure is computationally efficient, it ignores the performance enhancement of efficiently encoding solid areas.

In the example, the highlighted area represents a modification to the standard JBIG flow chart. If a most probable symbol is encoded and a renormalization is not required, then the highlighted region is reached. In this block, the context of the next pixel is built and compared with the current pixel. If they are the same, then the algorithm jumps to the label `QUICK_LOOP`, and if they are different, the algorithm jumps to the label `EXIT_LOOP`. (Unfortunately, there is no efficient looping structure that will perform this loop, so a `goto` is used.)

To understand why this loop improves the performance significantly, consider the characteristics of solid areas. In solid areas, the pixels are almost always encoded as most probable symbols with very highly skewed probabilities. Since the probabilities are highly skewed, it is rare that a renormalization needs to be performed; therefore, a very high percentage of the time that solid areas are encoded, the highlighted region is reached. Most non-solid areas do not have probabilities that are as highly skewed; therefore, even when most probable symbols are encoded, there is a good chance that a renormalization will be required, so the highlighted region will not be reached. Since

solid areas represent a large percentage of most documents anyway, a very high percentage of the cases that reach the highlighted region will be solid areas. When the new block is reached, it builds the new context in exactly the same way as at the top of the for-loop. The only extra instructions are an assignment and a comparison that involve the register variables `context` and `old_context` (the index decrement, check for end of line, and `goto` to loop back would have been done by the for-loop anyway), so if the new context is not the same as the previous context, there is little damage. On the other hand if the new context is the same as the previous context, then the four statements between the `EXIT_LOOP` and `QUICK_LOOP` labels are skipped for a considerable savings.

3.4 Fast Deterministic Prediction

Deterministic prediction uses a set of tables to discover which pixels are predictable by a JBIG decoder. For the default resolution reduction algorithm, the standard defines four tables, one for each spatial phase, that are used for DP. Each of the tables uses only the causal pixels (the ones that will be known by the decoder) to create its index. If a pixel is found to be deterministically predictable, it is skipped, otherwise it is arithmetically encoded and transmitted.

This can be improved by combining the four JBIG tables into a single table. The four single bit values in the original table are combined into a single four bit value where each bit represents the predictability of its corresponding pixel. The advantage of this technique is that only one deterministic context needs to be built to index a table for every four pixels. Figure 24 demonstrates how this can be done. In this example, the determinstic context is built as previously shown in figure 18 on page 39. This context is used to index a table of values from 0 to 15 that indicate which pixels must be encoded. In this example, a `switch` statement is used to quickly jump to a code segment

that will build the probability estimation contexts only for the pixels that need to be encoded.

While this has the advantage of only needing one table lookup to determine which pixels need to be encoded, it has a more subtle charm on some platforms. Since a `switch` is used to jump to a piece of software that will build contexts for all necessary pixels, only one jump is required instead of four `if`'s. In processors with a pipelined architecture, a series of `if` statements requires the pipe to wait to see if a branch is required. This causes the pipe to empty making the processor less efficient. By performing only one jump to a block with no other branches, the pipe will remain full while the pixels are processed, thereby improving the overall speed.

3.5 Removing the SWTCH from the Probability Estimation

In section 2.5, JBIG adaptive probability estimation was described. The JBIG standard clearly defines the tables and procedures required to perform this task in an efficient

```
DP_context = (hi_context3 & DP_MASK) |
             DPlo_context[ lo_context ];
switch ( DP_table[ DP_context )
{
  case 0:  /* all pixels predictable */
    break;

  case 1:  /* encode phase 0 only */
    process the phase 0 pixel;
    break:

  ...

  case 15: /* encode all four phases */
    process all four pixels;
    break;
}
```

Figure 24 Efficient deterministic prediction

way. Figure 13 on page 29 illustrates the process. A 113 state probability estimation table is defined that typically requires five bytes per state in a software implementation (two for LSZ, one for NMPS, one for NLPS, and one for SWTCH) for a total of 565 bytes. For each resolution layer and for each bit plane being processed, a 4096 byte context array and a 4096 byte MPS array is maintained (1024 bytes each for the lowest resolution layer). It is possible to make a slight change to this configuration that removes a couple of steps from the procedure for encoding least probable symbols, at the expense of making the state table larger.

When a least probable symbol is encoded enough times, the probability estimation tables may find it necessary to change the polarity of most probable symbol. This is done by checking the SWTCH byte in the state table for the current context before changing the state to NLPS. Every time a least probable symbol is encoded, the SWTCH byte must be checked.

Another way this can be done is to include the MPS in the state table. This can be done by doubling the size of the state table so that each state in the original table has two states in the new table, one with MPS = 0, and one with MPS = 1. The new NMPS points to the corresponding state in the table with the same MPS polarity. Each new NLPS points to the corresponding state in the table with the same MPS polarity, unless the SWTCH bit was set in the original table. If the SWTCH bit was set, the NLPS point to the corresponding state with the opposite MPS.

This is useful because it simplifies the procedure for encoding least probable symbols. It removes the need to specifically check the SWTCH byte for each LPS, conditionally branch, and occasionally change the MPS byte for the context. All of this is done by simply changing to the NLPS state. As a consequence of this, when each pixel is compared with the MPS, it must get the value from the state table; however, this is no differ-

ent from the previous design. There is no difference between accessing an MPS array using the context as an index and accessing it using the state as an index. The improved probability estimation process is shown in figure 25

As far as memory is concerned, this new design actually requires less memory. When the SWTCH field in the state table is replaced by the MPS field, the table still requires 5 bytes per state. Since the new state table has twice as many states, it requires 1130 bytes. However, it is no longer necessary to create the 4096 byte MPS array for each layer and bit plane.

If memory size is not a major concern, having the MPS in the state table can be slightly faster if the state table is implemented as a linked list. In this case, the address of a state structure is stored in the state array instead of the state number, and each NMPS and NLPS in just a pointer to the next state structure. The state pointer already needs to be known (and possibly stored in a register variable), and the MPS field is a short fixed offset from pointer. This type of indexing can be done very quickly in some processor architectures.

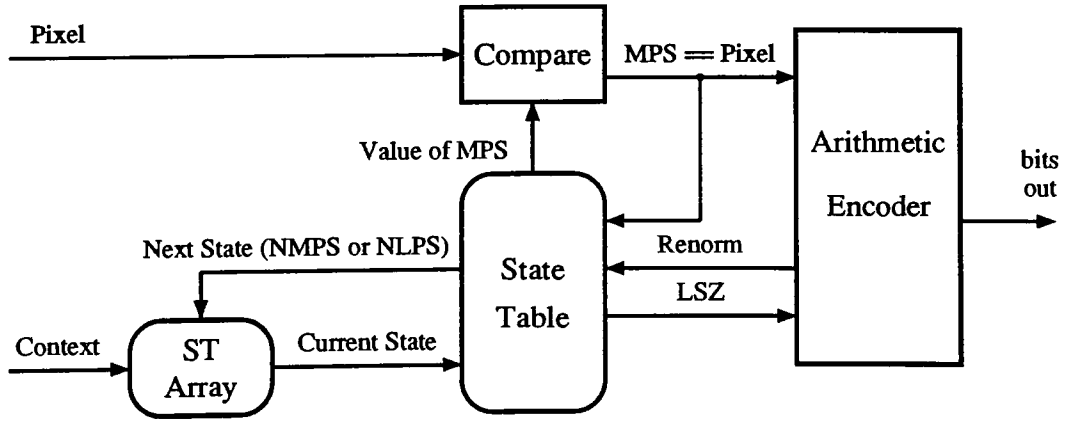


Figure 25 Improved structure of probability estimation

While this change is an interesting and appropriate change to the JBIG design, it only has a major benefit when a least probable symbol is encoded. By definition, the least probable symbols are encoded must less often than the most probable symbols. As a result, this change does not have as dramatic of an effect as some of the other proposed optimizations.

4 Results

4.1 Implementation of Software

As a part of this project, a full JBIG compressor and decompressor has been written. These programs meet the requirements for minimum support of free parameters as suggested in Annex A of the JBIG specification [1], with the exception of DPPRIV and DPLAST. These parameters are not supported because an absence of any private deterministic prediction tables makes it impossible to effectively debug the option; although implementation of the option would not be difficult.

Figure 26 shows a diagram of the modules that were generated to implement the JBIG encoder and decoder algorithms. In this diagram, each box represents a group of functions that perform a required task. The arrows represent the direction of the interaction between the modules. In other words, the application layer calls the image sequencer,

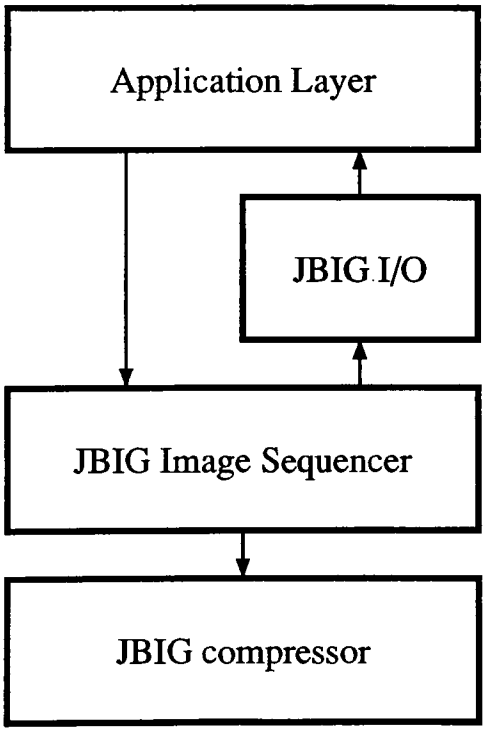


Figure 26 Module Diagram for the JBIG Compressor

the image sequencer calls the compressor and the I/O modules, and the I/O module calls the application layer. The following is a description of each of the modules:

JBIG Compressor - The compressor module performs the entire JBIG compression operation on a single stripe of one resolution layer and one bit-plane. It is intended to be a very fast, portable implementation of JBIG algorithm.

JBIG Image Sequencer - The image sequencer module is responsible for providing the compressor module with valid data to compress. It performs the resolution reduction process to generate low resolution images, and it columnizes the pixel data (section 3.1.1). The sequencer is also responsible for looping through the image in a convenient manner. In this implementation, it compresses the image in the order described by HITOLO=0, SEQ=1, ILEAVE=0, and SMID=0. This ordering is convenient for a software application because only enough memory needs to be allocated to store each resolution layer for one stripe and one bit-plane.

JBIG I/O - The I/O module calls the application layer to make a request for image data as needed. It also sends the compressed data to the application in the correct order. It is responsible for any buffering that needs to be done if the order requested by the application is different from the order of processing performed by the sequencer. In the current application, the buffering is done by storing the stripes of compressed data in temporary disk files until it is needed.

Application Layer - The application layer is intended to be any program that needs to use the JBIG algorithm to compress an image. It must be able to define the JBIG parameters that will be used to compress the image, access an input image to be compressed (from a disk file, video memory, scanner, ...), and receive compressed image data (to be transmitted or stored). In the case of this project, the application layer is a

simple user interface module with routines to read an image file and create a compressed image file.

The software described above has been implemented and debugged. All of the techniques documented in the previous chapter are used. The final software has been tested using the compliance tests defined in section 7.2 of the JBIG specification. It passes all tests with no errors.

4.2 Performance of the JBIG Algorithm

To test the execution speed of the JBIG software implementation, a set of 200 DPI (Dots Per Inch) text images was selected and processed on several 32-bit computing platforms. The test images, shown in Appendix C.1, are standard CCITT text documents that are commonly used of testing compression algorithms. The results of the tests are shown in tables 2, 3, 4, and 5. In these tables, the execution times do not include time spent in the I/O routines, and the times for the progressive modes include the time for the resolution reduction algorithm. Four reduced resolution image layers were produced so that the lowest resolution layer was 12.5 DPI.

CCITT Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	4.17	7.73	5.16	5.28	18.78
2	6.53	7.62	4.37	4.29	18.49
3	6.90	8.01	7.43	7.64	19.08
4	7.35	8.95	9.88	10.33	20.99
5	7.13	8.01	7.37	7.71	20.48
6	6.36	7.80	5.75	5.75	19.37
7	8.00	8.80	10.30	10.51	21.31
8	6.69	7.75	5.51	5.65	19.28
Average	6.64	8.08	6.97	7.15	19.72

Table 2 JBIG execution times on a VAXstation 4000/60, in seconds.

CCITT Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	2.43	4.38	3.35	3.15	10.93
2	3.37	4.18	2.80	2.82	10.83
3	3.90	4.55	4.78	4.73	11.25
4	4.17	4.93	6.28	6.15	11.87
5	4.03	4.63	4.70	4.63	11.32
6	3.50	4.22	3.58	3.52	10.98
7	4.53	4.93	6.52	6.43	11.87
8	3.87	4.40	3.50	3.53	11.18
Average	3.72	4.53	4.44	4.37	11.28

Table 3 JBIG execution times on a Sun Sparc 2, in seconds

CCITT Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	1.43	2.40	1.53	1.85	5.73
2	1.90	2.30	1.48	1.52	5.77
3	2.30	2.40	2.23	2.42	6.02
4	2.22	2.63	3.15	3.15	6.18
5	2.20	2.32	2.37	2.38	5.88
6	1.90	2.30	1.80	1.88	5.83
7	2.45	2.67	3.37	3.32	6.42
8	2.17	2.42	1.87	1.90	6.05
Average	2.40	2.43	2.22	2.30	5.98

Table 4 JBIG execution times on a Sun Sparc 10, in seconds

CCITT Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	11.48	19.54	18.14	18.50	86.56
2	16.16	18.86	14.36	14.58	86.24
3	17.72	20.64	28.84	29.60	87.58
4	21.18	24.82	39.62	39.76	90.04
5	18.72	21.00	28.06	28.72	87.70
6	16.02	19.56	20.70	20.86	86.88
7	21.72	24.06	40.60	41.30	89.88
8	17.20	19.74	20.18	20.42	87.52
Average	17.52	21.03	26.31	26.72	87.80

Table 5 JBIG execution times on an Amiga 3000, 25MHz 68030, in seconds

Several interesting trends can be noticed by studying these tables:

- Typical prediction in sequential mode improves the speed of the software somewhat. On documents that have a large number of all-white lines, like CCITT1, it has a much more dramatic effect.
- Typical prediction in progressive mode accounts for a huge reduction in the execution time. The reason for this is that a very high percentage of the lines in a differential layer are found to be typical. All of the solid area in these lines are skipped very efficiently.
- Deterministic prediction in progressive mode has very little effect on the compression speed.
- When all of the predictors are enabled, sequential mode is only slightly faster than progressive mode even though the progressive mode requires the generation and compression of each of the lower resolution layers. The reason for this is that the typical prediction algorithm allows a large part of the image to be skipped.

CCITT Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	35.03	35.02	30.60	29.05	29.10
2	60.24	60.66	57.45	55.24	55.04
3	23.38	23.39	21.64	20.70	20.75
4	9.45	9.46	8.75	8.41	8.41
5	19.87	19.90	18.23	17.47	17.47
6	40.84	40.99	38.14	36.43	36.44
7	9.13	9.13	8.45	7.97	7.97
8	36.08	36.14	33.77	32.48	32.43
Average	19.71	19.74	18.20	17.37	17.37

Table 6 JBIG compression ratios for test images

The compression ratios for these images, for each of the various compression options that were tested, are stored in table 6. Examination to tables reveals the following trends:

- For both sequential and progressive modes, disabling the typical prediction had virtually no effect on the compression ratios.
- Enabling deterministic prediction in the progressive mode accounted for a 4.8% improvement in the average compression ratio..
- In this case, in which five separate resolution layers are compressed, the benefits of progressive transmission and storage, with TP and DP on, are achieved at a cost of 7.7% loss in compression when compared to sequential mode, with TP on.

4.3 Comparison of JBIG and CCITT Group 3

The JBIG compression standard has been compared to the CCITT group 3 [17] and group 4 [18] standards for binary image compression[19]. The JBIG algorithm is expected to outperform the previous standards on a wide range of images, although this is

achieved at the price of extra complexity. To provide a benchmark for this comparison in both compression ratio and software execution speed, each of the test images shown in Appendix C.1 was compressed by the CCITT group 3, CCITT group 4, and JBIG sequential, and JBIG progressive algorithms. For the JBIG progressive test, the original 200 DPI images were reduced to a lowest resolution layer of 12.5 DPI.

Table 7 shows the compression ratios for each of the images and algorithm. On average, the sequential JBIG compression ratio was about 35% higher that the CCITT Group 4 ratio, 182% better than the Group 3 ratio. The average progressive JBIG compression ratio was 17% better than Group 4, and 145% better than Group 3.

CCITT Images	CCITT Group 3	CCITT Group 4	JBIG Sequential TP on	JBIG Progressive TP on DP on
1	12.78	28.35	35.03	30.60
2	14.06	47.51	60.24	57.45
3	7.64	17.88	23.38	21.64
4	4.65	7.41	9.46	8.75
5	7.28	15.93	19.87	18.27
6	9.61	30.82	40.84	38.14
7	4.72	7.41	9.13	8.45
8	7.91	26.87	36.08	33.77
Average	7.44	15.55	20.99	18.20

Table 7 Comparison of compression ratios for binary image compress algorithms

Table 8 shows the software execution times on a Sun Sparc 2 workstation. In this test, the optimized JBIG implementation discussed in this paper was compared with an unoptimized version of the JBIG algorithm*. These times are compared with execution times

* The unoptimized JBIG software was kindly provided by Don Duttweiler from AT&T. This software is available for the purpose of evaluating the JBIG algorithm and is known to be inefficient. Other known optimized versions of JBIG are proprietary.

of CCITT group 3 and group 4 compression software of unknown complexity. As expected, the CCITT group 3 and group 4 algorithms ran significantly faster than the optimized JBIG implementation. Group 3 was about 4 times faster, and group 4 was about 2.5 times faster. It is also apparent that the optimizations discussed significantly improve the execution speed of the software over the unoptimized software. For sequential mode, the optimized version is almost 3 times faster; for the progressive mode the optimized version is almost 5 times faster.

CCITT Images	CCITT Group 3	CCITT Group 4	Unoptimized JBIG		Optimized JBIG	
			Seq.	Prog.	Seq.	Prog.
1	0.66	1.07	7.3	19.5	2.43	3.35
2	0.51	1.07	10.1	18.4	3.37	2.80
3	0.96	1.55	10.3	22.6	3.90	4.78
4	1.79	2.74	10.3	25.6	4.17	6.28
5	1.05	1.66	10.6	22.3	4.03	4.70
6	0.74	1.21	9.8	20.3	3.50	3.58
7	1.64	2.54	11.1	26.2	4.53	6.52
8	0.75	1.25	10.3	20.0	3.87	3.50
Average	1.01	1.64	9.97	21.86	3.72	4.44

Table 8 Comparison of execution times for binary image compression algorithms on a Sun Sparc 2

4.4 Effect of Image Noise on Compression Times

Image noise is a problem in many imaging systems. It can be caused by environmental problems such as dust on the imaging array, or dust on the image surface. It can also be caused by problems with the imaging electronics. Whatever the source, it can cause a degradation of the quality of the resulting image.

Image noise can also have a negative effect on the compression algorithms that are used on the images. In general, the more noise an image has, the less compressible it is. For a JBIG software implementation, there can be an even larger problem. One of the major sources of optimization discussed in this paper was a result of taking advantage of large solid areas. When white space is broken up by the presence of image noise, part of the advantage of the optimization is negated. In addition, image noise can cause the typical prediction assumption to fail in a much higher percentage of the lines, negating the advantage of typical prediction.

To simulate the problem of noise in an image system, random noise was added to the 8 CCITT test documents. This was done by randomly selecting a percentage of pixels in each image and changing their value. These images were then compressed using the JBIG algorithm in both sequential and progressive modes on a VAXstation 4000/60. Table 9 lists the average execution times and compression ratios for the images which can be compared with the original results in table 2.

For both JBIG compression modes, adding image noise reduces the resulting compression ratio as expected. This reflects the fact that the extra noise represents extra infor-

% Noise	# Pixels Changed	Sequential Mode		Progressive Mode	
		Time (sec.)	Comp. Ratio	Time (sec.)	Comp. Ratio
0.01%	411	7.02	19.24	9.15	17.69
0.03%	1232	7.65	18.24	12.40	16.90
0.10%	4106	8.21	16.08	16.78	14.98
0.30%	12317	8.49	12.47	18.62	11.76
1.00%	41057	9.27	7.59	19.66	7.27
3.00%	123172	10.68	4.08	20.34	3.97

Table 9 Effect of image noise on JBIG compression for the CCITT image set.

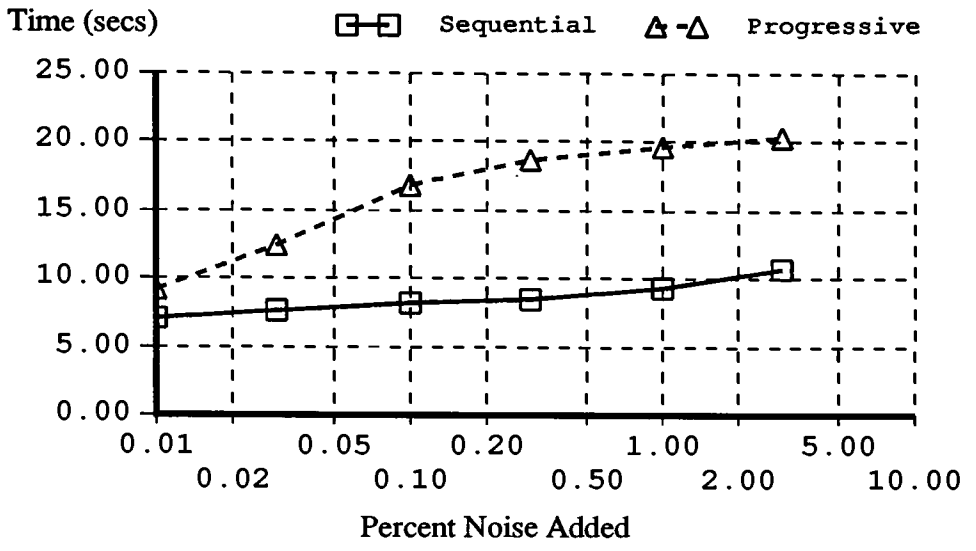


Figure 27 The effect of image noise on JBIG compression speed for progressive and sequential transmission

mation that needs to be encoded. The added noise also effects the execution time of both JBIG modes, but in different ways as seen in figure 27. The execution times for the progressively encoded images are much more strongly effected by the noise. To understand why this is the case, consider the effect of disabling typical prediction in tables 2-5. Disabling the typical prediction option in progressive mode, has a much larger impact on the execution time than it did in sequential mode. Since the primary effect of adding the noise is to effectively disable the typical prediction algorithm, it is not surprising that the effect is similar to completely disabling the algorithm.

In systems where image noise of the type shown in Appendix C.3 is a problem, it would be useful to try to remove the noise from the images. Since most of the noise is represented as single isolated black dots on a white background (or isolated white dots on a black background), an attempt can be made to remove the noise by removing single isolated dots. The images in Appendix C.4 represent an attempt to filter the images in Appendix C.3. In each of these images, each pixel was considered as the center pixel in a 5x5 square of pixels. If all of the other pixels in the square were the opposite of the

center pixel, then the center pixel was changed. This procedure should remove most of the noise, but it should not have a large effect on any text in the image.

To test this concept, each of the noisy images was filtered and compressed using the JBIG algorithm. The results of this test are shown in table 10. In the filtered image, the compression ratios and execution times are much more consistant for low levels of noise. When the noise levels become high enough that more than one noise pixel is

% Noise	# Pixels Changed	Sequential Mode		Progressive Mode	
		Time (sec.)	Comp. Ratio	Time (sec.)	Comp. Ratio
0.01%	411	6.54	19.69	6.86	18.20
0.03%	1232	6.49	19.59	6.88	18.09
0.10%	4106	6.61	19.16	7.30	17.75
0.30%	12317	7.28	17.79	7.80	17.27
1.00%	41057	8.39	13.04	9.80	12.13
3.00%	123172	9.46	6.09	19.56	5.76

Table 10 Effect of removing image noise on JBIG compression (CCITT Test Set)

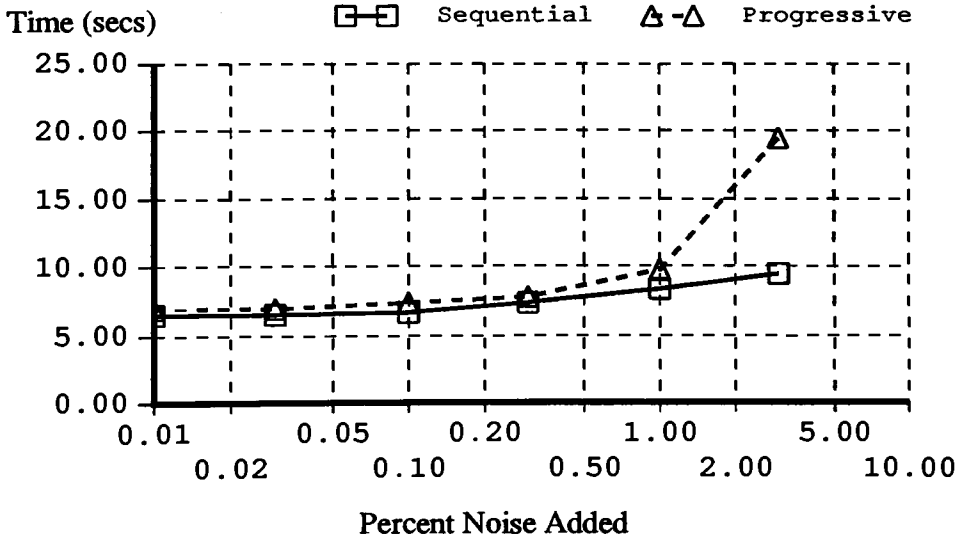


Figure 28 The effect of filtering noisy images on JBIG compression speed

found in a 2x2 region, the benefit of the filter disappears. This can be seen graphically in figure 28.

4.5 Halftone Images

Digital halftoning is a technique that allows continuous tone images to be display or printed as binary images by printing patterns of dots that simulate gray levels [20]. In terms of image compression, halftoned images are particularly challenging because they have characteristics that are drastically different from other types of images, such as text or line art. Appendix C.5 contains two different examples of halftoned images. The first image is an example of 'ordered-dither' in which dots are placed in an ordered pattern to simulate gray levels. The second image is an example of 'error-diffusion' in which dots are placed in a more random fashion.

These halftoned images were compressed on a Sun Sparc 2 workstation and the execution times and compression ratios are shown in tables 11 and 12. Since the images are different sizes, it is not relevant to compare the execution times between the two images,

Halftoned Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	9.93	9.87	26.58	27.12	26.22
2	2.40	2.32	5.00	4.88	4.82

Table 11 JBIG halftone execution times on a Sun Sparc 2, in seconds

Halftoned Images	Sequential TP on	Sequential TP off	Progressive TP on DP on	Progressive TP on DP off	Progressive TP off DP off
1	6.04	6.07	8.36	7.53	7.53
2	1.76	1.76	1.34	1,20	1.20

Table 12 JBIG halftone compression ratios

although it is possible to compare the compression ratios. From these two examples, it is obvious that the amount of compression that can be acheived in an image varies dramatically depending on the halftoning technique used to generate the image. It is also possible to see the trends caused by enabling or disabling certain features. Probably the most important difference between the halftoned images and the text images that were tested previously, is the fact that typical prediction does not improve the execution speed in halftoned images. This is because many of the lines are not typical, and even if they were, there is almost no solid area in the images to predict. Another important item to note is that enabling deterministic prediction improved the compression ratio in both images by over 11%. This is compared to an average improvement of about 4.8% in the text images.

It is worth noting that the JBIG algorithm could perform even better if an algorithm for AT pixel movement were used. In the current implementation, an algorithm for automatic AT pixel placement is not used, however, it is possible to move the AT pixel to any position within 16 pixels horizontally of the pixel being encoded. If an optimal position for the AT pixel is known beforehand, the AT pixel can be moved to improve the encoding performance. As an example, since the width of the dither pattern in the first test image is four pixels, the AT pixel can be moved to a position four pixels before the pixel being encoded. When this is done in sequential mode, the compression ratio improves from 6.04 to 10.34, or an improvement of 71%. In progressive mode, the compresion ratio improves from 8.36 to 9.47, or an improvement of 13%.

The two test images were also processed using the CCITT Group 3 and Group 4 algorithms on a Sun Sparc 2. The result of the tests are summarized in tables 13 and 14. As indicated in table 13, the compression ratios for both CCITT algorithms is less than one, meaning that the encoded image is larger than the original image. For these same images, the JBIG algorithm performed significantly better. On the order-dither image, the JBIG algorithm compressed at a level similar to some of the complex text documents. Even on the difficult error-diffused image, the JBIG algorithm provided a small amount of compression.

When comparing the execution time for the algorithms, it is interesting to note that the JBIG software, in sequential mode, was significantly faster than the previous CCITT standards, which exactly the opposite of the situation for text documents. When comparing the optimized version of JBIG with the unoptimized version, the optimized version is about twice as fast.

Halftoned Images	CCITT Group 3	CCITT Group 4	JBIG Sequential	JBIG Progressive
1	0.83	0.71	6.04	8.36
2	0.5	0.46	1.76	1.34

Table 13 Comparison of halftone compression ratios for binary image compression algorithms

Halftoned Images	CCITT Group 3	CCITT Group 4	Unoptimized JBIG		Optimized JBIG	
			Seq.	Prog.	Seq.	Prog.
1	16.78	36.02	20.4	62.6	9.93	26.58
2	3.66	5.06	4.2	12.3	2.40	5.00

Table 14 Comparison of halftone compression execution times for binary image compression algorithms

5 Conclusions

Several techniques have been developed to efficiently implement, in software, the encoding half of the JBIG binary image compression standard. The most important techniques are 1) a procedure for efficiently building the required JBIG contexts by creating columns of three pixels, and 2) procedures for efficiently processing solid areas in the resolution reduction algorithm, differential layer typical prediction algorithm, and the lowest resolution layer arithmetic encoding algorithm. Techniques are also presented for fast deterministic prediction, and improved handling of the conditional exchange condition.

These techniques have been implemented in portable software and shown to be fully JBIG compliant. They have been tested on several 32-bit computing platforms and the execution times have been recorded. When compared with CCITT Group 3 and Group 4 compression algorithms on text documents, JBIG is found to perform significantly better from a compression standpoint, although the JBIG software takes significantly longer to execute. When the new optimized JBIG software is compared to the unoptimized JBIG evaluation software, it is found be almost 3x faster on sequentially processed images, and almost 5x faster on progressively processed images. On halftoned images, JBIG compressed an ordered-dither image about 10x better than the older standards, while on an error-diffused image it compressed about 4x better. It is also interesting to note that for both halftoned images, the optimized JBIG software ran significantly faster than the CCITT Group 3 and Group 4 software. The optimized software also ran about twice as fast as the unoptimized software, even though most of the optimization techniques are negated by the characteristics of the halftones.

The JBIG algorithm was also tested in the presence of image noise. The speed of the software was found to drop off significantly, especially in progressive transmissions. A simple filtering concept was tested to show that noise could be removed so that the images could be compressed without suffering the problem of reduced execution speed.

References

[1] CCITT Draft Recommendation T.82 ISO/IEC Draft International Standard 11544, Coded Representation of Picture and Audio Information - Progressive Bi-level Image Compression, WG9-S1R5.1, April 3, 1992.

[2] P. Elias, in N. Abramson, *Information Theory and Coding*, McGraw-Hill Book Co., Inc., New York, 1963.

[3] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.* 27, 379 (1948).

[4] M. Rabbani and P. W. Jones, *Digital Image Compression Techniques*, SPIE Optical Engineering Press, Bellingham Washington, 1991.

[5] A. I. Khinchin, *Mathematical Foundations of Information Theory*, Dover, New York, 1957.

[6] S. Goldman, *Information Theory*, Prentice-Hall, New York, 1953.

[7] D. A. Huffman, "A Method for the Construction of Minumum-Redundancy Codes," *Proc. IRE* **40**, 1098 (1952).

[8] S. W. Golomb, "Run-Length Encodings," *IEEE Trans. Inform. Theory*, **IT-12** (4), 399-401 (1966).

[9] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop.*, 23 (2), 149-162 (1979).

[10] G. G. Langdon, "An Introduction to Arithmetic Coding," *IBM J. Res. Develop.* **28**, 135 (1984).

[11] G. G. Langdon and J. J. Rissanen, "Compression of Black-White Images with Arithmetic Coding,", *IEEE Trans. Commun.* **COM-29**, 858 (1981).

[12] C. B. Jones, "An Efficient Coding System for Long Source Sequences," *IEEE Trans. Info. Theory*, **IT-27** (3), 280-291 (1981).

[13] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of ACM*, **30** (6), 520-540 (1987).

[14] W. B. Pennebaker, J. L. Mitchel, G. G. Langdon, Jr., and R. B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic coder," *IBM J. Res. Develop.* **32** (6), 717-726 (1988).

[15] J. L. Mitchell and W. B. Pennebaker, "Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder," *IBM J. Res. Develop.* **32** (6), 727-736 (1988).

[16] J. L. Mitchell and W. B. Pennebaker, "Software Implementations of the Q-Coder," *IBM J. Res. Develop.* **32** (6), 753-774 (1988).

[17] CCITT Recommendation T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission," Vol VII-Fascicle VII.3, 21-47.

[18] CCITT Recommendation T.6, "Facsimile Coding Schemes and Coding Control Functions for Group 4 facsimile apparatus," Vol. VII-Fascicle VII.3, 48-57.

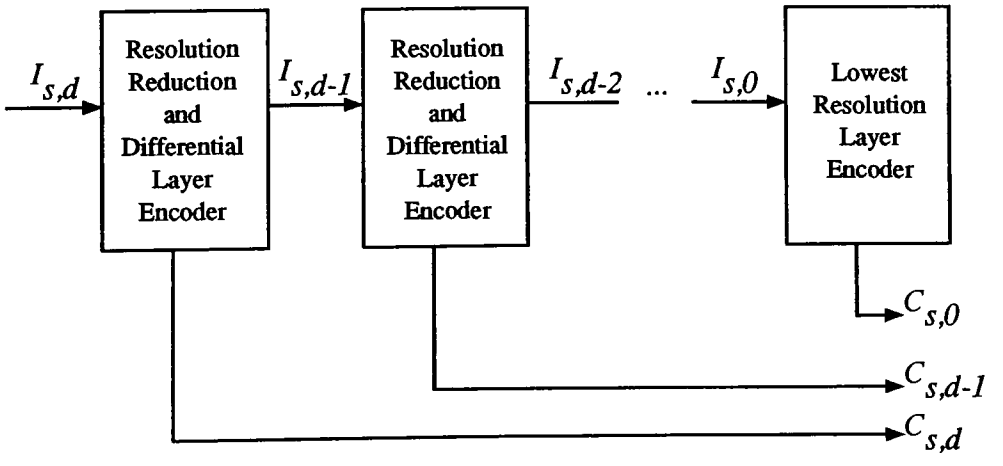
[19] S. J. Urban, "Review of Standards for Electronic Imaging for Facsimile Systems," *Journal of Electronic Imaging* **1** (1), 5-21 (1992).

[20] R. Ulichney, *Digital Halftoning*, MIT Press, Cambridge, Massachusetts, 1987.

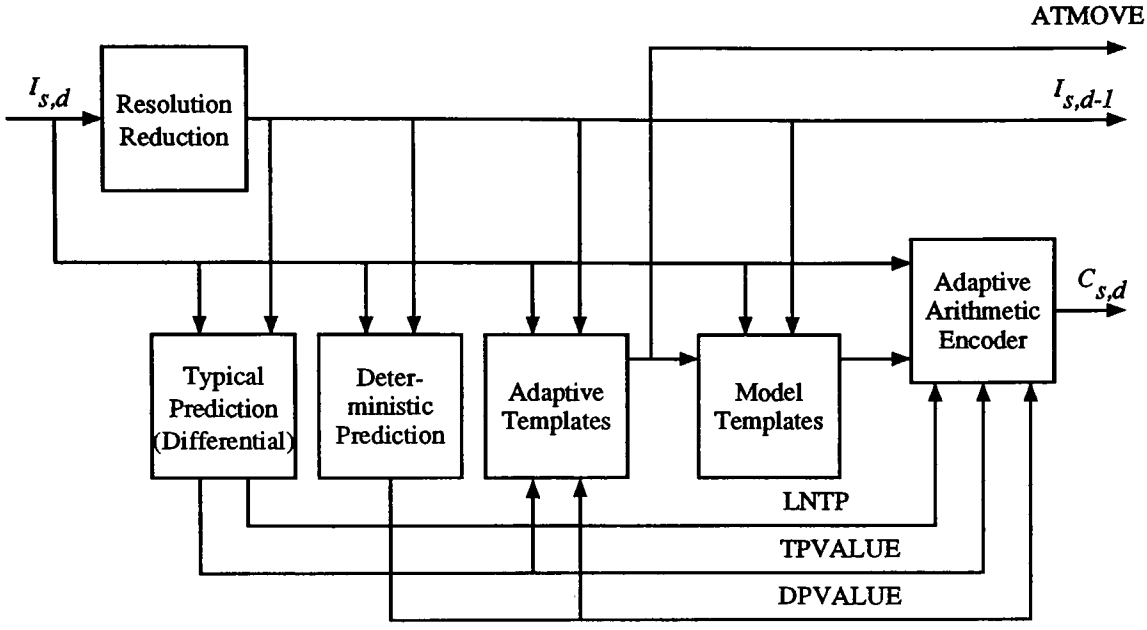
Appendix A Block Diagrams of the JBIG Compression Algorithm

The block diagrams in this appendix were taken from the JBIG specification [1]. They illustrate the flow of information in the JBIG algorithm. The following table lists the symbols that are used in the diagrams and their meaning.

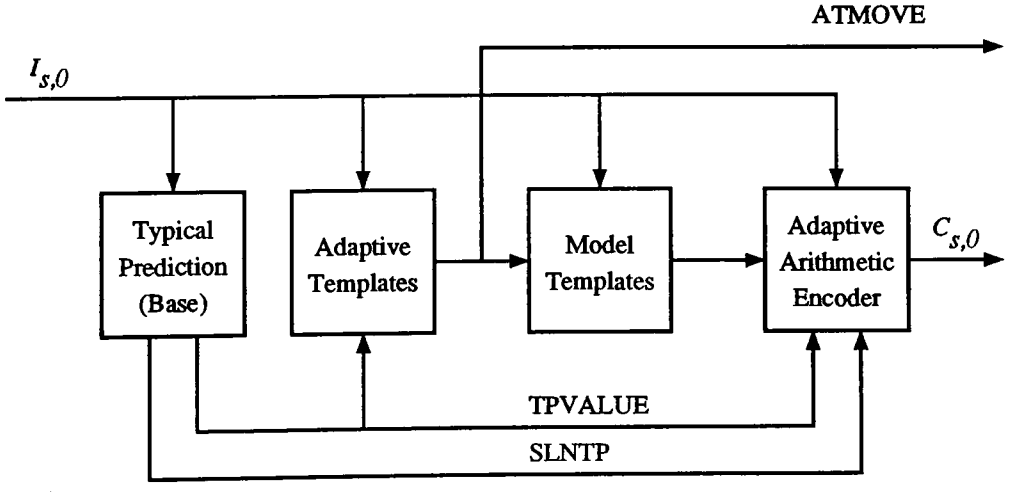
Symbol	Description
$I_{s,d}$	image data for stripe s and resolution layer d
$C_{s,d}$	compressed data for stripe s and resolution layer d
LNTP	line not typical
SLNTP	same as LNTP
TPVALUE	typical prediction value
DPVALUE	deterministic prediction value
ATMOVE	flag AT pixel movement



Decomposition of the JBIG encoder



JBIG Differential Layer Encoder



JBIG Lowest Resolution Layer Encoder

Appendix B Routines for Reordering JBIG Tables

B.1 Reordering Resolution Reduction Tables

The following routine is an example function to show how standard resolution reduction tables can be reordered to be used with the context ordering defined in this paper.

```
/*****************************************************************************
*
* Function Name:        JBIG_reorder_rrtables
*
* Description:          This function takes as input the standard JBIG
*                       resolution reduction tables and creates new tables
*                       that can be used in the optimized implementation.
*                       The difference between the two tables is that
*                       the bits in the index of the original table have
*                       been shuffled around to form a new index that is
*                       used for the reordered table.
*
* Call Parameters:      original_table  - resolution reduction table as
*                                         defined in the JBIG spec.
*                       reordered_table - new table for resolution reduction
*                                         for use with the optimized context.
*
* Returns:              nothing
*
* Revision History
*    Author             Date        Description
* 0. Craig Smith        05/13/93    original version
*
*****************************************************************************/

#define RR_TABLE_LENGTH             4096

void JBIG_reorder_rrtables( const unsigned char *original_table,
                                  unsigned char *reordered_table )

{
/*
*   Define a union between a long integer that can be used as an array index,
*   and a bit field that can be used to reorder the bits in the index.
*   Note:  The order of the bits in the bit field is system dependent.  This
*          routine works on system in which the first bit in the field is the
*          least significant.
*/
    union
    {
        long i;
        struct
```

```
        {
            unsigned b0:  1;
            unsigned b1:  1;
            unsigned b2:  1;
            unsigned b3:  1;
            unsigned b4:  1;
            unsigned b5:  1;
            unsigned b6:  1;
            unsigned b7:  1;
            unsigned b8:  1;
            unsigned b9:  1;
            unsigned b10: 1;
            unsigned pad: 20;
        } b;
    } in, out;

/*
* For every possible index to the original table, a new index is
* created for the new table. The value of the original table is
* then copied to the new table.
*/
    out.i = 0;
    for ( in.i = 0 ; in.i < RR_TABLE_LENGTH ; in.i++ )
    {
        out.b.b0 = in.b.b10;
        out.b.b1 = in.b.b0;
        out.b.b2 = in.b.b3;
        out.b.b3 = in.b.b6;
        out.b.b4 = in.b.b1;
        out.b.b5 = in.b.b4;
        out.b.b6 = in.b.b7;
        out.b.b7 = in.b.b9;
        out.b.b8 = in.b.b11;
        out.b.b9 = in.b.b2;
        out.b.b10 = in.b.b5;
        out.b.b11 = in.b.b8;

        reordered_table[ out.i ] = original_table[ in.i ];
    }

    return;
}
```

B.2 Reordering Deterministic Prediction Tables

The following routine is an example function to show how standard deterministic prediction tables can be combined and reordered to be used with the context ordering and fast deterministic prediction defined in this paper.

```
/*****************************************************************************
*
* Function Name:        JBIG_reorder_dptables
*
* Description:          This function takes as input the standard JBIG
```

```
*                       deterministic prediction tables and creates new tables
*                       that can be used in the optimized implementation.
*                       The difference between the tables is that the bits
*                       in the index of the original table have been shuffled
*                       around to form a new index that is
*                       used for the reordered table.
*
* Call Parameters:      phase0_table    - phase 0 DP table as defined in the
*                                         JBIG spec.
*                       phase1_table    - phase 1 DP table as defined in the
*                                         JBIG spec.
*                       phase2_table    - phase 2 DP table as defined in the
*                                         JBIG spec.
*                       phase3_table      phase 3 DP table as defined in the
*                                         JBIG spec.
*                       reordered_table - new table for deterministic prediction
*                                         for use with the optimized context.
*
* Returns:              nothing
*
* Revision History
*    Author             Date        Description
* 0. Craig Smith        05/13/93    original version
*
*****************************************************************************/

#define DP_TABLE_LENGTH             4096

void JBIG_reorder_rrtables( const unsigned char *phase0_table,
                            const unsigned char *phase1_table,
                            const unsigned char *phase2_table,
                            const unsigned char *phase3_table,
                                  unsigned char *reordered_table )

{
/*
*   Define a union between a long integer that can be used as an array index,
*   and a bit field that can be used to reorder the bits in the index.
*   Note:  The order of the bits in the bit field is system dependent.  This
*          routine works on system in which the first bit in the field is the
*          least significant.
*/
    union
    {
        long i;
        struct
        {
            unsigned b0:  1;
            unsigned b1:  1;
            unsigned b2:  1;
            unsigned b3:  1;
            unsigned b4:  1;
            unsigned b5:  1;
            unsigned b6:  1;
            unsigned b7:  1;
            unsigned b8:  1;
            unsigned b9:  1;
            unsigned b10: 1;
            unsigned pad: 20;
        } b;
    } in, out;

/*
```

```
*   The out variable is the index to the new reordered table.  Take the
*   bits in out.i and rearrange them to create the variable in.i that is
*   used to index each of the JBIG standard tables for deterministic
*   prediction.  The final value in the reordered_table is a number between
*   0 and 15 that indicates which combination of pixels should be encoded.
*/
    in.i = 0;
    for ( out.i = 0 ; out.i < DP_TABLE_LENGTH ; out.i++ )
    {
        in.b.b0  = out.b.b10;
        in.b.b1  = out.b.b5;
        in.b.b2  = out.b.b8;
        in.b.b3  = out.b.b11;
        in.b.b4  = out.b.b0;
        in.b.b5  = out.b.b2;
        in.b.b6  = out.b.b1;
        in.b.b7  = out.b.b3;
        in.b.b8  = 0;
        in.b.b9  = 0;
        in.b.b10 = 0;
        in.b.b11 = 0;

        reordered_table[ out.i ]  = (phase0_table[ in.i ] < 2 ? 0 : 1);

        in.i <<= 1;
        in.b.b0 = out.b.b7;

        reordered_table[ out.i ] += (phase1_table[ in.i ] < 2 ? 0 : 2);

        in.i <<= 2;
        in.b.b0 = out.b.b9
        in.b.b1 = out.b.b4

        reordered_table[ out.i ] += (phase0_table[ in.i ] < 2 ? 0 : 4);

        in.i <<= 1;
        in.b.b0 = out.b.b6;

        reordered_table[ out.i ] += (phase1_table[ in.i ] < 2 ? 0 : 8);
    }

    return;
}
```

Appendix C Test Images

C.1 Original Test Images

On the following pages, the set of test images that were used to perform the evaluations in Chapter 4 are printed. These images are the standard CCITT test set and they are described in the following table.

CCITT Image	Description
1	typewritten letter
2	hand-drawn electrical schematic
3	typewritten invoice
4	typewritten French
5	line art and text
6	plot and graph
7	Chinese
8	handwriting and banner

Each of these images are 1728 pixels wide and 2376 pixels high. They are printed at a resolution of 200 dots per inch.

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

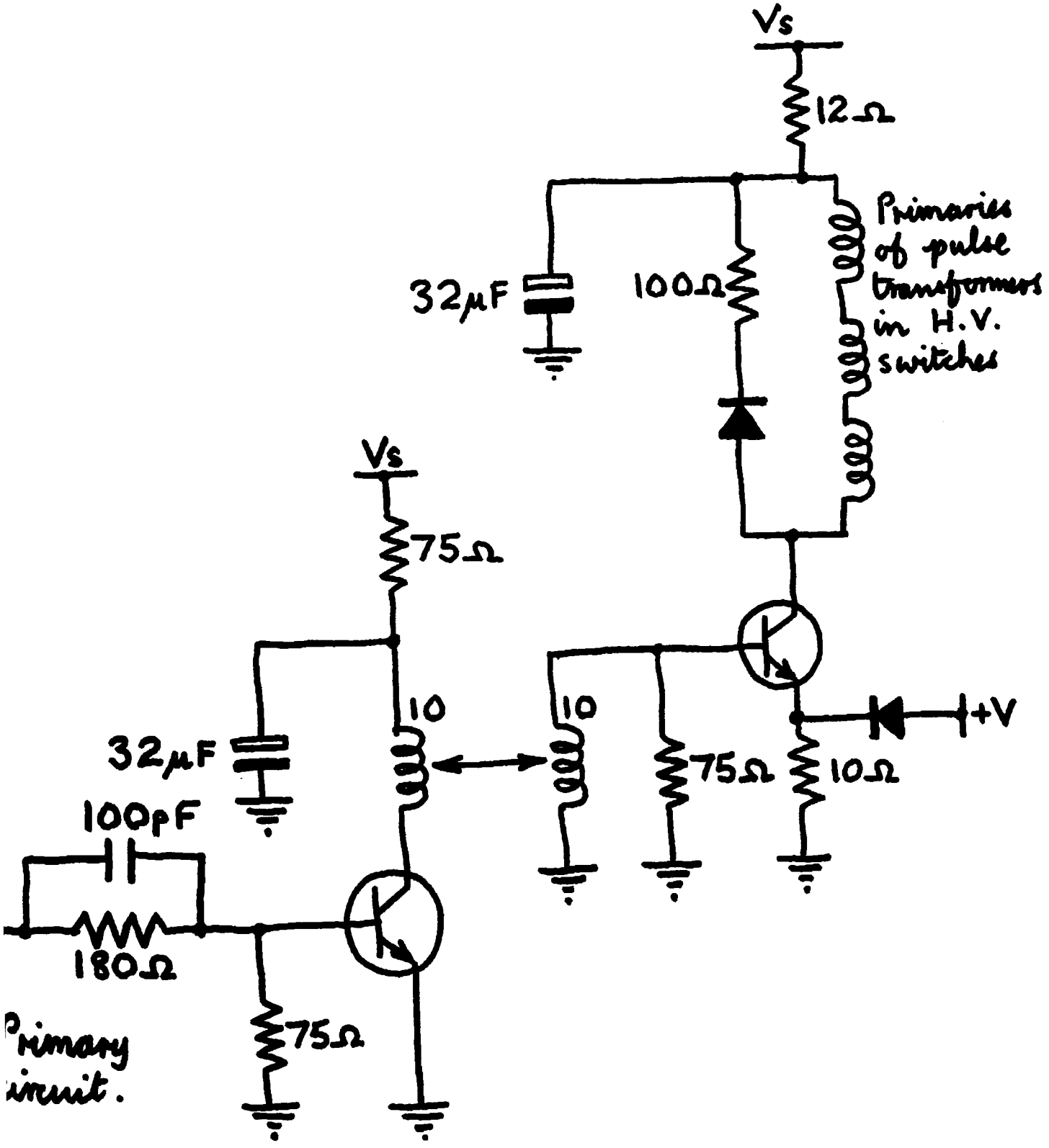
At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research



This is current driver circuit.

Phil.

Mot directeur

ETABLISSEMENTS ABCDEFG
SOCIETE ANONYME AU CAPITAL DE 300 000 F
39, RUE DU XYUTRSTBBL F 69000 NTBCLAG
Tél. : (35) 24.46.32 Adr. Tg. : NRVLJNOLM
Télex : 31586 F IN : 710490070257
Transporteur (ou Transitaire)
M. M. DUPONT Frères
8 quai des biodfsh F 6009 NTBCLAG

CLASSEMENT	FACTURE INVOICE	Exemplaire 15	
CODE CLIENT Z 04599	DATE 7-7-74	NUMERO 06	FEUILLET 01

Votre commande du 74-2-2 numéro 438

Notre offre AZ/B7 du 74-1-1 numéro 12

LIVRAISON
5, rue XYZ
99000 VILLE

FACTURATION
12, rue ABCD BP 15
99000 VILLE

DOMICILIATION BANCAIRE DU VENDEUR

CODE BANQUE | CODE GUICHET | COMPTE CLIENT

PAYS D'ORIGINE | PAYS DE DESTINATION

CONDITIONS DE LIVRAISON DATE 74-03-03

LICENCE D'EXPORTATION | NATURE DU CONTRAT (monnaie)

CONDITIONS DE PAIEMENT (échéance, %...) FAB

TRANSPORTS ORIGINE	TRANSPORTS DESTINATION	MODE
Pays 1	Etat 2	Air

MARQUES ET NUMEROS MARKS AND NUMBERS	NOMBRE ET NATURE DES COLIS : DÉNOMINATION DE LA MARCHANDISE NUMBER AND KING OF PACKAGES: DESCRIPTION OF GOODS	NOMENCLATURE STATISTICAL No.	MASSE NETTE NET WEIGHT / MASSE BRUTE GROSS WEIGHT	VALEUR VALUE / DIMENSIONS MEASUREMENTS
74.21.456.44.2 A	1 Composants	U 123/4	5 kg / 8 kg	1400 X 13x10x6

QUANTITÉ COMMANDÉE ET UNITÉ QUANTITY ORDERED AND UNIT	N° ET RÉF. DE L'ARTICLE	DÉSIGNATION	QUANTITÉ LIVRÉE ET UNITÉ QUANTITY DELIVERED AND UNIT	PRIX UNITAIRE UNIT PRICE	MONTANT TOTAL TOTAL AMOUNT
2	AF-809	Circuit intégré	2	104,33 F	208,66 F
10	58-T4	Connecteur	10	83,10 F	831,00 F
25	ZI07	Composant indéterminé	20	15,00 F	300,00 F

Costs	Débours	Inclus	Non inclus
Packing	Emballages		92,14
Freight	Transport		
Insurance	Assurances		

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet,et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger,et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

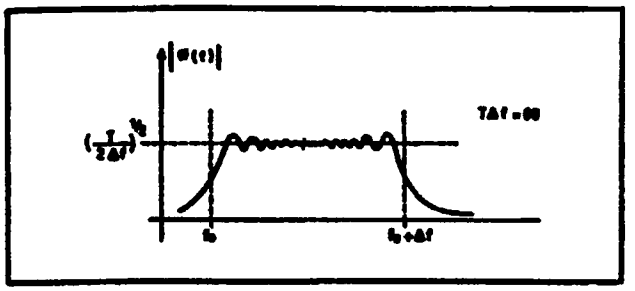


FIG. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leqslant f \leqslant f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

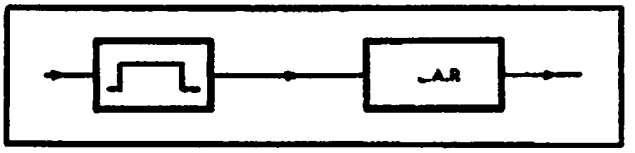


FIG. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_R décroissant linéairement avec la fréquence f suivant l'expression :

$$T_R = T_0+(f_0-f)\frac{T}{\Delta f} \quad (\text{avec } T_0 > T)$$

(voir fig. 4).

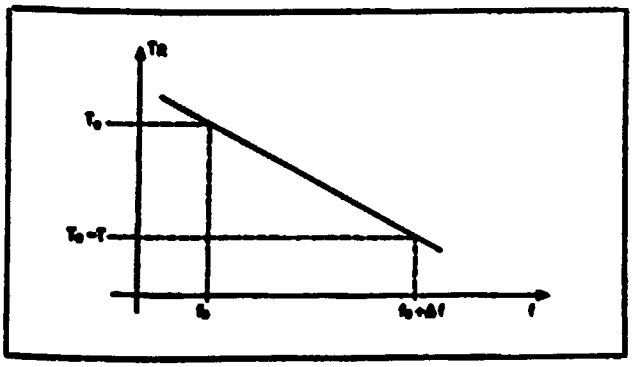


FIG. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi\int_0^f T_R\,df$$

$$\varphi = -2\pi\left[T_0+\frac{f_0T}{\Delta f}\right]f+\pi\frac{T}{\Delta f}f^2$$

Et cette phase est bien l'opposé de $\underline{/\phi(f)}$, à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de récepteur à compression d'impulsion donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\dfrac{T}{1/\Delta f} = T\Delta f$

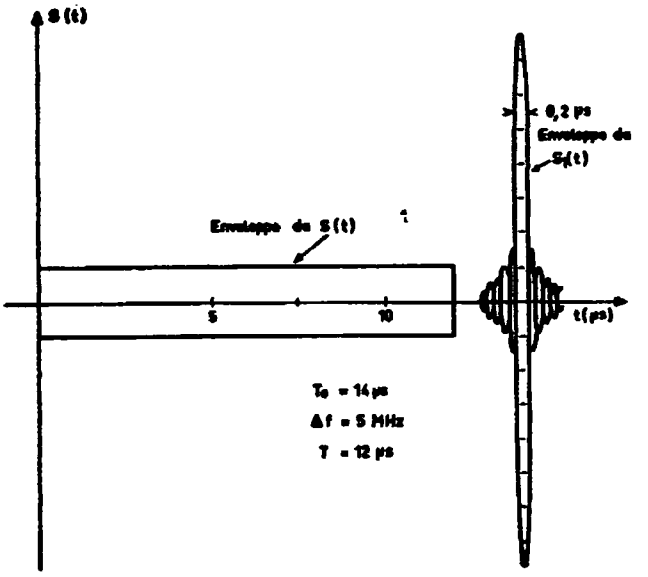
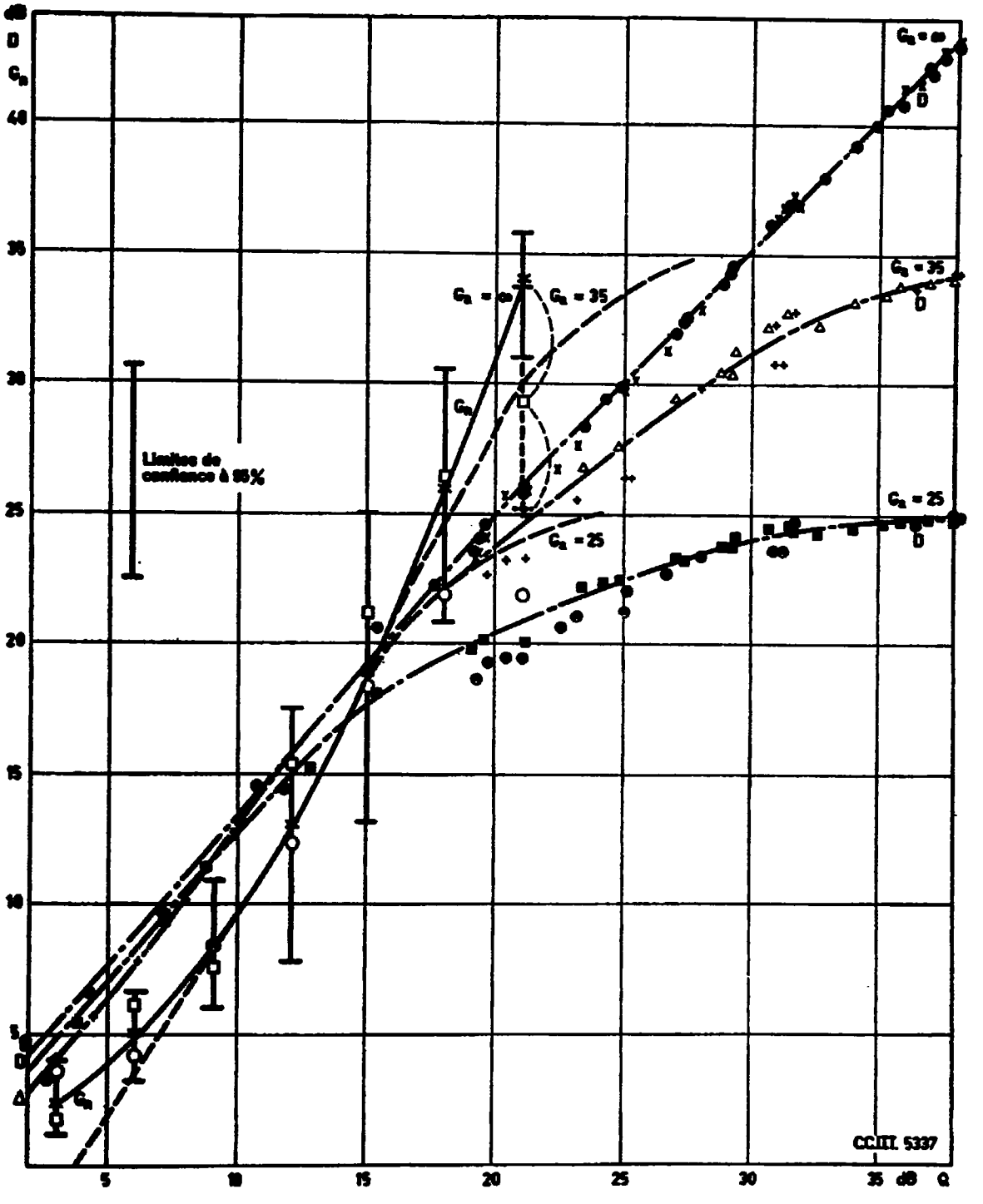


FIG. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\dfrac{T}{\Delta f}$ et elle met un temps $T_0-(f-f_0)\dfrac{T}{\Delta f}$ pour traverser, ce qui la fait ressortir



Courbes adaptées G_n (essais subjectifs) pour

	$G_n = \infty$	$G_n = 35$dB	$G_n = 25$dB
[10] ———	×	□	○
[23] — — —			

Points calculés $D\,(Q, G_n)$ pour

$G_n = \infty$ $G_n = 25$ $G_n = 35$ dB

● ■ △ – dans la partie montante

× ● + – dans la partie descendante

——×—— — ——×——△

Courbes ——— — ——— — ——— $D\,(Q, G_n)$

FIGURE 3

CCITTの概要

沿革

CCITTは、国際電気通信連合（ITU）の四つの常設機関（事務総局、国際周波数登録委員会、CCIR、CCITT）の一つとして、ITUの中でも、世界の国際通信上の諸問題を真先に取上げ、その解決方法を見出して行く重要な機関である。日本名は、国際電信電話諮問委員会と称する。

CCITTの前身は、CCIF（国際電話諮問委員会）とCCIT（国際電信諮問委員会）である。CCIFは、1924年にヨーロッパに「国際長距離電話通信諮問委員会」が設置され、これが1925年のパリ電信電話会議のとき、正式に、「国際電話諮問委員会」として万国電信連合の公式機関となったものである。CCITは、同じく1925年の会議のとき、CCIFと併立するものとして設置された。

そして、CCIFは、1956年の12月に第18回総会が開催されたのち、CCITは、同年同月に第8回総会が開催されたのち、併合されて現在のCCITTとなった。このCCITTは、CCIFとCCITが解散した直後、第1回総会を開催し、第2回総会は、1960年にニューデリーで、第3回総会は、1964年、ジュネーブで、第4回総会は、1968年、アルゼンチンで開催された。

CCIFとCCITが合併したのは、有線電気通信の分野、とくに伝送路について電信回線と電話回線とを技術的に分ける意味がなくなってきたこと、各国とも大体において、電信部門と電話部門は同一組織内にあること、CCIFの事務局とCCITの事務局の合併による能率増進等がおもな理由であった。

CCITTは、上述のように、ヨーロッパ内の国ぐにによって、ヨーロッパ内の電信・電話の技術・運用・料金の基準を定め、あるいは統一をはかってきたので、現在でも、その影響を受け、会合参加国は、ヨーロッパの国が多く、ヨーロッパで生起する問題の研究が多い。たとえば、1960年のCCITT勧告の中で、技術上配慮する距離は約2,500kmであったが、これはヨーロッパ内領域を想定したものである。

しかしながら、1956年9月に敷設された大西洋横断電話ケーブルは、大陸間電話通信の自動化および半自動化への技術的可能性を与え、CCITTがこの問題を取り上げるに及び、CCITTの性格は漸次、汎世界的色彩を実質的に帯びるに至った。この汎世界的性格は第2次世界大戦後目ざましくなったアジア・アフリカ植民地の独立に伴ってITUの構成員の中にこれらの国が加わり、ITUの中に新しい意見が導入されたことにも起因して、技術面、政治面の双方から導入されてきた。CCITTの汎世界化は、1960年の第2回総会がニューデリーで開催されたことにもあらわれている。この総会までは、CCIT、CCIFのいずれにアメリカやアジアで総会が開催されたことがなく、CCITT委員長も、ニューデリー総会の準備文書で、この点には注目すべきであるとのべている。

任務

ITUは、全権委員会議、主管庁会議を始めとして、七つの機関をもち、それの機関の権限と任務は国際電気通信条約に明記されている。そこで条約をみてみるならば、CCITTの任務は、つぎのとおりとなっている。

「国際電信電話諮問委員会（CCITT）は、電信および電話に関する技術、および料金の問題について研究し、および意見を表明することを任務とする。」（1965年モントルー条約第187号）

「各国際諮問委員会は、その任務の遂行に当たって、新しい国または発展のにある国における地域的および国際的分野にわたる電気通信の創設、発達および善に直接関連のある問題について研究し、および意見を作成するように妥当な処を払わなければならない。」（同第188号）

「各国際諮問委員会は、また、関係国の要請に基づき、その国内電気通信のについて研究し、かつ、勧告を行なうことができる。」（同第189号）

上記第187号と第188号にいわれる「意見」とは、フランス語のAvisを訳したもので、英語では、「勧告（Recommendation）」となっている。CCITT表明する意見は、国際法的には強制力をもたないものであって、この点が、条約信規則、電話規則等各国を拘束する力をもっているものと異なる。もっとも音は称しても、技術的分野では、電信規則のごとき、各国政府が承認してその内実施する強制規則をもたないので、実際にある機器の仕様を定める場合には、の国の意見が統一されたこの「意見」に従わなければ、円滑な国際通信を行なとができない場合が多い。この意見（または勧告）は、国際通信を行なう場合が直面する問題について、具体的意見を表明するもので、たとえば、大陸間ルで大陸間通話を半自動化しようとする場合、その信号方式や取り扱う通話のおよび料金は、どのようにするかを研究して意見を表明する。したがって、CTTの活動は、つねに時代の最先端を行くもので、CCITTの活動方向は、まま世界の国際通信の活動方向であるともいえる。

この意見は、また、電信規則以下のその他の規則のごとく、数年以上の間隔って開催される主管庁会議というような大会議の決定をまたなくても表明することができ、また、その改正も容易であるので、現在のように進歩の早い国際通信は、関係国の意見を統一した国際的見解としては非常に便利である。

memorandum

FROM: A.P. Sproggs Research	TO: G.V. Smith Project Planning
TEL EXTN: 2041	DATE: 1-9-71

We know that, where possible, data is reduced to alphanumeric form for transmission by communication systems. However, this can be expensive, and also some data must remain in graphic form. For example, we cannot key-punch an engineering drawing or weather map.

I think we should realise that high speed facsimile transmissions are needed to overcome our problems in efficient graphic data communication. We need research into graphics data compression.

Any comments?

Albert.

WELL, WE ASKED FOR IT!

C.2 Example of Resolution Reduction for Progressive Transmission

On the following five pages, an example of an image as it would be displayed in a progressive transmission system are presented. The first image represents a 12.5 DPI version of the image that would be displayed very quickly. Each successive image represents twice the resolution of the previous image, but takes longer to transmit. The final (fifth) image is the original 200 DPI version of the image. These images are generated by the default JBIG resolution reduction algorithm.

[redacted]

[redacted]

[redacted]

THE SLEREXE COMPANY LIMITED

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

THE SLEREXE COMPANY LIMITED

SAUNDERS LANE · BOOKLE · DORSET · BH15 4RR

TELEPHONE BOOKLE (0945) 51617 · TELEX 12345

Our Ref. 150/RJC/PAC 16th January, 1971.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Estroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile communication.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH 25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

C.3 Examples of Images with Random Noise

In section 4.4 on page 63, an experiment was performed in which random noise was added to the set of test images. In this experiment, noise levels of 0.01%, 0.03%, 0.1%, 0.3%, 1%, and 3% were simulated. Examples of these noise levels are given on the following six pages for one of the test images.

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (94513) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (94513) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE · BOOLE · DORSET · BH25 8ER

TELEPHONE BOOLE (945 13) 51617 · TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE · BOOLE · DORSET · BH25 8ER

TELEPHONE BOOLE (945 13) 51617 · TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

C.4 Examples of Noisy Images after Filtering

In section 4.4 on page 63, an experiment was performed in which random noise was added to the set of test images. In this experiment, noise levels of 0.01%, 0.03%, 0.1%, 0.3%, 1%, and 3% were simulated. These images were then filtered to remove isolated pixels. Examples of the filtered images for each of the noise levels are given on the following six pages for one of the test images.

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

THE SLEREXE COMPANY LIMITED

SAPORS LANE · BOOLE · DORSET · BH25 8ER

TELEPHONE BOOLE (945 13) 51617 · TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

C.5 Halftone Test Images

The images on the following pages are examples of halftoned images that are used in section 4.5. These images are part of the 'Stockholm' image set that was created for testing the JBIG algorithm. The following table describes each of the images. The name refers to the standard name of the image.

Image #	Name	Width (pixels)	Height (pixels)	Description
1	S04A	3072	2048	halftoned sailboat
2	S09	1024	1024	error-diffused building

The first image is been rotated 90 degrees and printed at 400 dots per inch. The second image is printed at 200 dots per inch.

0.90
0.90