

**UNCONSTRAINED BLOCK TILING  
FOR COMPRESSION OF HIGH RESOLUTION  
BI-LEVEL IMAGES**

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE  
UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

AUGUST 1991

by

Gopal Krishna

Thesis Committee:

Kenneth Zeger, Chairperson  
Edward J. Weldon Jr.  
Norman Abramson  
Anthony Kuh

We certify that we have read this thesis and that, in our opinion, it is satisfactory in scope and quality as a thesis for the degree of Master of Science in Electrical Engineering.

Thesis Committee

---

Chairperson

©Copyright 1991  
by  
Gopal Krishna

**To my parents with love**

# Acknowledgements

I would like to express my sincere gratitude to Professor Zeger for his invaluable guidance and advice throughout my thesis. Thanks to him, I learned several lessons which I am sure will come a long way in my life. A big fat mahalo to Professor Abramson, Professor Weldon and Professor Kuh for agreeing to serve on my thesis committee. I feel privileged to have come in contact with and worked on small but interesting projects with Professor Abramson and Professor Weldon. Although not directly related to my thesis, the support and guidance I recieved from Professor Holm-Kennedy influenced me in more than one way. My conversations with him were always moral boosters for me. For their help and support in spurring me through, a sometimes frustrating path of research, I would like to thank all of my friends, especially Paul, VG, DP, Hari, Marc, Chak and Vik. Special thanks to Dan for providing a code to display images on X-windows. Last, but not the least, I would like to thank my parents and family for their confidence in me. I dedicate this thesis to my parents.

# Abstract

In this thesis, a new simple-to-implement technique, called Unconstrained Block Tiling (UBT), for the lossless compression of bi-level images for facsimile transmission and storage is proposed that exploits two-dimensional neighboring pixel redundancies. A bi-level image is initially partitioned into non-overlapping rectangles of arbitrary dimensions that cover either only white pixels or a mixture of black and white pixels. An entropy coded index is used to specify the quantized length and width of each rectangle. The rectangles containing some black pixels are encoded using spatial prediction and known coding techniques. The idea of partitioning the image into rectangles extends the notion of run-length coding into two dimensions. The key feature of the proposed UBT is that it reduces the problem of compression of a bi-level image to that of coding of pixels in well defined regions which cover a relatively very small area in a typical facsimile document.

# Table of Contents

<b>Acknowledgements</b> .....	<b>v</b>
<b>Abstract</b> .....	<b>vi</b>
<b>List of Tables</b> .....	<b>x</b>
<b>List of Figures</b> .....	<b>xi</b>
<b>1 Introduction</b> .....	<b>1</b>
1.1 Previous Work .....	2
1.2 Background .....	4
1.2.1 Definitions .....	4
1.2.2 Prediction Techniques .....	5
1.2.3 Residual Coding .....	6
1.2.4 CCITT Recommendations .....	8
1.3 Outline of the thesis .....	9
<b>2 Predictive Compression</b> .....	<b>11</b>
2.1 Definitions .....	11
2.1.1 Prediction Error .....	11
2.1.2 Prediction Error Probability .....	12
2.1.3 Prediction Factor .....	12

2.2	Prediction Techniques .....	13
2.2.1	Vertical XOR .....	13
2.2.2	Horizontal XOR .....	14
2.2.3	Two-dimensional Prediction .....	15
<b>3</b>	<b>Unconstrained Tiling .....</b>	<b>19</b>
3.1	Introduction .....	20
3.2	Sequential Tiling .....	22
3.2.1	Limitation of Sequential Tiling .....	22
3.3	Block Growing .....	26
3.3.1	Maximal One-Dimensional Growing .....	26
3.3.2	Omni-dimensional Growing .....	29
3.3.3	Choice of Birth Location .....	31
3.4	Non-white Rectangles (NWR) .....	32
<b>4</b>	<b>Coding of Non-white Regions .....</b>	<b>35</b>
4.1	Entropy Coding .....	35
4.2	Block Coding .....	36
4.3	Run-length Coding .....	37
4.4	Run-length Block Coding .....	39
4.5	Conditional Entropy Coding .....	40
<b>5</b>	<b>Implementation Issues .....</b>	<b>41</b>
5.1	Prediction .....	42
5.2	Covering of Black Pixels: Non-white Regions .....	43
5.3	Removal of White Rows .....	44
5.4	Tiling of White Space .....	44



<b>6</b>	<b>Results and discussion</b> .....	<b>46</b>
6.1	Prediction .....	46
6.2	NWR Compression Ratio .....	49
6.3	Contribution of different techniques .....	54
6.4	A Comparative Study .....	65
<b>7</b>	<b>Conclusion and Future Directions</b> .....	<b>71</b>
	<b>Appendix A</b> .....	<b>73</b>
	<b>Appendix B</b> .....	<b>89</b>
	<b>Bibliography</b> .....	<b>91</b>

# List of Tables

1.1	Test Image Descriptions .....	10
2.1	Comparison of Prediction Techniques.....	14
5.1	Gain in the number of all white rows following prediction .....	44
6.1	Compression ratios achieved by various techniques for <b>letter type</b> image	69
6.2	Compression ratios achieved by various techniques for <b>roman type</b> image .....	70
6.3	Compression ratios achieved by various techniques for <b>handwriting</b> <b>type</b> image .....	70

# List of Figures

1.1	Differential Pulse Code Modulation . . . . .	5
2.1	Definition of immediate and secondary neighbors . . . . .	16
2.2	Two-dimensional predictor . . . . .	17
2.3	A 7-pixel prediction window . . . . .	17
3.1	Sequential Tiling Technique . . . . .	21
3.2	An example of Sequential Tiling Technique . . . . .	23
3.3	Modified tiling of the image shown in figure 3.2 . . . . .	24
3.4	Limitations of sequential tiling technique . . . . .	25
3.5	Maximal One-dimensional Growing Technique . . . . .	27
3.6	Omni-dimensional Growing Technique . . . . .	28
3.7	An example of Block Growing Technique . . . . .	30
3.8	Covering of black pixels: non-white regions . . . . .	34
4.1	Block Coding . . . . .	36
4.2	Run-length coding . . . . .	38
4.3	Run-length block coding . . . . .	39
6.1	Prediction window size vs. prediction factor . . . . .	47
6.2	Prediction windows . . . . .	48
6.3	NWR compression ratio vs. length of the square block . . . . .	50
6.4	NWR compression ratio vs. # of elements in the sliding window . . . . .	51
6.5	NWR compression ratio vs. length of longest run . . . . .	52

6.6	NWR compression ratio vs. block length . . . . .	53
6.7	Share of WRRC, BT, and CNWR in covering the Letter type image . . .	55
6.8	Share of WRRC, BT, and CNWR in covering the Roman type image . .	56
6.9	Share of WRRC, BT, and CNWR in covering the Handwriting type image . . . . .	57
6.10	Share of WRRC, BT, and CNWR in saving bits: Letter type image . . .	58
6.11	Share of WRRC, BT, and CNWR in saving bits: Roman type image . .	59
6.12	Share of WRRC, BT, and CNWR in saving bits: Handwriting type image . . . . .	60
6.13	Share of WRRC, BT, and CNWR in the total number of transmitted bits: Letter type image . . . . .	62
6.14	Share of WRRC, BT, and CNWR in the total number of transmitted bits: Roman type image . . . . .	63
6.15	Share of WRRC, BT, and CNWR in the total number of transmitted bits: Handwriting type image . . . . .	64
6.16	Compression ratio vs. techniques for compression: Letter type image . .	66
6.17	Compression ratio vs. techniques for compression: Roman type image .	67
6.18	Compression ratio vs. techniques for compression: Handwriting type image . . . . .	68
.1	Decimated image of the document generating <i>Letter Type</i> image . . . . .	74
.2	Decimated image of the document generating <i>Roman Type</i> image . . . . .	75
.3	Decimated image of the document generating <i>Handwriting Type</i> image .	76
.4	A section of Letter type image . . . . .	77
.5	A section of Roman type image . . . . .	78
.6	A section of Handwriting type image . . . . .	79
.7	Residual Letter type image after Vertical-XOR . . . . .	80

.8	Residual Roman type image after Vertical-XOR .....	81
.9	Residual Handwriting type image after Vertical-XOR.....	82
.10	Residual Letter type image after Horizontal-XOR .....	83
.11	Residual Roman type image after Horizontal-XOR .....	84
.12	Residual Handwriting image after Horizontal-XOR .....	85
.13	Residual Letter type image after 12 element 2-D prediction.....	86
.14	Residual Roman type image after 12 element 2-D prediction.....	87
.15	Residual Handwriting type image after 12 element 2-D prediction .....	88

# Chapter 1

## Introduction

The need for electronic storage and transmission of bi-level images such as line drawings, letters, newsprint, maps, and other documents has been increasing rapidly. Bi-level images contain only one bit per pixel<sup>1</sup>, or in other words consist of only black and white pixels. One of the main applications of bi-level images is in facsimile transmission, where a typical image is represented by several megabits of information. *Lossless data compression techniques* attempt to reduce the average number of bits required to store or transmit images without any distortion to the image. Most bi-level image compression algorithms exploit the fact that most pixels are white and that black pixels occur with a regularity, manifested in the form of characters, symbols, or connected boundaries. There are three basic concepts behind most of the current compression algorithms for coding of bi-level images: coding only transition points<sup>2</sup> between black and white, skipping white, and pattern recognition.

In this thesis a new, simple-to-implement, technique for the lossless compression of bi-level images for facsimile transmission is proposed which takes advantage of both transitional point coding, and skipping white. This technique, called *Unconstrained Block Tiling* (UBT), divides the problem of encoding for compression into two parts:

---

<sup>1</sup>A picture-element or dot is called a pixel.

<sup>2</sup>A transition point is a pixel whose value is different from the previous pixel in the horizontal direction.

partitioning of an image into “white” and “non-white” rectangles, and compression of non-white rectangles or regions. The partitioning of the image is done such that white rectangles cover only white pixels, and non-white rectangles or regions cover black pixels and also some of the surrounding white pixels.

This introductory chapter describes the background and motivation for new work, and outlines the main contributions in the following chapters.

## 1.1 Previous Work

The compression of bi-level images for facsimile transmission has received considerable attention in the literature [1][2][3][4]. Most of the proposed data compression schemes attempt to reduce the redundancy in the image either by using statistical codes for encoding the positions of transitive elements[5], or by reducing the number of transitive elements[6]. A transitive element is defined as one whose value is different from the previous element in the horizontal direction. Run-length coding[7], predictive coding[8], and READ coding[9] are well-known techniques which encode the positions of transitive elements. In run-length coding, the length of each black or white run is encoded by means of Huffman codes[10], Modified Huffman codes[27] or Wyle codes[11], while scanning a picture sequentially line by line from top to bottom. In prediction coding, each pixel is predicted by using some function of neighboring elements and the predicted value is compared with the actual value of the pixel. If the predicted value is different from the actual value of the pixel, an error is committed. The error sequence resulting from prediction is then encoded using run-length coding. READ coding, as well as RAC[12] and EDIC[13], encodes the distance between the start position of a run<sup>3</sup> and that of a certain previously scanned run which satisfies some prescribed conditions.

---

<sup>3</sup>A run is a set of consecutive single valued pixels.

These coding schemes contain two common operations. The first operation is to select the starting position of each run from a given picture. The second operation is to determine the positions of transitive elements in some way, e.g., the distance between two consecutive transitive elements, or equivalently, the length of a run. The data compression in these conventional coding schemes is achieved only through the latter operation.

On the other hand, a technique called Selective Element Coding (SECT)[14] reduces the transitive elements to be selected. Essentially, each selected element corresponds to a “corner” of an object in the picture, but the number of selective elements is smaller than that of corners. The number of selective elements is bounded by the number of 0, 45, and 90 degree boundary lines which form the edges of objects in a picture. Since the number of boundary lines is small relative to the total number of picture elements, SECT reduces the redundancies.

The limitation with the above schemes is that they use one-dimensional redundancies only, and therefore, lack the capability of exploiting the dependencies of a pixel in two dimensions.

Kunt[15] and Johnsen[16] proposed a two-dimensional Block coding which partitions each picture into a set of blocks of size  $n \times m$ . Codeword for an all white block is 0, whereas codewords for other configurations are composed of the  $nm$  bits of the block preceded by the prefix 1. Cohen, et.al.[17] examined a class of two-dimensional image coding methods based upon a recursive hierarchical decomposition of the image into uniform areas, which extends the idea of representing pictures by binary trees[18]. In a binary tree scheme, if the image is uniformly black or white, only one symbol is transmitted – the grey level of the image. If not, the transmitter sends a metasymbol which indicates that the picture is not uniform, and that the code that follows represents two subareas of the picture. The two subareas are created by a



single line (a “cut”) that divides the whole picture into two equal parts. The coding proceeds in this fashion recursively until the transmitted code exhausts all the uniform subareas and sub-subareas of the picture. In the worst case, the coding descends down to the level of single pixels which are, of course, uniform.

## 1.2 Background

Some important terms are introduced below in the format of definition for the ease of reading.

### 1.2.1 Definitions

#### Compression Ratio

Compression ratio, as used in this thesis, is defined as a ratio of the total number of bits in an image to the number of bits required to encode the image.

#### Entropy

Entropy is a measure of randomness of a random variable. For a random variables with  $M$  outcomes  $x_1, x_2, \dots, x_M$  with probabilities  $p_1 = p(x_1), p_2 = p(x_2), \dots, p_M = p(x_M)$ , the *entropy* in bits is defined as

$$H = - \sum_{k=1}^M p_k \log_2 p_k$$

Entropy represents the amount of information associated with the set of coder input values and gives a lower bound on the average number of bits required to code those inputs[19].

#### Block Quantizer

A *noiseless source code* is a code which maps each of the symbols of a set  $S$  into a fixed sequence of the code set  $X$ [20]. These fixed sequences of the code set are called

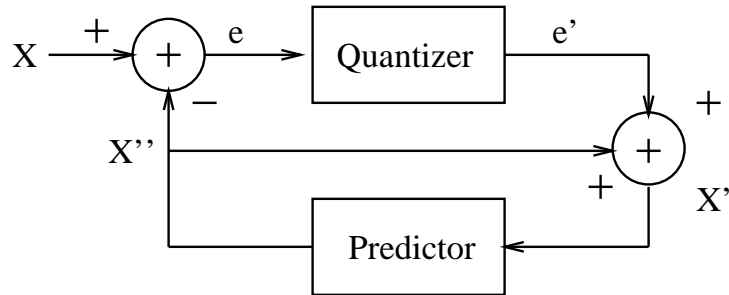


Figure 1.1. Differential Pulse Code Modulation

*code words.*

A *fixed length code* has a set of code words each of which has the same number of bits, whereas a *variable length code* has code words with different number of bits. If a code has the property that a sequence of codewords can be decoded in only one way, the code is called a *uniquely decodable code*.

### 1.2.2 Prediction Techniques

Prediction techniques exploit the inter-dependencies of the neighboring pixels to create a *predicted image*, based on the original image. The value of each element in the predicted image is determined by a function, called the *prediction function*, of some neighboring elements which form the *prediction window*. A *residual image* is obtained by subtracting the predicted image from the original image. Since binary subtraction is same as binary addition, residual image is obtained by simply adding the predicted and original image. The idea of prediction is analogous to that of Differential Pulse-

Code Modulation (DPCM) (Figure 1.1)[21]. If the original image has redundancy, the residual image will have large runs of 0s (or 1s).

A predicted image can be created by using one of many techniques. Simple one-dimensional prediction techniques, like vertical-XOR<sup>4</sup> and horizontal-XOR, are very easy to implement as the prediction window contains only one pixel, namely the previous pixel in the vertical or the horizontal direction. Two-dimensional prediction windows with more than one element reduce the number of prediction errors and thus achieve better prediction, but are more complex to implement. Chapter 2 deals with prediction techniques in detail.

### 1.2.3 Residual Coding

Residual coding is used to encode the residual image after the prediction. There are several useful coding techniques; some of the most known techniques are outlined in the following sections.

#### Entropy Coding

The technique of encoding discrete data into uniquely decodable variable-length code-words in an invertible manner is called *entropy coding*. In entropy coding as applied to noiseless image compression, a block of  $M$  pixels with block probabilities  $p_i$ ,  $i = 0, 1, \dots, L - 1$ ,  $L = 2^M$ , is coded such that the average bit rate is

$$\sum_i p_i (-\log_2 p_i) = H$$

This gives a variable-length code for each block, where highly probable blocks (or symbols) are represented by small-length codes, and vice versa. If  $-\log_2 p_i$  is not an integer, the achieved rate exceeds  $H$  but approaches it asymptotically with increasing block size.

---

<sup>4</sup>XOR refers to the binary *exclusive-OR* operation.

## Huffman Coding

For a given block size the most efficient fixed-to-variable length encoding technique is *Huffman Coding*[10]. In the Huffman coding algorithm, first all the symbol probabilities  $p_i$  are arranged in decreasing order and are considered as leaf nodes of a tree. Then, while there is more than one node the following two steps are executed: two nodes with smallest probability are merged to form a new node whose probability is the sum of the two merged nodes; 1 is assigned to the left and 0 to the right or vice versa to each pair of branches merging into a node. Finally the leaf node where the symbol is located, is read sequentially from the root node.

This algorithm gives the *Huffman code book* for any given set of probabilities. Coding and decoding is done simply by looking up values in a table.

## Arithmetic Coding

Arithmetic coding[22] is another variable length encoding technique. The basic idea of arithmetic coding of a binary source can be described by a binary tree.

The structure of the tree can be viewed as a *classification tree*[21] for points in the unit interval  $[0, 1)$ . The input to the tree is a real number  $r \in [0, 1)$ . Each node makes a binary decision based on  $r$ . The classifier, based on this decision, either outputs a 1 and advances along the (say) right branch emanating from the node or outputs a 0 and advances along the left.

Arithmetic coding is more complicated to implement than Huffman coding, but its compression is typically greater and hence it is a popular approach for entropy coding where the extra compression justifies the extra complexity.

## Run-length Coding

In *run-length coding* (RLC), the output of a binary source is coded in terms of the number of 0s between two successive 1s. In other words, the lengths of the runs of 0s are encoded. This technique is simple to implement and is particularly useful whenever there is a high probability of large runs of 0s in the symbols coming out of a binary source.

### 1.2.4 CCITT Recommendations

The CCITT (Consultative Committee for International Telephone and Telegraph) has recommended international standards for Group 1 facsimile to Group 4 facsimile, and developing recommendations for a new generation of facsimile equipment.

Group 1 and Group 2 are analogue facsimile with transmission times of approximately 6 and 3 minutes, respectively. Group 3 is digital facsimile with a transmission time of approximately 1 minute and uses a one-dimensional run-length coding scheme. An optional data-compression technique specified for Group 3 is a two-dimensional coding scheme known as the modified READ code (MRC)[23]. The compression technique for Group 4 recommendations is known as modified READ code II (MRC II), and the recommendations include a mode of operation known as Mixed Mode where the information printed on a page is divided into two parts – symbols and graphics[24]. The recommendations being developed use a progressive encoding system which transmits a compressed image by first sending the compressed data for a reduced-resolution version of the image and then enhances it as needed by transmitting additional compressed data[25].

## 1.3 Outline of the thesis

The proposed Unconstrained Block Tiling (UBT) scheme consists of three major steps : (i) formation of a predicted image based on the original image, (ii) tiling of the predicted image with white and non-white rectangles, and (iii) coding of the pixels covered by the non-white rectangles (NWR). The subsequent chapters follow this sequence.

Chapter 2 deals with various prediction techniques which exploit the inter-dependencies of the neighboring pixels to form a residual image containing large runs of 0s (or 1s). Some basic definitions related to prediction are given, followed by the description of one- and two-dimensional prediction techniques.

Chapter 3 introduces a novel idea of partitioning an image into two types of rectangles: white and non-white rectangles. Two algorithms for the tiling of white space, called *Block Tiling* (BT) (*sequential tiling* and *block growing*) along with the related issues, are described. The key feature of this technique, besides yielding high compression ratios and being easy to implement, is that it isolates all the black pixels within well-defined regions. The locations and dimensions of these regions are easily communicated to the decoder.

Chapter 4 covers some possible techniques for coding of the pixels covered by the non-white rectangles. It also introduces a new technique called Run-length Block Coding (RLBC).

Chapter 5 addresses the issues related to the implementation of the proposed UBT technique. This chapter describes all the steps and the related trade-offs involved in the implementation of the UBT technique. In addition, the issue of complexity is addressed.

Chapter 6 discusses the results obtained from the simulations of UBT. A com-

<i>No.</i>	<i>Type</i>	<i>Dimensions width x height</i>	<i>% of black pixels</i>
1	Letter type	3072 x 4352	1.768
2	Roman type and figures	3072 x 4352	2.434
3	Hand writing	3072 x 4352	3.217

Table 1.1. Test Image Descriptions

parative study of various prediction, tiling, and coding techniques is given. Three standard bi-level images used for the comparison and an evaluation of the proposed technique are described in the following section.

Chapter 7 draws conclusions based on the results obtained in chapter 6. In addition, some of the directions for future work in this area are given.

### **Test Images**

Three images from the set of Stockholm JBIG images[26] were selected for the evaluation of the proposed technique. These images are summarized in Table 1.1. The table also shows the percent of black pixels in the each image. The resolutions of all these images are 400 dots per inch, and the dimensions are 3072 columns x 4352 rows. In this thesis, these images are referred to by their type, namely *letter type*, *roman type*, or *handwriting type*. Decimated<sup>5</sup> versions of the image generating documents are shown in Appendix A.

---

<sup>5</sup>The images were "decimated" only for the purpose of display

# Chapter 2

## Predictive Compression

Prediction techniques exploit the inter-dependencies of neighboring pixels to create a predicted image, based on the original image.

The first step of the proposed technique is to obtain a residual image by subtracting the predicted image from the original image. A predicted image can be created by one of many techniques. Various prediction techniques and the associated prediction error probabilities are dealt with in this chapter.

### 2.1 Definitions

Before describing prediction techniques, some important terms related to prediction are defined in the following sections.

#### 2.1.1 Prediction Error

For a bi-level image  $u(m, n)$ ,  $m \in 1, \dots, M$ ,  $n \in 1, \dots, N$ , let  $v(m, n)$  denote its predicted value based on the values of pixels in a prediction window  $W$ . A prediction window contains some of the pixels which have already been coded and is defined by the prediction technique. The prediction error can be defined as

$$e(m, n) = \begin{cases} 1, & v(m, n) \neq u(m, n) \\ 0, & v(m, n) = u(m, n) \end{cases}$$

The image is reconstructed from  $e(m, n)$  simply as



$$u(m, n) = v(m, n) \oplus e(m, n),$$

where  $\oplus$  is a binary addition or an exclusive-OR operation.

### 2.1.2 Prediction Error Probability

One reasonable prediction criterion is to minimize the prediction error probability, or the maximum likelihood prediction. For an  $N$ -element prediction window, there are  $2^N$  different states<sup>6</sup>. Let  $S_k$ ,  $k=0,2,\dots,2^{N-1}$  denote the  $k^{\text{th}}$  state of the prediction window  $W$  with probability  $p_k$  and define

$$q_k = \text{Prob}[u(m, n) = 1 | S_k]$$

Then the optimum hard decision prediction rule having minimum prediction error probability is

$$v(m, n) = \begin{cases} 1, & \text{if } q_k \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

If the random sequence  $u(m, n)$  is strict-sense stationary, then the various probabilities will remain constant at every  $(m, n)$ , and therefore the prediction rule need not change.

### 2.1.3 Prediction Factor

For an easy comparison of various prediction techniques, we introduced a quantity that measures a predictor's ability to minimize the prediction error probability:

$$\text{Prediction Factor} = \frac{(\text{number of black pixels in the residual image})}{(\text{number of black pixels in the original image})}$$

---

<sup>6</sup>A state is defined as a valid binary word which characterizes a prediction window.

The above definition assumes that the errors in the residual image are denoted by black pixels (i.e. by 1s). For a predictor to be useful, the value of the prediction factor should be less than 1. For a binary memoryless source, the smaller the prediction factor, the better the predictor. Empirically and by heuristics, in general, this is also the case for a binary source with memory.

## 2.2 Prediction Techniques

The following sections describe various prediction techniques.

### 2.2.1 Vertical XOR

”Vertical XOR” is a simple one-step prediction technique. In this technique, the prediction window contains only one pixel, the pixel above the predicted pixel. If  $u(i, j)$ , and  $u(i, j + 1)$  are the two pixels in the consecutive rows of an image  $X$ , then vertical XOR is defined as

$$e(i, j + 1) = u(i, j) \oplus u(i, j + 1),$$

where  $e(i, j + 1)$  is the residual pixel value.

A pixel in the first row of the residual image  $X$  is obtained by XORing it with an imaginary white pixel.

The residual image can also be created by subtracting a predicted image from the original image. Here, the predicted image is simply the original image shifted downward by one row. This technique is particularly useful for the prediction of an image full of vertical lines, for example engineering drawing. Figures A.7-A.9 show the vertical XORed images. The prediction factors obtained using this technique are given in Table 2.1.

<i>Technique</i>	<i>Image</i>	<i>% of black pixels</i>		<i>Prediction factor</i>
		<i>original image</i>	<i>residual image</i>	
Vertical-XOR	Letter type	1.768	0.36	0.2
	Roman type	2.434	0.45	0.18
	Handwriting	3.217	0.91	0.28
Horizontal-XOR	Letter type	1.768	0.34	0.19
	Roman type	2.434	0.43	0.177
	Handwriting	3.217	0.76	0.238
2-D prediction	Letter type	1.768	0.217	0.123
	Roman type	2.434	0.26	0.107
	Handwriting	3.217	0.56	0.176

Table 2.1. Comparison of Prediction Techniques

### 2.2.2 Horizontal XOR

Horizontal XOR is similar to Vertical XOR, except that the prediction window pixel is one that is to the immediate left of the predicted pixel. If  $u(i, j)$  and  $u(i + 1, j)$  are two consecutive pixels in a row, then the horizontal XOR is defined as

$$e(i + 1, j) = u(i + 1, j) \oplus u(i, j),$$

where  $e(i + 1, j)$  is the residual pixel value or the prediction error.

A pixel in the first column of the residual image  $X$  is obtained by XORing it with an imaginary white pixel.

In the horizontal-XOR technique, a predicted image is simply the original image shifted by one column to its right. This technique is particularly useful for the prediction of an image full of horizontal lines. Figures A.10-A.12 show the horizontal XORed images. The prediction factors obtained using this technique are given in Table 2.1

### 2.2.3 Two-dimensional Prediction

Simple one-dimensional techniques (e.g. Horizontal- and Vertical-XOR techniques) do not necessarily take full advantage of a pixel's dependency on all of its neighbors. They only consider one pixel while predicting a pixel's value. In this context, we define the 'neighborhood' of a pixel as shown in Figure 2.1. The eight pixels marked by the  $\alpha$  form in the 8-neighborhood of the pixel marked by "?".

Two-dimensional prediction techniques utilize the predicted pixel's dependency on not only the immediate neighbors, but also on one or more secondary neighbors. The secondary neighbors are defined in Figure 2.1. The sixteen pixels marked by  $\beta$  are the secondary neighbors of the pixel marked by "?". A two-dimensional prediction window for the prediction of a pixel  $p$  can contain only those neighbors of  $p$  which follow the *causality*<sup>7</sup> principle.

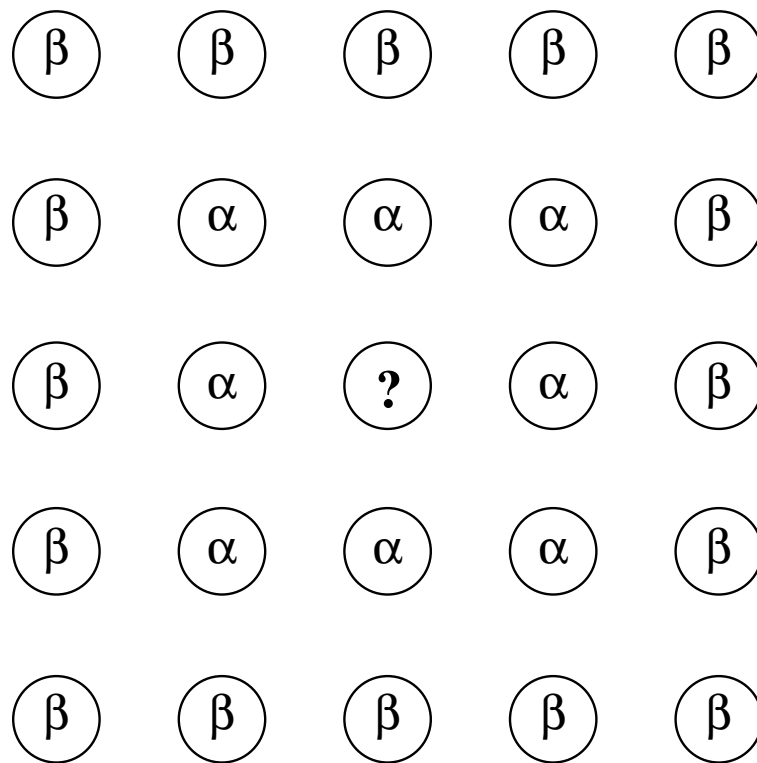
A two-dimensional predictor function is shown in Figure 2.2. The function  $f(a, b, c, d)$  maps a quadruplet  $(a, b, c, d)$  to 0 or 1, whichever has a higher relative frequency conditioned on the event that this quadruplet occurs. The conditional relative frequencies are obtained from experimental data. If the prediction is correct,  $e$  is set to 0, otherwise  $e$  is set to 1. As a result, the original image is transformed into an error image which contains a significantly smaller number of 1s as compared to the original image or the XORed image.

Figures A.12-A.14 show the residual images after using a 7-element prediction window which is shown in Figure 2.3 . The predictor factors obtained from using this 7-element prediction window are given in the Table II.1.

Since for an  $N$ -element prediction window, there are  $2^N$  different states, in practice a suitable choice of  $N$  has to be made to achieve a trade-off between prediction factor

---

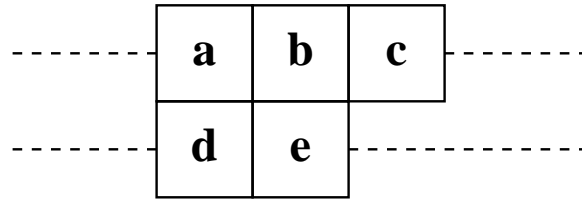
<sup>7</sup>The *causality* principle states that for a system to be causal, the output for any  $n = n_0$  must depend on the input for  $n \leq n_0$  only.



$\alpha$ : **Immediate neighbors**

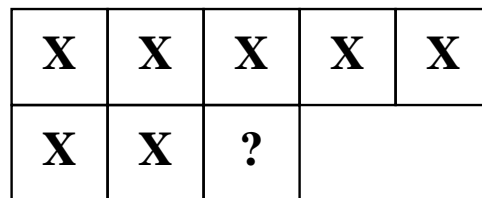
$\beta$ : **Secondary neighbors**

Figure 2.1. Definition of immediate and secondary neighbors



Prediction function =  $f(a,b,c,d)$

Figure 2.2. Two-dimensional predictor



? : pixel to be predicted  
 X : elements of prediction window W

Figure 2.3. A 7-pixel prediction window

and the complexity of the predictor due to large values of  $N$ .

# Chapter 3

## Unconstrained Tiling

Unlike current bi-level compression techniques, the UBT technique divides the problem of coding for compression into two parts:

(i) isolating the black pixels: all the black pixels in the image are isolated within rectangles; sizes and locations of which are known to both the encoder and the decoder. These rectangles, called *non-white rectangles* (NWR) or *non-white regions*, cover all the black pixels and some of the surrounding white pixels. The regions not covered by the non-white rectangles are tiled with *white-rectangles*. The dimensions of the white rectangles are chosen from a set of pre-defined quantized dimensions.

(ii) encoding of covered pixels: pixels covered by non-white rectangles are then encoded using known techniques described in Chapter 4.

In this thesis, a new technique is introduced for tiling<sup>8</sup> bi-level images into rectangles of quantized dimensions for high data compression is introduced. This chapter presents the idea of tiling and describes various techniques for the tiling of bi-level images for data compression.

---

<sup>8</sup>In this thesis, tiling refers to covering of an uniform area (i.e. black or white) with non-overlapping rectangles of pre-defined dimensions.



## 3.1 Introduction

Unconstrained Tiling partitions the image into two sets of rectangles, the dimensions of which conform to a pre-defined set of quantized dimensions: one covering only white pixels, and the other covering a mixture of black and white pixels. While scanning from left to right and top to bottom, first all the black pixels and some of the surrounding pixels are covered by non-white rectangles (e.g. in a business document, each letter, word or sentence can be covered by the non-white rectangles). Then, the remaining pixels are tiled with white rectangles of quantized dimensions, such that each white rectangle covers the *maximum possible area*. The motivation behind tiling white space with the rectangles of maximum possible area is to maximize the *white region compression ratio*<sup>9</sup> by minimizing the total number of rectangles which tile the white space. We call this technique of tiling an image with rectangles *block tiling*. The idea of partitioning the image into rectangles extends the notion of run-length coding into two dimensions and utilizes the structured distribution of white space in a typical facsimile transmission. A similar technique for compression of two-dimensional data using Peano-Hilbert plane-filling curves [29] was investigated by A. Lempel and J. Ziv [28].

The following sections describe the algorithms for the covering of non-white regions and the tiling of white space. Two techniques for the tiling of white space, namely *sequential tiling* and *block growing*, are introduced in the first two sections. The last section addresses the issue of covering non-white regions.

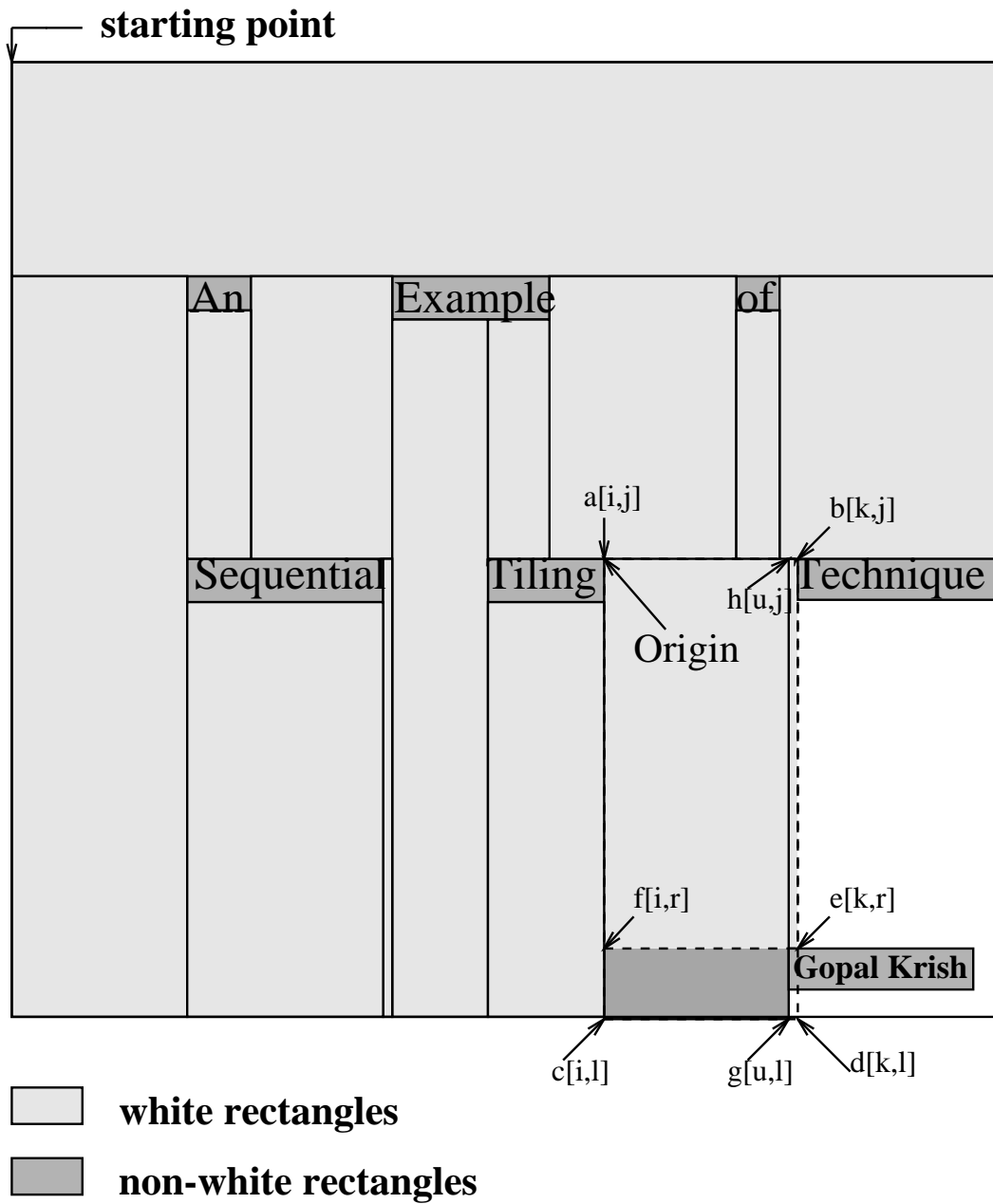


Figure 3.1. Sequential Tiling Technique

## 3.2 Sequential Tiling

Sequential Tiling is a technique for tiling the white space of an image into rectangles of quantized dimensions. After isolating non-white regions (described in Section 3.4), the image is raster scanned from left to right. If a pixel  $a[i, j]$ , shown in Figure 3.1, where  $i$  and  $j$  denote the  $i$ th column and  $j$ th row of the image, respectively, is white and not covered by any rectangle, it is designated as an *origin* at a particular time. The origin is always the left hand upper corner of any valid rectangle. Scanning is continued starting from an origin until reached (1) a boundary pixel of a non-white rectangle, or (2) a pixel at the image boundary, or (3) a black pixel,  $b[k, j]$ . Starting from  $a[i, j]$  all the pixels in the  $i$ th column are scanned until a boundary pixel  $c[i, l]$  (same as  $b[k, j]$ ) is reached. Now, a rectangle with the origin as its upper lefthand corner is determined such that it covers the maximum area within the region defined by the points  $a[i, j]$ ,  $b[k, j]$ ,  $c[i, l]$  and  $d[k, l]$ , as illustrated in Figure 3.1. The area is defined as the product of the *Length* and *Width*, where the length is the distance of the upper right hand corner pixel from the origin, and the width is the distance of the lower left hand corner pixel from the origin. The algorithm for finding rectangles of maximum area is described in Appendix B.

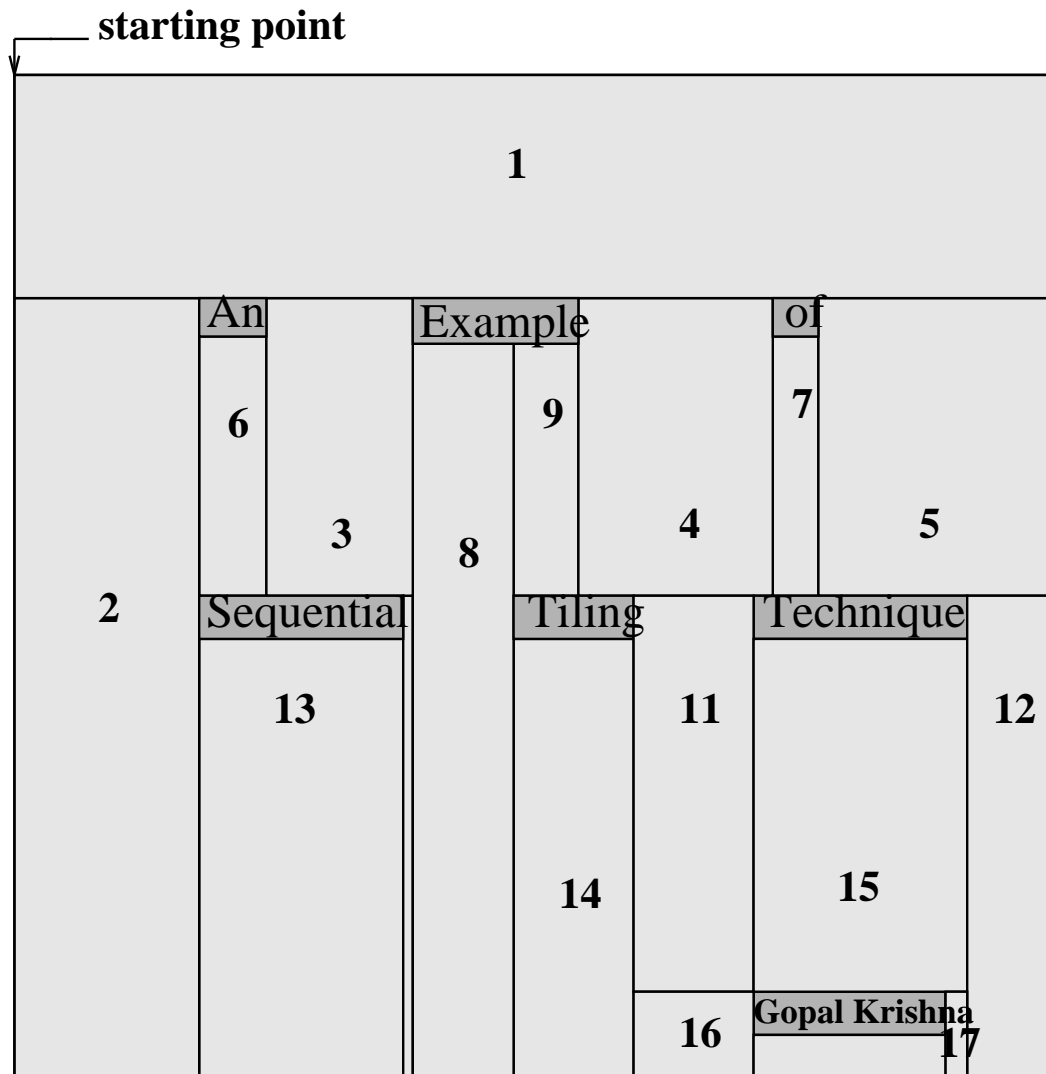
Figure 3.2 shows an image tiled using this technique. The number on each rectangle in the figure indicates the order in which rectangles were tiled.

### 3.2.1 Limitation of Sequential Tiling

Due to the sequential nature of raster scanning, the area covered by the rectangle determined by the sequential tiling algorithm is very often not maximized. In order to achieve high compression ratios it is desirable to tile an image with the minimum

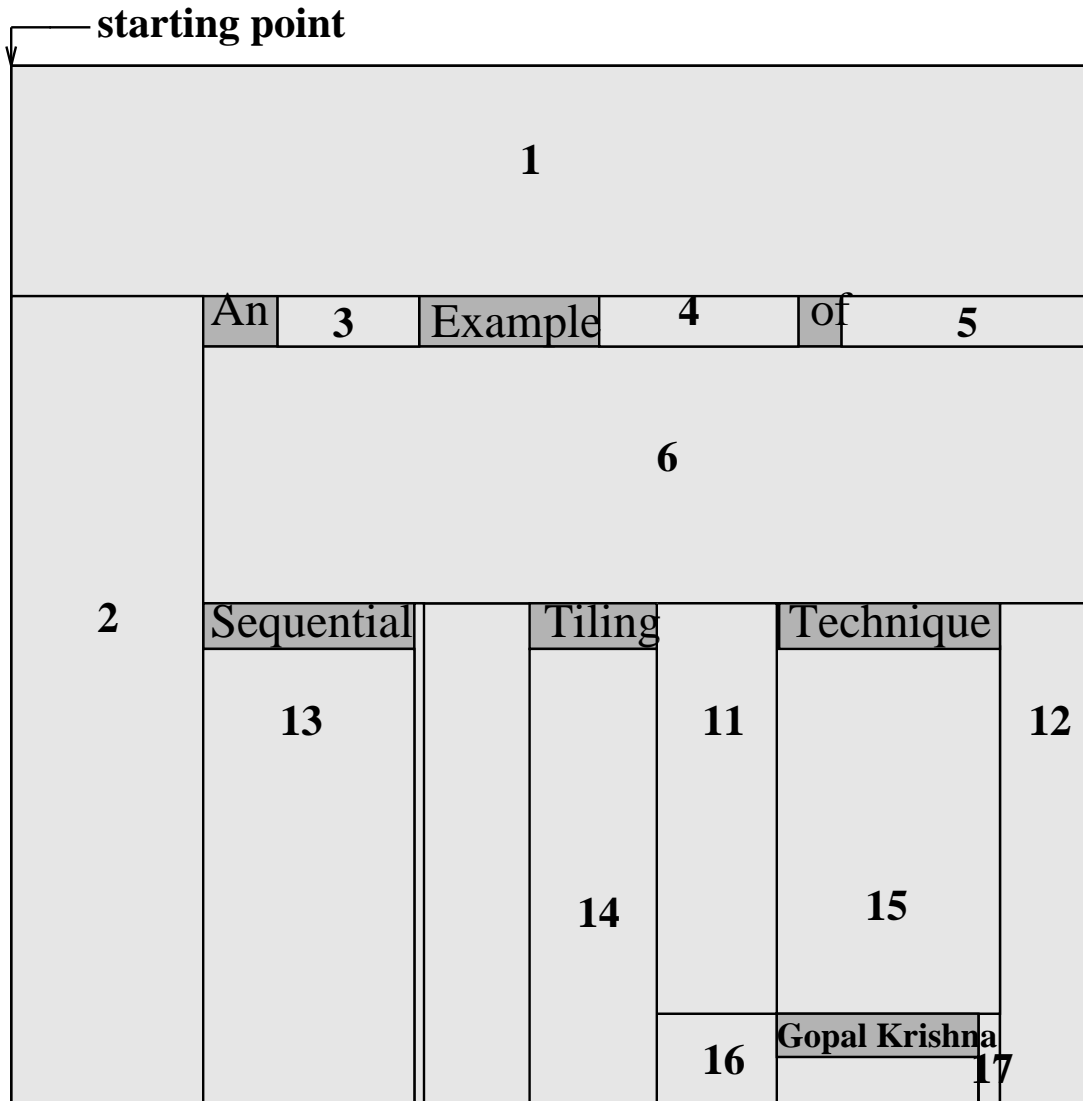
---

<sup>9</sup>White region compression ratio is defined as a ratio of the total number of bits in the white space of an image to the number of bits required to tile the entire white space.



- white rectangles**
- non-white rectangles**

Figure 3.2. An example of Sequential Tiling Technique



- white rectangles
- non-white rectangles

Figure 3.3. Modified tiling of the image shown in figure 3.2

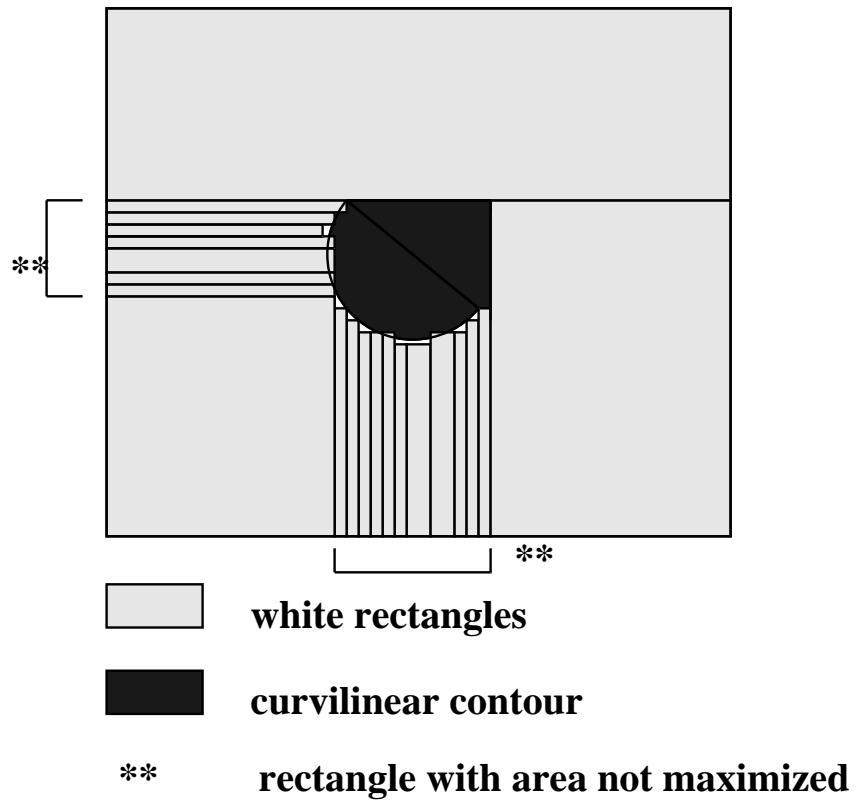


Figure 3.4. Limitations of sequential tiling technique

number of rectangles. In Figure 3.2 the space between the first two sentences is tiled with seven rectangles. Figure 3.3 shows the tiling of the same space with only four rectangles. However, the sequential nature of this technique prohibits the tiling of the white space as done in Figure 3.3. The rectangles cover even smaller areas if the black pixels follow curvilinear contours, as shown in Figure 3.4. As the curvature of the contour increases, the rectangles become increasingly thin stripes.

### 3.3 Block Growing

Another technique of tiling the white space of the image is called *block growing*. In the block growing technique, a rectangle is grown around a white pixel  $p$ , called the *birth location* of the grown rectangle, until it covers the maximum possible area. A rectangle thus generated is called a *grown rectangle*. The four boundaries (left, right, top and bottom) of the grown rectangle define the scan length and the scan width of the rectangle. The scan length and width are the x and y dimensions of the grown rectangle. These scan lengths and scan widths are truncated to the nearest of the defined quantized lengths and widths. Block growing is a powerful technique for tiling which doesn't suffer from the limitations of sequential tiling.

Several methods of block growing were investigated in this research. Two of the most efficient methods are described in the following sections.

#### 3.3.1 Maximal One-Dimensional Growing

In the maximal one-dimensional<sup>10</sup> growing technique, a rectangle is allowed to grow in only one dimension (x or y-direction) until it has grown to its maximum limit in that dimension (i.e. until it has hit a transition or image boundary element on the both sides of the birth location) before any growth is allowed in the remaining dimension.

---

<sup>10</sup>The word "dimension" refers to the length and width of a grown rectangle.

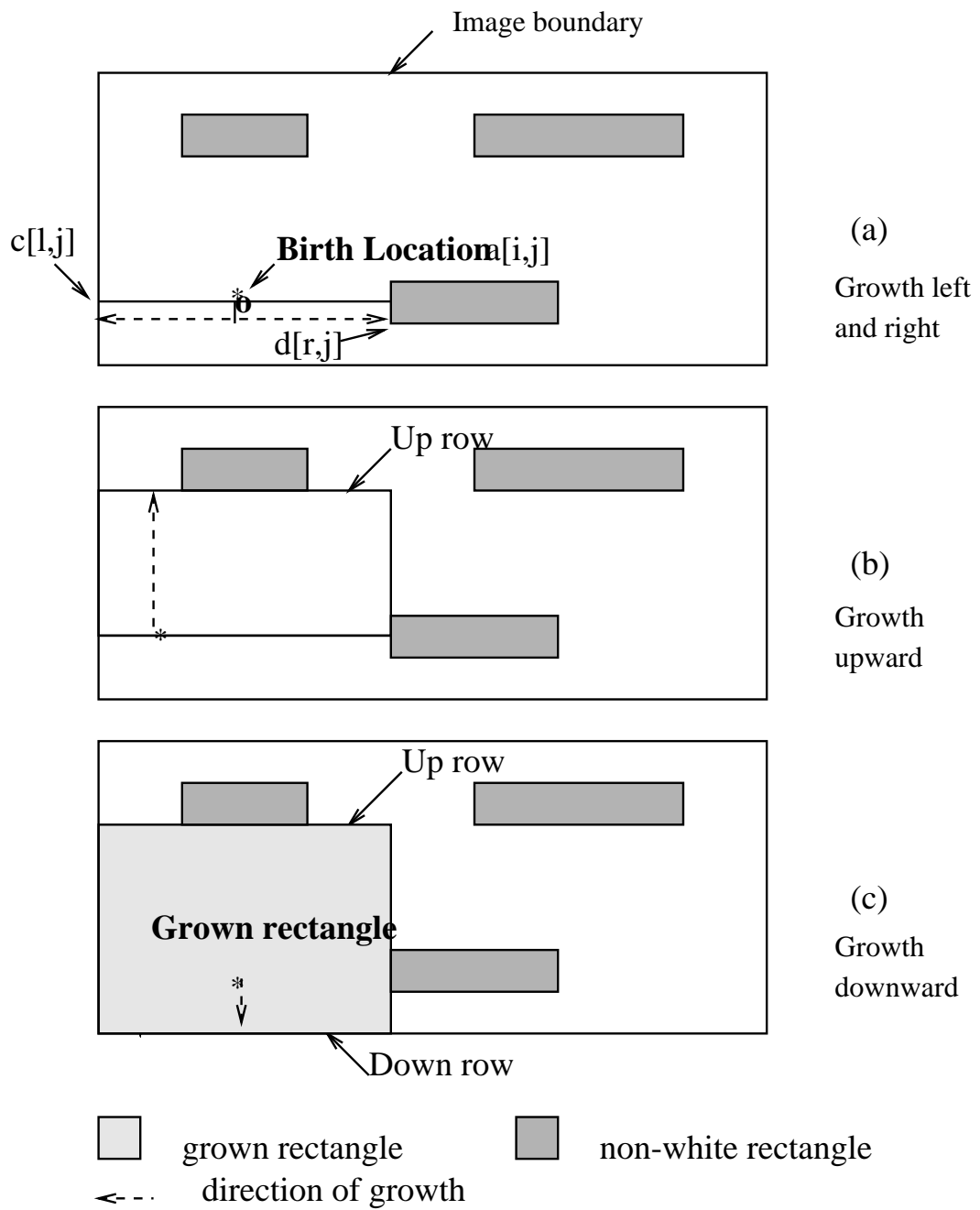


Figure 3.5. Maximal One-dimensional Growing Technique



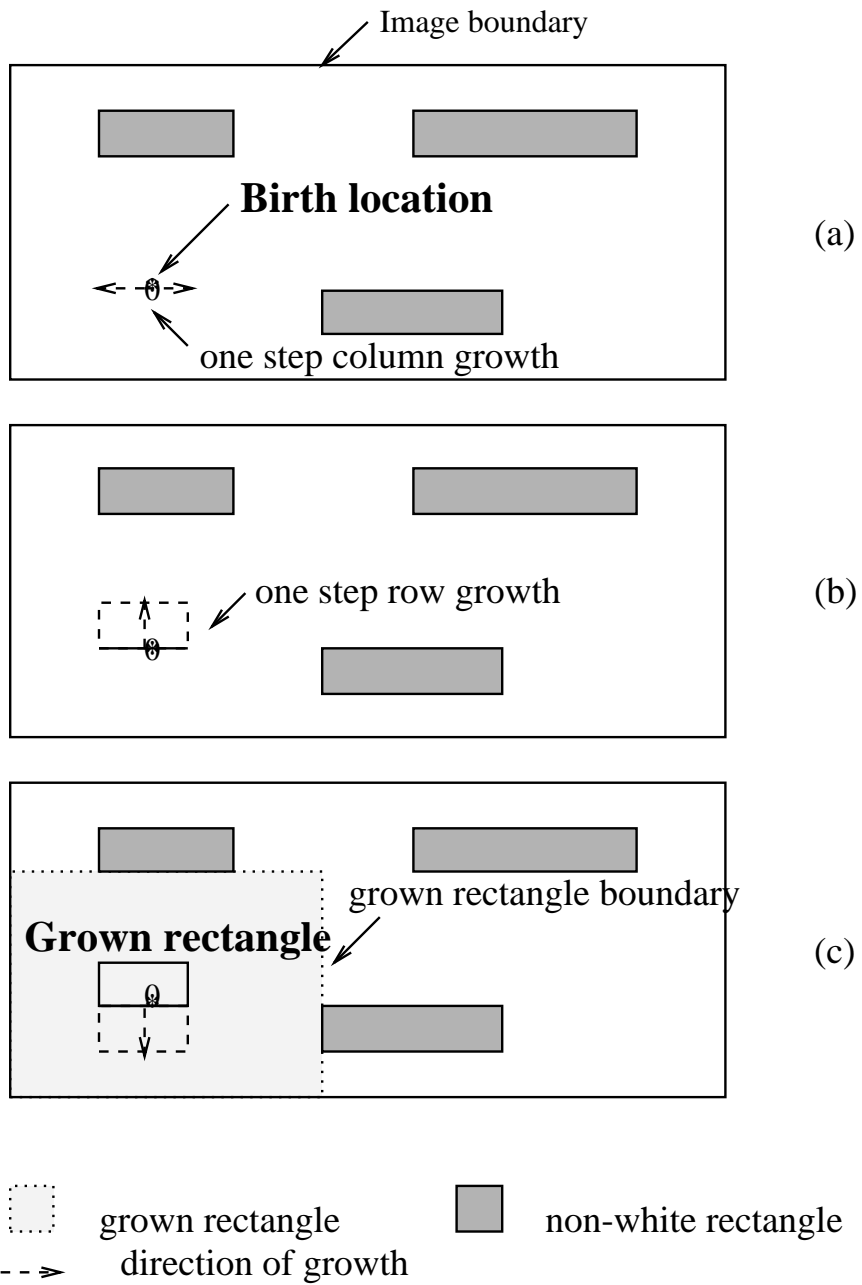


Figure 3.6. Omni-dimensional Growing Technique

For example, as shown in Figure 3.5a, let  $a[i, j]$  be a birth location, where  $i$  and  $j$  denote column and row, respectively. Suppose the algorithm first allows only the horizontal growth. Then, a rectangle can grow to the left and right directions of its birth location, such that it meets  $c[l, j]$  and  $d[r, j]$  as its left and right boundary elements.  $c[l, j]$  is either a transition element left of birth location, or a left hand image boundary element in the horizontal direction. Similarly,  $d[r, j]$  is either a transition element right of birth location, or a right hand image boundary element. Once the rectangle has grown to the maximum possible dimension in the horizontal direction, it is allowed to grow in the vertical direction. Vertical growth can be understood by imagining that  $c[l, j]$  and  $d[r, j]$  are connected by a string. The string is moved one row at a time in the upward direction, until it hits either a row, *up row*, as shown in Figure 3.5b, with at least one transition element between the columns  $l$  and  $r$ , or the upper boundary of the image. Similarly, *down row* can be determined by moving the string in the downward direction.  $c[l, j]$ ,  $d[r, j]$ , *uprow*, and *downrow* define the dimensions of the grown rectangle. Figure 3.5c shows the grown rectangle.

### 3.3.2 Omni-dimensional Growing

Unlike maximal one-dimensional growing, omni-dimensional growing allows rectangles to grow in all the dimensions around the birth location (BL) in a round-robin manner. Sequence assignment for round-robin growth is arbitrary. Each round allows a rectangle to grow in each dimension by only one step. A step is equivalent to one column or one row increment depending on whether the step is for a horizontal or vertical growth. Any directional growth which has reached (i) a transition element, or (ii) an image boundary pixel, or (iii) a non-white rectangle boundary element is eliminated from further participation in the round-robin growth. Round-robin growth terminates when all four directional growths get eliminated. Figure 3.6 illustrates the

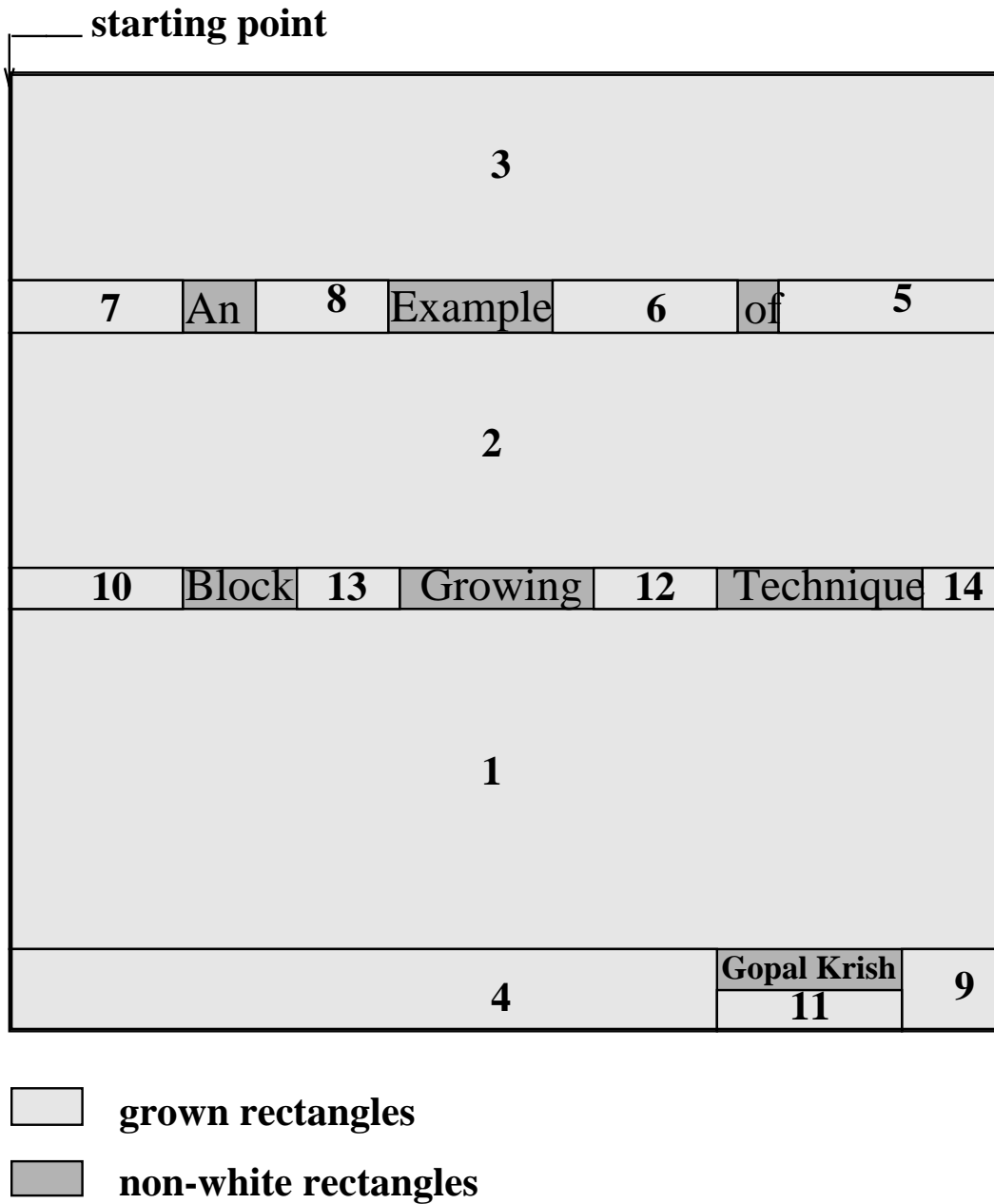


Figure 3.7. An example of Block Growing Technique

idea of the omni-dimensional growing technique. In Figure 3.6a, the growing starts from the birth location with a one step column growth located to the right of the birth location, followed by a one step column growth located to the left of the birth location. Next, (Figure 3.6b) a one step row growth in the upward direction is allowed, followed by one in the downward direction. The directional growths are eliminated in the following order: (1) one step column growth in the negative  $x$  direction, (2) one step row growth in the negative  $y$  direction, (3) one step row growth in the positive  $y$  direction, and finally (4) one step column growth in the positive  $x$  direction.

Figure 3.7 shows an example of the block growing technique. The numbers on the rectangles represent the order in which the rectangles were grown. In this example, the entire white space is tiled with only fourteen rectangles. The sequential tiling technique needs seventeen rectangles to tile the same white space (Figure 3.2).

### 3.3.3 Choice of Birth Location

The distribution of birth locations across white regions for potential grown rectangles determines the efficiency of the growing technique in the tiling of the white regions of an image. The efficiency of a growing technique is defined here by the number of rectangles required to cover all the white regions in the image. The smaller the number of rectangles required, the higher the efficiency, as the objective is to reduce the number of rectangles.

Two techniques for determination of birth locations are described below.

#### Random Jump

We investigated a technique we call the "random jumping", in which a pixel is randomly picked and its eligibility as a birth location is checked. If the pixel is white and is not covered by any other grown rectangle, it qualifies as a birth location.

It was found that in some cases the random nature of the selection of birth location may reduce the efficiency and may also lead to an undeterministic mode, while introducing a random delay in tiling of the white regions of an image.

### **Raster Scan**

The other technique investigated determines the birth locations by "raster scanning". In this technique, the image is raster scanned from left to right and top to bottom. Every pixel which is white and not covered by any other grown rectangle qualifies as a birth location. This technique of finding birth locations may lead to the problem faced by the sequentially tiled rectangles – they may not cover the maximum possible area. This will result in a lower efficiency.

One technique we chose to use to overcome this problem is to grow rectangles in several *passes*, with a defined *threshold area* for each pass. If the area of a grown rectangle is less than that of the threshold for a pass, then the rectangle is ignored. The pixels covered by the ignored rectangle are still marked as eligible birth locations and are available to be covered by other valid grown rectangles. For instance, Figure 3.7 was tiled in three passes. The first pass yielded only one rectangle (#1). The second yielded #2 and #3. The rest were grown in the third pass.

## **3.4 Non-white Rectangles (NWR)**

In the proposed UBT technique, all the black pixels are covered by non-white rectangles, before any tiling of white space is done. The object is to cover each *target* with the smallest possible rectangle allowed by the pre-defined quantized dimensions. Here, a *target* is a cluster of pixels, and is defined by the UBT algorithm. For instance, in an image consists of only English text a target can be a letter, word or sentence (including numbers and figures). Pixels belonging to a target are called the

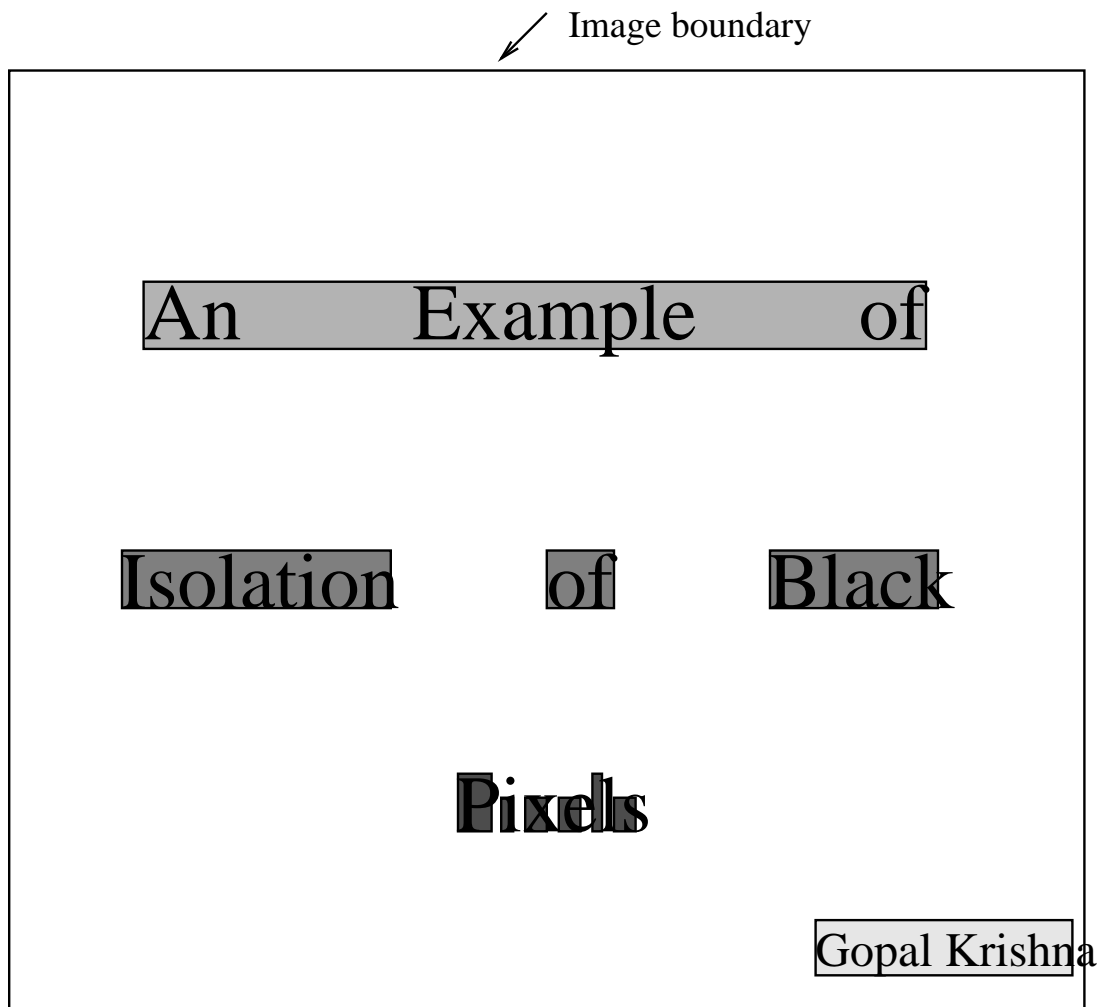
*target pixels*. In contrast to white rectangles which only contain single shaded pixels (white), non-white rectangles cover a mixture of black and white pixels. Figure 3.8 shows an example of the covering of the black pixels in an image consisting of only English text, when the target is a letter, word and sentence, respectively.

Isolation of black pixels with non-white rectangles serve two purposes:

(i) It allows a structured and efficient tiling of the white space which results in high compression ratios, and

(ii) The black pixels are localized within well defined regions, locations and dimensions of which are very easily known to the decoder.

Non-white rectangles can be determined by modifying Sequential tiling or Block Growing techniques discussed in the previous two sections.






-  Non-white rectangle allowed to cover a complete sentence
-  Non-white rectangle allowed to cover individual words
-  Non-white rectangle allowed to cover a letter only

Figure 3.8. Covering of black pixels: non-white regions

# Chapter 4

## Coding of Non-white Regions

Once the predicted image is tiled with white and non-white rectangles using the techniques described in the previous chapter, any of the known bi-level image coding techniques (e.g. run-length coding, READ coding and entropy coding) can be used to encode the target pixels (pixels covered by the non-white rectangles). No coding is required for the pixels covered by the white rectangles. After tiling, the image is raster scanned and the dimensions of the white and non-white rectangles are entropy coded and transmitted. Finally, the target pixels are encoded using a modified version of any of the several known bi-level image coding techniques.

In this chapter several coding techniques to encode pixels covered by non-white rectangles are discussed.

### 4.1 Entropy Coding

One of the simplest techniques of encoding target pixels is entropy coding. Assuming the image to be a discrete memoryless source, the probability of 0 among the target pixels is ascertained. This probability is used to compute the approximate compression ratio achieved in encoding the target pixels using Arithmetic coding. We define the compression ratio in non-white rectangles as the ratio of total number of target pixels to the total number of bits required to encode all the target pixels.



1 1	1 0	0 0
1 0	1 0	1 1
1 1	1 0	1 0
0 1	0 1	1 0
0 1	1 1	0 0
0 1	1 1	0 0

Image partition by 2x2 blocks or patterns

14,10,3,13,9,10,5,15,0

Pattern numbers for the corresponding codeword assignment from the look-up table

Figure 4.1. Block Coding

## 4.2 Block Coding

The basic idea of block coding is to partition an image into a set of two-dimensional blocks or patterns of size  $n \times m$ . This concept of block coding can be used for the encoding of target pixels. Each pattern consists of  $nm$  bits and can be considered as a symbol of a set  $S$ . Then, there are  $2^{nm}$  possible symbols in the set  $S$ . The probability of occurrence of each symbol can be determined by raster scanning the non-white rectangles. Compression is achieved by entropy coding of the patterns. Larger values of  $n$  and  $m$  generally yield higher compression ratios. However, with increasing values

of  $n$  and  $m$ , the total number of possible patterns increase exponentially; this results in an exponential increase in complexity.

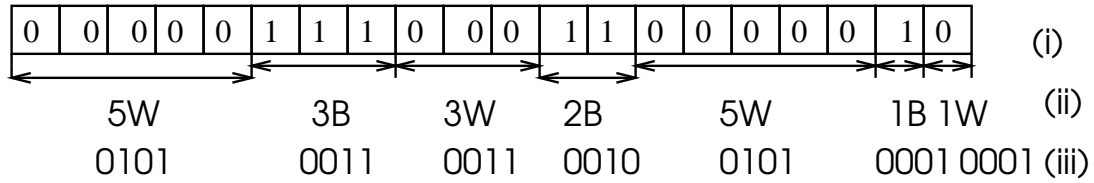
A special case of block coding is square block coding in which only square blocks are used, i.e.  $n = m$ . In this case the lengths and widths of the non-white rectangles are quantized such that they are multiples of  $n$ .

Figure 4.1 illustrates the idea of block coding. The image is partitioned by  $2 \times 2$  blocks (figure 4.1a) and there are 16 possible patterns. On raster scanning, each of the  $2 \times 2$  block is matched with one of 16 possible patterns and a codeword corresponding to the pattern matched is assigned to the block. The patterns and their corresponding codewords are stored in a look-up table. Figure 4.1b shows the pattern numbers matched to the blocks in figure 4.1a.

### 4.3 Run-length Coding

In the proposed BCITT technique, the locations of all the non-white rectangles are known to the decoder. Therefore, all the target pixels can be treated as elements of a long one-dimensional array. The elements of the one-dimensional array are left to right and top to bottom raster scanned pixels belonging to the non-white rectangles, starting from the upper left hand corner of the predicted and tiled image. These pixels can be encoded by run-length coding using a modified Huffman code[27] with the advantage that no End-of-Line (EOL) codeword is required. The end-of-line codeword is required in the known run-length coding techniques to indicate that the end of the row (or line) being coded has been reached, and that the following codeword starts a new row.

An example of run-length coding with fixed length codewords is shown in Figure 4.2. Since white (0s) and black (1s) runs alternate, the color of the run need not be



- (i) Data
- (ii) Run lengths
- (iii) Run-length coding, fixed length

Figure 4.2. Run-length coding

coded. If necessary, the first run is always a white run with length zero. If run-lengths are coded by fixed-length  $m$ -bit codewords, each representing a block of maximum run-length  $M - 1$ , ( $M = 2^m$ ) then  $M$  can be chosen to maximize compression.

A more-efficient technique is to use Huffman coding. A large code book is avoided by using truncated or modified Huffman codes[27]. The truncated Huffman code assigns separate codewords for white and black runs of lengths up to  $L_w$  (white run) and  $L_b$  (black run). Longer runs, which have lower probabilities, are assigned a fixed-length codeword, which consists of a prefix code plus an 11-bit binary code of the run-length.

The modified Huffman code, recommended by the CCITT as a one-dimensional standard code for Group 3 facsimile transmission, uses  $L_w = L_b = 63$ . Run-lengths smaller than 64 are Huffman coded to give the *terminator* code. The remaining runs are assigned two codewords, consisting of a *make-up* code and a terminator code[27].

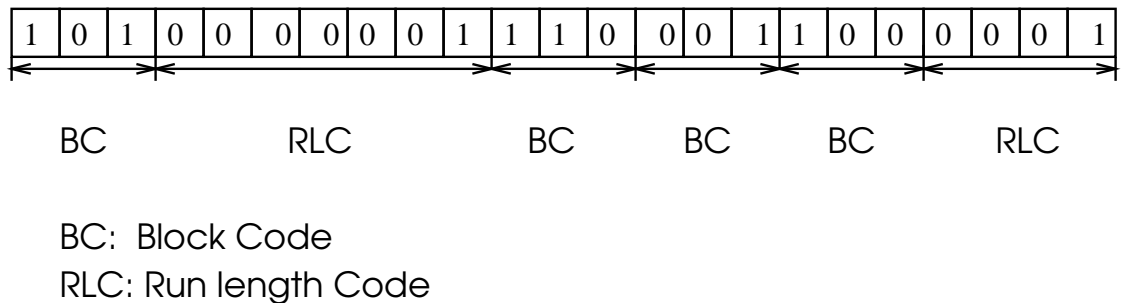


Figure 4.3. Run-length block coding

## 4.4 Run-length Block Coding

Run-length coding yields high compression ratios if the average length of runs is long. In other words, runs of short length decrease the compression ratio. On the other hand, block coding increases the compression ratio, if some of the block codes are more probable than others. Block coding and run-length coding can be combined to take advantage of the both techniques. We call the combination of run-length and block coding, Run-length Block Coding (RLBC).

In this technique, pixels are encoded by run-length coding as long as the length of the run is greater than or equal to a threshold  $N$ . If the length of a run is less than  $N$ , then that run is discarded and one of the  $2^N$  one-dimensional block patterns is used to encode  $N$  consecutive pixels. The pattern for encoding is selected by simply finding the one which matches with  $N$  consecutive pixels to be encoded. Figure 4.3 illustrates this technique, where  $N = 4$ .

## 4.5 Conditional Entropy Coding

Unlike entropy coding, conditional entropy coding assumes that the source has memory and it can be modeled as a markov process. If  $W$  is the size of a one-dimensional sliding window, then there are  $2^W$  possible window patterns. The probability of the  $W$ th pixel being 0, conditioned on the occurrence of one of  $2^W$  patterns is computed. Probabilities are compiled by sliding the window one pixel at a time over all the non-white region and then are used to entropy code the non-white regions.

# Chapter 5

## Implementation Issues

The algorithm for the proposed Unconstrained Block Tiling Technique (UBT) consists of the following six steps:

- A *predicted image*, based on the image to be encoded, is created using one of the prediction techniques described in Chapter2.
- All the black pixels of the residual image are then covered by the non-white rectangles (described in Chapter3).
- All the rows in the residual image which consist of only white pixels are removed and a White Row Removal Code (WRRC) is generated.
- The white space that remains after the covering of black pixels is tiled with white rectangles; first by block growing and then by sequential tiling (described in Chapter3).
- The horizontal and vertical dimensions of white and non-white rectangles are entropy coded.
- Finally, the target pixels (those covered by the non-white rectangles) are encoded using one of the techniques discussed in Chapter 4.

This chapter discusses various issues related to the implementation of some of the steps outlined above and addresses some of the complexity issues .

## 5.1 Prediction

Most of the compression in the UBT technique is achieved from the tiling of the white space in an image to be encoded. The image predicted from the original image generally contains a significantly higher number of white pixels than that in the original image. This, coupled with the ease of creating predicted images from the original image, makes prediction a reasonable first step.

Simple one-dimensional predictors, like vertical- and horizontal-XOR predictors (discussed in Chapter 2), help to remove either vertical or horizontal redundancies (e.g. vertical and horizontal lines) and are very easy to implement. Two-dimensional predictors, on the other hand, can remove both the horizontal and vertical redundancies, but are usually more complex to implement. For instance, it has been observed that a 12-element predictor can result in a prediction factor which is smaller than that obtained from a one-dimensional predictor by as much as a factor of two.

For an  $N$ -element predictor window, there are  $2^N$  different states. It has been observed that for increasing  $N$ , the prediction factor decreases linearly. However, on the other hand, due to the exponential increase in the number of states of predictor window, the complexity increases exponentially. Therefore, in practice, a suitable choice of  $N$  has to be made to achieve a trade-off between the prediction factor and the complexity of the predictor.

## 5.2 Covering of Black Pixels: Non-white Regions

The black pixels of a predicted image are covered by the non-white rectangles. The non-white rectangle dimensions correspond to the quantized dimensions defined by the UBT algorithm. These dimensions determine the number of non-white rectangles needed to cover all the black pixels, and hence the number of bits required to specify all the non-white rectangles. Expanding the set of quantized dimensions, although increasing the entropy of the dimensions, results in a smaller number of non-white rectangles. Therefore, this results in a smaller number of total bits required to specify all the non-white rectangles. The number of bits required is computed as given below:

$$\text{Number of bits} = (\text{entropy of the dimensions of the NW Rs}) * (\text{number of NW Rs})$$

However, increasing the number of defined dimensions increases the complexity. The pixels covered by the non-white rectangles (or the target pixels) are encoded using one of the techniques described in Chapter 4 and usually yield a very low compression ratio as compared to that yielded by the white rectangles. Higher compression ratios are usually achieved when an image is tiled with a few white rectangles possessing a larger average area than that achieved when the image is tiled with a large number of white rectangles possessing a smaller average area. If the dimensions of the non-white rectangles are small, the effect of the target pixels' compression ratio on the overall compression ratio may reduce due to a reduction in the number of target pixels. However, as a result, the number of white rectangles may increase and the compression ratio yielded by the white rectangles may decrease. Hence, the overall compression ratio may decrease. Similarly, if the dimensions of the non-white rectangles are allowed to be relatively large, the influence of the target pixels' compression ratio on the over-all compression ratio may increase due to the increase in the number of target pixels. This in turn may again reduce the over-all compression. Therefore,



<i>Image</i>	<i># of all white rows</i>		<i>gain</i>
	<i>before prediction</i>	<i>after prediction</i>	
Letter type	2957	3003	46
Roman type	2227	2553	326
Handwriting	1524	1569	45

Table 5.1. Gain in the number of all white rows following prediction

there is a trade-off between the defined dimensions of non-white rectangles and the number of white rectangles used.

### 5.3 Removal of White Rows

The predicted image is raster scanned to generate a White Row Removal Code (WRRC). Each bit of WRRC corresponds to a row in the predicted image. If a row being scanned has at least one black pixel, it is retained for further processing and a code 1 is assigned to WRRC corresponding to this row; otherwise, the row is removed and a code 0 is assigned. Thus, the number of one bit codes in the WRRC code for an image is equal to the number of rows in the predicted image.

The all white rows are removed only after the prediction and the covering of black pixels, rather than from the original image, because in certain types of images (e.g. business letters) following prediction a few more rows become all white rows. Table 5.1 shows the *gain* in the number of all white rows following prediction. A 12-element two-dimensional prediction window was used to compile the table.

### 5.4 Tiling of White Space

Since *block growing* with several *passes* can overcome the limitations of *sequential tiling*, we decided to tile the white space first with grown rectangles, as described in the Chapter 3. The number of *passes* and the *threshold area* for each pass are defined by the UBT algorithm. The efficiency of tiling generally improves as the

number of passes increases. The *efficiency* is defined in terms of the number of white rectangles needed to tile the entire white space. The smaller the number, the higher the efficiency. In other words, the smaller the number of rectangles needed to tile the entire white space, the more the average area covered by each of the white rectangles, and therefore, the smaller the number of bits required to encode their dimensions. Hence, the compression ratio will be higher. However, the passes can cause a delay. For instance, in block growing with 7 passes, some of the grown rectangles cover as many as 700,000 pixels each. On the average, block growing yields a compression ratio of 1500 within the grown regions, when the minimum threshold area allowed is 100.

There is a trade-off between the size of the code book for the quantized dimension and the number of rectangles needed to tile the white space. Increasing the codebook size generally increases the compression ratio by reducing the number of rectangles needed to tile the white space. It unfortunately may also exponentially increase the complexity. The selection of the elements for the set of defined quantized dimensions also determines the total number of rectangles which tile the white space. The elements are chosen such that they minimize the total number of white rectangles.

Once the major area of the white space is tiled by the block growing technique, it may be advantageous to fill the remaining white space slots by the sequential tiling technique, since the sequential tiling technique is faster and the remaining slots are usually relatively very small in size and number.

# Chapter 6

## Results and discussion

The proposed UBT technique was simulated on a SUN Microsystem SPARCstation 2 to demonstrate its applicability and to evaluate its performance. The program code was written in the *C* programming language. The major functions of the code are given in Appendix C. Three standard images (described in Table 1.1) were used as the test images.

In this chapter the results of these simulations are presented. A comparison of the results obtained from the UBT technique is done with some known techniques for the compression of bi-level images. The implications of the results in light of the issues outlined in Chapter 5 are discussed.

### 6.1 Prediction

#### **Prediction window size vs. prediction factor**

Figure 6.1 shows the variation of prediction factor with the number of elements in the prediction window for the test images. The prediction windows are shown in Figure 6.2. As can be seen from Figure 6.1, the prediction factor decreases with the increase in the number of elements in the prediction window. It should be noted that the positions of the prediction window elements relative to the pixel to be predicted (or in other words, the shape of the prediction window) have a considerable effect on the

Figure 6.1. Prediction window size vs. prediction factor

<b>X</b>	<b>X</b>	<b>X</b>
<b>X</b>	<b>X</b>	<b>?</b>

(i)

<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>
<b>X</b>	<b>X</b>	<b>?</b>		

(ii)

<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>
<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>
<b>X</b>	<b>X</b>	<b>?</b>		

(iii)

**X**: prediction window pixels  
**?** pixel to be predicted

Figure 6.2. Prediction windows

prediction factor.

## 6.2 NWR Compression Ratio

In this section, the effect of the change in a variable in each of the four techniques, implemented for the coding of NWRs on *NWR compression ratio*, is discussed.

### **Block coding: length of square block**

Figure 6.3 shows that the NWR compression ratio increases linearly with the block size. As an example, the variation of NWR compression ratio with the block size is plotted for the letter type image with only a 5-element prediction window.

### **Conditional entropy coding: size of sliding window**

As shown in the Figure 6.4 the NWR compression ratio increases with an increase in the number of elements in the sliding window. It should, however, be noted that the NWR compression ratio increases by only 1.96% when the number of elements in the sliding window is increased from 3 to 10. The letter type image with a 12-element prediction window was used.

### **Run-length coding: length of longest run**

In the run-length coding technique implemented, one code book for runs of  $N$  or fewer 0s was used. An extra code was used to specify runs of 0s longer than  $N + 1$ .

Figure 6.5 shows the increase in NWR compression ratio with increases in  $N$ . The NWR compression ratio increases rapidly until  $N$  is less than 400, thereafter, its growth saturates. The letter type image with a 5-element prediction window was used.

Figure 6.3. NWR compression ratio vs. length of the square block

Figure 6.4. NWR compression ratio vs. # of elements in the sliding window



Figure 6.5. NWR compression ratio vs. length of longest run

Figure 6.6. NWR compression ratio vs. block length

### **Run-length block coding: block length**

Figure 6.6 is a graph of the NWR compression ratio vs. the length of the block. It shows that the compression ratio remains almost constant with increases in the block length.

## **6.3 Contribution of different techniques**

In this section, the contributions of the three different parts of UBT towards the over-all compression ratio, saving of bits, and the total number of bits covered by them, are discussed. A 12-element prediction window (Figure 6.2c) and a sentence blocking technique were used for the results of this section.

### **Share in covering the image**

Figures 6.7-6.9 show the area covered by the three parts of UBT, namely white row removal code (WRRC), block tiling (BT), and coding of non-white regions (CNWR) for the three test images.

Most of the area in the letter and roman type images is covered by WRRC – 69% and 58.66%, respectively. In the letter type image, BT covers 21.76% of the area leaving only 9.22% of the area for CNWR. On the other hand, in the roman type image BT and CNWR cover nearly the same amount of area – 21.31% and 19.99%, respectively. However, in the handwriting type image, WRRC and CNWR cover about the same amount of area.

### **Share in saving bits**

Figures 6.10-6.12 show the contribution of WRRC, BT, and CNWR, in reducing the total number of bits needed to be transmitted. For the letter type image the number of coded bits needed to be transmitted is 1.275% of the total bits in the original

Figure 6.7. Share of WRRC, BT, and CNWR in covering the Letter type image

Figure 6.8. Share of WRRC, BT, and CNWR in covering the Roman type image

Figure 6.9. Share of WRRC, BT, and CNWR in covering the Handwriting type image

Figure 6.10. Share of WRRC, BT, and CNWR in saving bits: Letter type image

Figure 6.11. Share of WRRC, BT, and CNWR in saving bits: Roman type image



Figure 6.12. Share of WRRC, BT, and CNWR in saving bits: Handwriting type image

image. As shown in Figure 6.10, most of the compression (or the savings in the number of bits to be transmitted) came from WRRC and BT. In the roman type image the corresponding number of bits needed to be transmitted is 1.69%. WRRC saves 58.63% of the total number of bits in the original image, followed by BT and CNWR which save 21.26% and 18.42%, respectively. In the handwriting type image CNWR saves 32% of the total bits, more than that saved by BT: this occurs because CNWR covers more area than that covered by BT (Figure 6.9).

### **Share in the total number of transmitted bits**

Figures 6.13-6.15 show the share of WRRC, BT, CNWR, and NWRT (tiling of non-white regions) in the total number of bits transmitted. In all of these three images, almost all of the bits transmitted came from CNWR – 94.81%, 95.12%, and 97.9% for letter, roman and handwriting type image, respectively. It should be noted that NWRT takes less than 0.3% of the total number of bits to isolate all the black pixels within non-white rectangles.

On comparing Figures 6.7, 6.10 and 6.13 it can be seen that for the letter type image WRRC covers 69% of the area but uses only 2.59% of the total number of transmitted bits, i.e. it yields a compression ratio more than 2119. Similarly, BT achieves a compression ratio more than 754. Although WRRC and BT cover more than 90% of the area, they take up less than 5% of the total number of transmitted bits. On the other hand, CNWR covers only 9.22% of the area but contributes more than 94% of the total transmitted bits. Similar inferences can be drawn for the roman and handwriting type images by comparing the other pie-charts.

It can be concluded that the concept of tiling the white space and that of WRRC results in high compression ratios. The covering of black pixels by non-white rectangles requires a negligible number of bits and helps in the structured and efficient

Figure 6.13. Share of WRRC, BT, and CNWR in the total number of transmitted bits: Letter type image

Figure 6.14. Share of WRRC, BT, and CNWR in the total number of transmitted bits: Roman type image

Figure 6.15. Share of WRRC, BT, and CNWR in the total number of transmitted bits: Handwriting type image

tiling of the white space. This reduces the problem of bi-level image compression to that of coding only target pixels ( the number of which can be as small as 7% of the total bits in the original image) which comprise of all the black and some of the white pixels of the image.

## 6.4 A Comparative Study

In this section the results obtained from using different algorithms for the UBT are presented and compared with some of the known techniques. Figures 6.16-6.18 compare each of the test images with some of the known techniques.

The legend for these figures is as follows:

*a = Entropy coding of the image*

*b = Entropy coding of the horizontal-XORed image*

*c = Entropy coding of the residual image using a 2-D predictor*

*d = Sentence blocking and tiling, entropy coding the NWR pixels*

*e = Sentence blocking and tiling, RLBC of the NWR pixels*

*f = Word blocking and tiling, entropy coding of the NWR pixels*

*g = Word blocking and tiling, RLBC of the NWR pixels*

In *d, e, f, and g* a two-dimensional 12-element prediction window was used to create the predicted image. Figure 6.16 shows that the UBT technique increases the compression ratio by more than 56% of that achieved by the predictive coding; i.e. by more than 929% that of achieved by simple entropy coding only. Similarly, it can be seen from Figure 6.17 that the UBT technique increases the compression ratio for the roman type image by more than 42%. Here it needs to be emphasized that these

Figure 6.16. Compression ratio vs. techniques for compression: Letter type image

Figure 6.17. Compression ratio vs. techniques for compression: Roman type image



Figure 6.18. Compression ratio vs. techniques for compression: Handwriting type image

<i>Technique</i>			<i>Compression ratio</i>
<i>Step I</i>	<i>Step II</i>		
Entropy coding			7.798
Prediction	vertical-XOR	entropy coding	28.89
	horizontal-XOR	entropy coding	30.43
	5-elements	entropy coding	44.2
	7-elements	entropy coding	44.76
	12-elements	entropy coding	51.35
BCITT	sentence blocking	entropy coding	72.39
		conditional entropy coding	74.9
		runlength coding	78.8
		block runlength coding	79.1
	word blocking	entropy coding	74.72
		conditional entropy coding	76.6
		runlength coding	80.04
		block runlength coding	80.3

Table 6.1. Compression ratios achieved by various techniques for **letter type** image increases in the compression ratios are not the best that can be achieved using the UBT technique.

The compression ratios achieved by all the techniques simulated in this study are given in Tables 6.1-6.3. Step I in these tables refers to the prediction for the predictive compression technique; and to prediction, sentence/word blocking and tiling for the UBT technique. Step II refers to the coding of the entire image for the predictive compression technique, and to the coding of the target pixels for the UBT technique.

<i>Technique</i>			<i>Compression ratio</i>
<i>Step I</i>	<i>Step II</i>		
Entropy coding			6.053
Prediction	vertical-XOR	entropy coding	23.99
	horizontal-XOR	entropy coding	24.88
	5-elements	entropy coding	37.69
	7-elements	entropy coding	38.33
	12-elements	entropy coding	43.31
BCITT	sentence blocking	entropy coding	53.56
		conditional entropy coding	57.15
		runlength coding	60.26
		block runlength coding	60.20
	word blocking	entropy coding	55.89
		conditional entropy coding	58.95
		runlength coding	61.20
		block runlength coding	61.13

Table 6.2. Compression ratios achieved by various techniques for **roman type** image

<i>Technique</i>			<i>Compression ratio</i>
<i>Step I</i>	<i>Step II</i>		
Entropy coding			4.87
Prediction	vertical-XOR	entropy coding	13.34
	horizontal-XOR	entropy coding	15.38
	5-elements	entropy coding	19.47
	7-elements	entropy coding	19.85
	12-elements	entropy coding	23.12
BCITT	sentence blocking	entropy coding	27.13
		conditional entropy coding	28.32
		runlength coding	29.44
		block runlength coding	29.46
	word blocking	entropy coding	27.23
		conditional entropy coding	28.38
		runlength coding	29.51
		block runlength coding	29.54

Table 6.3. Compression ratios achieved by various techniques for **handwriting type** image

# Chapter 7

## Conclusion and Future Directions

In this thesis a new simple-to-implement technique for the compression of high-resolution bi-level images for facsimile transmission and storage was presented. This chapter summarizes the work and outlines future directions for research.

The proposed UBT technique divides the problem of coding for compression into two parts: tiling of the image with white and non-white rectangles, and coding of pixels covered by non-white rectangles. The idea of partitioning the image into rectangles of pre-defined dimensions extends the notion of run-length coding to two-dimensions.

In the first part of this thesis the notion of prediction was presented. It was shown that increasing the size of a prediction window generally decreases the prediction factor and results in higher compression ratios.

The second part introduced a new concept of partitioning an image into white and non-white rectangles. Two new algorithms for tiling the white space, namely sequential tiling and block growing were presented. In block growing, an omni-dimensional growing method was found to be an efficient way of tiling the white space. It was shown that block tiling within the tiled regions yields very high compression ratios.

The third part of the thesis was concerned with coding of the pixels covered by the non-white rectangles. A number of different simple coding techniques for coding of the target pixels were discussed. A new technique which combines the concepts

of block and run-length coding was also introduced. It was found that of all the simulated coding techniques, run-length coding and run-length block coding gave the best compression ratios.

It can be concluded that the concepts of tiling of the white space with rectangles and that of WRRC yield very high compression ratios. Although the covering of black pixels with non-white rectangles takes up a negligible part of the total number of transmitted bits, it results in a higher compression, and reduces the problem of compression to that of coding of pixels in well defined regions which cover a relatively very small area of a typical facsimile document.

Although the primary aim of this thesis was compression of bi-level images, it should be emphasized that the proposed technique can effectively be used for multi-level image by simply encoding each bit plane independently as though it were itself a bi-level image.

A number of different directions can be suggested for future work in all the three parts of the UBT technique. In prediction, instead of using a fixed prediction window and trying to minimize the prediction factor by increasing the number of prediction window elements, an adaptive prediction window can be used. The adaptive window should be able to change the shape and size of the prediction window according to the local characteristics of the image being predicted.

In this thesis the set of defined quantized dimensions for the white rectangles was selected by repeated trials. A mathematical base for the selection of the set can be developed or an algorithm similar to the Generalized Lloyd Algorithm (GLA)[30] might be developed. The issue of codebook size should be settled by more simulations.

The main focus of any future study of UBT technique should be the development of an efficient technique for the coding of target pixels.

# Appendix A

## Images

In this appendix, decimated images of the document generating images, and a section of residual images obtained from prediction are shown. Decimated images were created only for display purposes.

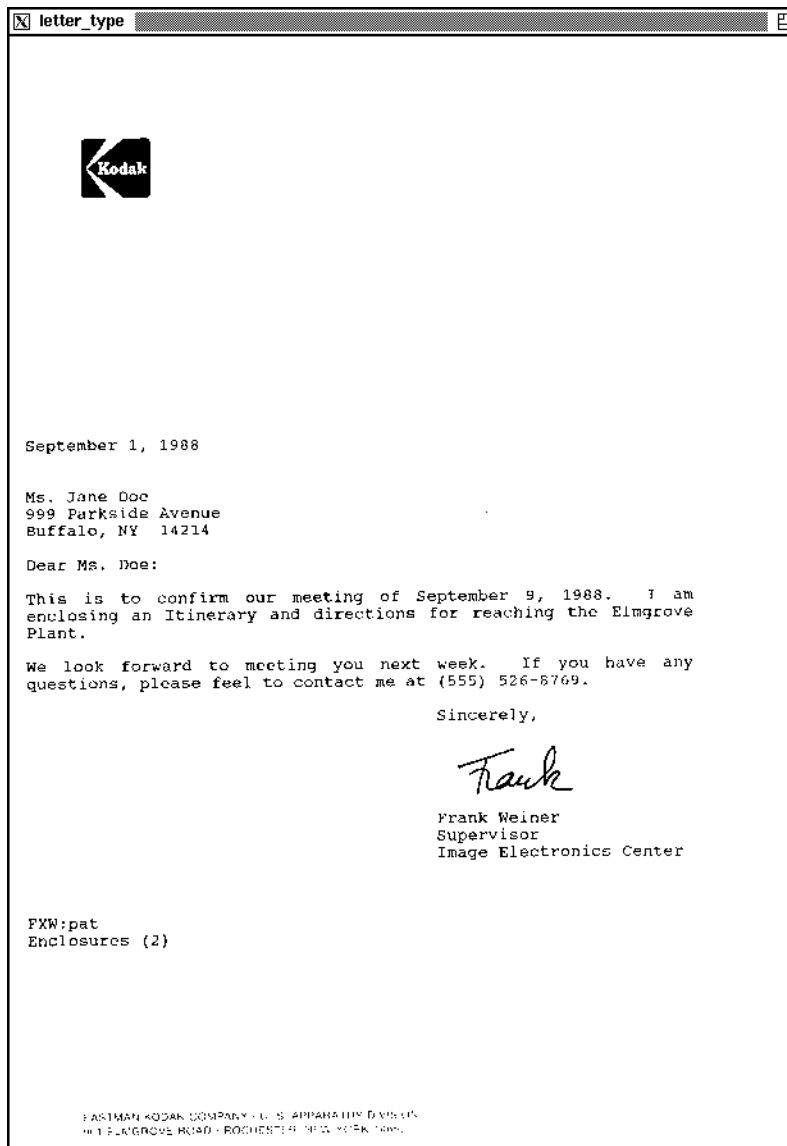



Figure .1. Decimated image of the document generating *Letter Type* image

roman\_type

## Information Theory Concepts



A rectangular box labeled "Source S" has an arrow pointing to the right towards the text "s<sub>i</sub>, s<sub>j</sub>, ...".

Source alphabet of size  $n$  with probabilities  $p(s_1), p(s_2), \dots, p(s_n)$ .

The information (in bits) provided by the occurrence of source symbol  $s_i$  is given by

$$I(s_i) = \log_2 \frac{1}{p(s_i)} \text{ bits}$$

The average amount of information obtained per symbol from the source is called the **entropy**  $H(S)$  of the source:

$$H(S) = \sum_{i=1}^n p(s_i) \log_2 \frac{1}{p(s_i)}$$

**example:**

$$S = \{A, B, C, D\}$$

$$P(A) = \frac{1}{2}, P(B) = \frac{1}{4}, P(C) = \frac{1}{8}, P(D) = \frac{1}{8}$$

$$H(S) = 1\frac{3}{4} \text{ bits}$$

5

Figure .2. Decimated image of the document generating *Roman Type* image



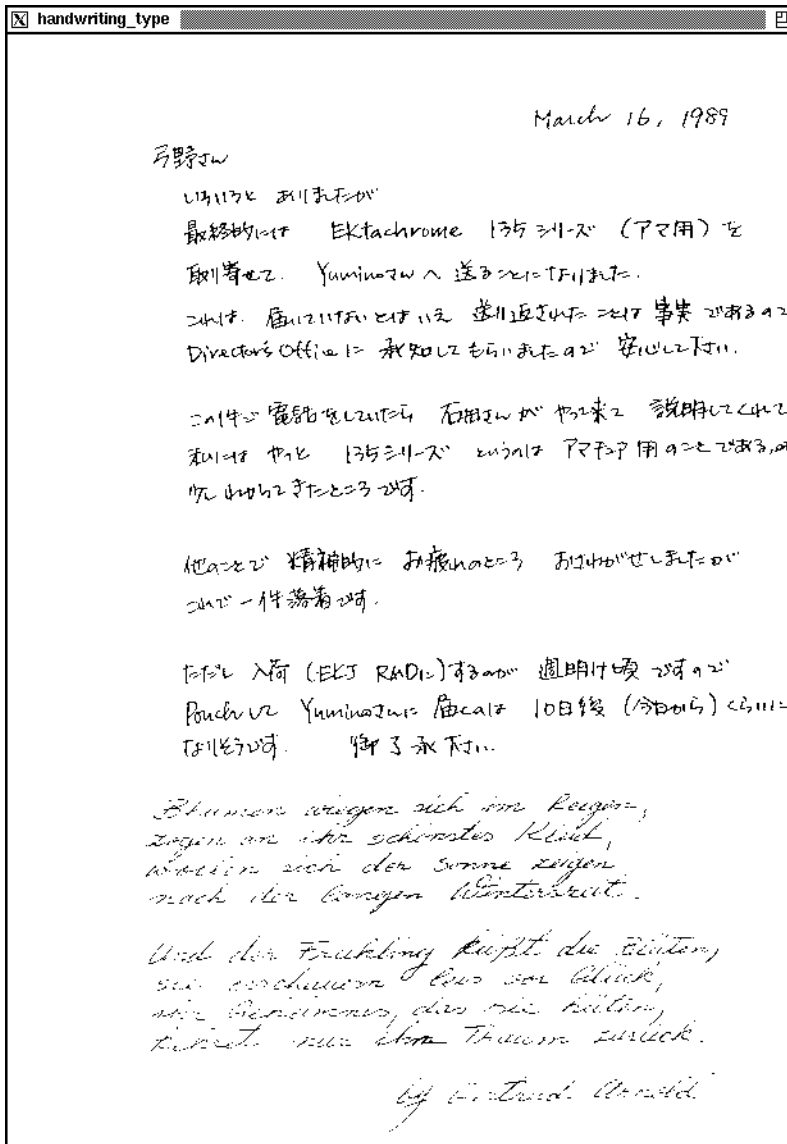


Figure .3. Decimated image of the document generating *Handwriting Type* image

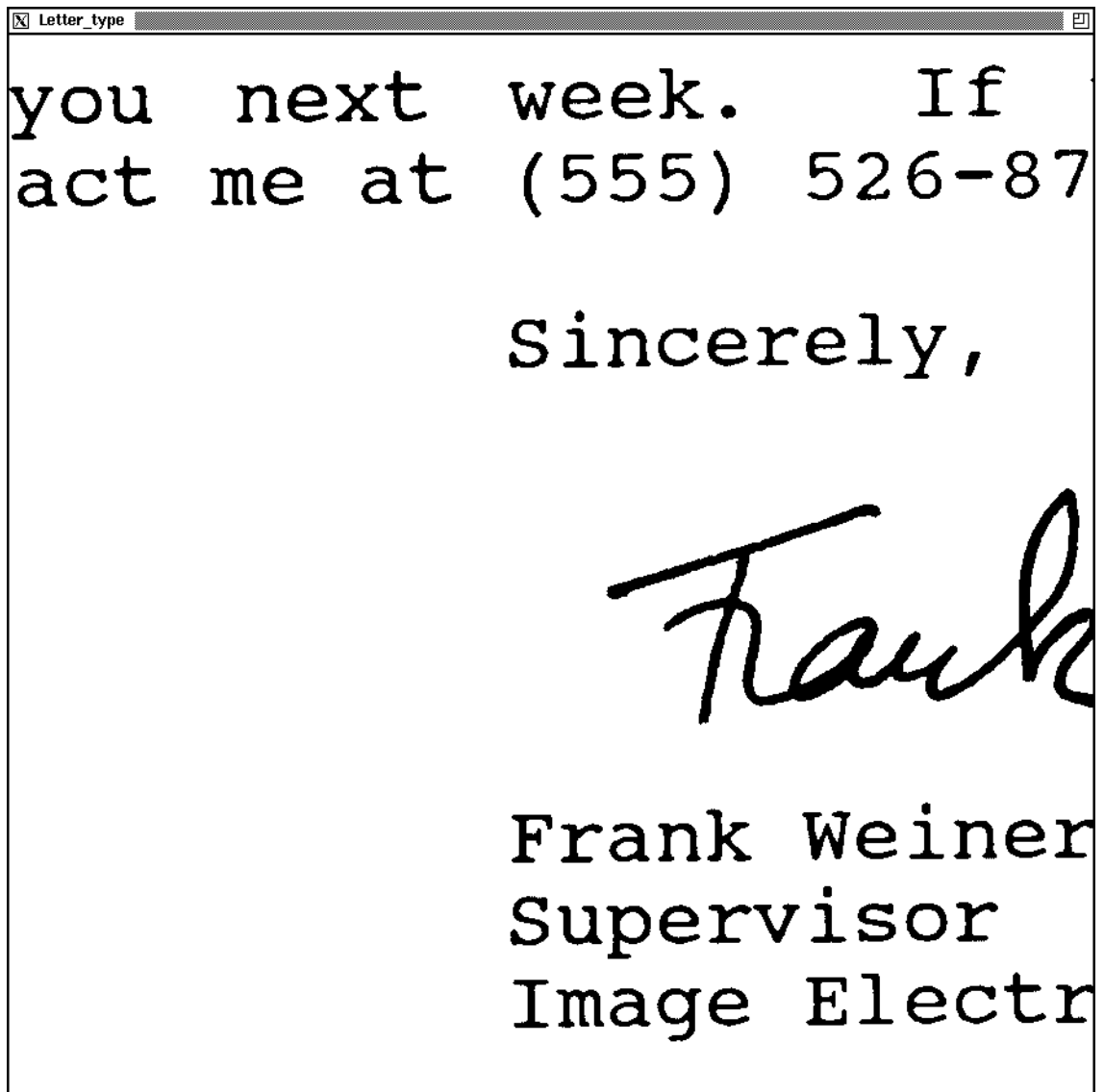


Figure .4. A section of Letter type image

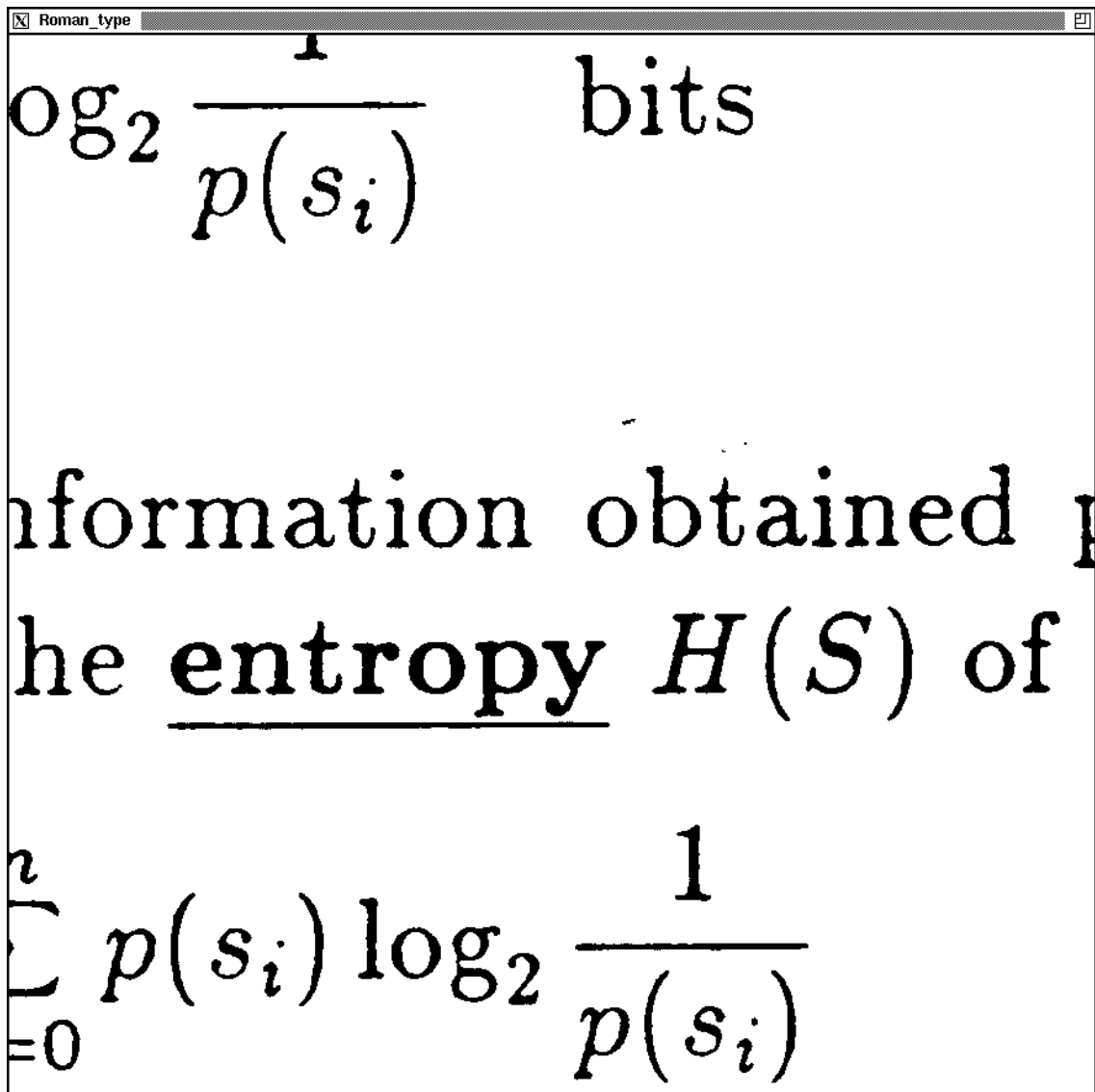


Figure .5. A section of Roman type image

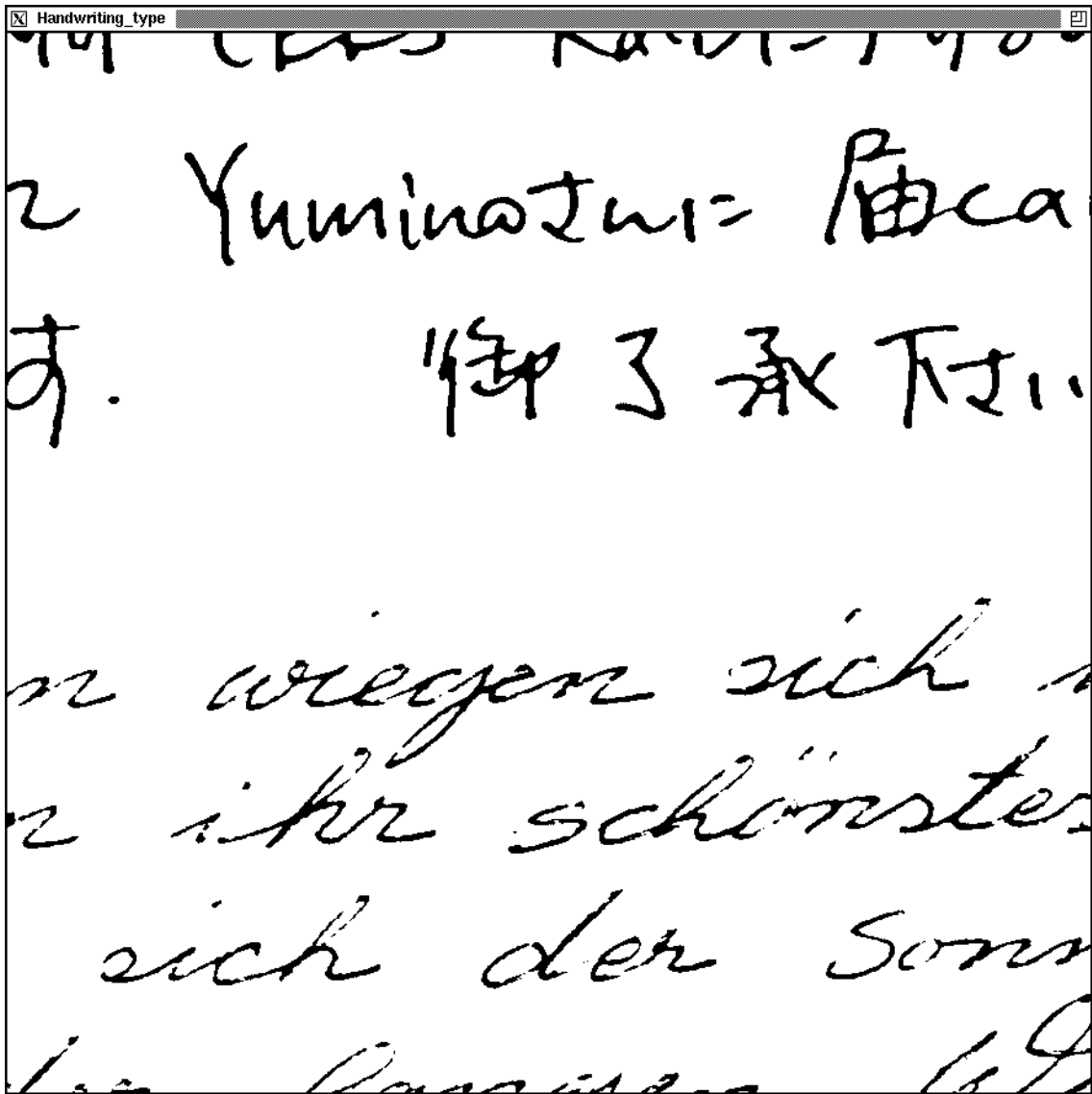


Figure .6. A section of Handwriting type image

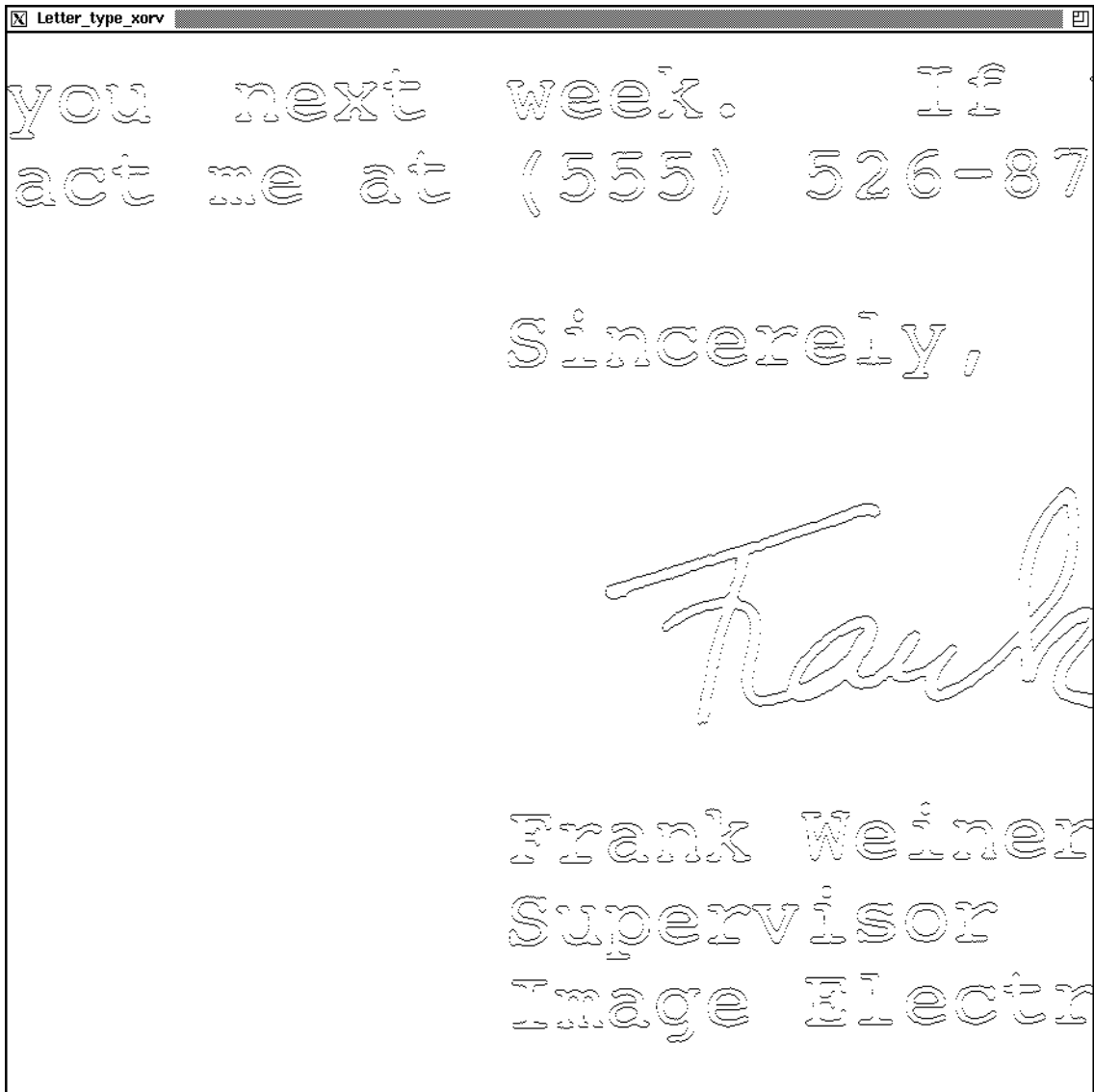


Figure .7. Residual Letter type image after Vertical-XOR

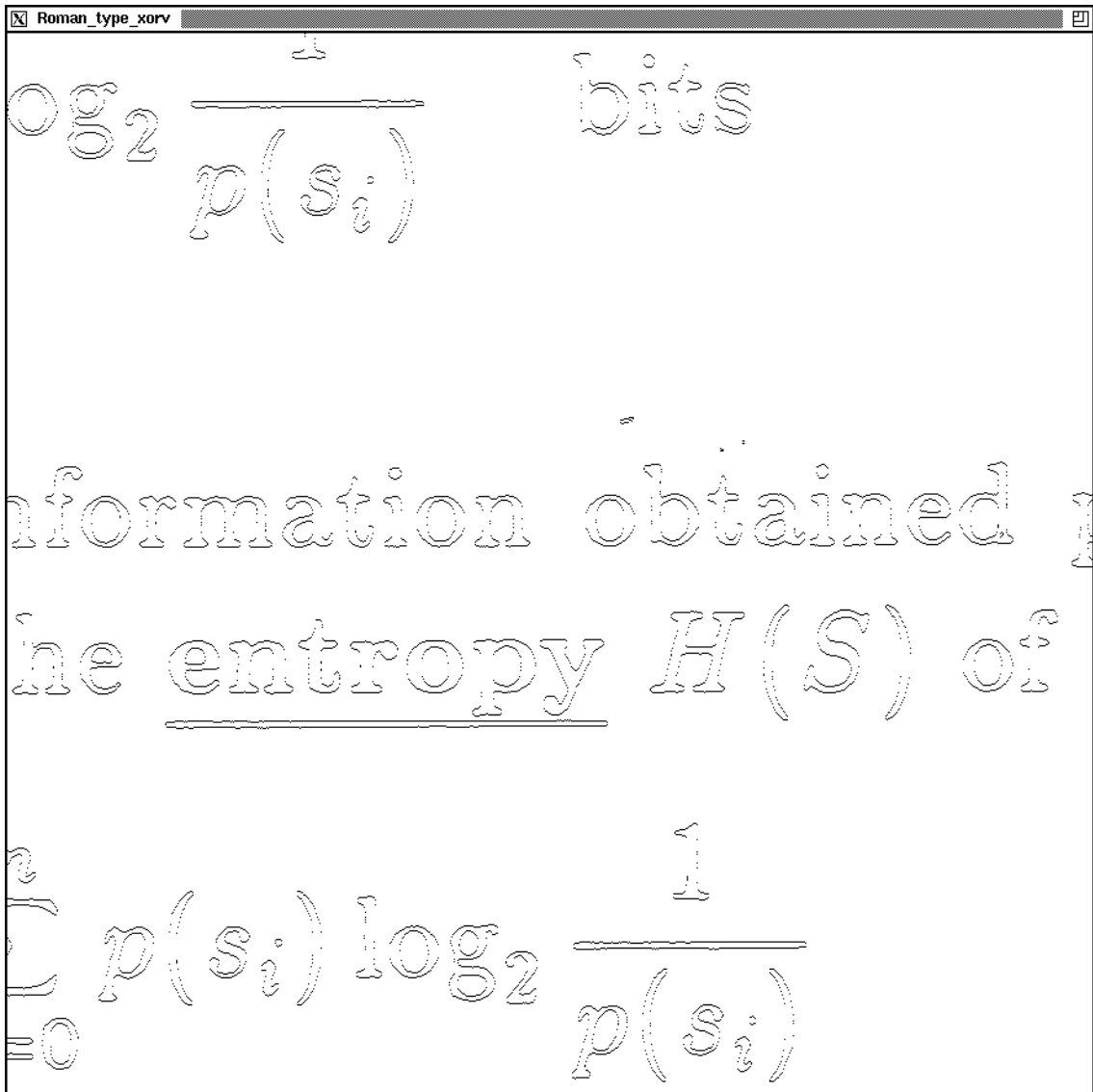


Figure .8. Residual Roman type image after Vertical-XOR

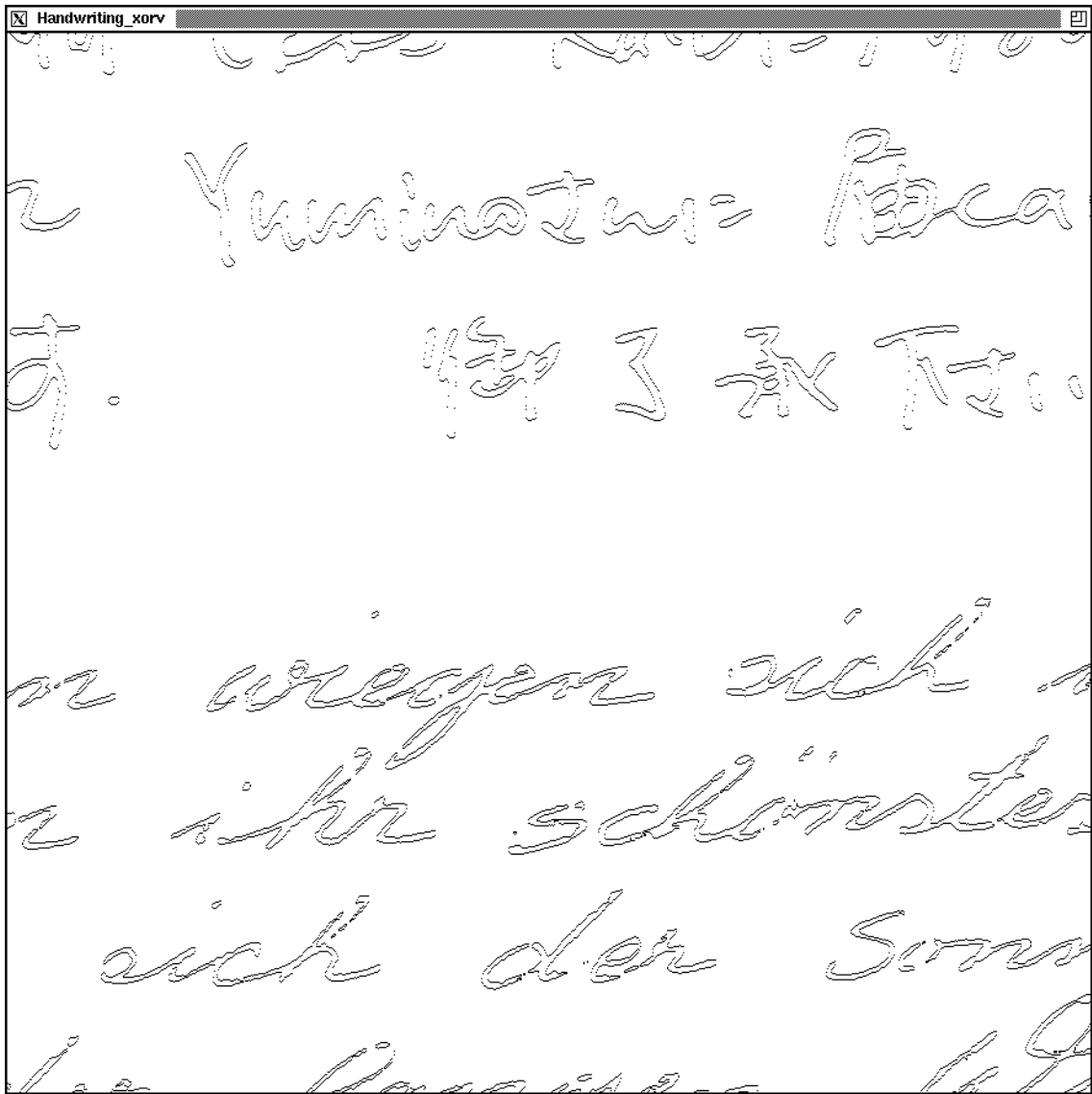


Figure .9. Residual Handwriting type image after Vertical-XOR

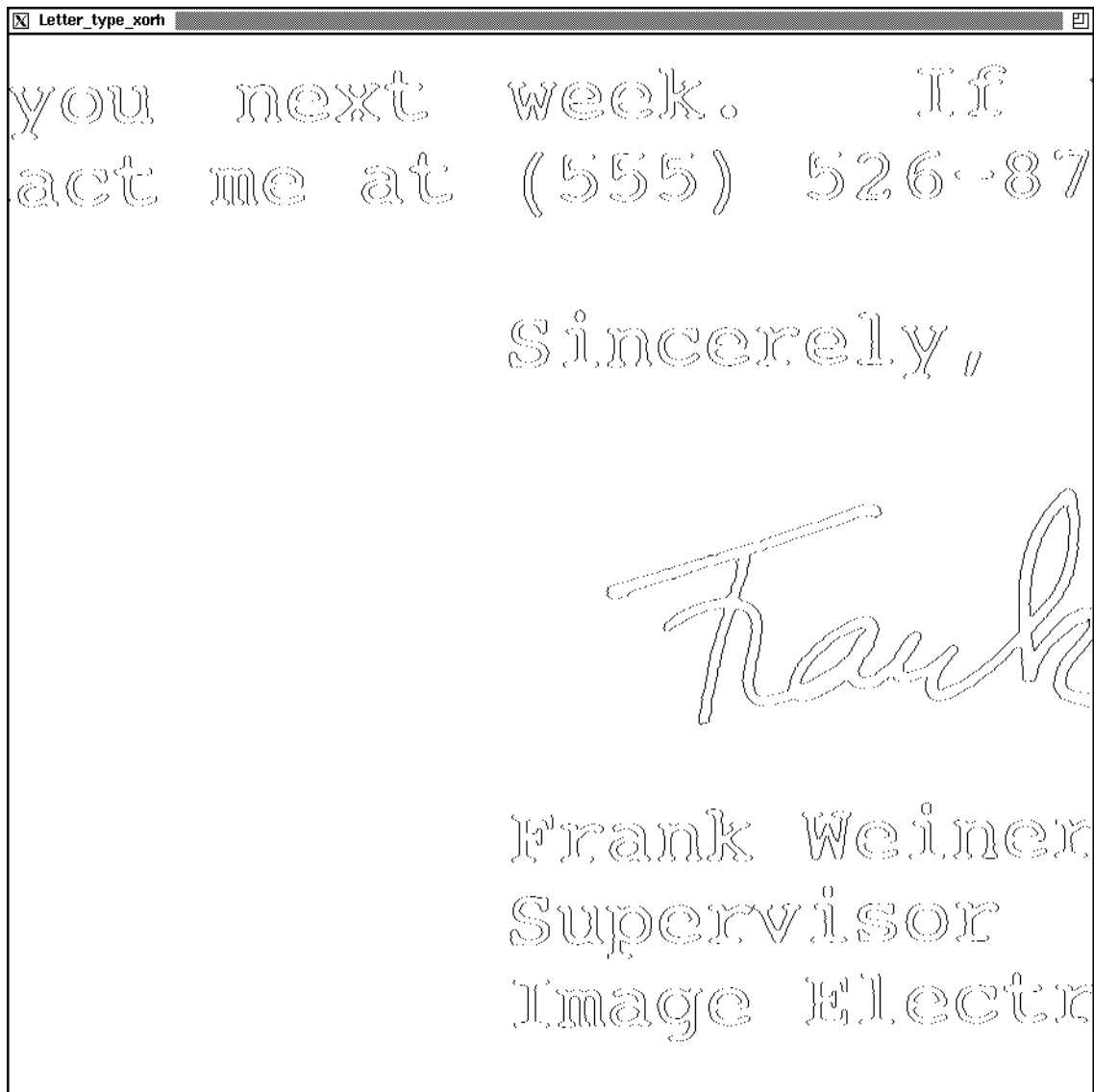


Figure .10. Residual Letter type image after Horizontal-XOR



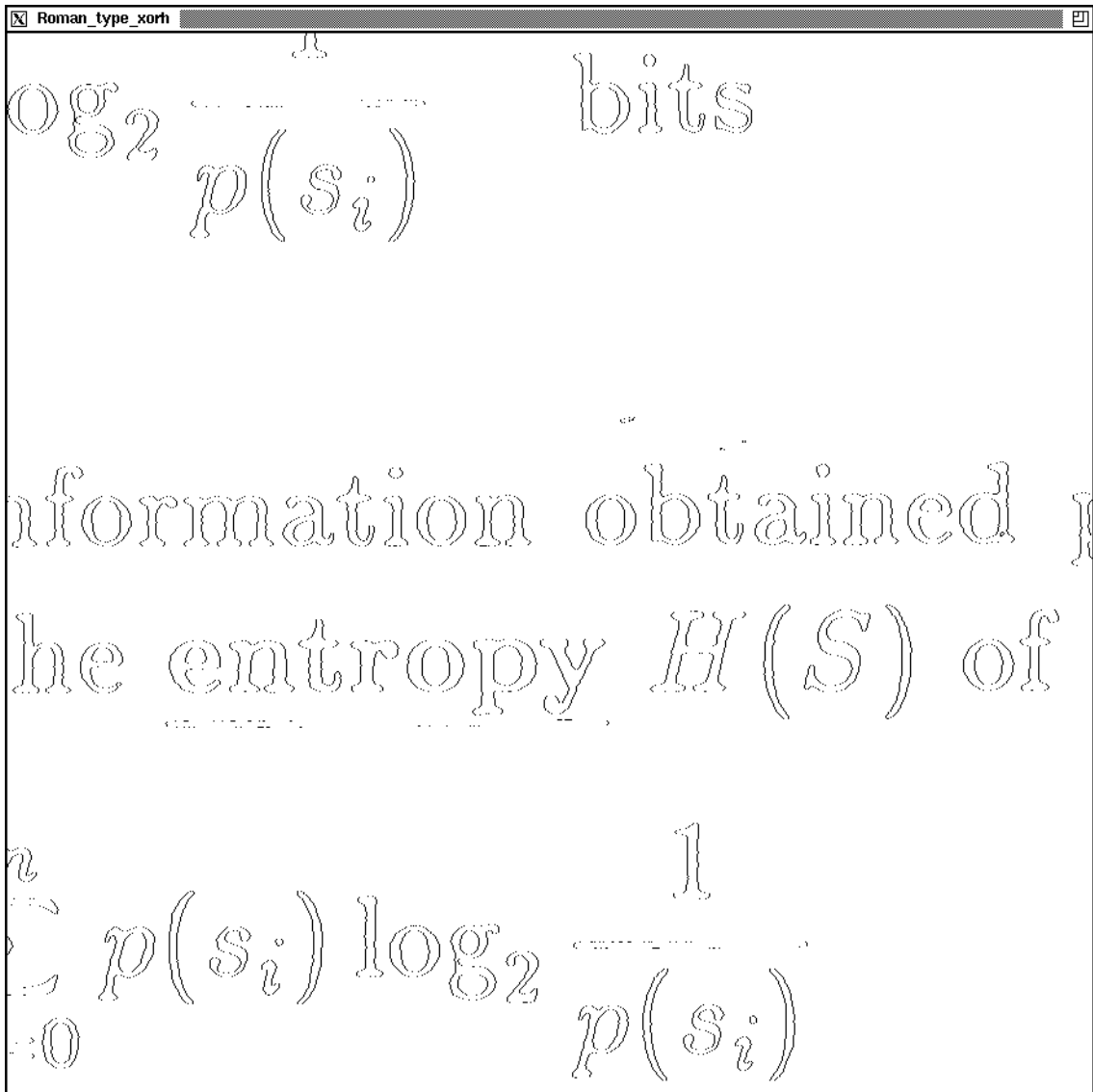


Figure .11. Residual Roman type image after Horizontal-XOR

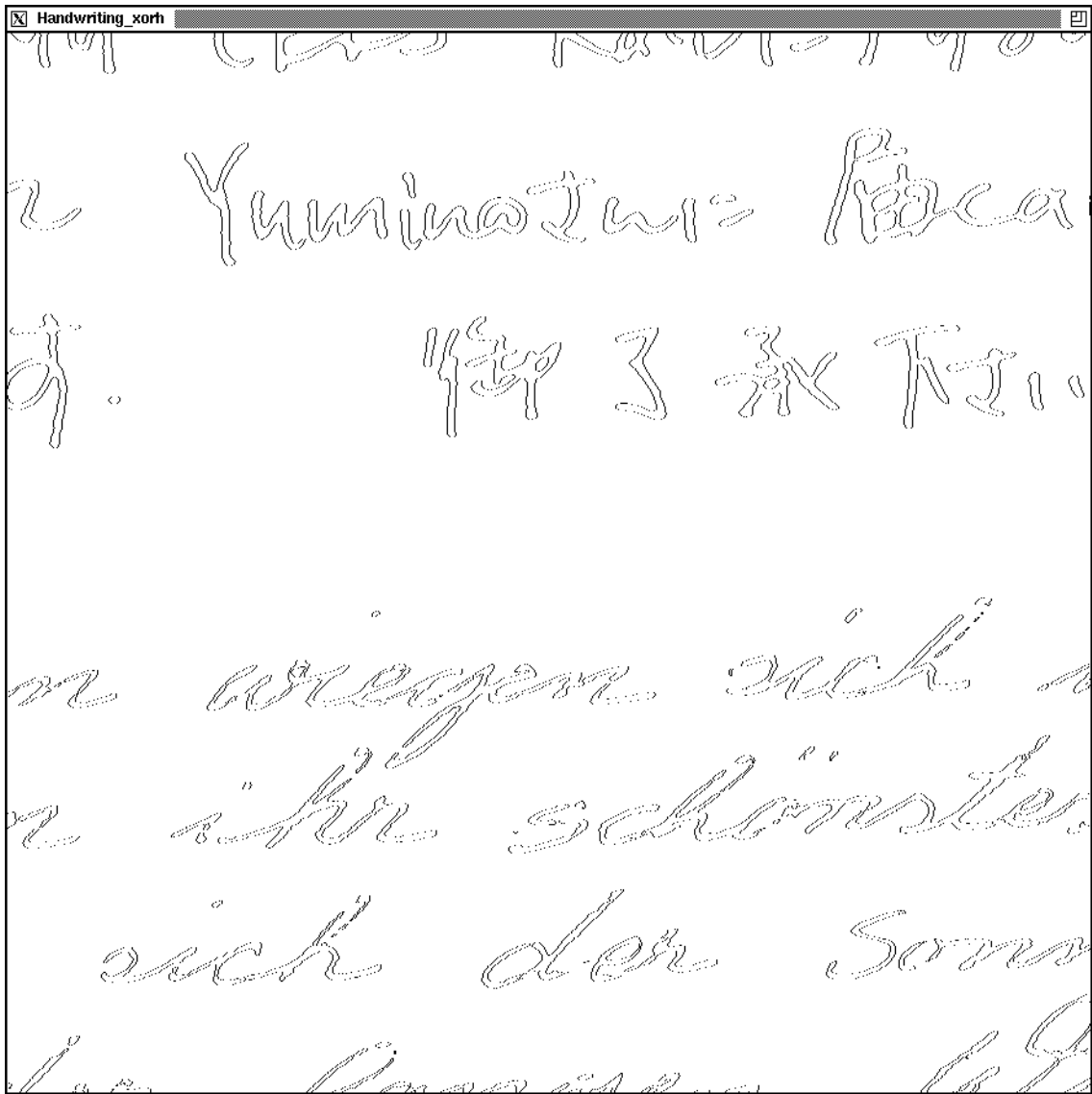


Figure .12. Residual Handwriting image after Horizontal-XOR

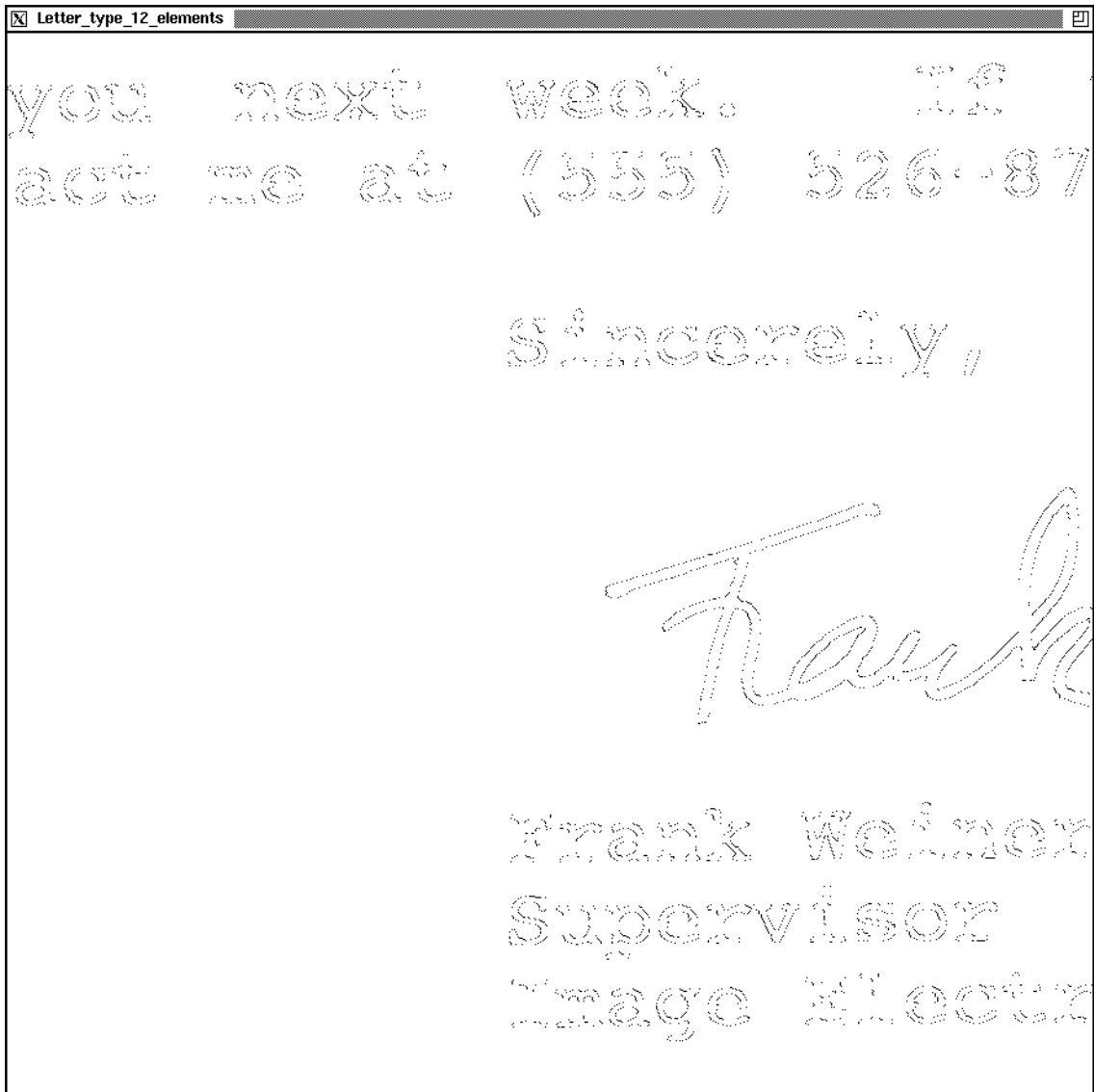


Figure .13. Residual Letter type image after 12 element 2-D prediction

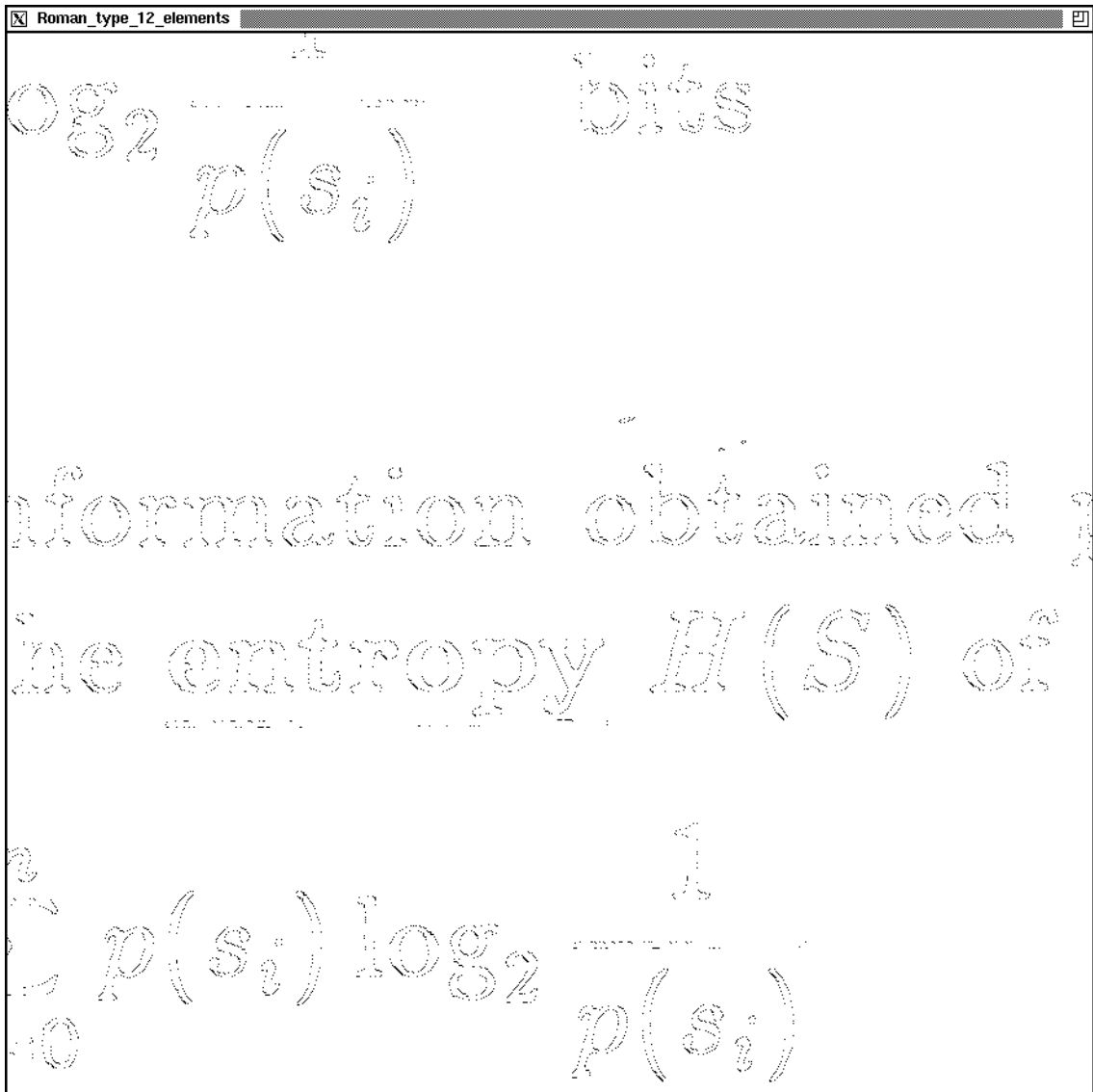


Figure .14. Residual Roman type image after 12 element 2-D prediction

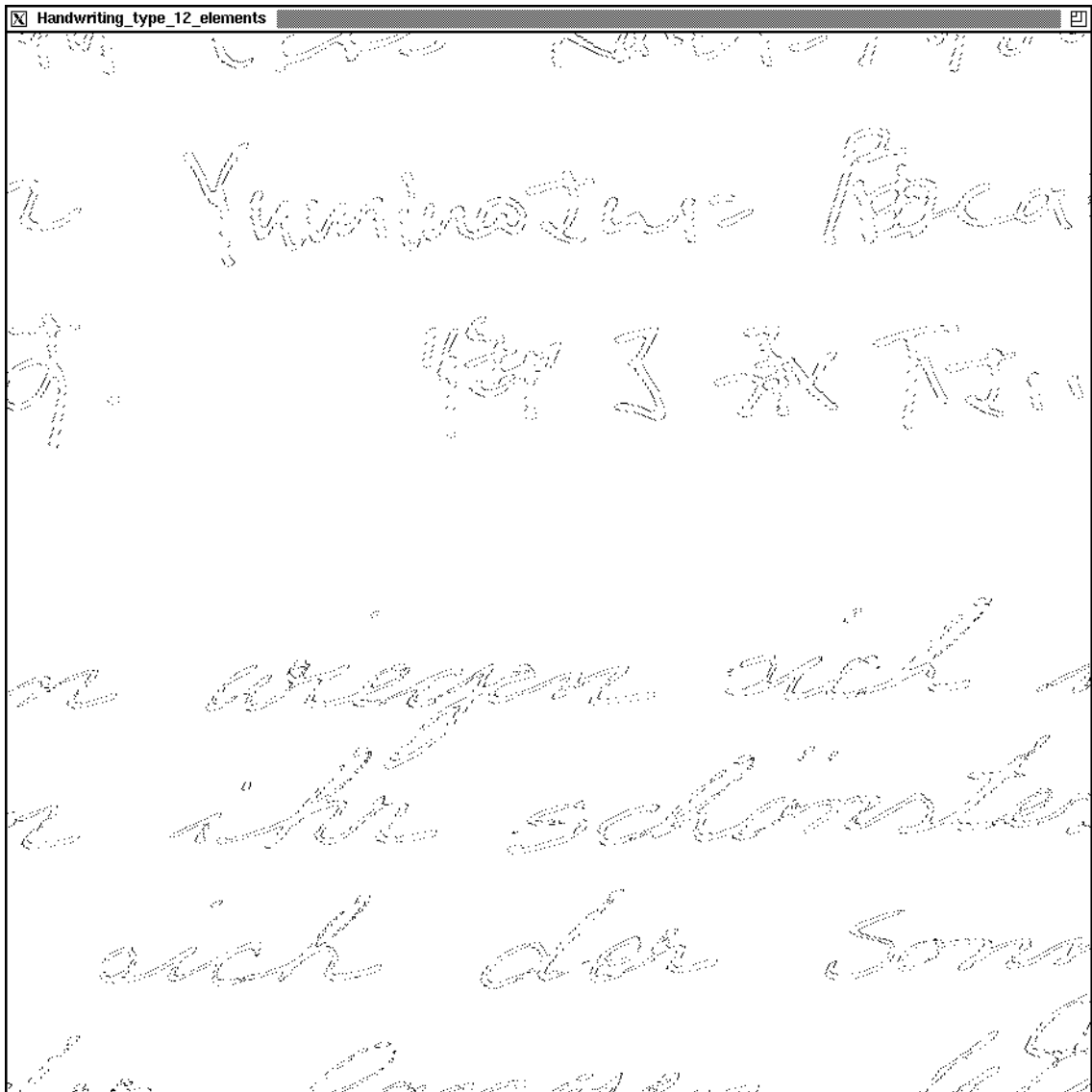


Figure .15. Residual Handwriting type image after 12 element 2-D prediction

# Appendix B

## Finding Maximum Area Rectangles

In this appendix, the algorithm for finding rectangles of maximum area using Sequential Tiling technique is described.

There are two constraints in finding a rectangle of the maximum area: firstly, the origin must be the upper left hand corner of the rectangle; secondly, the rectangle must cover white pixels only. Working under these constraints, all the pixels starting from the origin are raster scanned and the dimensions of the largest area rectangle, defined by a variable `largest_area`, are continuously updated until the pixel  $b[k, j]$  is reached. At every stage of scanning, only two rectangles need to be compared to find a rectangle with the largest area: one with the present pixel as its upper right hand corner, and the other with the pixel to the immediate left of the present pixel as its upper right hand corner. For example, in Figure 3.1 the two contending rectangles are defined by the points  $a[i, j]$ ,  $h[u, j]$ ,  $g[u, l]$  and  $c[i, l]$ , and  $a[i, j]$ ,  $b[k, j]$ ,  $e[k, r]$  and  $f[i, r]$ . The algorithm for finding maximum area rectangles is as follows:

initialize:

*previous\_max\_row = bottom boundary of the image (number of rows)*

*column = current\_column*

*row = current\_row*

```

    last_column = k

while ((column < last_column)and(row < previous_max_row)){
    if (hit by either a black pixel or a boundary pixel){
        area_present_pixel = (column - current_column) * (row - current_row);
        area_prev_pixel = (column - current_column - 1) * (previous_max_row);
        largest_area = larger(area_present_pixel, area_prev_pixel);
        previous_max_row = row;
    }
}

```

The column and row corresponding to the largest\_area found from the above code determine the lower right hand corner, as well as the *scan length* and *scan width* of the rectangle. These scan length and scan width are then truncated to the nearest of the defined quantized lengths and widths. Similarly, the sizes and locations of all the other rectangles that tile the image are determined.

# Bibliography

- [1] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520-540, June 1987.
- [2] V.R. Algazi, P.L. Kelly, and R.R. Estes, "Compression of binary facsimile images by preprocessing and color shrinking," *IEEE Trans. Commun.*, vol. COM-38, no. 9, pp. 1592-1598, Sept. 1990.
- [3] Y.Yasuda, "Overview of digital facsimile coding techniques in Japan," *Proc. IEEE*, vol.68, no. 7, pp. 830-845, July 1980.
- [4] H.G. Musmann and D. Preuss, "Comparison of redundancy reducing codes for facsimile transmission of documents," *IEEE Trans. Commun.*, vol. COM-25, no. 11, pp. 1425-1433, Nov. 1977.
- [5] D.C.V. Voorhis, "An extended run-length encoder and decoder for compression of black/white images," *IEEE Trans. Inform. Theory*, vol. IT-22, no. 2, pp. 190-199, Mar. 1976.
- [6] F.W. Mounts, E.G. Bowen, and A.N. Netravali, "An ordering scheme for facsimile coding," *Bell Syst. Tech. J.*, vol. 58, no. 9, pp. 2113-2128, Nov. 1979.



- [7] H. Meyr, H.B. Rodolsky, and T.S. Huang, "Optimum run-length codes," *IEEE Trans. Commun.*, vol. COM-22, no. 6, pp. 826-835, June 1974.
- [8] M. Takagi and T. Tsuda, "Bandwidth compression for facsimile using two-dimensional prediction," *Syst., Comput., Contr.*, vol. 4, no. 2, pp. 16-23, 1973.
- [9] "Proposal for draft recommendation of two-dimensional coding scheme," CCITT SG XIV Contribution, no. 42, Nov. 1978.
- [10] D.A. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, pp. 1098-1101, Sept. 1952.
- [11] H. Wyle, T. Erb, and R. Barow, "Reduced time facsimile transmission by digital coding," *IRE Trans. on Commun. Syst.*, vol. CS-9, no. 3, pp. 215-222, Sept. 1961.
- [12] Y. Wakahara, Y. Yamazaki, H. Teramura, and Y. Nakagome, "Data compression factors of relative address coding scheme for facsimile signals," *J. IIEE Jap.*, vol. 5, no. 3, pp. 92-100, Oct. 1976.
- [13] T. Yamada, "Edge-difference coding – A new, efficient redundancy reduction technique for facsimile signals," *IEEE Trans. Commun.*, vol. COM-27, no. 8, pp. 1210-1217, Aug. 1979.
- [14] H. Morita and S. Arimoto, "SECT – A coding technique for black/white graphics," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 4, pp. 559-570, July 1983.

- [15] M. Kunt, "Comparaison de techniques d'encodage pour la reduction de redondance d'images facsimile a deux niveaux," *Thesis*, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, no. 183, 1974.
- [16] O. Johnsen, "Etude de strategies adaptatives pour la transmission d'images facsimile a deux niveaux," *AGEN Mitteil.*, no. 20, pp.41-53, June 1976.
- [17] Y. Cohen, M.S. Landy, and M. Pavel, "Hierarchical coding of binary images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-7, no. 3, pp. 284-298, May 1985.
- [18] K. Knowlton, "Progressive transmission of grey scale and B/W pictures by simple, efficient and lossless encoding schemes," *Proc. IEEE* vol. 68, pp.885-896, 1980.
- [19] C.E Shannon, "A mathematical theory of communication," *Bell Sys. Tech. J.*, vol. 27, pp. 379-423, July 1948.
- [20] N. Abramson, "Information theory and coding," *McGraw- Hill Book Co., Inc.*, New York, 1963.
- [21] A. Gersho and R. M. Gray, "Vector quantization and signal compression," *Kluwer Academic Press*, Massachusetts, 1990.
- [22] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop.*, vol. 23, no. 2, pp. 149-162, Mar. 1979.
- [23] Y. Yamazaki, "Standardization activities in image communication for telematic services," *Signal Proc.: Image Commun.*, vol. 1, no. 1, pp. 55-73, June 1989.

- [24] D. Bodson, S. Urban, A.R. Deutermann, and C.E. Clarke, "Measurement of data compression in advanced group 4 facsimile systems," *IEEE Proc.*, vol. 73, no. 4, April 1985.
- [25] "ISO/IEC international standard XXXXX coded representation of picture and audio information – progressive bi-level image compression standard," *Early draft*, WG9-S1R2, Dec. 1990.
- [26] "Description of JBIG evaluation images for July meeting," *JBIG Document 105 Rev. 2*, Stockholm, June 6, 1989.
- [27] A.K. Jain, "Fundamentals of digital image processing," *Prentice Hall*, New Jersey, 1989.
- [28] A. Lempel and J. Ziv, "Compression of two-dimensional data," *IEEE Trans. Info. Theory*, vol. IT-32, no. 1, pp. 2-8, Jan. 1986.
- [29] M. Gardner, "Mathematical games," *Sci. Amer.*, pp. 124-133, Dec. 1976.
- [30] S.P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Info. Theory*, vol. IT-28, pp. 129-137, Mar 1982.