Image halftoning and inverse reconstruction problems with considerations to image watermarking

Thesis by

Murat Meşe

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2001

(Submitted May 3, 2001)

Acknowledgments

First, I would like to thank to my advisor Professor P.P. Vaidyanathan for his encouragement, teaching, patience, time and friendship. P.P., as we all call him, is the best advisor anyone can ever find. I have greatly benefited from his academic excellence, his style of thinking. Thank you P.P., for everything you taught me and everything you gave me.

I would also like to thank the members of my candidacy exam and my PhD defense committee members for their interest and valuable comments on my research: Prof. R.J. McEliece, Dr. Dariush Divsalar, Prof. J. Franklin, Prof. Pietro Perona, Dr. Igor Djokovic. I would like to acknowledge the generous support from the National Science Foundation and the Office of Naval Research which were crucial for my work at Caltech.

The people in Caltech Digital Processing Group made my stay at Caltech enjoyable and entertaining. Sony Akkarakaran, Dr. Ahmet Kirac, Dr. Jamal Tuqan, Dr. Yuan Pei Lin, Dr. See May Phoong, and Bojan Vrcelj, thanks for everything. I especially want to thank Sony, my office mate for five years, for his friendship and constant feedback and discussions. I hope I have still some credit to tease him. I would also like to thank to my friends at Caltech: Dr. Ayhan İrfanoğlu, Dr. Zehra Çataltepe, Dr. Adam Rasheed, Dr. Doruk Engin, Dr. Slim Alouini. Finally, I would like to thank my family for their love, support and for providing me the opportunity to pursue my goals: My parents, my brother Mithat, and my sisters Melek and Meral, thanks a lot.

Abstract

In this work, we first discuss the image halftoning problem. Halftoning is the rendition of continuous-tone pictures on displays that are capable of producing only two levels. There are several well-known algorithms for halftoning. The dot diffusion method for digital halftoning has the advantage of pixel-level parallelism unlike the error diffusion method, which is a popular halftoning method. The image quality offered by error diffusion is still regarded as superior to most of the other known methods. We show how the image quality obtained using the dot diffusion method can be improved by optimization of the so-called class matrix. By taking the human visual characteristics into account we show that such optimization consistently results in images comparable to error diffusion, without sacrificing the pixel-level parallelism. The dot diffusion algorithm will be discussed and by modifying the algorithm, embedded multiresolution property will be added. Later, we introduce LUT (Look Up Table) based halftoning and tree-structured LUT (TLUT) halftoning. We demonstrate how error diffusion characteristics can be achieved with this method. Afterwards, our algorithm will be trained on halftones obtained by Direct Binary Search (DBS) which is an algorithm with high computational complexity. The complexity of TLUT halftoning is higher than that of error diffusion but much lower than that of the DBS algorithm. Thus, halftone image quality between that of error diffusion and DBS will be achieved depending on the size of tree structure in TLUT algorithm.

We also discuss the inverse halftoning problem. Inverse halftoning is the reconstruction of a continuous tone image from its halftoned version. We propose two methods for inverse halftoning of dot diffused images. The first one uses Projection Onto Convex Sets (POCS) and the second one uses wavelets. We then propose a novel and fast method for inverse halftoning called the **L**ook **U**p **T**able (LUT) Method. The LUT for inverse halftoning is obtained from the histogram gathered from a few sample halftone images and the corresponding original images. For each pixel, the algorithm looks at the pixel's neighborhood (template) and depending upon the distribution of pixels in the template, it assigns a contone value from a precomputed LUT. The method is extremely fast (no filtering is required) and the image quality achieved is comparable to the best methods known for

inverse halftoning. The LUT inverse halftoning method does not depend on the specific properties of the halftoning method, and can be applied to any method. An algorithm for template selection for LUT inverse halftoning is introduced. We also extend LUT inverse halftoning to color halftones.

The next topic is image watermarking and effects of halftoning on watermarked images. Watermarking is the process of embedding a secret signal into a host signal in order to verify ownership or authenticity. We discuss the effects of applying inverse halftoning before detection of watermark in halftoned images and offer methods to improve watermark detection from halftoned images.

Finally, we consider the optimal histogram modification with MSE metric and optimal codebook selection problem. Watermarking with histogram modification is one of the few watermarking methods which is robust to rotation and scaling. We formulate histogram modification problem as finding a transformation such that the error between the input and the output signal is minimized and the output signal has the desired histogram. It turns out that this problem is equivalent to the integer linear programming problem. Then, we formulate the problem of finding the optimal codebook where the codewords can come from a finite set. The equivalent problem again turns out to be a linear integer programming problem and the solution is guaranteed to be globally optimal.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

In this thesis, we first introduce the halftoning problem and review different standard methods to solve it. Then we describe novel halftoning algorithms and demonstrate their performance. Another interesting problem addressed here is the inverse halftoning of images. We will discuss the importance of this problem and describe the existing solutions. This will be followed by an introduction of our novel inverse halftoning algorithms. The performance of these algorithms will be assessed by applying them to different kinds of halftones. Then we will study the watermarking problem and examine the effects of halftoning and inverse halftoning on watermark detection. The last topic will be a specific way of embedding watermarks into images through histogram modification. In the following we give a brief explanation of the topics discussed in the thesis.

1.1 The history of halftoning

Halftoning is the rendition of continuous-tone pictures on media on which only two levels can be displayed. The problem arose in the late 19th century when printing machines attempted to print images on paper. This was accomplished by adjusting the size of the dots according to local image intensity. This process is called analog halftoning and the details of this process are given below. With the proliferation of bilevel devices, the digital halftoning problem also gained importance. Some of these bilevel devices are fax machines, printers, plasma display panels, etc. Halftoning is vastly used in printing newspapers, magazines, etc. The first commercial halftone screen was used in 1866 to print continuous tone images on paper by arranging dots for a specific image. Some of the screen densities used are 50-85 lpi (line per inch) for newspapers, 100-120 lpi for highly polished papers and for some magazines, 120-150 lpi for color illustrations in magazines and for books printed on coated papers. This work will focus on digital halftoning.

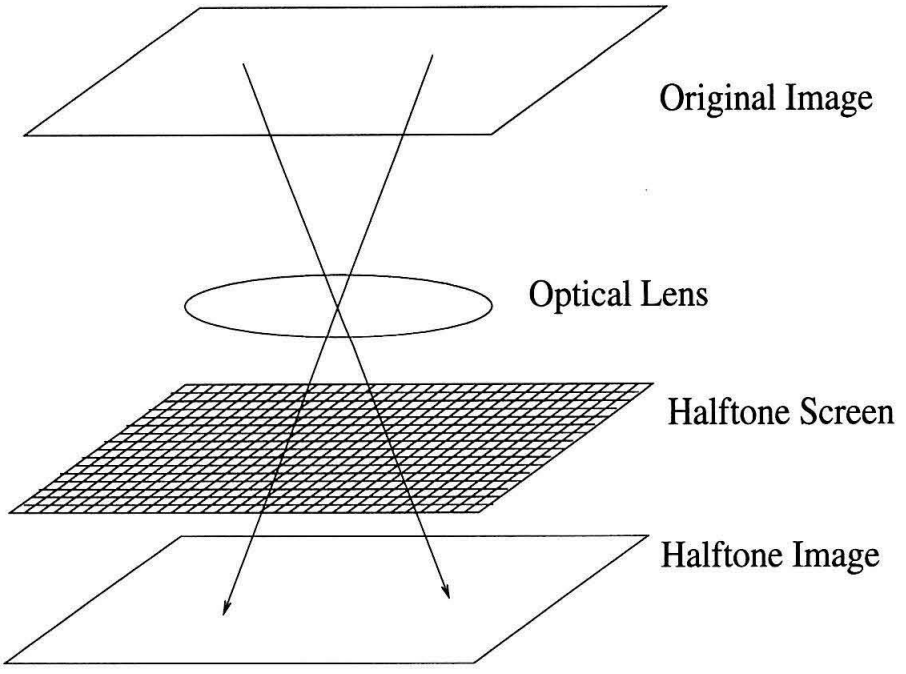

Figure 1.1: Analog halftoning process.

1.1.1 Analog halftoning

An analog halftone is a photoengraving made from an image photographed through a screen and then etched so that the details of the image are reproduced in dots. This is useful in rendering smooth variations of color in the original image by means of dots assigned to areas of the image. This process is depicted in Fig. 1.1. An image is illuminated by a light source. The light coming from the image passes through a halftone screen. The halftone screen consists of a series of regular spaced opaque lines on glass, crossing at right angles, producing transparent apertures between intersections. This screen breaks up a solid or continuous tone image into a pattern of small dots. This is depicted in Fig. 1.2. The incident wave goes through the openings of the halftone screen and creates a luminance pattern on the surface of the high contrast film due to diffraction. The brightness of the incident wave modulates the peak of this luminance pattern. Then this pattern translates into black and white areas due to the high contrast nature of the film.

1.1.2 Digital halftoning

Digital halftoning is the rendition of continuous-tone pictures on displays that are capable of producing only two levels. This process is depicted in Fig. 1.3. The input is an image

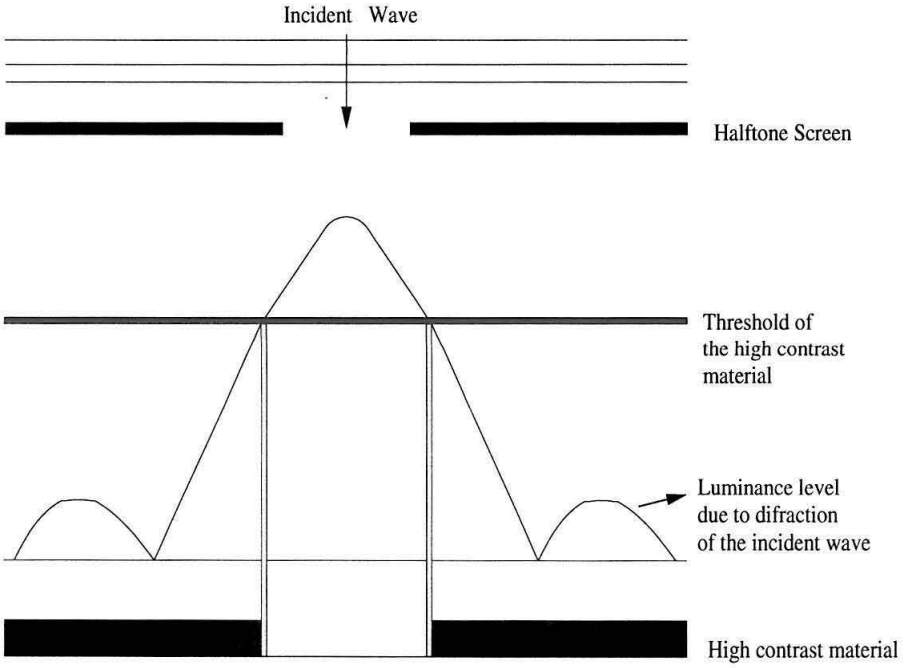

Figure 1.2: The diffraction pattern due to halftone screen.

whose pixels have more than two levels, e.g., 256 levels, and the result of the halftoning process is an image which has only two levels. These two images look nearly the same to human eye when viewed from a distance. Thus halftoning process exploits the low pass characteristics of the human visual system.

There are different kinds of halftoning algorithms. Some of these algorithms are summarized in Table 1.1, categorized by their computational complexity, halftone quality that they produce, and amount of parallelism they offer. Among these algorithms, the ordered dither algorithm is the simplest one and it has complete parallelism but the worst halftone

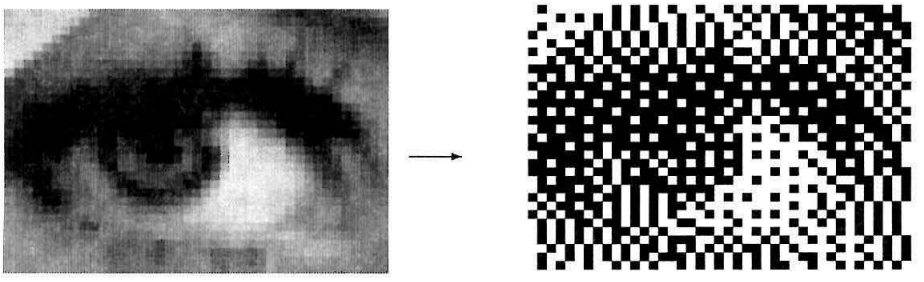

Figure 1.3: Digital halftoning process.

Method	Complexity	Halftone Quality	Parallelism
Ordered Dither	extremely low	poor	complete
Dot Diffusion	low	poor	substantial
Error Diffusion	low	good	none
Direct Binary Search	high	best	none

Table 1.1: Comparison of different halftoning algorithms.

quality. On the other hand error diffused halftones are good, but error diffusion is an inherently serial algorithm. Notice that the best halftone quality is achieved by direct binary search algorithm, but this algorithm is computationally very intense. The details of these algorithms are given in Chapter 2.

The dot diffusion method for digital halftoning has the advantage of pixel-level parallelism unlike the popular error diffusion halftoning method. However, image quality offered by error diffusion is still regarded as superior to most of the other known methods. In Chapter 2 we show how the dot diffusion method can be improved by optimization of the so-called class matrix. By taking the human visual characteristics into account, we show that such optimization consistently results in images comparable to error diffusion, without sacrificing the pixel-level parallelism. This is followed by a discussion on special cases of dot diffusion. Adaptive dot diffusion is also introduced and then a mathematical description of dot diffusion is derived. Embedded multiresolution dot diffusion is also discussed, which is useful for rendering at different resolutions and transmitting images progressively.

In the second part of Chapter 4, we introduce LUT (Look Up Table) based halftoning and tree-structured LUT (TLUT) halftoning. Pixels from a causal neighborhood (template) and contone value of the current pixel will be included in LUT. This process is depicted in Fig. 1.4 and details of this algorithm are given in Chapter 4. We will demonstrate how error diffusion characteristics can be achieved with this method. The performance of LUT halftoning will be improved by TLUT halftoning. This is achieved by making the templates adaptive. Even though TLUT method is more complex than LUT halftoning, it produces better halftones and requires much less storage than LUT halftoning. Afterwards, our algorithm will be trained on halftones obtained by Direct Binary Search (DBS). The complexity of TLUT halftoning is higher than that of error diffusion algorithm but much lower than that of the DBS algorithm. Also, the quality of TLUT halftones increases if

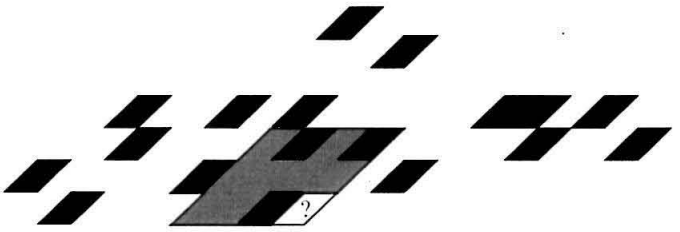

Figure 1.4: LUT halftoning.

the size of TLUT gets bigger. Thus, diffent halftone image qualities between that of error diffusion and DBS can be achieved depending on the size of tree structure in the TLUT algorithm.

1.2 Inverse halftoning problem

Inverse halftoning is the reconstruction of a continuous tone image from its halftoned version. Inverse halftoning has a wide range of applications. Examples include image compression, printed image processing, scaling, enhancement, etc. In these applications, operations cannot be done on the halftone image directly, and inverse halftoning is mandatory.

Since there can be more than one continuous tone image giving rise to a particular halftone image, there is no unique inverse halftone of a given halftoned image. Thus, extra properties of images are needed in order to do inverse halftoning. The basic assumption in all inverse halftoning algorithms is that "natural" images have "mostly lowpass" characteristics. Simple low pass filtering can remove most of the noise due to halftoning but it also removes edge information. Besides lowpass filtering, there are more sophisticated approaches for inverse halftoning. The method of projection onto convex sets (POCS) has been used by Analoui and Allebach [4] for halftone images produced by ordered dithering. For error diffused halftones, Hein and Zakhor [22] have successfully used the POCS approach. A different method called logical filtering has been used by Fan [16] for ordered dither images. Wong [62] has used an iterative filtering method for inverse halftoning of error diffused images. The method of overcomplete wavelet expansions has been used in [64] to produce inverse halftones with good quality for error diffused images by separating the halftoning noise from the original image through edge detection. Another method for inverse halftoning of error diffused images was introduced by Kite et al. in [27]. This method is not only fast but also yields images of very good quality. The method uses space varying filtering based

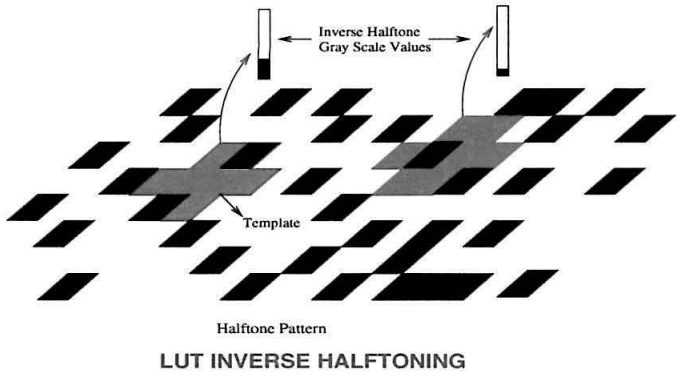

Figure 1.5: LUT inverse halftoning.

on gradients obtained from the image.

In Sections 2.7-2.8, inverse halftoning of dot diffused images is addressed and two methods are discussed. The first one uses Projection Onto Convex Sets (POCS) and the second one uses wavelets.

In Chapter 3 we propose LUT (Look Up Table) based methods for inverse halftoning of images. The LUT for inverse halftoning is obtained from the histogram gathered from a few sample halftone images and corresponding original images. For each pixel, the algorithm looks at pixel's neighborhood (template) and depending upon distribution of pixels in template, it assigns a contone value from a precomputed LUT. The algorithm is depicted in Fig. 1.5. The method is extremely fast (no filtering is required) and the image quality achieved is comparable to the best methods known for inverse halftoning. The LUT inverse halftoning method does not depend on the specific properties of the halftoning method, and can be applied to any halftoning method. An algorithm for template selection for LUT inverse halftoning is introduced. We demonstrate the performance of the LUT inverse halftoning algorithm on error diffused images and dithered images. We also extend LUT inverse halftoning to color halftones.

Many of the entries in the LUT are unused because the corresponding binary patterns hardly occur in commonly encountered halftones. These are called nonexistent patterns. In Chapter 4, we propose a tree structure which will reduce the storage requirements of an LUT by avoiding nonexistent patterns. First a small template LUT will be used to get a crude inverse halftone. Then this value will be refined by adaptively adding pixels to the template depending on context. The TLUT algorithm is depicted in Fig. 1.6. We will

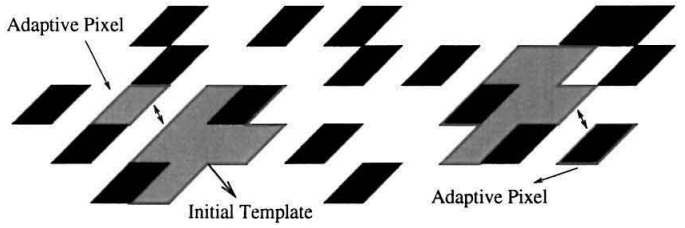

Figure 1.6: TLUT inverse halftoning.

demonstrate its performance on error diffused images and ordered dither images.

1.3 Watermarking and effects of watermarking on halftoned images

Watermarking is the process of embedding a secret signal into a host signal in order to verify ownership or authenticity. The watermark in an image should be detectable robustly even after common image processing algorithms are applied on the watermarked image. These algorithms include compression, scaling, cropping, printing and scanning, etc. Application of these algorithms can be thought of as a distortion to the watermarked images.

It is sometimes necessary to watermark halftoned images. In Chapter 5 we discuss the effects of applying inverse halftoning before detection of watermark in halftoned images. Inverse halftoning can be thought of as another distortion to the watermarked image. We will show an enhancement in watermark detector performance if inverse halftoning is used on smooth error diffused images and a deterioration in performance if it is used on high frequency error diffused images. We also observe that inverse halftoning of clustered dot and ordered dither images deteriorates the watermark detection. These results are obtained when a specific watermarking scheme is used, and we also report experimental results when other watermarking schemes are used.

1.4 Optimal histogram modification with MSE metric and the optimal codebook selection problem

Most of the watermarking methods such as spread spectrum watermarking cannot detect the watermark if an unknown amount of rotation or scaling is applied on the watermarked

images. Watermarking with histogram modification is one of the watermarking methods which is robust to rotation and scaling [10]. However, so far the histogram modification problem has been solved in an adhoc manner without defining an error criterion between the original and modified images.

In the first part of Chapter 6 we propose a method to modify the histogram of a signal to a desired specific histogram. Traditionally, points having the same value in the input signal are all mapped to same value in the output signal. Hence, the desired histogram can only be approximated. Here we formulate the histogram modification problem as finding a transformation such that the error between the input and the output signal is minimized and the output signal has the desired histogram. It turns out that this problem is equivalent to an integer linear programming problem. This method might be specifically useful for histogram based watermarking and compression. In the second part of Chapter 6 we formulate the problem of finding the optimal codebook where the codewords can come from a finite set. The equivalent problem turns out to be a linear integer programming problem and the solution is guaranteed to be globally optimal.

Chapter 2 Optimized halftoning using dot diffusion and methods for inverse halftoning of dot diffused images

2.1 Introduction

Digital halftoning is the rendition of continuous-tone pictures on displays that are capable of producing only two levels. There are many good methods for digital halftoning: ordered dither [8], error diffusion [17], neural-net based methods [5], and more recently direct binary search (DBS) [54]. Ordered dithering is a thresholding of the continuous-tone image with a spatially periodic screen [8]. In error diffusion [17], the error is 'diffused' to the unprocessed neighboring pixels.

Ordered dithering is a parallel method, requiring only pointwise comparisons. But the resulting halftones suffer from periodic patterns. On the other hand, error diffused halftones do not suffer from periodicity and offer blue noise characteristic [58] which is found to be desirable.[1]

The main drawback is that error diffusion is inherently serial.[2] Also there occur worm-like patterns in near mid-gray regions and resulting halftones have ghosting problems [28]. Mitsa and Parker have optimized the ordered dither matrix [51] for large sizes like 256x256 to get the blue noise effect. This is a compromise between parallelism and image quality.

The dot diffusion method for halftoning introduced by Knuth [28] is an attractive method which attempts to retain the good features of error diffusion while offering substantial parallelism. However, surprisingly, not much work has been done on optimization of the so-called class matrix. In this work we will show that the class matrix (see below) can further be optimized by taking into account the properties of human visual system (HVS). The resulting halftones will then be of the similar quality as for error diffusion. Since dot

[1] More recently, it has been shown that **green noise** is more appropriate for non-ideal printers, which suffer from dot gain [31]. In this chapter we consider ideal printer models.

[2] It can be shown [15] that error diffusion for an $M \times N$ image can, in principle, be implemented in $M+N$ steps by using sufficient number of parallel computations.

diffusion also offers increased parallelism, it now appears to be an attractive alternative to error diffusion.

In this chapter, we first review the dot diffusion method in Section 2.2. In the following section, the optimization of class matrix will be discussed and adaptive dot diffusion will be introduced. We will discuss special cases of dot diffusion algorithm in Section 2.4. Then, in Section 2.5, we will give a mathematical description of dot diffusion method. Furthermore, we will address the inverse halftoning problem in Sections 2.6-2.8. Inverse halftoning has a wide range of applications such as compression, printed image processing, scaling, enhancement, etc. In these applications, operations cannot be done on the halftone image directly, and inverse halftoning is mandatory. For inverse halftoning two methods are discussed. One of the methods uses POCS (Projection Onto Convex Sets) which is an iterative algorithm. The other one is based on wavelet decomposition of images to differentiate the halftoning noise from the original image. Then a simple yet efficient algorithm for inverse halftoning of dot diffused images is proposed and compared to other methods. In Section 2.9 embedded multiresolution dot diffusion is discussed. Parts of this chapter have been presented at conferences [36],[37],[38] and [39] and published in two journal magazines [40],[41].

2.2 Review of dot diffusion

The dot diffusion method for halftoning has only one design parameter, called **class matrix C**. It determines the order in which the pixels are halftoned. Thus, the pixel positions (n_1, n_2) of an image are divided into IJ classes according to (n_1 mod I, n_2 mod J) where I and J are constant integers. Table 1 is an example of the class matrix for $I = J = 8$, used by Knuth. There are 64 class numbers. Let $x(n_1, n_2)$ be the continuous tone (contone) image with pixel values in the normalized range $[0, 1]$. Starting from class $k = 1$, we process the pixels for increasing values of k. For a fixed k, we take all pixel locations (n_1, n_2) belonging to class k and define the halftone pixels to be

$$h(n_1, n_2) = \begin{cases} 1 & \text{if } x(n_1, n_2) \geq 0.5 \\ 0 & \text{if } x(n_1, n_2) < 0.5 \end{cases} . \tag{2.1}$$

We also define the error $e(n_1, n_2) = x(n_1, n_2) - h(n_1, n_2)$. We then look at the eight neighbors of (n_1, n_2) and replace the contone pixel with an adjusted version for those neigh-

bors which have a higher class number (i.e., those neighbors that have not been halftoned yet). To be specific, neighbors with higher class numbers are replaced with

$$x(i,j) + 2e(n_1,n_2)/w \quad \text{(for orthogonal neighbors)} \tag{2.2(a)}$$

$$x(i,j) + e(n_1,n_2)/w \quad \text{(for diagonal neighbors)} \tag{2.2(b)}$$

where w is such that the sum of errors added to all the neighbors is exactly $e(n_1,n_2)$. The extra factor of two for orthogonal neighbors (i.e., vertically and horizontally adjacent neighbors) is because vertically or horizontally oriented error patterns are more perceptible than diagonal patterns.

The contone pixels $x(n_1,n_2)$ which have the next class number $k+1$ are then similarly processed. The pixel values $x(n_1,n_2)$ are of course not the original contone values, but the adjusted values according to earlier diffusion steps (2.2). When the algorithm terminates, the signal $h(n_1,n_2)$ is the desired halftone.

This diffusion process is illustrated in Fig. 2.1. The numbers in the matrix are elements of a class matrix and the integers in the bubbles are relative weights of diffusion coefficients. The neighbors of 33 with higher class numbers are those labeled as 58, 45, 42, 40, 63, 47. The error created at 33 is divided by the sum of relative weights of diffusion coefficients, which is $2+1+2+1+2+1=9$ in this case. The result of the division, ϵ, is the error to be diffused to diagonal neighbors, and 2ϵ is diffused to orthogonal neighbors. Since there are 64 classes, the algorithm completes the halftoning in 64 steps.

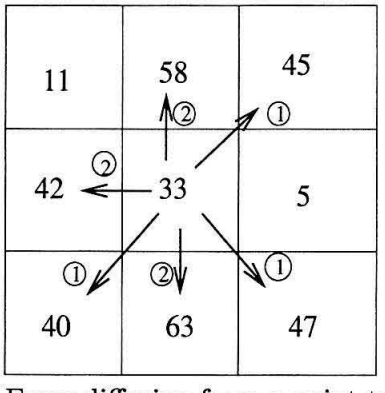

Figure 2.1: Error diffusion from a point to the neighbor points.

Usually an image is enhanced [28] before dot diffusion is applied. For this the continuous image pixels $C(i,j)$ are replaced by $C'(i,j) = \frac{C(i,j)-\alpha\bar{C}(i,j)}{1-\alpha}$ where $\bar{C}(i,j) = \frac{\sum_{u=i-1}^{i+1}\sum_{v=j-1}^{j+1} C(u,v)}{9}$.

Here the parameter α determines the degree of enhancement. If $\alpha = 0$, there is no enhancement, and the enhancement increases as α increases. If $\alpha = 0.9$ then the enhancement filter simplifies to

$$C'(i,j) = 8C(i,j) + C(i,j) - \sum_{0<(u-i)^2+(v-j)^2<3} C(u,v).$$

This algorithm is completely parallel requiring 9 additions per pixel, and no multiplications.

2.3 Optimization of class matrix

Knuth introduced the notion of **barons** and **near-barons** in the selection of his class matrix. A baron has only low-class neighbors, and a near-baron has one high class neighbor. The quantization error at a baron cannot be distributed to neighbors, and the error at a near-baron can be distributed to only one neighbor. Knuth's idea was that the number of barons and near-barons should therefore be minimized. He exhibited a class matrix with two barons and two near-barons (Table 1). The quality of the resulting halftones still exhibits periodic patterns similar to ordered dither methods (see Fig. 2.6). Knuth has also produced a class matrix with one baron and near-baron, but unfortunately these were vertically lined up to produce objectionable visual artifacts. In our experience, the baron/near-baron criterion does not appear to be the right choice for optimization. To explain this, define a k-baron to be a position which has k high-level neighbors. Thus $k = 0$ corresponds to a baron, $k = 1$ to a near baron, and $k = 8$ to an **antibaron.** We have produced a class matrix which minimizes the number of k-barons sequentially for $0 \leq k \leq 8$. The resulting halftone quality was found in most cases to be slightly worse than Knuth's original results, leading us to conclude that baron minimization is not the right approach. In Section 2.3.1 we introduce a different optimization criterion based on the HVS, and show that the image quality is significantly improved, though the class matrix does not minimize barons.

2.3.1 Objective function based on blue noise

It has been observed in the past that the error in a good halftone should have the **blue noise** property [58]. This means that the noise energy should mostly be in the high frequency region where it is known to be less perceptible. We will show how to incorporate blue noise

characteristics into the class matrix optimization.

Imagine that we have a constant gray image $x(n_1, n_2) = g$ where $0 \leq g \leq 1$. Let $h(n_1, n_2)$ denote the halftoned version. Since the halftone is supposed to create the perception of the gray level, the average number of dark pixels should be equal to the original gray level.[3] Typically, therefore, the dark pixels are spatially distributed with a certain average frequency f_g called the **principal frequency**, which increases with gray level g. The preference for blue noise [58] (high frequency white noise) in halftoning arises because noise energy at a significantly higher spatial frequency than f_g is less perceivable. Thus, we can optimize a halftoning method for a particular gray level g by forcing the noise spectrum to be concentrated above f_g.

This does not, however, imply optimality at other gray levels. Interestingly, however, if the gray level g during the optimization phase is chosen carefully, the resulting halftones for arbitrary natural images are excellent. For example, we optimized the class matrix in the dot diffusion method for the gray level $g = \frac{1}{16}$ and obtained very good halftones for natural images as we demonstrate in this section.

Calculating the noise spectrum. In order to implement the optimization, we first need to compute the noise spectrum. The halftone pattern $h(n_1, n_2)$ for the gray level $x(n_1, n_2) = g$ has the error $e(n_1, n_2) = g - h(n_1, n_2)$, which is an $N \times N$ image. Imagine that this is divided into $L \times L$ blocks so there are $B = (N/L)^2$ blocks. (In our experiment $N = 256, L = 64, B = 16$.) Let $E_m(l_1, l_2)$ be the $L \times L$ DFT of the the mth block of $e(n_1, n_2)$. We define the average noise spectrum as

$$P(l_1, l_2) = \frac{1}{B} \sum_{m=0}^{B-1} |E_m(l_1, l_2)|^2.$$

From this we compute the so-called **radially averaged power spectrum** $P_r(k_r)$ where k_r is a scalar called the radial frequency. Since $|l_1|$ and $|l_2|$ range from 0 to $L/2$, k_r ranges from 0 to $L/\sqrt{2}$. We take specific integer values for k_r and calculate $P_r(k_r)$ as follows. For each chosen k_r define an annulus $\mathcal{A}(k_r)$ in the (l_1, l_2) plane by the equation

$$\left| \sqrt{l_1^2 + l_2^2} - k_r \right| < \Delta/2.$$

The quantity Δ, which determines the width of the annulus, is chosen as unity in our

[3] Note that a grey level of 0 represents white and a grey level of 1 represents black.

calculation. With $N(k_r)$ denoting the number of elements in $\mathcal{A}(k_r)$, the radially averaged power spectrum of the error for gray level g is then

$$P_r(k_r) = \frac{1}{N(k_r)} \sum_{l_1, l_2 \in \mathcal{A}(k_r)} P(l_1, l_2).$$

The class matrix in the dot diffusion method should be optimized such that this radial spectrum is appropriately shaped for a well-chosen fixed gray level g. In terms of the radial frequency variable k_r, the principal frequency for the halftone of gray level g is given by

$$f_g = k_{max}\sqrt{g}$$

where $k_{max} = L$. In fact, for $g > 0.5$, since black pixels are more in number, the halftone is perceived as a distribution of white dots [58] and we have to take $f_g = k_{max}\sqrt{1-g}$.

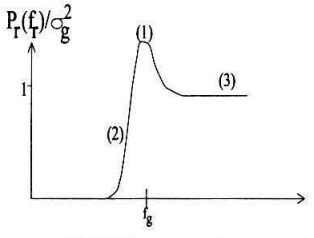

Figure 2.2: A typical desired radial spectrum characteristics.

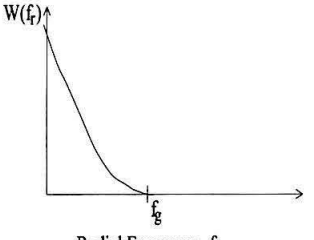

Figure 2.3: The weight function used in the optimization.

The aim of the optimization is to shape $P_r(k_r)$ by choice of the class matrix $\mathbf{C}$ so that most of its energy is moved to the region $k_r > f_g$ (as demonstrated in Fig. 2.2). We therefore define the cost function

$$\Phi(\mathbf{C}, g) = \int_0^{f_g} P_r(k_r) w(k_r) dk_r.$$

The idea is to choose the weighting function $w(k_r)$ such that upon minimization of the above function, $P_r(k_r)$ has a low frequency cutoff at principal frequency f_g, sharp transition region, and a flat high frequency region. The weight function was chosen to be $w(k_r) = (k_r - f_g)^2$ for $0 \leq k_r \leq f_g$ and zero outside. (In Section 2.3.2 we consider more sophisticated weighting functions.) In the optimization the integral was replaced with a discrete sum. The choice of the class matrix that minimizes this sum was performed using the **pairwise exchange algorithm** [2] described below:

1) Randomly order the numbers in the class matrix.

2) List all possible exchanges of class numbers.

3) If an exchange does not reduce cost, restore the pair to original positions and proceed to the next pair.

4) If an exchange does reduce cost, keep it and restart the enumeration from the beginning.

5) Stop searching if no further exchanges reduce cost.

6) Repeat the above steps a fixed number of times and keep the best class matrix. [4]

Choice of gray level. Since the algorithm can be applied only to a given gray level, the gray level should be chosen wisely, in order to get good halftones for other gray levels also. In our experience if we perform this optimization for a fixed small gray level (e.g., $g = \frac{1}{16}, \frac{1}{8}$ etc.), we get good halftones for natural images also. Class matrices obtained from optimization with a very small gray level will not work, because there is not much error to diffuse to other points during the dot diffusion process. Mid gray levels are not suitable, first because there are huge diffusions between points, and second, even unoptimized algorithms yield perceptually pleasing halftones for mid gray anyway. The actual gray level used in the optimization was $g = \frac{1}{16}$, and it was found experimentally. The optimized class matrix is shown in Table 2. Notice that the optimal class matrix has *several barons and near barons.*

Example: The 512×512 continuous tone peppers image was halftoned by using Knuth's class matrix (Fig. 2.6), and by the optimized class matrix (Fig. 2.7). It is clear that the new method is visually superior to unoptimized dot diffusion method. In fact, the new method offers a quality comparable to Floyd-Steinberg error diffusion method (Fig. 2.5). Error diffused images suffer from worm-like patterns which are not in the original image, whereas dot diffused halftones do not contain these artifacts. Notice that the artificial periodic patterns in Fig. 2.6 are absent in Fig. 2.5 and in the new method (Fig. 2.7).[5]

[4]Note that pairwise exchange algorithm yields a local minimum of the cost function. We start the pairwise exchange with random class matrices and take the class matrix having the least local minimum in order to get closer to the global minimum. Global minimum is not guaranteed.

[5]The halftone and inverse halftone images can be found at [66].

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

Table 1: Class matrix C used in Knuth's method.

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

Table 2: Class matrix C obtained by parabolic weighting function.

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

Table 3: Class matrix C obtained by HVS function.

2.3.2 Other choices for the weighting function

For simplicity we have chosen our weighting function above to be the parabola $w(k_r) = (k_r - f_g)^2$ for $0 \leq k_r \leq f_g$ and zero outside. Another alternative is to use the HVS function as the weighting function. The images are passed through a model of the HVS function. Since our model is linear, we apply the HVS function to the difference between the original and halftone images. The energy of the resulting image is defined to be the perceived halftoning error (PHE). The calculation of PHE of a given image $x[m,n]$ for a given HVS function $h[m,n]$ is depicted in Fig. 2.8. In the figure the output of the energy calculator is: $c = \Sigma_{m=-\infty}^{\infty}\Sigma_{n=-\infty}^{\infty}|e[m,n]|^2$. The image used in the optimization should be chosen wisely. For example, in this section we have chosen a gray scale ramp because the cost of a gray scale ramp is the average value of the costs of gray scales which exist in the gray scale ramp.

We will use a specific HVS model in the optimization. In the frequency domain the HVS model is defined as follows:

$$H_c(u,v) = aL^b e^{-\frac{1}{s(\phi)}\frac{\sqrt{u^2+v^2}}{clog(L)+d}}.$$

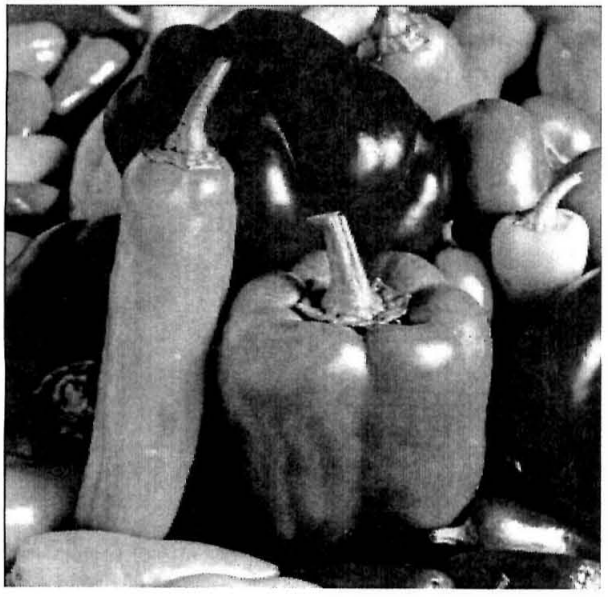

Figure 2.4: Original image, peppers.

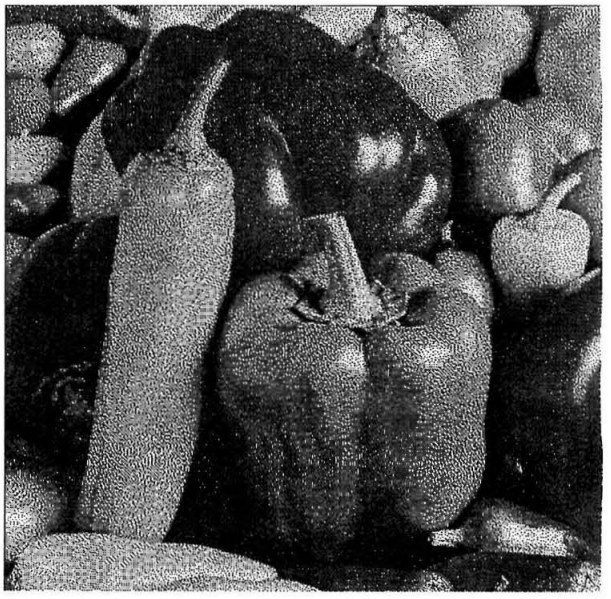

Figure 2.5: Floyd-Steinberg error diffusion.

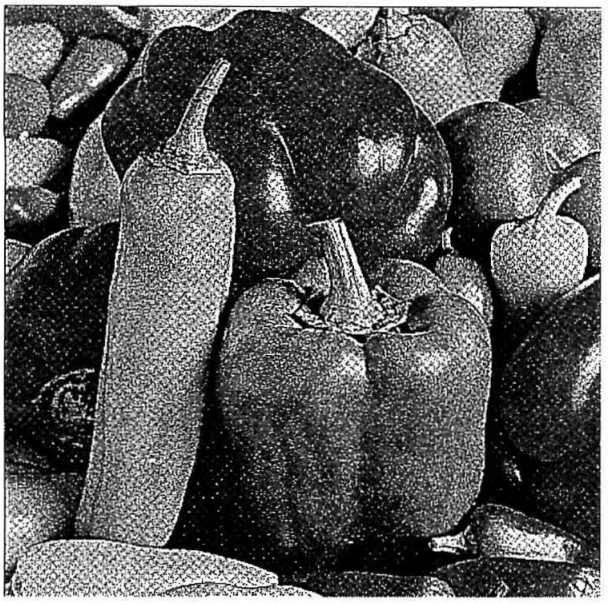

Figure 2.6: Dot diffusion with Knuth's class matrix.

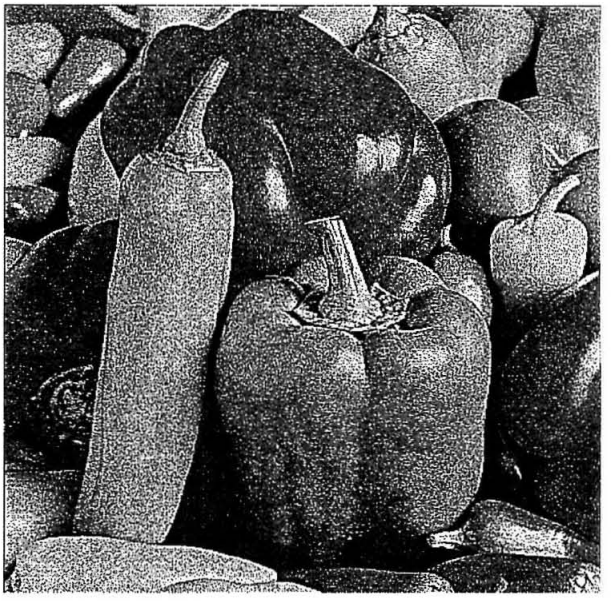

Figure 2.7: Dot diffusion with enhancement and 8×8 class matrix optimized using parabolic weighting function.

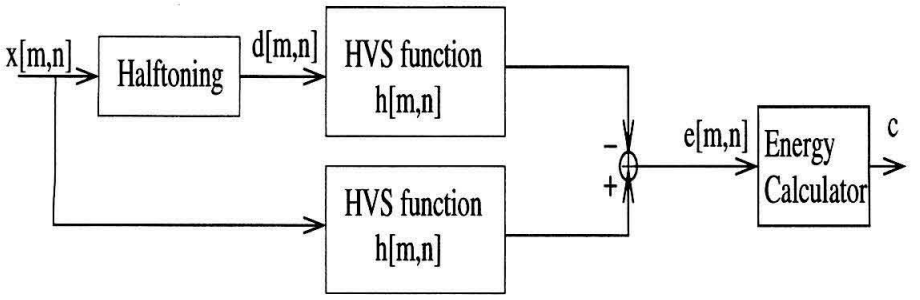

Figure 2.8: Perceived halftoning error of an image for a given HVS function.

Here, u and v are the frequency variables in cycles/degree subtended at the retina and L is the average luminance in $candela/m^2$. The quantity $\sqrt{u^2+v^2}$ is therefore the radial frequency. The quantity ϕ is the angular frequency defined as $\phi = atan(\frac{u}{v})$. The various constants and the function $s(\phi)$ are defined as follows:

- $a = 131.6$, $b = 0.3188$, $c = 0.525$, $d = 3.91$.
- $s(\phi) = \frac{1-w}{2}cos(4\phi) + \frac{1+w}{2}$ where $w = 0.7$.

The dependence on radial frequency $\sqrt{u^2+v^2}$ was developed in [52]. Then the angular dependence of the model, i.e., $s(\phi)$ was introduced in [3] following Daly's model [12]. We used $L = 10$ $candela/m^2$ in our experiments. With $h_c(x,y)$ denoting the inverse Fourier transform of $H_c(u,v)$, the discretized version $h[m,n] = h_c(Tm,Tn)$ is used in the calculations. The relation between $H(w_1,w_2)$ (the discrete fourier transform of $h[m,n]$) and $H_c(u,v)$ is as follows:

$$H(w_1,w_2) = \frac{1}{T}\Sigma_{k=-\infty}^{\infty}\Sigma_{l=-\infty}^{\infty}H_c(\frac{w_1-2\pi k}{2\pi T}, \frac{w_2-2\pi l}{2\pi T}). \tag{2.3}$$

Sampling the inverse transform at interval $T = 0.0165$ corresponds to a certain printer resolution, R dpi, viewed at a specific distance, D inches. Since a length x viewed at a distance D subtends an angle of $\Theta = tan^{-1}(x/D) \approx x/D$ radians for $x \ll D$, the spacing of the dots will be

$$T = \frac{1}{RD}\text{radians} = \frac{180}{\pi}\frac{1}{RD}\text{degrees}. \tag{2.4}$$

This clarifies the relation between T, D and R. In particular, $T = 0.0165$ corresponds to $RD = 300\text{dpi} \times 11.5827\text{in}$. In Fig. 2.9, the normalized HVS function is shown for this value

of T.

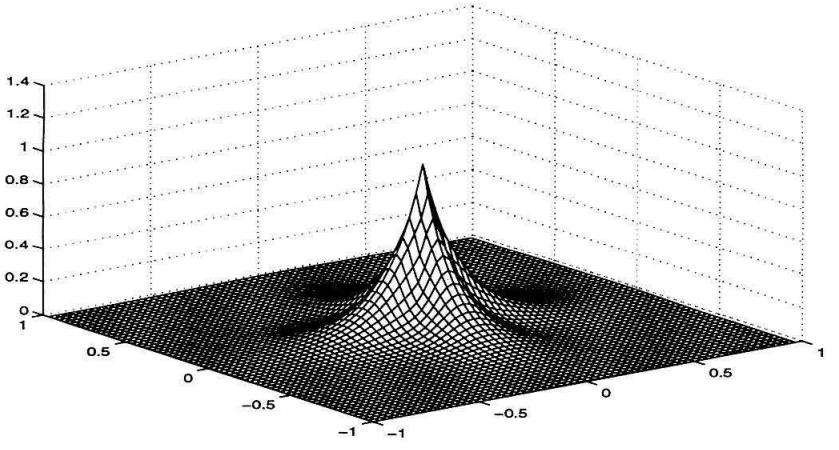

Figure 2.9: Normalized HVS function $H(w_1, w_2)$ for $T = 0.0165$ ($RD = 300\text{dpi} \times 11.5827\text{in}$). The axes are $\frac{w_1}{\pi}$ and $\frac{w_2}{\pi}$.

Notice that the HVS weighting filter has three basic properties: (1) It is a decaying function in terms of frequency. (2) The HVS response along the 45-degree line is $\frac{1}{\sqrt{2}}$ of the response to horizontal and vertical lines. (3) Weights are nonzero for all frequencies. The parabolic weight function is circularly symmetric, and becomes zero after a certain frequency. So, the parabolic weighting function does not have properties 2 and 3. We optimized the class matrix with this HVS function as the weighting function.

In the optimization we have used a gray scale image. First we have done the optimization for 8×8 class matrix. We have optimized this class matrix for $RD = 300\text{dpi} \times 11.5827\text{in}$. This class matrix is shown in Table 2.1. The dot diffused image of a constant gray level $g = \frac{1}{16}$ with the class matrix optimized using HVS function and the dot diffused image of the same constant gray level with the class matrix optimized using the parabolic weight function are shown in Fig. 2.10. From the dot patterns it can be seen that the HVS function aligns the pattern in diagonal directions, and the dot pattern looks more irregular.

We have summarized the perceived halftoning errors in Table 2.2. In this chapter, perceived errors are normalized so that perceived error of a gray scale ramp halftoned by the Floyd Steinberg error diffusion algorithm is unity. As it can be seen from the table, our optimized class matrix achieves 40.04% less PHE than Knuth's class matrix and 51.61% more PHE than error diffusion.

Figure 2.10: Optimized dot patterns for g=1/16. Top: using HVS function, bottom: using the parabolic weighting function.

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

Table 2.1: 8×8 optimized class matrix.

Example: The 512×512 continuous tone peppers image was halftoned by using Knuth's class matrix (Fig. 2.6, $PHE = 30.77$), and by the optimized 8×8 class matrix (Fig. 2.11, $PHE = 30.35$).[6] The images in this chapter are printed out with $R = 150$ *dpi*, thus they should be viewed from a distance $D \approx 23$ *inches*. It is clear that the dot diffusion method with the optimized 8×8 class matrix is visually superior to dot diffusion method with Knuth's class matrix. In fact, dot diffusion with the optimized 8×8 class matrix offers a quality comparable to Floyd-Steinberg error diffusion method (Fig. 2.5, $PHE = 3.86$). Note that we cannot use the PHE values of error diffused images and images obtained by dot diffusion with enhancement to compare the visual quality of these two methods because of the enhancement step. Thus visual inspection is necessary. Error diffused images suffer from worm-like patterns which are not in the original image, whereas dot diffused halftones do not contain these artifacts. Notice that the artificial periodic patterns in Fig. 2.6 are absent in Fig. 2.5 and in the dot diffusion with the optimized 8×8 class matrix (Fig. 2.11).

[6]We observed that the enhancement step in dot diffusion is the cause of higher PHE values. In the next section, enhancement step will be removed from dot diffusion algorithm.

Halftoning Method	Dot Diffusion Knuth's Class Matrix	Dot Diffusion Optimized Class Matrix	Error Diffusion (Floyd Steinberg)
Perceived Halftoning Error	2.53	1.52	1.00

Table 2.2: Perceived halftoning errors (PHE) for 8×8 class matrices.

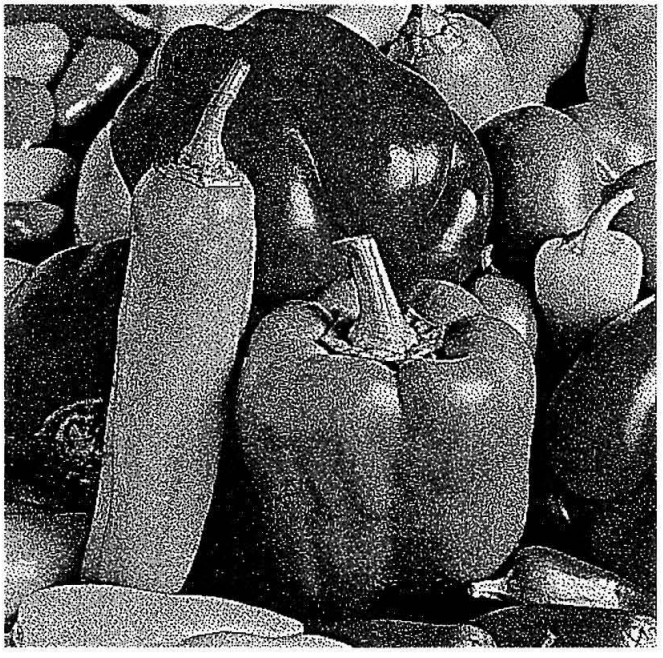

Figure 2.11: Dot diffusion with HVS optimized 8×8 class matrix and enhancement.

2.3.3 Effect of diffusion constants

The diffusion constant β is the ratio between the horizontal diffusion coefficient and the diagonal diffusion coefficient of the dot diffusion process. The diffusion constant has been chosen to be 2.0 by Knuth. The reasons for this selection are as follows:

- It is desirable to diffuse more errors in the vertical and horizontal directions, and keep more errors in the diagonal directions where the eye sensitivity is known to be lower.
- It reduces the number of multiplications if β is chosen as a power of 2.

In Knuth's method the choice $\beta = 2$ was rather crucial because there was no optimization of the class matrix. We have found experimentally that when the class matrix is optimized for a chosen β, the results are relatively insensitive to β as long as it is in the range $1 \leq \beta \leq 2$.

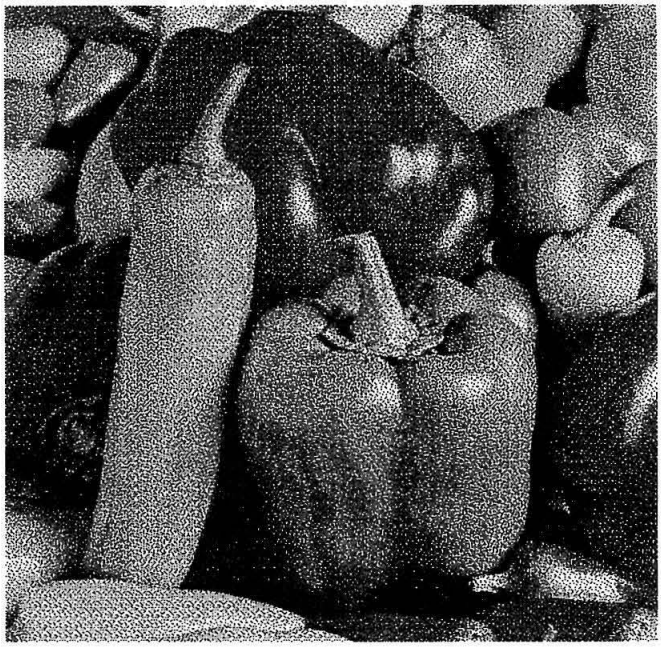

Figure 2.12: Dot diffusion with HVS optimized 8×8 class matrix and no enhancement.

However, there are slight vertical and horizontal patterns when $\beta = 0.5$. For $\beta \leq 1/4$ and $\beta \geq 4$ more serious artifacts are noticeable.

In principle the diffusion coefficient can be chosen with greater degree of freedom. For example we can choose it such that it depends on the class number as well as the direction of diffusion. At this time, however, we do not have an optimization algorithm to optimize such a general set of diffusion coefficients.

2.3.4 Remarks

The optimization of class matrix was done in Section 2.3.1 for constant gray level images only when parabolic weighting function is used. For this, it is necessary to pick the constant gray level strategically such that, for most natural images with multiple gray levels, the halftone quality is good. A natural question here is can we make the class matrix adaptive? For example imagine a library of optimal class matrices $\{C_i\}$ with C_i optimized for the ith gray level. We can divide the image into 8×8 blocks, and for each block a different class matrix can be used depending on the average gray level there. We have done experiments with this idea. Assuming that the image is enhanced prior to halftoning (as in Section

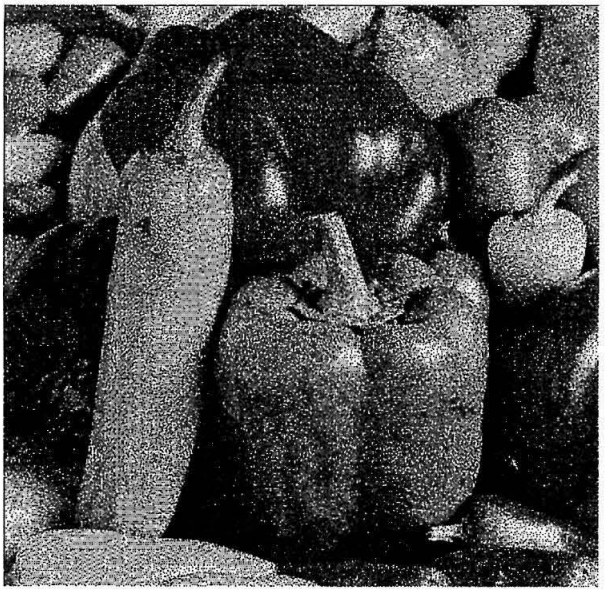

Figure 2.13: Adaptive dot diffusion with no enhancement and 8×8 class matrices optimized using HVS function.

2.2), we found the advantage of the adaptive scheme to be insignificant. However, if there is no enhancement prior to halftoning, then adaptive dot diffusion is significantly better than non adaptive (compare Figs. 2.12 and 2.13). Besides the obvious advantages brought into play by adaptation, there is another reason why this is so. Namely, the grid of pixels formed by a given class number is not a periodic grid in the adaptive case. Any periodicity artifacts created by the periodic class matrix are therefore broken. But there is another problem, namely the boundary effects between the blocks having different class matrices are apparent.

Another parameter in the dot diffusion is the enhancement filter. The enhancement lessens the objectionable halftoning artifacts in other grey levels which are not close to $g = \frac{1}{16}$. This can be seen from the resulting image obtained by dot diffusion with enhancement (Fig. 2.7) and the image obtained by the dot diffusion without enhancement (Fig. 2.12). The periodic patterns show up almost everywhere in the dot diffused images if enhancement is not done prior to halftoning. The enhancement filter used has a parameter α (see Section 2.2) which controls the degree of enhancement where $\alpha = 0$ means no enhancement and $\alpha = 0.9$ is the value used in our experiments. The enhancement parameter α can be lessened

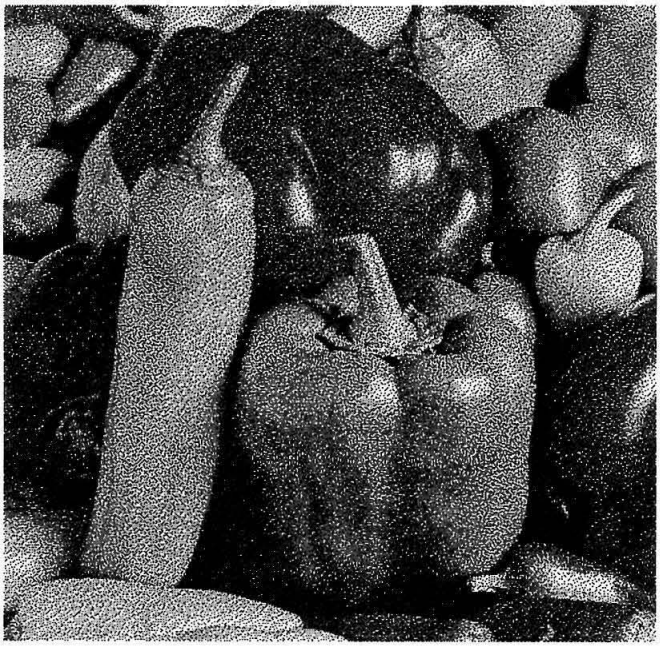

Figure 2.14: Dot diffusion with HVS optimized 16×16 class matrix and no enhancement.

to 0.8 without any perceivable difference.

2.3.5 Dot diffusion without enhancement

In all the discussions so far the halftoning step is preceded by an enhancement filter (described in Section 2.2). The enhancement step reduces halftoning noise, but might be objectionable in some applications because of its very visible sharpening effect (e.g., see Fig. 2.11). It turns out that we can get good halftones without use of the enhancement step provided we make the class matrix larger than the standard 8×8 size. The price paid for the larger class matrix is that the parallelism of the algorithm is compromised. However, in practice, even with a 16×16 class matrix, we have plenty of parallelism for any desktop printing implementation: Assume that we want to render the image at 600 dpi, and process 16 rasters (lines or rows) of an 8.5×11 inch square page simultaneously. Then we will have (600*8*16)/256=300 pixels that can be processed simultaneously.[7] The real disadvantage of increasing the size of the class matrix is the fact that the number of rasters that must be

[7]The number 8 in the numerator of this expression is based on the assumption of an 8 inch wide active printing area.

processed at the same time increases.

We found that if a 16×16 matrix is used, the halftone images resulting from the optimization of this matrix are very good even without the enhancement step. (For comparison we note here that whenever enhancement is used, the class matrix can be as small as 5×5 without creating noticeable periodicity patterns.) Such optimization was carried out using a gray scale ramp as the training image. The HVS function was used in the optimization, and the associated cost was optimized using the pairwise exchange algorithm. The 16×16 optimized class matrix is shown in Table 2.3. The PHE of a gray scale halftoned by dot diffusion with 16×16 optimized class matrix is 1.19. We can compare this PHE value with the PHE values in Table 2.2. The PHE for optimized 16×16 class matrix is only 19.38% worse than the error diffusion. Also, the PHE for optimized 8×8 class matrix is 27.00% worse than the PHE for optimized 16×16 class matrix.

202	1	14	18	51	56	45	105	74	98	75	145	150	170	171	173
4	7	24	37	57	52	66	88	146	103	138	159	183	185	198	222
8	15	25	38	68	70	87	6	107	153	144	166	184	193	225	2
16	27	44	54	29	102	116	132	140	137	167	120	196	224	227	5
23	40	53	72	85	104	165	136	158	174	131	200	223	226	228	17
41	86	73	84	114	118	168	134	169	181	201	220	232	229	13	22
48	121	55	106	124	133	147	177	180	203	221	231	246	3	21	42
77	82	128	110	139	135	179	182	207	197	230	245	247	20	43	50
81	100	113	148	143	172	178	204	219	233	244	250	248	34	49	69
109	108	141	151	186	164	208	218	234	243	249	256	19	46	71	80
111	142	89	76	176	206	215	235	242	251	255	39	47	78	117	101
112	149	161	175	205	216	236	241	252	253	254	62	63	94	95	126
152	160	190	191	209	217	237	240	26	32	61	83	93	96	125	115
157	189	192	210	214	238	239	30	33	60	65	92	119	79	129	156
188	195	199	213	10	11	31	36	59	64	91	97	123	130	155	162
194	211	212	9	12	28	35	58	67	90	99	122	127	154	163	187

Table 2.3: 16x16 class matrix.

The peppers image halftoned with the resulting class matrix is shown in Fig. 2.14 ($PHE = 5.90$). There are no periodic artifacts in this result. While the overall visible noise level appears to be higher than for error diffusion, the problematic halftone patterns of error diffusion in the mid gray level are eliminated here. (Examine the body of the middle pepper in Fig. 2.5). By comparing Fig. 2.6 and Fig. 2.14, we see that 16×16 dot diffusion without enhancement is also superior to 8×8 enhanced dot diffusion using Knuth's matrix

...	$y(n_1-2)+n_2-1$	$y(n_1-2)+n_2$	$y(n_1-2)+n_2+1$	...
...	$y(n_1-1)+n_2-1$	$\mathbf{y(n_1-1)+n_2}$	$y(n_1-1)+n_2+1$	...
...	yn_1+n_2-1	yn_1+n_2	yn_1+n_2+1	...

Table 2.4: A part of the class matrix (defined in Eqn. (2)) which is not close to the boundaries of the class matrix.

because there are no noticeable periodic patterns any more, and there are no enhancement artifacts.

In order to obtain an authentic comparison with good printing quality, we have produced three images in Figs. 2.15– 2.17 using 150 dpi resolution. These are halftoned versions of the Parrot image. Figure 2.15 shows the result of error diffusion, Fig. 2.16, the result of direct binary search (DBS) obtained from the website of the authors of [3], and Fig. 2.17 shows the result of using the 16×16 optimized dot diffusion described above. In terms of image quality, the DBS method is evidently the best one. The dot diffusion output appears to be comparable to error diffusion in most areas of the image. Dot diffusion has the advantage that the complexity is much lower than that of DBS. Moreover, it offers parallelism of implementation unlike error diffusion.

2.4 Special cases of dot diffusion

Commonly used sizes for the class matrix are 8×8 and 16×16. A trivial special case of the dot diffusion algorithm arises when the class matrix is of size 1×1. In this case the algorithm reduces to direct pixel by pixel binary quantization.

Next, what happens when the class matrix is made arbitrarily large? Assume that the image is of size $x \times y$ and let the class matrix of size $x \times y$ be defined as follows:

$$C(n_1, n_2) = (n_1 - 1)y + n_2 \quad \text{for} \quad n_1 = 1, \ldots, x, n_2 = 1, \ldots, y. \tag{2.5}$$

Because of the structure of this class matrix, the pixels are processed in raster scan order the same way the pixels are processed in error diffusion. In Table 2.4 we have shown a part of the class matrix which is not close to the boundaries of the class matrix. In step $(n_1 - 1)y + n_2$, the error due to quantization at (n_1, n_2) is diffused only to the pixels at

Figure 2.15: Floyd-Steinberg error diffusion.

Figure 2.16: Direct binary search (DBS).

Figure 2.17: 16×16 dot diffusion without enhancement.

locations (n_1, n_2+1), (n_1+1, n_2-1), (n_1+1, n_2) and (n_1+1, n_2+1). Furthermore, the error will be diffused to the four neighbor pixels with the diffusion coefficients shown in Table 2.5. Thus, the dot diffusion method becomes identical to error diffusion if the class matrix is as large as the image itself, and defined as in Eqn. (2). The only difference between this special case and the Floyd-Steinberg error diffusion is in the values of the filter coefficients as summarized in Tables 2.5 and 2.6. The filter in Table 2.5 is referred to as the DD (dot diffusion) filter to distinguish it from the FS (Floyd-Steinberg) filter in Table 2.6. Using these two sets of filters in error diffusion, we found that the image qualities are nearly identical. There may be slight differences in the implementation complexities (e.g., the denominators in FS filters are powers of two which makes divisions very easy). The main point of this discussion, though, is to make the conceptual connection between dot diffusion and error diffusion.

	×	$\frac{2}{6}$
$\frac{1}{6}$	$\frac{2}{6}$	$\frac{1}{6}$

Table 2.5: Diffusion coefficients for the case where dot diffusion reduces to error diffusion (infinite size class matrix). These are called the DD filter coefficients.

	×	$\frac{7}{16}$
$\frac{3}{16}$	$\frac{5}{16}$	$\frac{1}{16}$

Table 2.6: Floyd-Steinberg error diffusion coefficients. These are called the FS filter coefficients.

2.5 Mathematical description of dot-diffusion

We have defined the dot diffusion process in Section 2.2, but we also want to give a mathematical description of the dot diffusion. With the aid of this description, we can relate the quantizer error to halftone error. In addition to providing further insight, this will also be useful in Section 2.7 for inverse halftoning.

Let us denote the number of classes by L. For example, if the class matrix is 8×8 as in Section 2.2, then $L = 64$. Let $\mathbf{x}_k$ denote a vector whose elements are the pixels of the original contone image belonging to class k in some order. Let $\mathbf{x}$ denote a vector whose elements are the pixels, in some order, of the contone image. For example,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_L \end{bmatrix}.$$

Each of the vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L$, can be regarded as a polyphase component [59] of the contone image.

2.5.1 Quantizer error q and halftone error e

In the dot diffusion process, the pixels which are quantized by the two-level quantizer are modified versions $\mathbf{y}_i$ of the original vectors $\mathbf{x}_i$, the modification being that we diffuse the quantization errors from lower classes processed earlier. Since the pixels in class 1 are quantized directly, we have

$$\mathbf{y}_1 = \mathbf{x}_1.$$

Let $\mathbf{h}_1$ denote the halftone vector obtained from quantizing this to two levels. The quantizer error $\mathbf{q}_1 = \mathbf{y}_1 - \mathbf{h}_1$ is then diffused to those neighbors of the pixels of $\mathbf{x}_1$, which have a higher class number. For example, $\mathbf{x}_2$ is replaced with

$$\mathbf{y}_2 = \mathbf{x}_2 + \mathbf{D}_{21}(\mathbf{y}_1 - \mathbf{h}_1)$$

where $\mathbf{D}_{21}$ is a matrix representing the diffusion coefficients (i.e., quantities like $2/w$ and $1/w$ in equations (2.2(a)) and (2.2(b)). We then quantize $\mathbf{y}_2$ with the two level quantizer to produce the halftone $\mathbf{h}_2$ for all the pixels in class 2. The quantizer error $\mathbf{q}_2 = \mathbf{y}_2 - \mathbf{h}_2$ is then diffused to the higher class pixels. For example, consider class 3 pixels. In general these pixels receive diffused error from $\mathbf{q}_1$ and $\mathbf{q}_2$ so that the modified class 3 pixels are represented by the vector

$$\mathbf{y}_3 = \mathbf{x}_3 + \mathbf{D}_{31}(\mathbf{y}_1 - \mathbf{h}_1) + \mathbf{D}_{32}(\mathbf{y}_2 - \mathbf{h}_2).$$

Two-level quantization of $\mathbf{y}_3$ then produces the halftone $\mathbf{h}_3$ for class 3 pixel positions, and so forth. Thus, in general, the class vector $\mathbf{x}_k$ is modified to

$$\mathbf{y}_k = \mathbf{x}_k + \mathbf{D}_{k1}(\mathbf{y}_1 - \mathbf{h}_1) + \mathbf{D}_{k2}(\mathbf{y}_2 - \mathbf{h}_2) + \ \ldots \ + \mathbf{D}_{k,k-1}(\mathbf{y}_{k-1} - \mathbf{h}_{k-1}) \qquad (2.6)$$

and then quantized to obtain the halftone $\mathbf{h}_k$. Proceeding in this way, the halftone pixels $\mathbf{h}_k$ for all classes $1 \leq k \leq L$ are generated. The quantizer error vector $\mathbf{q}_k$ and halftone error vector $\mathbf{e}_k$ for class k are given by

$$\begin{aligned}\mathbf{q}_k &= \mathbf{y}_k - \mathbf{h}_k \qquad \text{(quantizer error)}.\\ \mathbf{e}_k &= \mathbf{x}_k - \mathbf{h}_k \qquad \text{(halftoning error)}.\end{aligned}$$

Subtracting $\mathbf{h}_k$ from both sides of (2.6) we get $\mathbf{q}_k = \mathbf{e}_k + \mathbf{D}_{k1}\mathbf{q}_1 + \mathbf{D}_{k2}\mathbf{q}_2 + \ldots + \mathbf{D}_{k,k-1}\mathbf{q}_{k-1}$, that is,

$$\begin{aligned}\mathbf{q}_1 &= \mathbf{e}_1.\\ \mathbf{q}_2 &= \mathbf{e}_2 + \mathbf{D}_{21}\mathbf{q}_1.\\ \mathbf{q}_3 &= \mathbf{e}_3 + \mathbf{D}_{31}\mathbf{q}_1 + \mathbf{D}_{32}\mathbf{q}_2.\\ &\vdots \qquad\qquad (2.7)\end{aligned}$$

By starting from the first equation, we can sequentially replace $\mathbf{q}_i$ in terms of $\mathbf{e}_i, \mathbf{e}_{i-1} \ldots$, on the right side of Eq. (2.7), resulting in an expression of the form

$$\begin{bmatrix}\mathbf{q}_1\\ \mathbf{q}_2\\ \vdots\\ \mathbf{q}_L\end{bmatrix} = \mathbf{A}_L \begin{bmatrix}\mathbf{e}_1\\ \mathbf{e}_2\\ \vdots\\ \mathbf{e}_L\end{bmatrix}$$

where $\mathbf{A}_L$ is a matrix depending on the elements of the smaller matrices $\mathbf{D}_{ij}$. We now show that $\mathbf{A}_L$ can be generated from $\mathbf{A}_{L-1}$ as follows: from Eq. (2.7) we can express $\mathbf{q}_L$ as

$$\begin{aligned}\mathbf{q}_L &= [\mathbf{D}_{L1} \quad \mathbf{D}_{L2} \quad \ldots \quad \mathbf{D}_{L,L-1}] \begin{bmatrix}\mathbf{q}_1\\ \mathbf{q}_2\\ \vdots\\ \mathbf{q}_{L-1}\end{bmatrix} + \mathbf{e}_L\\ &= [\mathbf{D}_{L1} \quad \mathbf{D}_{L2} \quad \ldots \quad \mathbf{D}_{L,L-1}]\, \mathbf{A}_{L-1} \begin{bmatrix}\mathbf{e}_1\\ \mathbf{e}_2\\ \vdots\\ \mathbf{e}_{L-1}\end{bmatrix} + \mathbf{e}_L\end{aligned}$$

$$= [\mathbf{D}_{L1} \quad \mathbf{D}_{L2} \quad \dots \quad \mathbf{D}_{L,L-1} \quad \mathbf{I}] \begin{bmatrix} \mathbf{A}_{L-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_{L-1} \\ \mathbf{e}_L \end{bmatrix}$$

which shows that

$$\underbrace{\begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_L \end{bmatrix}}_{\mathbf{q}} = \underbrace{\begin{bmatrix} & & \mathbf{I} & & \mathbf{0} \\ & & & & \\ \mathbf{D}_{L1} & \mathbf{D}_{L2} & \dots & \mathbf{D}_{L,L-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{L-1} & \mathbf{0} \\ & \\ \mathbf{0} & \mathbf{I} \end{bmatrix}}_{\mathbf{A}_L} \underbrace{\begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_L \end{bmatrix}}_{\mathbf{e}}$$

that is,

$$\mathbf{q} = \mathbf{A}_L \mathbf{e}.$$

Similarly $\mathbf{A}_{L-1}$ can be expressed in terms of $\mathbf{A}_{L-2}$, and so forth. This gives an expression for $\mathbf{A}_L$ as a product of simple matrices, that is, $\mathbf{A}_L = \mathbf{B}_L \mathbf{B}_{L-1} \dots \mathbf{B}_2 \mathbf{B}_1$, where $\mathbf{B}_k$ represents diffusion of error to class k from all lower classes, and $\mathbf{B}_1 = \mathbf{I}$. For example

$$\mathbf{A}_4 = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{D}_{41} & \mathbf{D}_{42} & \mathbf{D}_{43} & \mathbf{I} \end{bmatrix}}_{\mathbf{B}_4} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_{31} & \mathbf{D}_{32} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}}_{\mathbf{B}_3} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_{21} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}}_{\mathbf{B}_2}. \tag{2.8}$$

The matrix $\mathbf{A}_L$ has determinant equal to unity as seen from the factored expression (2.8). It is therefore invertible, and we can obtain

$$\mathbf{e} = \mathbf{T}_L \mathbf{q}$$

where $\mathbf{T}_L = \mathbf{A}_L^{-1}$. Here $\mathbf{T}_L$ can be regarded as the transfer function from the quantizer error $\mathbf{q}$ to the actual halftoning error. The total halftoning error-squared, defined as $\mathbf{e}^T\mathbf{e}$, can readily be computed from this if we know the quantizer error $\mathbf{q}$.

2.5.2 Expression for diffused image

Here is a summary of the main points of the preceding discussions. The **original contone image** is made from pixels in the vectors $\mathbf{x}_k$. The **diffused image** $\mathbf{y}$ is made from the pixels in $\mathbf{y}_k$ which are inputs to the two level quantizer. The image $\mathbf{h}$ whose pixels come from $\mathbf{h}_k$ is the **halftone image**. The pixels from the original contone, diffused, and halftone images can be arranged in the form of vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{h}$ as

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_L \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_L \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_L \end{bmatrix}.$$

The quantizer error vector $\mathbf{q}$ and halftone error vector $\mathbf{e}$ are defined as

$$\mathbf{q} = \mathbf{y} - \mathbf{h}, \qquad \mathbf{e} = \mathbf{x} - \mathbf{h}.$$

The diffusion process is schematically depicted in Fig. 2.18. We can now express the diffused image $\mathbf{y}$ in terms of the original contone image $\mathbf{x}$ and the halftone image $\mathbf{h}$ as follows: $\mathbf{y} = \mathbf{q} + \mathbf{h} = \mathbf{A}_L\mathbf{e} + \mathbf{h} = \mathbf{A}_L(\mathbf{x} - \mathbf{h}) + \mathbf{h}$, that is,

$$\mathbf{y} = \mathbf{A}_L\mathbf{x} + (\mathbf{I} - \mathbf{A}_L)\mathbf{h}. \tag{2.9}$$

This expression allows us to characterize the so-called inverse halftone set in a nice way. Let y_i and h_i denote, respectively, the ith scalar component of $\mathbf{y}$ and $\mathbf{h}$. Since y_i is directly quantized to yield h_i, we see that

$$y_i \begin{cases} \geq 0.5 & \text{if } h_i = 1 \\ \\ < 0.5 & \text{if } h_i = 0 \end{cases}. \tag{2.10}$$

Given a halftone image $\mathbf{h}$ and the halftone algorithm (e.g., dot diffusion with class matrix (as in Table 2)), the **inverse halftone set** $\mathcal{C}$ is the collection of all image vectors $\mathbf{x}$ which yield the halftone image $\mathbf{h}$. That is, an image $\mathbf{x}$ belongs to $\mathcal{C}$ if and only if the vector $\mathbf{y}$ computed using (2.9) satisfies Eq. (2.10).

Notice that if we set $\mathbf{x} = \mathbf{h}$ in Eq. (2.9), then we get $\mathbf{y} = \mathbf{h}$ which implies in particular

that (2.10) holds. Thus the halftone image $\mathbf{h}$ itself is a member of $\mathcal{C}$. That is, if we perform dot diffusion again on the halftone image $\mathbf{h}$, the result is still $\mathbf{h}$.

2.5.3 A closed convex subset of the inverse halftone set

We will see now that the inverse halftone set $\mathcal{C}$ is convex but not closed. We then show how to construct a subset $\mathcal{S}_1 \subset \mathcal{C}$ which is closed and convex. This will be useful Section 2.7 where we create a contone image from the dot-diffused halftone using the POCS method. For convenience of discussion let us renumber the elements of the halftone $\mathbf{h}$ such that it can be partitioned as

$$\mathbf{h} = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \qquad \text{where} \qquad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

The elements of $\mathbf{x}, \mathbf{y}$ and the matrix $\mathbf{A}_L$ are also renumbered accordingly. Then the diffused image vector $\mathbf{y}$ has the form

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_a \\ \mathbf{y}_b \end{bmatrix} = \begin{bmatrix} \mathbf{A}_a \\ \mathbf{A}_b \end{bmatrix} \mathbf{x} + \mathbf{c}$$

where $\mathbf{A}_a, \mathbf{A}_b$ and $\mathbf{c}$ do not depend on $\mathbf{x}$ or $\mathbf{y}$.

The diffused image vector $\mathbf{y}$ is such that $\mathbf{y}_a \geq 0.5 \times \mathbf{1}$ and $\mathbf{y}_b < 0.5 \times \mathbf{1}$ where the vector inequalities in the previous equation should be interpreted on an element by element basis. That is, the inverse halftone set $\mathcal{C}$ is the set of all image vectors $\mathbf{x}$ such that

$$\mathbf{A}_a\mathbf{x} \geq \mathbf{d}_a \qquad \text{and} \qquad \mathbf{A}_b\mathbf{x} < \mathbf{d}_b. \tag{2.11}$$

Given two image vectors $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ satisfying (2.11), we can readily verify that the linear combination $\alpha\mathbf{x}^{(1)} + (1-\alpha)\mathbf{x}^{(2)}$ also satisfies Eq. (2.11) whenever $0 \leq \alpha \leq 1$. This shows that the set $\mathcal{C}$ is convex.

The set of all $\mathbf{x}$ satisfying $\mathbf{A}_a\mathbf{x} \geq \mathbf{d}$ is interpreted geometrically as in Fig. 2.19 for the case where $\mathbf{x}$ is a two-dimensional vector. Since the boundary $\mathbf{A}_b\mathbf{x} = \mathbf{d}$ is included, this is a closed set [19]. The set $\mathbf{A}_b\mathbf{x} < \mathbf{d}$ has a similar interpretation, but since the boundary is not included, it is not closed. The intersection of the two sets described by $\mathbf{A}_a\mathbf{x} \geq \mathbf{d}_a$ and

$\mathbf{A}_b\mathbf{x} < \mathbf{d}_b$ is therefore not closed. Summarizing, *the inverse halftone set C for a dot-diffused halftone* **h** *is a convex set but it is not closed.*

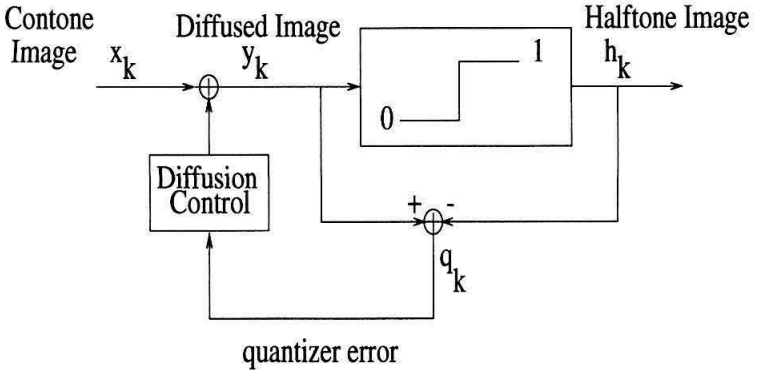

Figure 2.18: Schematic representation of the dot diffusion process. Here x_k represents a vector of all pixels belonging to class k.

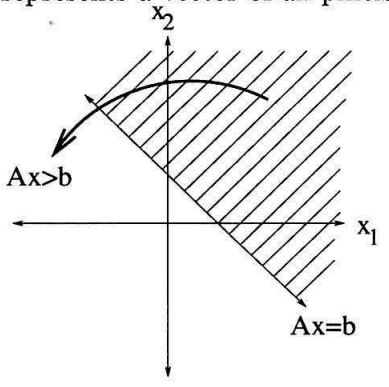

Figure 2.19: Ax $\geq$ b is a closed set.

2.5.4 The digitized subset

Now consider a subset $\mathcal{D} \subset \mathcal{C}$ such that all images in $\mathcal{D}$ are digitized to, say, 8 bits/pixel. The set $\mathcal{D}$ is clearly not empty because the halftone image **h** is certainly a member of $\mathcal{D}$. With **x** chosen from this digitized subset $\mathcal{D}$, the elements y_i of **y** also take values from a discrete set. So we can always find an $\epsilon > 0$ such that none of the y_i's fall in the open interval $(0.5 - \epsilon, 0.5)$. That is, not only is Eq. (2.10) satisfied, but the following stronger condition holds:

$$y_i \begin{cases} \geq 0.5 & \text{if } h_i = 1 \\ \\ \leq 0.5 - \epsilon & \text{if } h_i = 0 \end{cases} . \tag{2.12}$$

for some fixed $\epsilon > 0$ that can be precalculated from $\mathbf{A}$ and $\mathbf{h}$. By following the kind of reasoning that resulted in Eq. (2.11), we see that if $\mathbf{x}$ is in the digitized subset $\mathcal{D}$, then

$$\mathbf{A}_a\mathbf{x} \geq \mathbf{d}_a \qquad \text{and} \qquad \mathbf{A}_b\mathbf{x} \leq \mathbf{d}_b, \tag{2.13}$$

where the vector $\mathbf{d}_b$ now depends on ϵ as well. Notice that the strict inequality $<$ of (2.11) has now been replaced with $\leq$.

2.5.5 The closed convex subset

We have just shown that every element of the digitized set $\mathcal{D}$ satisfies Eq. (2.13). The set $\mathcal{D}$ is trivially "closed" because it is finite [p. 15, [7]]. However, $\mathcal{D}$ is evidently not convex because a linear combination $\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2$ of 8 bit images $\mathbf{x}_1$ and $\mathbf{x}_2$ is not an 8 bit image for arbitrary α in $0 \leq \alpha \leq 1$. Now consider a set $\mathcal{S}_1$ that is bigger than $\mathcal{D}$ by defining it to be the *set of all image vectors* $\mathbf{x}$ for which Eq. (2.13) holds, or equivalently, Eq. (2.12) holds. By slight modification of the arguments at the end of Section 4.4 we see that this set is both closed and convex. Since Eq. (2.12) holds, it is also clear that Eq. (2.10) holds which shows that $\mathbf{x}$ continue to belong in the inverse halftone set $\mathcal{C}$. Summarizing, we have three sets $\mathcal{D}, \mathcal{S}_1$ and $\mathcal{C}$ with the inclusion relationship

$$\mathcal{D} \subset \mathcal{S}_1 \subset \mathcal{C}.$$

$\mathcal{C}$ is the set of all image vectors which result in the given halftone image $\mathbf{h}$ using the given dot diffusion algorithm. The set $\mathcal{C}$ is convex but not closed. The digitized subset $\mathcal{D}$ is closed but not convex. The intermediate set $\mathcal{S}_1$ is closed and convex. Notice finally that the closed convex set $\mathcal{S}_1$ described by Eq. (2.13) can be described more compactly as

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}$$

where $\mathbf{A} = \begin{bmatrix} -\mathbf{A}_a \\ \mathbf{A}_b \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} -\mathbf{d}_a \\ \mathbf{d}_b \end{bmatrix}$. Note that the above vector-inequality is interpreted componentwise.

These ideas will be useful in Section 2.7 where we apply the iterative POCS technique to derive an approximation of the original contone image from a halftone. Assuming that an 8-bit image is a good approximation of the original contone, the distinction between

the three sets is minor. However the fact that we can work with the closed convex set $\mathcal{S}_1$ without much loss of generality is significant as we shall see in Section 2.7. It allows us to assume that the method of POCS converges to a good approximation of the contone image.

2.6 Inverse halftoning

In Section 2.7 we show how the POCS method can be applied for the inverse halftoning of a dot-diffused halftone. The mathematical characterization of dot-diffused images developed in Section 2.5 will be especially useful to construct the so-called space-domain constraint set, which is a key ingredient in the development of the algorithm. Even though the wavelet method gives better results, inverse halftoning using POCS method is added because of its elegance and generality. In Section 2.8 we show how the wavelet method for inverse halftoning [64] can be modified for the case of dot-diffused images. While the POCS method often produces better PSNR, the wavelet method typically yields a more pleasing visual quality.

2.7 Inverse halftoning using POCS

The method of POCS, which is an acronym for *Projection Onto Convex Sets*, is a powerful algorithm for the approximate recovery of a signal from partial information. It has been used very widely in many applications, as elaborated in authoritative references [55],[65]. To explain the idea in its simplest form, assume that the unknown signal is known a-priori to belong to the intersection of two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. Assume these are closed convex sets (see below). Then starting from an arbitrary initial guess for the signal and performing successive projections onto these two sets, we can converge to a point in the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$. Even though this intersection may have many elements and therefore the original signal not exactly recoverable, careful choice of $\mathcal{S}_1$ and $\mathcal{S}_2$ often leads to satisfactory results.

We will first state the POCS method and the associated convergence theorem more precisely. After this we describe how the method can be applied for recovering a continuous tone image from its halftoned version. We will see that the specific details of the convex set $\mathcal{S}_1$ depend on the details of the dot diffusion procedure and the class matrix (Section 2.7.4). Finally in Section 2.7.5 we will show experimental results.

2.7.1 Mathematical background

The mathematical setting for the POCS method is the following. Let the unknown signal $\mathbf{f}$ be a vector in a Hilbert space H, e.g., ℓ_2 space of images. For example, it could be a vector constructed from some arrangement of the pixels in a contone image. In view of the physical constraints that we happen to know, let us assume that $\mathbf{f}$ is in the intersections of known subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_m$ in H. (These may not be subspaces.) Assume that each of these is a **closed convex set**[8] and that their intersection $\mathcal{S}_{int}$ is nonempty. Let P_i be the **projection operator** from H to $\mathcal{S}_i$. The projection $\widehat{\mathbf{x}}$ of $\mathbf{x}$ onto $\mathcal{S}_i$ is defined to be the unique vector in $\mathcal{S}_i$ such that the error norm $\|\mathbf{x} - \widehat{\mathbf{x}}\|$ is minimized [[55], Section 2.2]. Define the composite operator

$$P \stackrel{\text{def}}{=} P_m P_{m-1} \ldots P_1$$

and consider the iteration $\mathbf{f_k} = P\mathbf{f}_{k-1}$, with initial vector $\mathbf{f}_0 \in \mathcal{H}$. Then, according to the **POCS theorem** (Theorem 2.4-1 in [55]), the vector $\mathbf{f}_k$ converges weakly[9] to some vector $\mathbf{f}_{lim}$ in the intersection $\mathcal{S}_{int}$. This result is true regardless of the initial vector $\mathbf{f}_0$, even though the limit $\mathbf{f}_{lim}$ can depend on $\mathbf{f}_0$. If this limit is an acceptable approximation of $\mathbf{f}$, then we are happy.

The convergence result continues to be true even if the projection operators P_i are replaced with so-called *relaxed projections* T_i [55]. These are defined as $T_i = 1 + \lambda_i(P_i - 1)$ where $0 < \lambda_i < 2$. We shall not require this stronger version.

2.7.2 Application of the POCS theorem for inverse halftoning

Consider applying the above result for the problem of inverse halftoning. The contone image $\mathbf{x}$ is halftoned with a known algorithm (e.g., dot diffusion with known class matrix), to yield a halftone $\mathbf{h}$. From this $\mathbf{h}$, and using our knowledge of the halftoning process, we have to construct a contone approximation $\mathbf{x}_{approx}$ subject to two conditions, namely (i) if it is halftoned, the result is again $\mathbf{h}$, and (ii) $\mathbf{x}_{approx}$ should be an "acceptable" approximation of $\mathbf{x}$.

[8] A set $\mathcal{S}$ is said to be convex if $\alpha\mathbf{f} + (1-\alpha)\mathbf{g}$ belongs to $\mathcal{S}$ for any α such that $0 \leq \alpha \leq 1$, whenever $\mathbf{f}, \mathbf{g}$ are in $\mathcal{S}$. A set $\mathcal{S}$ equipped with a metric (i.e., any measure of distance) is said to be closed if the limit of any convergent subsequence $\{\mathbf{f}_i\}$ in S also belongs to $\mathcal{S}$.

[9] The term "weakly" means that the inner product $\langle \mathbf{f}, \mathbf{f}_k \rangle$ converges to $\langle \mathbf{f}, \mathbf{f}_{lim} \rangle$ for any $\mathbf{f}$ in H. This is weaker than the requirement that $\mathbf{f}_k$ converges to $\mathbf{f}_{lim}$, which would mean that $\|\mathbf{f}_k - \mathbf{f}_{lim}\|$ goes to zero.

The first condition alone can be satisfied by many images. Let $\mathcal{S}_1$ be the set of all images such that the given halftoning algorithm yields the fixed halftone $\mathbf{h}$. The original contone image $\mathbf{x}$ evidently belongs to $\mathcal{S}_1$. Moreover, using the description of dot diffusion process in Section 4, it can be shown that the halftone $\mathbf{h}$ itself belongs to $\mathcal{S}_1$. We say that $\mathcal{S}_1$ is the **space domain constraint set.** For the second condition we have to define a set $\mathcal{S}_2$ which represents the set of "acceptable images" in some sense. For example, $\mathcal{S}_2$ could represent "natural images" which have certain smoothness properties. Since $\mathcal{S}_2$ is usually constructed with the help of lowpass operators (see below), it will be called the **frequency domain constraint set.** In the notation of Section 2.5, the parent Hilbert space H is ℓ_2, and $\mathcal{S}_1$ and $\mathcal{S}_2$ are the two subsets. If these are closed and convex, then we can start from an arbitrary initial image $\mathbf{f}_0$ in ℓ_2 and perform the projections

$$\mathbf{g}_k = P_1\mathbf{f}_{k-1} \qquad \text{(space-domain projection)},$$

$$\mathbf{f}_k = P_2\mathbf{g}_k \qquad \text{(frequency-domain projection)}.$$

That is, $\mathbf{f}_k = P_2P_1\mathbf{f}_{k-1}$. According to the POCS theorem this iteration converges to a member in the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$. If we are willing to accept any member in the intersection to be a valid approximation of the contone $\mathbf{x}$, then we are done.

The POCS algorithm has in the past been applied for inverse halftoning [22]. In the actual algorithm we have to identify the projection operators P_1 and P_2 which take an arbitrary image in ℓ_2 and project onto sets $\mathcal{S}_1$ and $\mathcal{S}_2$. For our application we already showed, using the mathematics of the dot diffusion algorithm (Section 2.5), that the set $\mathcal{S}_1$ is a closed convex set. The second point of novelty is with respect to the projection operator P_2. In the past, lowpass filtering has been used [Zakhor and Hein] as an approximation for P_2, the rationale being that many natural images are lowpass. But unfortunately LTI filtering is not a projection operator, that is $H^2(e^{j\omega}) \neq H(e^{j\omega})$, unless $H(e^{j\omega})$ is an ideal filter with passband response of unity and stopband response of zero. In [22] the authors use partial reconstructions from DCT and SVD (singular value decomposition) as other possible choices for the projection operator. In this chapter we use an operator which is not only an orthogonal projection but retains the properties of a good lowpass filter; this projection is constructed from an orthonormal multirate filter bank.

2.7.3 Implementation of the frequency domain projection

Consider Fig. 2.20 which shows the synthesis section of an M-band uniform orthonormal filter bank [59] with the perfect reconstruction property $y(n) = x(n)$. Note that the subbands $x_0(n), \ldots x_{L-1}(n)$ are obtained by feeding $x(n)$ to the analysis section (not shown) of the same filter bank. Suppose we delete the subband signals $x_L(n), \ldots x_{M-1}(n)$ and perform the synthesis by retaining only the subbands $x_0(n), \ldots x_{L-1}(n)$. Then the reconstruction $y(n) \neq x(n)$ in general, and $y(n)$ is called a partial reconstruction. If the filter bank has the orthonormal property [59],[60],[56], then we can give an orthogonal projection interpretation for this partial reconstruction. Thus, assume that the input $x(n)$ is in ℓ_2. Let $S \subset \ell_2$ be the subspace formed by the basis functions

$$\eta_{km}(n) = f_k(n - Mm), \quad L \leq k \leq M-1, \; -\infty \leq m \leq \infty$$

from the deleted channels. Then the partial reconstruction $y(n)$ belongs to $S^{\perp}$, the orthogonal complement of S. Since orthogonal complements are closed subspaces [33], and subspaces are automatically convex, it follows that $y(n)$ is the projection of $x(n)$ onto the closed convex set $S^{\perp}$. We can take the frequency domain constraint set to be

$$\mathcal{S}_2 = S^{\perp}.$$

As explained in Section 2.7.2, in order to implement the POCS method we have to know how to project an arbitrary intermediate image onto the closed convex set $\mathcal{S}_2$. It is clear that this can be done by decomposing the image into subbands using an orthonormal filter bank, and partially reconstructing it as above.

Figure 2.22 shows the actual two-dimensional filter bank used in our work for this frequency domain projection. Here $H_0(z)$ and $H_1(z)$ are one-dimensional filters, so the filter bank has separable two-dimensional analysis filters [59]. The notation $\downarrow (2,1)$ means decimation by two in the horizontal direction and no decimation in the vertical direction. The notation $\uparrow (2,1)$ similarly stands for the separable expander. With $H_0(z)$ and $H_1(z)$ denoting a lowpass/highpass pair, the signal $s_{00}(n_0, n_1)$ is the low-low subband. If $y(n_0, n_1)$ is reconstructed using this subband alone, then we can regard it as a "multirate" lowpass version, which at the same time is an orthogonal projection in the mathematical sense. In

our work we actually used Daubechies' 10-tap FIR filter [13] for the lowpass filter $H_0(z)$. The highpass filter $H_1(z)$ was chosen in the usual way [59] to obtain the orthonormal filter bank.

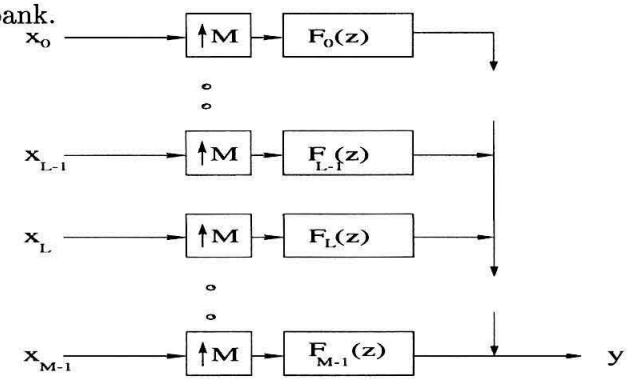

Figure 2.20: Synthesis section of an M channel filter bank.

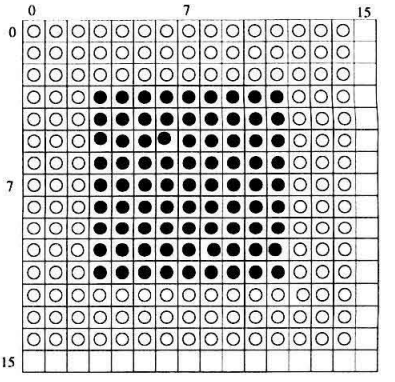

Figure 2.21: Overlapping blocks used in approximating the QP problem.

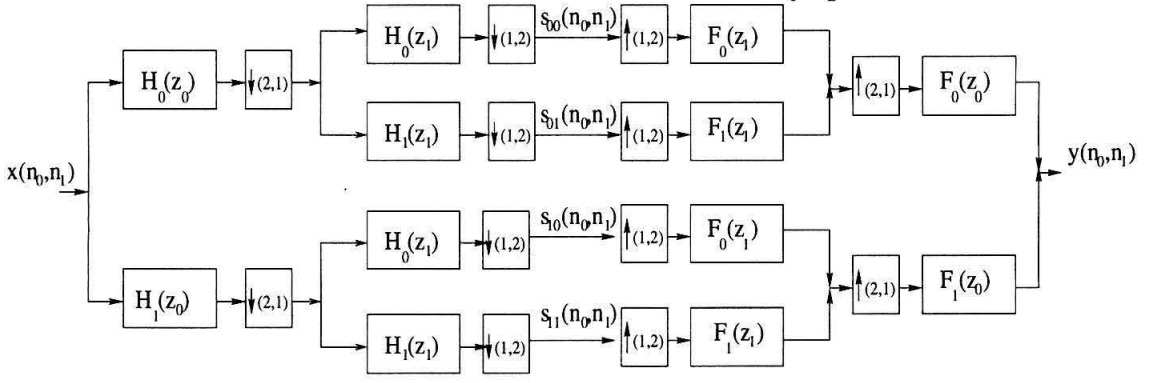

Figure 2.22: Two-dimensional separable PR filter bank.

2.7.4 Implementation of space domain projection

The space domain constraint on the inverse halftone is that it should lie in the closed convex set $\mathcal{S}_1$ defined in Section 2.5. This is essentially the set of all contone images which can give rise to the given halftone $\mathbf{h}$. As explained in Section 2.7.2, in order to implement the POCS method we have to know how to project an arbitrary intermediate image vector $\mathbf{v} \in \ell_2$ onto the closed convex set $\mathcal{S}_1$. The meaning of a projection was reviewed in Section 2.7.1: The projection $\hat{\mathbf{v}}$ of $\mathbf{v}$ onto $\mathcal{S}_1$ is the unique vector in $\mathcal{S}_1$ such that the error norm $\|\mathbf{v} - \hat{\mathbf{v}}\|$ is minimized. Here the notation $\|\mathbf{e}\|$ represents the ℓ_2 norm $\sqrt{\mathbf{e}^\dagger \mathbf{e}}$. In order to implement this projection, we simply solve a minimization problem subject to the constraint $\hat{\mathbf{v}} \in \mathcal{S}_1$. Thus, the projection $\hat{\mathbf{v}}$ of the image $\mathbf{v}$ onto the convex set $\mathcal{S}_1$ is the solution to the following constrained optimization problem:

$$\min_{\widehat{\mathbf{v}}} \|\mathbf{v} - \widehat{\mathbf{v}}\|^2 \qquad \text{subject to} \qquad \mathbf{A}\widehat{\mathbf{v}} \leq \mathbf{b}. \tag{2.14}$$

This follows because the elements $\widehat{\mathbf{v}}$ of the set $\mathcal{S}_1$ are completely characterized by the property $\mathbf{A}\widehat{\mathbf{v}} \leq \mathbf{b}$. This is a quadratic programming (QP) problem and can be solved using standard techniques. We used the Matlab optimization toolbox to solve for $\widehat{\mathbf{v}}$. In the interest of efficient programming, the QP problem was broken into several subproblems by partitioning the image into blocks. For this, overlapping blocks are used. In Fig. 2.21, the blocks used are shown. The black circles show the pixel positions that are changed after solving the QP subproblem, and white circles show the pixels used for boundary conditions of black circles, but they are not changed after solving this QP subproblem. Afterwards, the block location is moved 8 pixels to right or left, till the whole image is covered. The sizes of QP subproblems are 9×9. A further detail in the implementation is that the matrix $\mathbf{A}$ in the constraint equation must be modified to take into account the fact that the original image $\mathbf{x}$ is enhanced with a highpass filter before halftoning (as described in Section 2.2). Recall that the matrix $\mathbf{A}$ originated from the matrix $\mathbf{A}_L$ described in Section 2.5 where the key equation relating the original image $\mathbf{x}$ and the diffused image $\mathbf{y}$ was $\mathbf{y} = \mathbf{A}_L\mathbf{x} + (\mathbf{I} - \mathbf{A}_L)\mathbf{h}$. In this equation we have to replace $\mathbf{A}_L\mathbf{x}$ with $\mathbf{A}_L\mathbf{x}_e$ where $\mathbf{x}_e$ is the enhanced version of $\mathbf{x}$. The enhancing filter for $\alpha = 0.9$ (see Section 2.2) is the 2D filter

$$F_{enh}(z_0, z_1) = 10 - (z_0 + 1 + z_0^{-1})(z_1 + 1 + z_1^{-1})$$

and we can write $\mathbf{x}_e = \mathbf{E}\mathbf{x}$ where $\mathbf{E}$ is a square matrix (neglecting boundary details, such as lengthening of a signal due to filtering). The modified $\mathbf{A}$ matrix in the constraint $\mathbf{A}\widehat{\mathbf{v}} \leq \mathbf{b}$ in Eq. (2.14) can now be worked out.

2.7.5 Implementation details and experimental results

The frequency domain projection described above implicitly assumes that the original contone image is in the subset $\mathcal{S}_2$. Given an arbitrary image $x(n_0, n_1)$, we can replace it with its projection onto $\mathcal{S}_2$ before halftoning (i.e., compute the partial reconstruction $y(n_0, n_1)$ by using $s_{00}(n_0, n_1)$ alone, and then halftone $y(n_0, n_1)$). This **preconditioning** ensures that the desired inverse halftone is indeed in the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$. We found ex-

perimentally that for most natural images, the projection onto $\mathcal{S}_2$ is nearly as good as the original image, so such an initial conditioning is not a severe loss of information. Second, we found that in many examples the POCS algorithm converges to a good solution even without such preconditioning.

For the peppers and lena images, Figs. 2.23, 2.24, 2.25, 2.26 show the inverse halftoned images. In Figs. 2.23, 2.24, the original image was first projected onto the transform domain set $\mathcal{S}_2$ before halftoning. In Figs. 2.25, 2.26 this preconditioning was omitted. For completeness we mention the PSNR values for the reconstructed images. The PSNR values are as follows: peppers with preconditioning (PSNR=30.35dB with respect to original peppers and PSNR=32.39dB with respect to projection of peppers image onto $\mathcal{S}_2$), lena with preconditioning (PSNR=31.19dB with respect to original lena and PSNR=33.08dB with respect to projection of lena image onto $\mathcal{S}_2$), peppers without preconditioning (PSNR=29.44dB), and lena without preconditioning (PSNR=30.66dB). The images are obtained after 5 iterations.

2.8 Inverse halftoning using wavelets

An inverse halftoning algorithm which use wavelets was considered in [64] and [34]. In this method, wavelets are used to differentiate the halftoning noise from the edge information. The edges are detected at different scales with specific overcomplete wavelet transform. Since the edges are correlated at different scales whereas the noise is not, the halftoning noise is suppressed by thresholding operations wherever the edges are not prominent (These correspond to steps 2,3 in our inverse halftoning method). However, the algorithm in [64] is tailored for error diffusion, which has different characteristics than dot diffusion. If the method in [64] is used for dot diffusion, the result is not good. This can be seen from Fig. 2.30 which shows the result of inverse halftoning the dot diffused Lena by using the method in [64]. The image suffers from periodic patterns, which represent low frequency noise. There are basically two reasons for the inferior performance: The images are enhanced in dot diffusion before halftoning and there is more low frequency noise in dot diffusion.

In the new method, the specific properties of the dot diffusion algorithm are taken into account. The image is enhanced before dot diffusion, hence in the inverse halftoning, the dot diffused image should be de-enhanced using the inverse of the filter $F_{enh}(z_1, z_2) = 10 - (z_1 + 1 + z_1^{-1})(z_2 + 1 + z_2^{-1})$. Note that $F_{enh}(e^{jw_1}, e^{jw_2}) > 0$ for all $0 \leq w_1, w_2 \leq \pi$.

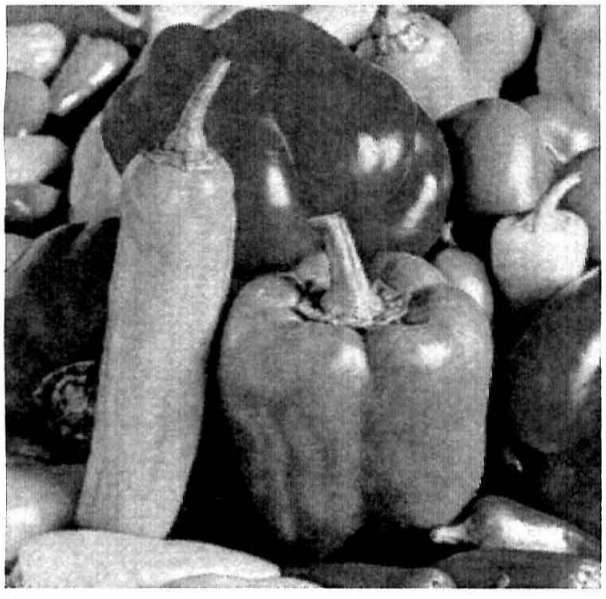

Figure 2.23: Inverse halftoned peppers with POCS (transform domain projection applied before halftoning).

Figure 2.24: Inverse halftoned Lena with POCS (transform domain projection applied before halftoning).

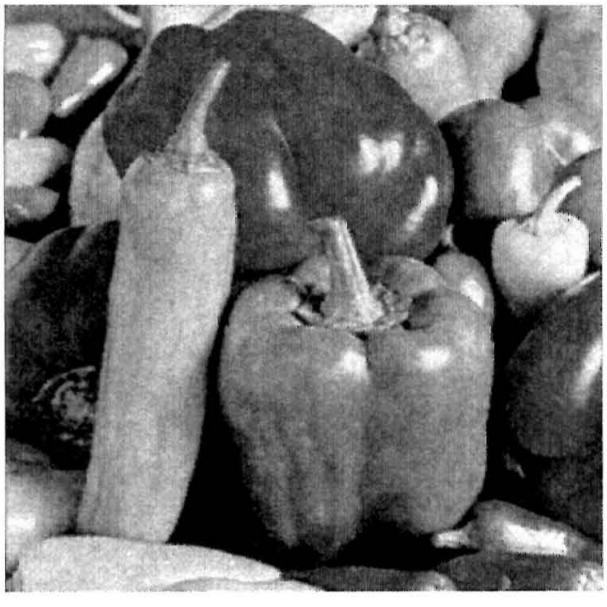

Figure 2.25: Inverse halftoned peppers with POCS (transform domain projection is not applied before halftoning).

Figure 2.26: Inverse halftoned Lena with POCS (transform domain projection is not applied before halftoning).

We use the wavelet tree built from the analysis block shown in Fig. 2.27. An image $X(x,y)$ is decomposed into $L(x,y)$, $H(x,y)$, and $V(x,y)$ using the undecimated wavelet transform. At scale 2^{i+1} (which will be described below) the filtering operations are as follows:

$$L(w_1,w_2) = F(2^i w_1)F(2^i w_2)X(w_1,w_2),$$

$$H(w_1,w_2) = G(2^i w_1)F(2^i w_2)X(w_1,w_2),$$

$$V(w_1,w_2) = F(2^i w_1)G(2^i w_2)X(w_1,w_2),$$

where G and F are derived from quadratic spline wavelets. These are tabulated along with the synthesis filters in Table 1 of [35] (our F is H in that table). The choice of filters given in [35] detect edges at different scales if they are used in the wavelet tree shown in Fig. 2.28 with scales 2^0,2^1,2^2,2^3 from left to right. For example, $H_i(x,y)$ and $V_i(x,y)$ represent the horizontal edges, and vertical edges of $L_{i-1}(x,y)$ at scale 2^{i-1} respectively, and $L_i(x,y)$ is the low pass version of $L_{i-1}(x,y)$.

The algorithm starts with a dot diffused image, $h(x,y)$. Then $h(x,y)$ is de-enhanced with the de-enhancement filter specified above. Let us call the resulting image $L_0(x,y)$. Afterwards, a 4-level wavelet decomposition is applied to $L_0(x,y)$. Then for each pixel location (x,y), the following is done:

1) Apply a symmetric FIR Gaussian filter, $f_g(n,m)$ to $V_1(x,y)$, and $H_1(x,y)$. ($f_g(n,m) = ce^{-\frac{n^2+m^2}{2\sigma^2}}$ for $-3 \leq n,m \leq 3$, and c is chosen such that the DC gain of the filter is unity). The first level edge images contain mostly the halftoning noise, thus low pass filtering these images reduces the blue noise without harming the edges significantly.

2) Let $E_{23}(x,y) = V_2(x,y)V_3(x,y)+H_2(x,y)H_3(x,y)$. If $E_{23}(x,y) \leq T_1$ then make $V_2(x,y) = 0$ and $H_2(x,y) = 0$.

3) Let $E_{34}(x,y) = V_3(x,y)V_4(x,y)+H_3(x,y)H_4(x,y)$. If $E_{34}(x,y) \leq T_2$ then make $V_3(x,y) = 0$ and $H_3(x,y) = 0$.

Steps 2 and 3 are the denoising steps in the algorithm where T_1 and T_2 are the thresholds determined experimentally. In order to discriminate the edges from the halftoning noise, we have to locate the edges. For this, the above steps perform a cross correlation between the edges at different scales. If there is a horizontal edge at scale i at (x,y), then $H_i(x,y)$ and $H_{i+1}(x,y)$ will be of the same sign [35]. The same is also true for vertical edges. Combining the horizontal and vertical edge correlations gives better results in detecting the diagonal

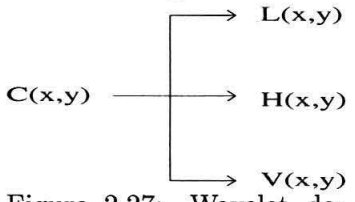

Figure 2.27: Wavelet decomposition of an image.

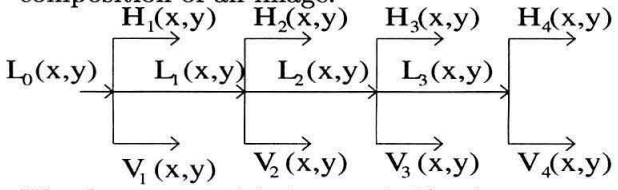

Figure 2.28: Wavelet tree used in inverse halftoning.

Figure 2.29: Result of simple deenhancement of dot diffused Lena image.

Figure 2.30: Result of inverse halftoning using an earlier method [27].

Figure 2.31: Inverse halftoned lena using the modified wavelet denoising method.

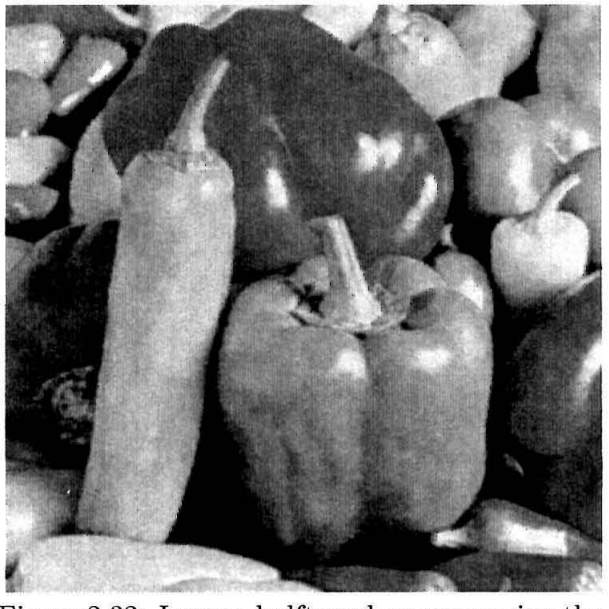

Figure 2.32: Inverse halftoned peppers using the modified wavelet denoising method.

edges.

4) The above steps have modified the subband signals L_i, H_i and V_i in certain ways. We now use the inverse filter bank (synthesis bank) corresponding to Fig. 2.28, and obtain a reconstructed version $\hat{L_0}(x,y)$. The image $\hat{L_0}(x,y)$ is the desired inverse halftone image.

In inverse halftoning, dot diffusion has an advantage, namely, even the simple de-enhanced image is a quite reasonable inverse halftone (PSNR=26.62dB for Lena image) (Fig. 2.29). The de-enhanced image is further processed as described above. The parameters used in the method are found experimentally. The variance of the Gaussian filter, σ^2, is chosen to be 0.5 and the thresholds are chosen to be $T_1 = 300$ and $T_2 = 20$. The results are shown in Fig. 2.31 (PSNR=30.58dB) and in Fig. 2.32 (PSNR=30.07dB).

Even though POCS gives higher PSNR values than the wavelet method, the wavelet method gives more pleasing results than the POCS. This is due to the space domain projection step in the POCS. Another advantage of the wavelet method is that, it is not iterative, whereas the POCS is inherently iterative. Thus the wavelet method is better than the POCS method for inverse halftoning. More recently a promising faster method has emerged for inverse halftoning of error diffused images [27]. We have not tried applying the algorithm for dot diffused images.

2.9 Embedded multiresolution dot diffusion

Another desired property of images is the embedded multiresolution property. If an image has embedded multiresolution property, the lower resolution images can be obtained from higher resolution images. Embedded images require less storage space, and embedding is also useful for progressive transmission.

As observed by [61], normal halftones do not have embedded multiresolution property. This can be seen from Fig. 2.33 where the 512×512 image is halftoned by dot diffusion and the lower resolution images are obtained by downsampling the higher resolution images by 2 in each direction. The lower resolution images are not good representations of the corresponding original images. But, the embedded multiresolution property can be imposed during the halftoning process. In [6], [29], and [61] this property is imposed on the halftone image as follows: First, the lowest resolution image is obtained using the halftoning algorithm. The higher resolution halftones are obtained from the lower resolution halftones, by

retaining the lower resolution image at the corresponding pixels, and halftoning the other pixels of the higher resolution. In [6] and [29], the halftoning method was an optimization-based one whereas in [61], the method was adaptive error diffusion. We will exploit the same idea, and we will show how to impose the embedded multiresolution property for dot diffused images.

Given a contone image $C_0(m,n)$ of size $M_0 \times N_0$, we want halftone images of smaller sizes $M_k \times N_k$, $k = 0,1,2,..,K$, to have a fair representation of the original image. We assume that there exist integers p_k and q_k such that $M_{k-1} = p_k M_k$ and $N_{k-1} = q_k N_k$ for $k = 1,2,..,K$. Then the halftoning algorithm will be as follows:

1) Obtain $C_K(m,n) = C(mp_1p_2...p_K, nq_1q_2...q_K)$ for $m = 1,2,..M_k$, $n = 1,2,..N_k$. Then initialize i as $K-1$.

2) Halftone $C_K(m,n)$, and let the resulting image be $h_K(m,n)$.

4) Define $C_i(m,n) = C(mp_0p_1...p_i, nq_0q_1...q_i)$ for $m = 1,2,..M_i$,$n = 1,2,..N_i$, where $p_0 = 1$, $q_0 = 1$.

5) For each pixel location (m,n) belonging to class i do

a) If $(m,n) = (ap_{i+1}, bq_{i+1})$ for some a,b integer, then $h_i(m,n) = h_{i+1}(m/p_{i+1}, n/q_{i+1})$ else

$$h_i(m,n) = \begin{cases} 1 & \text{if } C_i(m,n) \geq 0.5, \\ 0 & \text{if } C_i(m,n) < 0.5. \end{cases}$$

b) $e(m,n) = C_i(m,n) - h_i(m,n)$, and diffuse the error as in Eq. (2)

6) if $i = 0$, stop else let $i = i-1$ and go to step (3).

The multiresolution property is demonstrated in Fig. 2.34 where $K = 2$, $p_1 = p_2 = 2$, and $q_1 = q_2 = 2$. The lower resolution halftones are contained in higher resolution halftones and halftones at any resolution have a 'fair' representation of the original contone image. For the peppers image, the embedded multiresolution constraint did not affect the image quality at the highest resolution. However, there is a slight loss of quality in the highest resolution halftone image for the lena image because of the latter constraint. Thus, at the expense of a little quality loss, the embedded multiresolution property can be imposed on the dot diffusion method.

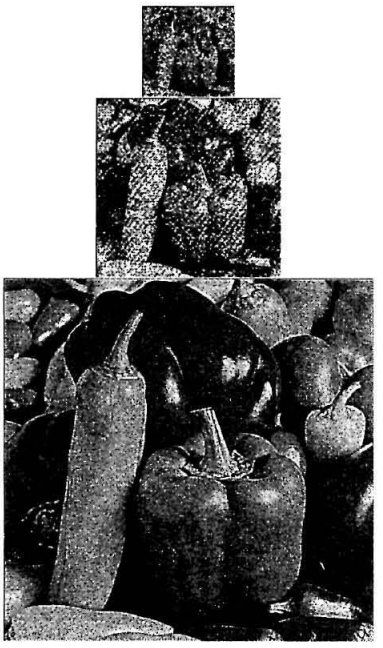

Figure 2.33: Dot diffused peppers image without embedded multiresolution property.

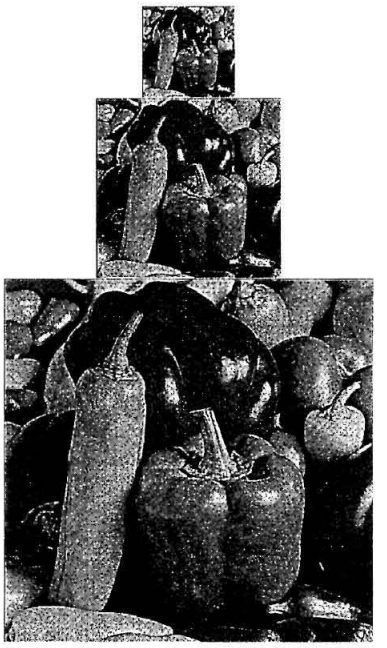

Figure 2.34: Dot diffused peppers image with embedded multiresolution property.

2.10 Concluding remarks

Even though dot diffusion offers more parallelism than error diffusion, it has not received much attention in the past. This is partly because the noise characteristics of error diffusion method are generally regarded as superior. Although the quality of dot diffused halftones is not as good as error diffusion, we have shown that it can be substantially improved over standard dot diffusion. With more optimization work, it should be possible to come even closer to error diffusion quality. Furthermore, the parallelism offered by dot diffusion is a great advantadge. The dot diffusion algorithm terminates in at most 64 steps for an 8×8 class matrix, compared to $M \times N$ steps needed for error diffusion algorithm for an $M \times N$ image. Moreover, as noticed in [36], the algorithm can in fact be terminated in about 50 steps. The conclusion is that Knuth's dot diffusion method with a carefully optimized class matrix is very promising; the image quality is comparable to error diffusion, and the implementation offers more parallelism than error diffusion. Since enhancement prior to halftoning can be objectionable in some cases, we also introduced and optimized 16×16 class matrix, which eliminated the need for enhancement. In this chapter, we first optimized the class matrix. Then a mathematical description of dot diffusion was

derived which was particularly useful in inverse halftoning. We also presented a wavelet-based inverse halftoning algorithm which works very well, even though the class matrix information is not used. Furthermore, we have shown that the dot diffusion algorithm can be easily modified to have the embedding property. This is useful for rendering at different resolution levels and for transmitting images progressively.

Chapter 3 Look up table (LUT) method for inverse halftoning

3.1 Introduction

In this chapter we introduce Look Up Table (LUT) methods for inverse halftoning of images.[1] The LUT method was first used by Netravali et al. to display dithered images [53]. However, in that work the authors assumed that the dither matrix and the registration of halftoned image with the dither matrix are known. Notice that this information may not be known for a particular halftone image. In another paper, Ting and Riskin used LUT method in halftone compression to get a temporary contone image [57]. However, obtaining high quality contone image was not the aim in that work.

Our LUT inverse halftoning method does not involve any linear filtering at all, and it does not explicitly assume any image model. To determine the inverse halftone value at a point, the algorithm looks at the pixel's neighborhood and, depending upon the distribution of pixels in the neighborhood, it assigns a contone value from a precomputed LUT. The number of pixels used in the neighborhood is relatively small, typically 16. For a 16 pixel neighborhood, we need $2^{16} = 64K$ bytes of LUT, and for a 19 pixel neighborhood, we need $2^{19} = 512K$ bytes of LUT.[2] The method is completely parallel and it does not require any computation because it is an LUT based algorithm. This makes it faster than the previously known methods. Moreover, the method provides very good image quality, often even better than the method in [27] as we shall demonstrate.

Since inverse halftoning cannot be done without using extra properties of images, we use sample images for training the LUT. In the training phase, sample images and corresponding halftone versions are used to obtain the LUT. The training phase is simple, and should be done once for a specific halftone method. In Section 3.2 the details of the algorithm are given. Then we show application of LUT inverse halftoning to actual images. Note that

[1] Preliminary versions of parts of this chapter are presented in [42] and [43], and part of this chapter is accepted for publication as a journal article [44].

[2] These storage requirements are calculated for 1 byte/pel contone images.

our LUT inverse halftoning algorithm accepts binary images only and extension to scanned halftones remains to be explored.

In Section 3.2, we use experimentally found templates for LUT inverse halftoning. However, these templates can be designed during the training phase. This problem is addressed in Section 3.3 and a recursive template selection algorithm is proposed. This algorithm minimizes the mean square error between the inverse halftoned and corresponding contone values when an additional pixel is added to the template. Then templates for error diffused halftones, clustered dither halftones and ordered dither halftones are designed and performances are presented. Afterwards, LUT inverse halftoning is extended for color halftones in Section 3.4. The template selection algorithm for black and white LUT inverse halftoning is then modified for the case of color LUT inverse halftoning. The performance of our algorithm on color images is demonstrated on error diffused color halftones.

3.2 LUT inverse halftoning

In the inverse halftoning method, we want to predict the contone value of a pixel from its surrounding neighbors. The idea we propose here has only superficial similarity to traditional linear filtering. In traditional filtering, the inverse halftone value of a pixel is a linear combination of the surrounding halftone pixels. But in the LUT method the estimated value is a nonlinear function of these pixels. In our method we have to keep the neighborhood to be used in the prediction relatively small. Otherwise, the size of the LUT will be extremely large.

The first step is to choose the neighborhood or template (a collection of pixels) for the pixels to be used in the prediction. We have used different templates as shown in Tables 3.1 and 3.2. In Table 3.1, a rectangular template is shown, and this template is abbreviated as "**Rect**" in comparison tables later. "**O**" denotes the estimated pixel. All of the pixels are used in the prediction. In Table 3.2 another template is shown. This pattern consists of a symmetric part and some additional pixels. In the 16 pixel pattern the "**a**" pixels and the "O" pixel are used in the prediction of the contone value of pixel "O." This template is denoted as "**16pels**" in comparison tables. Another example of a template is obtained by adding "**b**" pixels to the "**16pels**" template, and this template is referred to as "**19pels**" template.

a	a	a	a
a	a	a	a
a	a	O	a
a	a	a	a

Table 3.1: Neighborhood used in inverse halftoning (rectangular support, "Rect" template).

b	a	a	a	b
a	a	a	a	b
a	a	O	a	a
	a	a	a	
		a		

Table 3.2: Another possible neighborhood used in inverse halftoning ("19pels" template).

Let us assume that there are N pixels (including the pixel being estimated) in the neighborhood and that they are ordered in a specific way. Let us also call the pixel values as p_0,p_1,...,p_{N-1}. Note that there are 2^N different patterns since $p_i \in \{0,1\}$ for $i = 0,1,...,N-1$. Assuming that the original image is an 8 bit image, our LUT, T, should return a value for each pattern, i.e., $T(p_0,p_1,...,p_{N-1}) \in \{0,1,...,255\}$. During the inverse halftoning phase, the pixels in the region of support will be arranged in a specific order, and the contone value will be obtained from the LUT.

3.2.1 Design of LUT

We first obtain the expected contone value for each template pattern. Then this contone value will be assigned to the corresponding LUT position for that pattern. Let us denote the number of occurrences of pattern $(p_0p_1...p_{N-1})$ in the sample halftone images as $K(p_0,p_1,...,p_{N-1})$. Thus, $K(p_0,p_1,...,p_{N-1})$ is the histogram value of the pattern $(p_0p_1...p_{N-1})$. Notice that the pattern $(p_0p_1...p_{N-1})$ can occur in any of the sample images any number of times. Let us also denote the contone values in the original images corresponding to patterns $(p_0p_1...p_{N-1})$ as

$$C(p_0,p_1,...,p_{N-1},i) \quad \text{for} \quad i = 0,1,...,K(p_0,p_1,...,p_{N-1}) - 1.$$

If $K(p_0,p_1,...,p_{N-1}) > 0$, the LUT value for the pattern $(p_0,p_1,...,p_{N-1})$ will be the mean of the corresponding contone values, i.e.,

$$T(p_0,p_1,...,p_{N-1}) = \frac{\sum_{i=0}^{K(p_0,p_1,...,p_{N-1})-1} C(p_0,p_1,...,p_{N-1},i)}{K(p_0,p_1,...,p_{N-1})}.$$

If $K(p_0,p_1,...,p_{N-1}) = 0$, then the pattern $(p_0p_1...p_{N-1})$ does not exist and we explain next how to handle this situation.

−2	0	1	2	3	4	
−1	5	6	7	8	9	
0	10	11	12	13	14	
1		15	16	17		
2			18			
↑	−2	−1	0	1	2	← i_0
i_1						

Table 3.3: An example of correspondence between the integers i inside the table and (i_0,i_1) for "19pels" template.

Nonexistent Patterns: Some template patterns may not exist in any of the sample halftone images. Therefore, the contone values for nonexistent patterns should be estimated from the existent patterns. We applied three different methods to obtain the nonexistent values:

1.Low pass filtering: The contone value $T(p_0, p_1, ..., p_{N-1})$ corresponding to the missing pattern $(p_0 p_1 ... p_{N-1})$ is obtained as a linear combination of the binary pixels p_i, that is

$$T(p_0, p_1, ..., p_{N-1}) = \sum_{i=0}^{N-1} h(i) p_i.$$

The weights $h(i)$ are obtained by sampling a two-dimensional circularly symmetric Gaussian of the form $g(i_0, i_1) = ce^{-(i_0^2+i_1^2)/2\sigma^2}$, the parameter σ^2 being obtained experimentally by trial and error in order to achieve high PSNR values in the inverse halftoned images. The vector (i_0, i_1) is in the template, the exact correspondence between the integers i and the vectors (i_0, i_1) being fixed throughout the discussion. An example of the correspondence between the integers i and the vectors (i_0, i_1) is depicted in Table 3.3 for "19pels" template. For example, $i = 15$ corresponds to $(i_0, i_1) = (-1, 1)$ in the table. In our experiments the value $\sigma^2 = 0.586$ was used for "19pels" template. The constant c was chosen such that $\sum_i h(i) = 255$ so that the contone value is in the range $[0, 255]$. *2.Hamming distance*: In the sample halftone images, the existing pattern closest to the nonexistent pattern in hamming distance sense is searched. Then, the contone value corresponding to this existing pattern is assigned as the value for the nonexistent pattern.

3.Best linear estimator: Let us number all the patterns which exist in the sample halftone images as $(p_{i,0} p_{i,1} ... p_{i,N-1})$ for $i = 0, 1, ..., M-1$ where M is the number of existing patterns.

Define the pattern matrix **A** with elements

$$A(i,j) = p_{i,j} \text{ for } i = 0,1,...,M-1, j = 0,1,...,N-1$$

and LUT vector **b** with elements

$$b(i) = T(p_{i,0}, p_{i,1}, ..., p_{i,N-1}) \text{ for } i = 0,1,...,M-1.$$

Consider the overdetermined set of equations

$$\begin{matrix} Pattern & \#0 & \rightarrow \\ Pattern & \#1 & \rightarrow \\ \vdots & & \\ Pattern & \#\text{M-1} & \rightarrow \end{matrix} \underbrace{\begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,N-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M-1,0} & p_{M-1,1} & \cdots & p_{M-1,N-1} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b(0) \\ b(1) \\ \vdots \\ b(M-1) \end{bmatrix}}_{\mathbf{b}}$$

The best linear estimator will be the least squares solution to **x** and is given by

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}.$$

Then for each nonexistent pattern $(p_0p_1...p_{N-1})$, we obtain the contone value, $T(p_0,...,p_{N-1})$, as follows: Define $y = [p_0p_1...p_{N-1}]\mathbf{x}$. Then

$$T(p_0,p_1,...,p_{N-1}) = \begin{cases} 0, & if \quad y < 0 \\ 255, & if \quad y > 255 \\ round(y), & otherwise \end{cases}.$$

3.2.2 Experimental results on LUT inverse halftoning

Image Examples: In order to illustrate the performance of our method, we will use Lena and mandrill images in our experiments. The error diffused halftones of these images can be seen in Figs. 3.1 and 3.2. For visual comparison we now show the inverse halftone images for mandrill in Fig. 3.3 and Fig. 3.4. Figure 3.3 is obtained by using a recent method [27] called the "fastiht2" method (perhaps the best known method which achieves high PSNR

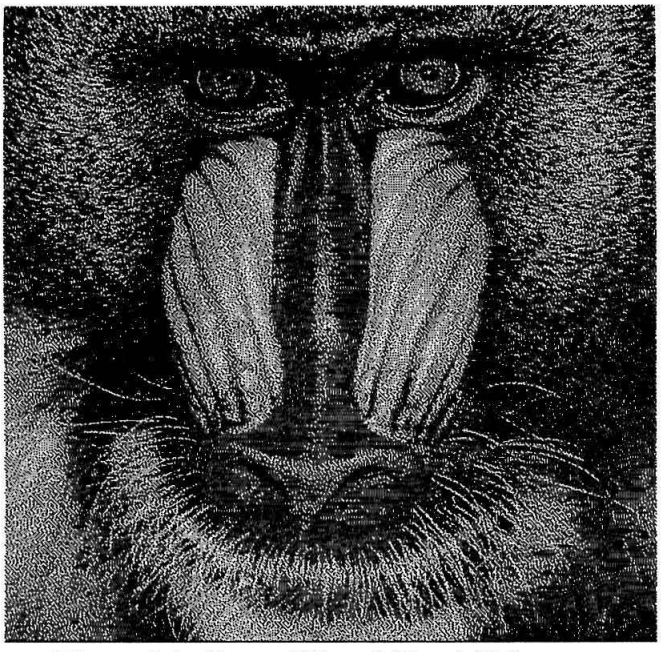

Figure 3.1: Error diffused Mandrill image.

in inverse halftoning or error diffused images and which is also quite fast). Figure 3.4 is obtained with our new LUT method. A clearer view can be obtained at our website [66]. It is clear that the LUT method performs very well compared to "fastiht2'. For example, it preserves high frequency information with nearly no loss (observe the hair on the cheeks). In this example the LUT method uses a training set containing 30 images which can be found at [66] and the template used was the "Rect" support. The training set does not include lena and mandrill images. The training time was 5 seconds on a Pentium II 450MHz computer. Note that the training is done only once for a specific halftoning algorithm. (Training is only required once for a specific halftoning method and training time should not be confused with inverse halftoning time of an image.) Similar comparison for the case of Lena (Figs. 3.5 and 3.6) confirms that the LUT method performs as well as the "fastiht2' method.[3]

We now explain how the inverse halftone quality is affected by various factors. This includes the handling of nonexistent patterns, the content of the training set, and template selection. These effects on inverse halftone quality are summarized in Tables 3.4 and 3.5 and will be explained in detail below.[4] The entries in these tables are the PSNR values

[3]More examples of images can be found at [66].

[4]In this section halftone images were obtained from original images by using the Floyd-Steinberg error diffusion.

Figure 3.2: Error diffused Lena image.

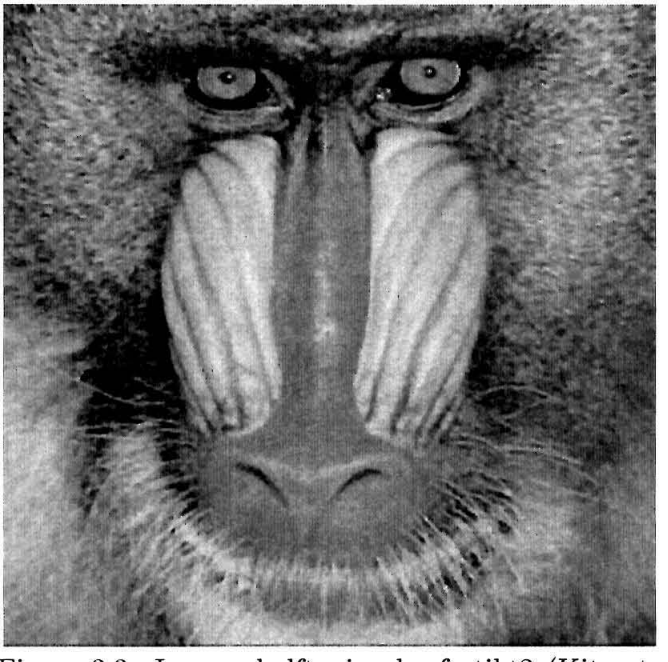

Figure 3.3: Inverse halftoning by fastiht2 (Kite et al.) (PSNR=22.59dB). The halftone was obtained by error diffusion.

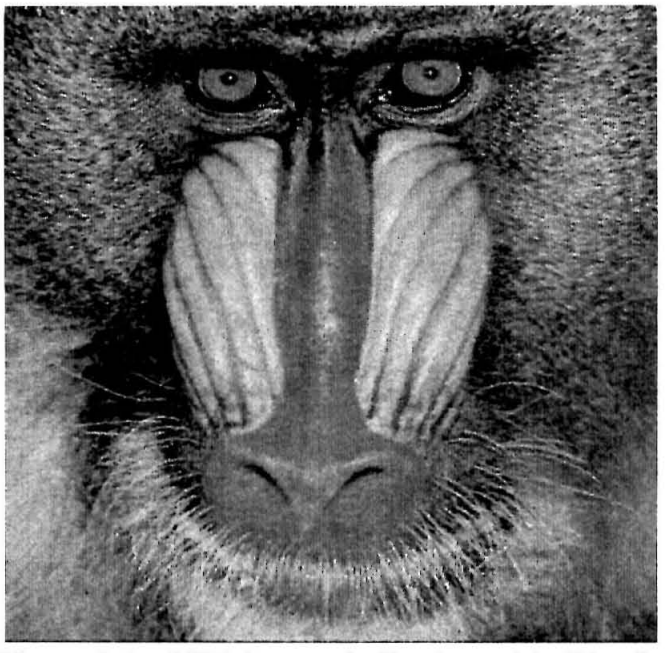

Figure 3.4: LUT inverse halftoning with "Rect" template (PSNR=24.42dB). The halftone was obtained by error diffusion.

Figure 3.5: Inverse halftoning by fastiht2 (Kite et al.)(PSNR=31.37dB). The halftone was obtained by error diffusion.

Figure 3.6: LUT inverse halftoning with "Rect" template (PSNR=30.41dB). The halftone was obtained by error diffusion.

(with respect to the original contone images). In all the tables, the row denoted as "Nonex" indicates the method used for handling the nonexistent patterns (Section 2.1). The last columns in Tables 3.4 and 3.5 show the performance of "fastiht2" method.

Handling the Nonexistent Patterns: In order to compare the methods for the estimation of the values of nonexistent patterns, we have trained our LUT with "19pels" template. The training set includes Lena, peppers, Barbara, mandrill, goldhill and boat images. When the nonexistent pattern estimation methods used are low pass filtering, hamming distance and best linear estimator, the PSNR values of the inverse halftoned airplane image (not included in the training set) are 29.79dB, 28.91dB, and 29.92dB respectively. The Hamming distance method does not work well. The performances of low pass filtering and best predictor methods are close to each other, the best predictor method being a bit better subjectively in image quality. Another disadvantage of low pass filtering method is that a two-dimensional Gaussian filter should be designed experimentally. Thus, only the results based on the best predictor are shown in Tables 3.4 and 3.5.

The number of nonexistent patterns depends on the template and the halftoning method.

If we train LUT for "Rect" template on error diffused images (Lena, peppers, Barbara, mandrill, Goldhill and boat images), 46.5% of the patterns do not exist in the training set. If we use the same template and train on clustered dither halftones, this percentage increases to 90%. Similarly, if we use the same template and train on ordered dither halftones with the 8×8 Bayer matrix, 95% of the patterns do not exist in the training set! However, if the LUT is applied on the same type of halftones used in the training, nonexistent patterns rarely occur. From the above percentage of nonexistent patterns, we can also conclude that in ordered dither and clustered dither halftones, the patterns are more restricted compared to patterns in error diffused halftones.

Effect of the Training Set: The second and third columns in Table 3.4 show results from our LUT methods. The difference between these two columns lies in the training set used for creating the LUT. The images used in the training set for the second column are Lena, Peppers, Goldhill, and Boat images, as indicated by the "x" symbols (mostly smooth images). Similarly the images in the training set for the third column are indicated by "x," and these are the images with more high frequency content (e.g., hairy cheeks in Mandrill, stripes on Barbara's clothing, etc.). The numbers in the table clearly show the importance of the training set. The inverse halftone quality of the LUT method is always better than that of the "fastiht2" method for images with high frequency content (mandrill, Barbara) even when these images are outside the training set. In fact, if these images are in the training set, the LUT method is *distinctly* better for these images. If the training set has only smooth images, then as shown by column 3, the LUT method is still better than "fastiht2" for these smooth images. Note that the airplane image is not in either of the two training sets we have used. Nevertheless, the inverse halftone quality is better than that of the "fastiht2"method if the training set is smooth (col. 2). This is because the airplane image is smooth. Thus a good training set should have enough images representing both smooth and nonsmooth images.

Effect of Template Selection: In Table 3.5 we investigate the effect of template selection on the quality of the inverse halftone. The training set includes 30 images and test set includes another 30 images. In these sets we have included smooth and non-smooth images and these images can be found at [66]. Between two templates having 16 pixels, "Rect" template gives better results than "16pels" when these templates are used to LUT inverse halftone the test images. If we add more pixels into the template, the quality of inverse

Method	LUT Method		fastiht2
Template	Rect	Rect	(Kite
Nonex.	3	3	et.al.[27])
Lena	31.16x	29.91	31.37
Peppers	30.92x	29.22	31.46
Barbara	25.79	27.01x	24.63
Mandrill	23.83	25.43x	22.59
Goldhill	29.67x	28.51	29.54
Boat	30.18x	28.78	29.28
Airplane	29.90	28.69	29.86

Table 3.4: Comparison of different training sets. (Note: 'x' near PSNR value denotes that the image is used in the training set.) The halftones were obtained by error diffusion.

Method	LUT Method			fastiht2
Template	16pels	Rect	19pels	(Kite
Nonex.	3	3	3	et.al. [27])
Average PSNR	26.43	26.50	26.61	25.95

Table 3.5: Comparison of different templates. Halftones are obtained by error diffusion.

halftone gets better as can be seen from the third column of the same table. But adding more pixels to the template makes the LUT bigger. Notice that LUT inverse halftoning even with "16pels" outperforms 'fastiht2" method. We have observed that for "19pels" the PSNR values of reconstructed non-smooth images are higher than the PSNR values of corresponding "fastiht2" reconstructed images and the PSNR values of reconstructed smooth images are close to the PSNR values of corresponding "fastiht2" reconstructed images.

3.3 Template selection

We have shown how to design the LUT for a given template. The next question is how we can choose the best template for inverse halftoning.[5] In the past, the selection of a

[5]The "Rect" and "16pels" templates are found by trial and error, but they are nearly as good as the optimal templates having 16 pixels as we will see at the end of this section.

suitable template has been discussed in the context of halftone compression [14]. In the latter problem, the halftone value of the current pixel should be estimated from a **causal** template as closely as possible. The correlation of halftone pixels with the current pixel is calculated and the highly correlated pixels are included in the template. On the other hand, in inverse halftoning we want to predict the contone value of the current pixel from a template which need not be causal.

Assume that the number of pixels to be used in the template is fixed. Our aim will be to choose the best template of size M. We will simplify the template selection problem by restricting the pixels to be in a fixed neighborhood, i.e., a rectangular $(L+1) \times (L+1)$ neighborhood. We will define our neighborhood as $N_L = \{(i,j) | i \in \{-L/2, ..., L/2\}$ and j $\in$ $\{-L/2, ..., L/2\}\}$. Let us call the pixel whose contone value is being estimated as the current pixel.

Here we give a recursive algorithm to choose the template. Let us denote a template having a pixels as $\mathcal{T}_a$. Assume that we have P images which have sizes $x_1 \times y_1$, $x_2 \times y_2$, ..., $x_P \times y_P$ in our training set. We will have both continuous tone images $D_l(n_1, n_2)$ and halftone images $H_l(n_1, n_2)$ for $l = 1, 2, ..., P$ in our training set where (n_1, n_2) denotes the pixel location in the images. Now let us define the mean square error between two image sets $\{D_l\}$ and $\{F_l\}$ as follows:[6]

$$Err(D, F) = \sum_{l=1}^{P} \sum_{i=1}^{x_l} \sum_{j=1}^{y_l} (D_l(i,j) - F_l(i,j))^2.$$

We can summarize our template selection algorithm in five steps:

Step 0. Let $a = 0$ and $\mathcal{T}_a = \emptyset$.

Step 1. Increment a by 1. Let us define $\mathcal{T}_{a,i,j}$ as $\mathcal{T}_{a,i,j} = \mathcal{T}_{a-1} + (i,j)$. For each $(i,j) \in N_L$ design an LUT using the template $\mathcal{T}_{a,i,j}$ to estimate contone value of a pixel from the pixels in the template $\mathcal{T}_{a,i,j}$ as explained in Section 3.2.1. Use this LUT to estimate the contone images $\{D_l\}$ from halftone images $\{H_l\}$. Let us call the estimate images as $\{\widehat{D_{a,i,j,l}}\}$.

Step 2. Calculate $Err(D, \widehat{D_{a,i,j}})$ for $(i,j) \in N_L$. Let

$$(p, r) = argmin_{(i,j) \in N_L} Err(D, \widehat{D_{a,i,j}}).$$

Include the pixel in the template: $\mathcal{T}_a = \mathcal{T}_{a-1} + \{(p,r)\}$.

[6] In our definition of error we did not incorporate the human visual system. We leave this topic for further research.

Step 3. If T_a does not have M elements, go to step 1. Otherwise stop.

In step 1, we obtain the estimated images using the pixels in the previous template and one more candidate pixel in N_L. In step 2, we choose one pixel from the candidate pixels *such that* when that pixel is added to the previous template in LUT inverse halftoning, the estimated images are closest to their corresponding contone images. Even though the template selection algorithm is a greedy algorithm, it does not find the globally optimum template. However, as we will demonstrate with the examples, it gives good results.

Designing better template selection algorithms is a topic for further research. Notice that finding the globally optimum template of size M and its pixels constrained to N_L neighborhood requires designing $\Pi_{i=1}^{M}(|N_L|+1-i)$ LUT's and testing these LUT's.[7] Hence, a sub-optimum algorithm is necessary for template selection. The algorithm given above minimizes the mean square error between inverse halftoned and corresponding contone images when an additional pixel is added to the template. In [43] we proposed another template selection method which exploits the correlation between the halftone values of pixels in the neighborhood and the contone value of the current pixel in the template selection process. Even though that method gave good results, it was based on the correlation between halftone and contone values rather than the error between inverse halftoned and corresponding contone images.

3.3.1 Experimental results on template selection

In this section, the training set contains different 30 images and the test set contains another different 30 images. Smooth and non-smooth images exist in both sets. These images can be found at [66].

Error Diffusion Example: We used our template selection algorithm to find the best template for error diffused images. The template is shown in Table 3.6. In the table '0' denotes the current pixel, and the numbers denote the order in which the pixel is added to the template. Thus if a smaller template of size K is needed, the first K pixels can be taken into the template.[8]

In Section 3.2 we showed two templates "Rect" and "16pels." These templates have 16 pixels and they are found by trial and error. Now, let us compare these two different

[7] Here $|N_L|$ is the number of elements in the set N_L.

[8] Small optimal templates may be useful for halftone compression as in [57] where a temporary contone image of the halftone is obtained by LUT inverse halftoning.

	18	16	15	17		
	10	5	3	4	14	
19	9	1	0	2	12	
	11	7	6	8	13	

Table 3.6: Template used for LUT inverse halftoning of FS error diffused images. '0' denotes the current pixel, and 'k'th pixel denotes the order in which the pixel is added to the template.

	16 pixel			19 pixel	
Template	16pels	Rect	16opt	19pels	19opt
Average PSNR	26.43	26.50	26.43	26.61	26.76

Table 3.7: Comparison of optimal template with others (error diffusion).

templates which have 16 pixels with the template ("16opt ") found using the optimization algorithm. The average PSNRs of LUT inverse halftoned images are shown in the second, third and fourth columns of Table 3.7. Notice that the average PSNR value is an average of inverse halftoned images which are not included in the training set. Hence even though the optimal templates achieve bigger average PSNR values inside the training set, this does not directly imply better performances of these optimal templates in the test set. However, for a bigger training set and a bigger test set, we would expect that both training and test sets give almost the same results. As it is seen from the table, the average PSNR values are close to each other for the 16 pixel templates. The average PSNR value of LUT inverse halftoned images with "19opt" template is better than the average PSNR value of LUT inverse halftoned images with "19pels" template. The performances of these two templates can be compared from the last two columns of Table 3.7. Notice that the improvements in PSNR by using the optimal templates rather than trial and error templates are small. But trial and error templates are found after extensive experiments, whereas template selection algorithm gives the optimum templates without any human interaction.

Clustered Dither Example: We have applied our LUT inverse halftoning algorithm on 3×6 clustered dither halftones [34]. The template designed is shown in Table 3.8. Notice

	18						
			21				
		19					
			7				
	17	3	11	14	2		
1		9	0	8	16		13
	5	15	12	4	10		
			6				
			20				

Table 3.8: Template used for LUT inverse halftoning of clustered dithered images.

that the order in which the pixels are added to the template is different than the one for error diffused images. Even though the PSNR values are high, the inverse halftoned images have a periodic frequency content which corresponds to the screen frequency as can be seen from Fig. 3.7. Increasing the number of pixels in a template up to 21 did not help to suppress the periodic frequency content fully even though the PSNR values of the inverse halftoned images increase. This is due to the fact that small templates cannot contain one full cycle of the lowest screen frequency. Increasing the template size to capture these low frequency content makes storage requirements infeasible. For comparison, the LUT inverse halftoned clustered dither image is shown in Fig. 3.7. Also, the PSNR values for different templates are shown in Table 3.9.

Our method works well on stochastic halftones (error diffusion, DBS, etc.) efficiently because stochastic halftones only introduce blue noise. For clustered and ordered dither halftones we use a median filtering step after LUT inverse halftoning. We have chosen the median filter as the postprocessor because a median filter smooths the image without smearing the edges. Thus, we get rid of the low screen-frequency content in the image.

We have experimented 3×3 and 5×5 median filters [24] as post processors. The 3×3 median filter was not big enough to suppress the periodic structures in the halftones, whereas 5×5 median filter was sufficient to suppress the periodic structures. In order to reduce the number of comparisons needed to implement the median filter, we used a

	16 pixel			19 pixel		21 pixel
Template	16pels	Rect	16opt	19pels	19opt	21opt
Average PSNR	23.19	22.30	23.30	23.49	23.71	23.78
Boat	26.00	25.03	26.07	26.50	26.49	26.55

Table 3.9: Comparison of optimal template with others (clustered dither).

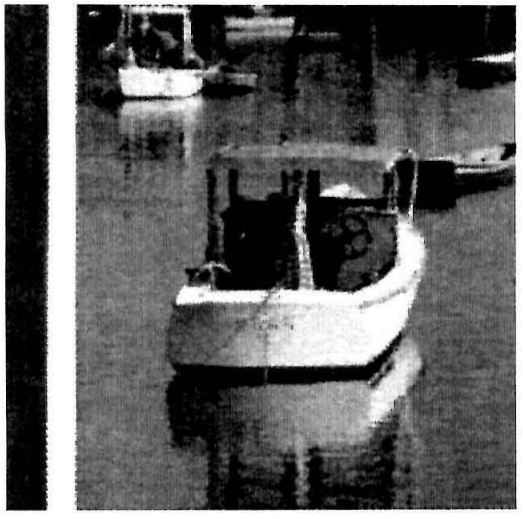

Figure 3.7: LUT inverse halftoning of boat image with "21opt" template. The halftone was a clustered dither image.

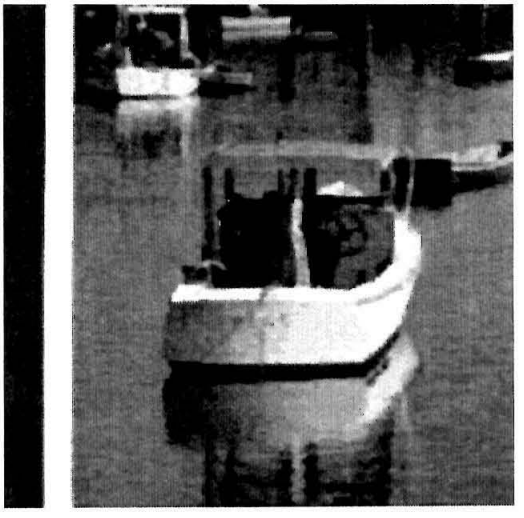

Figure 3.8: Two-step LUT inverse halftoning with "19opt" template. The halftone was a clustered dither image.

Template	16opt	19opt	21opt
Average PSNR	23.24	23.28	23.30
Boat	26.53	26.73	26.75

Table 3.10: Performance of two-step LUT inverse halftoning algorithm for clustered dither halftones.

5×5 separable median filter which requires only 18 comparisons per pixel. The result of two-step LUT inverse halftoning on clustered dither boat images is shown in Fig. 3.8 for template having 19 pixels. Increasing the template size also increases the image quality. In our experiments we use a 19 pixel template. The PSNR values for two step LUT inverse halftones which employs 16, 19, 21 pixel size templates are shown in Table 3.10. If we compare these PSNR values of two step LUT inverse halftones with the PSNR values of the one step LUT inverse halftones (Table 3.9), we see that median filtering does not increase the PSNR values; in fact it decreases them a bit. However, the image quality improves, since the periodic structures are suppressed. (These are not noticible here, but please see our website [66].) As a comparison we show the inverse halftoned images with algorithm II in [34] in Fig. 3.9 (PSNR=25.86dB). If this is compared with the image in Fig. 3.8, we can conclude that the former is more smoothed than the latter.

Another alternative post processing strategy would be to apply a notch filter to filter the screen frequency. But this means that we have to estimate the screen frequency and design the notch filter for different dither halftone methods. Also, we need to make the filter length quite large to get good performance. Hence we have chosen median filter as our post processing algorithm. Notice that we cannot apply median filtering directly on halftone images, because median filtering of halftone images means majority voting inside the median filter support and the output image is still a binary image.

Ordered Dither Example: The optimal template designed for ordered dither halftones using the method described in this section is shown in Table 3.11. In the halftoning process the 8×8 Bayer dither matrix [58] is used. We again used the two step LUT inverse halftoning algorithm. Experimentally we found out that it is a bit harder to inverse halftone ordered dither images. The two-step LUT inverse halftoning with 16 pixel template is not enough to suppress the periodic structures. With the 19 pixel template and 5×5 median filtering

		12	10	14	15	
19		9	2	8	11	
	7	1	0	4	16	
	12	3	5	6		20
		18	17			

Table 3.11: Template used for LUT inverse halftoning of dispersed dithered images (8×8 Bayer's matrix).

Figure 3.9: Inverse halftoned boat image with algorithm II in [34].

the periodic structures are mostly suppressed, but if closely examined some defects can be seen in the inverse halftoned image. This is due to the fact that the periodicity of 8×8 Bayer dither screen is longer than the periodicity of 3×6 clustered dither screen. Thus our two-step LUT inverse halftoning method will work for dither screens which are small enough so that our two step LUT algorithm can annihilate the periodic patterns in the halftones. We have verified this claim by inverse halftoning 4×4 Bayer dithered images: The periodic structures were absent in the inverse halftone images.

Remark: If the images are mostly smooth images, the inverse halftone quality of error diffused halftones can be improved by applying a simple smoothing algorithm on the LUT inverse halftoned image. Here we apply a small median filter (a 3×3 separable median filter which requires only 6 comparisons per pixel) on the LUT inverse halftone result to get

Figure 3.10: Result of LUT inverse halftoning with "Rect" template, followed by 3×3 median filtering (PSNR=31.50dB). The halftone was originally obtained by ED.

even smoother, high PSNR inverse halftone images. The Lena image LUT inverse halftoned with 'Rect' template (third column in Table 3.7) and then 3×3 median filtered is shown in Fig. 3.10 (PSNR=31.50dB).

3.4 Color inverse halftoning

In this section we will extend the LUT method for color halftones. In color inverse halftoning we will try to exploit the correlation between the color components of an image. Experimental results of template selection and color inverse halftoning will be presented in Section 3.4.2.

3.4.1 Template selection for color halftones

The simple extension of LUT inverse halftoning algorithm to color halftones is to treat the color planes separately. For each plane first choose the template and then design the LUT table for that template. This gives satisfactory results as we will illustrate in Section 3.4.2.

However, we can do better for color inverse halftoning. In most of the "natural images"

Template	Rect	19pels	19optc
Average PSNR	26.49	26.49	26.66

Table 3.12: Results of color inverse halftoning independently in each color plane using LUT inverse halftoning.

there is a high correlation between the color planes. This can be exploited in inverse halftoning. Since there is a correlation between the color planes, there is correlation between the halftone values of any color plane and the contone value of a specific color plane. During the LUT creation phase, if we carefully include the pixels from different color planes to predict the contone value of a particular color plane, this means that we are exploiting the correlation between the color planes.

If we decide to include pixels to our template from other color planes, we have to decide exactly which pixels from which color planes to add to our template. The template selection for color halftones can be made by simply changing the search space of the template selection algorithm. Let us denote the pixel locations in (R,G,B) planes as (R,p,r), (G,p,r), (B,p,r) for $(p,r) \in N_L$. We will define a fixed neighborhood around the current pixel to include pixels from its own color plane as well as other color planes. Let us define $N_{color,L}$ to be

$$N_{color,L} = \{(c,p,r)|c \in \{R,G,B\}, p \in \{-L/2,...,L/2\} \text{ and r} \in \{-L/2,...,L/2\}\}.$$

Then, the template selection algorithm given in Section 3.3 can be modified by changing N_L to $N_{color,L}$. The templates designed in this manner will be shown in Section 3.4.2.

3.4.2 Experimental results and conclusions on color inverse halftoning

Our training and test set include different 30 color images. These images can be found at [66]. We first apply Floyd-Steinberg error diffusion **separately** to red, green and blue planes to get the color halftone. Other types of halftoning methods can also be applied to these images. In Table 3.12 we show the average PSNR[9] values of the inverse halftoned images. In column 2 of Table 3.12 we show the average PSNR value of LUT inverse halftones obtained with "Rect" template applied separately to color planes. Similarly in column 3

[9]Here the PSNR value is obtained by averaging the PSNR values in the three color planes.

of Table 3.12 we show the average PSNR value of LUT inverse halftones obtained with "19pels" template applied separately to color planes.

We have applied template selection algorithm to our training set. In Tables 3.13, 3.14, and 3.15, we have shown the optimized templates which are used to estimate R, G, and B plane values respectively. In all these tables, there are three grids for each color plane. In these grids the middle cells denote the current pixel in different color planes and the 'k'th pixel denotes the order in which the pixel is added to the template. If the 'k'th pixel is in the (left most/middle, right most) grid, then 'k'th pixel added to the template is added from the (red, green, blue) plane. Since the tables are constructed recursively, any size n LUT can be obtained from the tables by keeping the first n pixels from the corresponding column. We have taken the first 19 pixels for each color plane to get "19optc" template. In the fourth column of Table 3.12 we have shown the average PSNR value of LUT inverse halftones obtained with the optimized "19optc" template. As expected, the inverse halftone quality for "19optc" template is better than "19pels" template. Notice that, since we allowed pixels from different color planes to be in the template for a color plane, two pixels from different color planes are used in the prediction of a specific color plane. In order to demonstrate the performance of color LUT inverse halftoning algorithm, we have shown the LUT inverse halftoned Lena image in Fig. 3.13. Color LUT inverse halftoned mandrill image is also shown in Fig. 3.14 (also see our website [66]). The color inverse halftone images obtained with LUT method are visually pleasing. For comparison, color halftones of Lena and mandrill images and original color Lena and mandrill images are shown in Figs. 3.15, 3.16 and 3.11, 3.12 respectively.

In another experiment we optimized the templates when only the mandrill image is used during the template selection algorithm. In the optimized templates (which are not shown here), there are a lot pixels from other color planes which are also used in the prediction of a particular color plane. Notice that, Mandrill has rapidly changing colors whereas most of the images in the training set have big color patches (almost constant color areas). This also explains why there are a lot of pixels from other color planes in the template when only mandrill is used in the template selection algorithm. Thus, if there are big color patches in the training images, there will be fewer pixels from other color planes in the prediction of a particular color plane, and we can separately inverse halftone the color planes without compromising the image quality. However, if we have rapidly changing colors in the training

Figure 3.11: Original color Lena image.

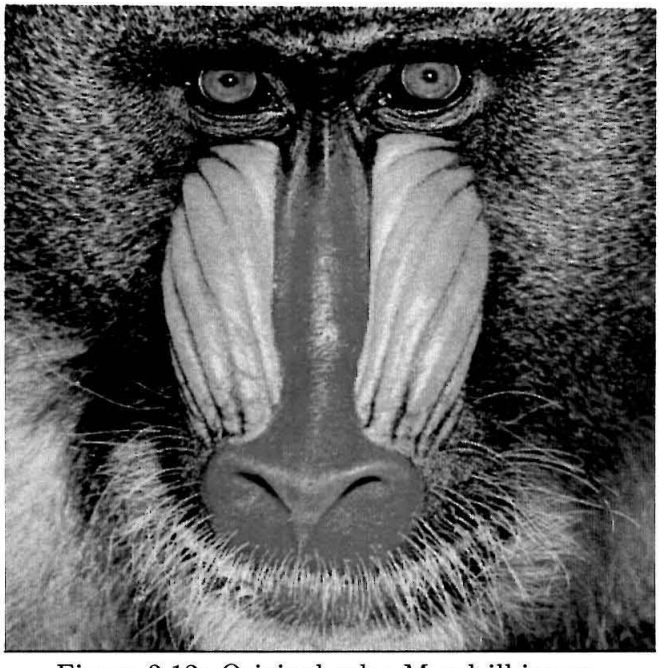

Figure 3.12: Original color Mandrill image.

Figure 3.13: Result of LUT inverse halftoning of color Lena with "19optc" template. The halftone was originally obtained by error diffusion.

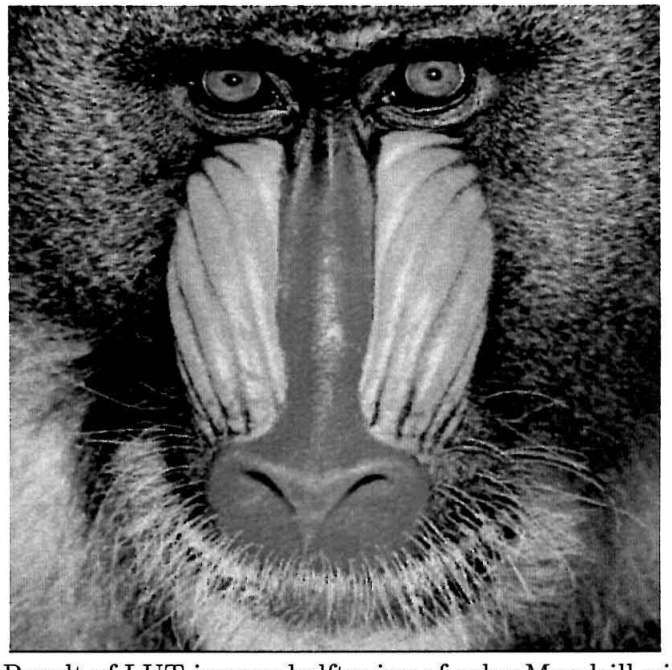

Figure 3.14: Result of LUT inverse halftoning of color Mandrill with "19optc" template. The halftone was originally obtained by error diffusion.

Figure 3.15: Color halftone Lena image. The halftone was originally obtained by error diffusion.

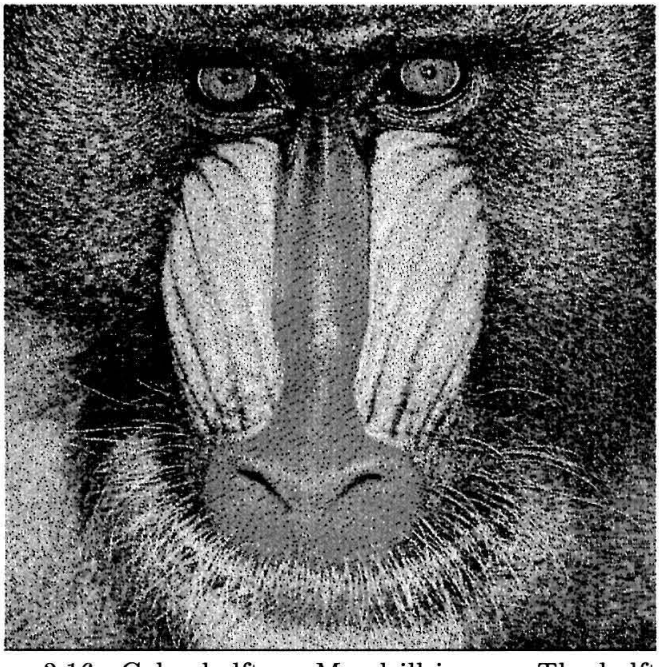

Figure 3.16: Color halftone Mandrill image. The halftone was originally obtained by error diffusion.

20	17	16	18	
10	5	3	4	15
9	1	**0**	2	12
11	7	6	8	19

R plane

		14		

G plane

		13		

B plane

Table 3.13: Template which is used to estimate R plane values for color inverse halftoning. The halftones are obtained by error diffusion.

		14		

R plane

20	18	17	19	
10	5	3	4	15
9	1	**0**	2	12
11	7	6	8	16

G plane

		13		

B plane

Table 3.14: Template which is used to estimate G plane values for color inverse halftoning. The halftones are obtained by error diffusion.

image, correlation between planes becomes more important.

3.5 Conclusion

In this chapter we have introduced a fast method for inverse halftoning which produces images of very good quality. We also proposed a recursive template selection algorithm which minimizes the mean square error between the inverse halftoned and original images. For

		12		

R plane

		17		

G plane

20	15	14	16	
10	5	3	4	18
9	1	**0**	2	11
13	7	6	8	19

B plane

Table 3.15: Template which is used to estimate B plane values for color inverse halftoning. The halftones are obtained by error diffusion.

coarse halftones like clustered dither halftones, a two-step LUT inverse halftoning algorithm was also introduced. Finally, the LUT inverse halftoning and template selection algorithm were extended for inverse halftoning of color halftones. Our inverse halftoning method uses a training mode whereby a look up table (LUT) is created from samples of halftone images in a particular training set. These training sets are obtained for each halftoning method separately. However, our LUT method works for any halftoning algorithm (error diffusion, screening [58], optimized dot diffusion [40], blue noise mask [51], DBS [54], etc.). With a properly chosen training set we demonstrated that the quality of the inverse halftones are at least as good as the best known methods. Note that the inverse halftone quality depends on the starting halftone, i.e., the inverse halftone quality will be better for better halftones.

The LUT method is very fast compared to other known methods because it does not require any computations, and is based entirely on memory access. However, our algorithm assumes bilevel halftones as input, thus it does not extend to scanned halftones yet and this topic is under investigation.

Chapter 4 Tree-structured method for LUT inverse halftoning and for image halftoning

4.1 Introduction

The LUT sizes in LUT inverse halftoning can become bigger if high quality inverse halftone is required. Each binary combination of pixels in the template corresponds to one entry in the LUT. But many of these combinations hardly arise in practice, and are referred to as **nonexistent patterns**. For example, in error diffusion with a sixteen pixel neighborhood, we found that 46.5% of the patterns are nonexistent. For clustered dither 90% are nonexistent, and for Bayer's ordered dithering 95%. In the next section we show how to take advantage of nonexistent patterns and reduce storage.

In this chapter, we will introduce tree-structured LUT (TLUT) inverse halftoning and show how to design TLUT in Section 4.2. The experimental results of TLUT inverse halftoning will be shown in Section 4.3. In the second part of the chapter, we will apply LUT and TLUT ideas to image halftoning. The aim of TLUT will be to imitate the computation-intensive good quality halftones. These will be discussed in Sections 4.4-4.6. Preliminary versions of this chapter were presented in [45] and [46].

4.2 Tree-structured LUT (TLUT) inverse halftoning

In this section, we show how to take advantage of nonexistent patterns and reduce storage. By using a tree structure, the storage requirements can be a fraction of its LUT equivalent. In this sense, the tree structure can be regarded as a 'compressed' version of the LUT. First a small template LUT will be used to get a crude inverse halftone for each pixel. Then this value will be refined by adaptively adding pixels to the template depending on the context. These adaptive pixels will be placed in a tree structure. Like the LUT method, tree-structured LUT inverse halftoning does not involve any filtering, but there will be more steps in inverse halftoning as outlined in Section 4.2.2.

x	x	x	x
x	x	x	x
x	x	O	x
x	x	x	x

Table 4.1: An example of template used in inverse halftoning.

In Section 4.2.1 we show the type of tree structure we are using in inverse halftoning. Then we will describe our inverse halftoning with tree structure in Section 4.2.2. Design of tree structure is discussed in Section 4.2.3 whereas assigning the contone values to tree leaves is outlined in Section 4.2.4. Then the experimental results on error diffused halftone images are reported and discussed in Section 4.3.

4.2.1 Tree structure

Let us denote the size of the initial template used as a (this is typically small, e.g., $a = 9$). We will define 2^a binary trees corresponding to the different patterns in the template. Each tree node is either split further or it is a leaf. Tree nodes are split so that the contone values of LUT obtained with initial template can be refined. If a node is split, then the location of additional pixel,(i, j), is stored in the node and two more nodes attached to this node as its children. If a tree node is a tree leaf, then a contone value is stored in the node.

In Fig. 4.1 we have illustrated a generic tree structure. The upper tree nodes are the tree roots. The black shaded nodes are the tree leaves and they store a contone value in $[0, 255]$. Unshaded tree nodes have two children and they store the location of the additional pixel as shown.

In order to store the tree, we need to record the tree structure, additional pixel locations, and contone values stored in the tree leaves. Let us assume that we have 2^a trees and b tree leaves. Then it can be shown that we need b bytes to store the contone values in the tree leaves, $(2b - 2^a)/8$ bytes to store the tree structure and $(b - 2^a)$ memory units to store the locations of additional pixels (See Appendix 4.8.1 for details). Memory unit usually corresponds to 1 or 2 bytes. The storage requirement of memory unit will depend on the number of possible locations for additional pixels.

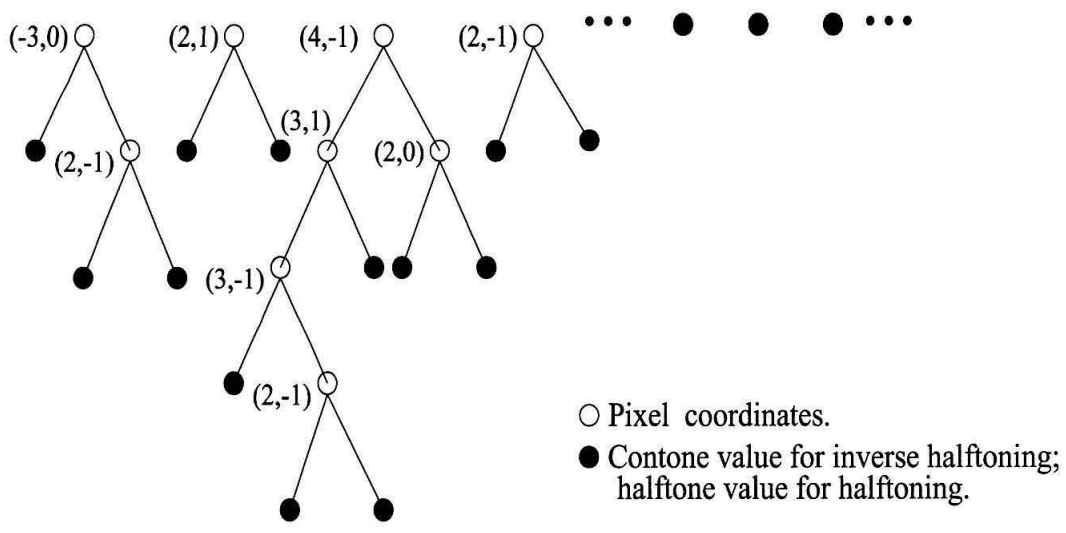

Figure 4.1: Generic tree structures used in inverse halftoning and halftoning.

4.2.2 Inverse halftoning with tree structure

In tree-structured LUT inverse halftoning, we try to find a tree leaf for each pixel in the halftone image. After finding the tree leaf, the contone value stored in the tree leaf will be assigned as the inverse halftone value of the pixel. To find the corresponding tree leaf for each halftone pixel location, we will do the following:

1. First look at the pattern inside the initial template $\mathcal{B}$ of size a. Each pattern will correspond to one of the binary trees. The root of the corresponding tree is declared as the current node.

2. Each node is either split into two nodes or it is a leaf. If a node is a leaf, then the average contone value is stored in the node. This value is assigned as the inverse halftone value at the pixel.

3. If a node is split into two, then the location (i, j) of the additional pixel is stored in the node. Get the halftone value of the pixel which is (i, j) away from the current pixel. If this value is 0(1), then the left (right) node is assigned as the current node. Then go to step 2.

4.2.3 Designing the tree structure

First, the initial template $\mathcal{B}$ of size a should be chosen. The pixels in $\mathcal{B}$ are chosen from a neighborhood $\mathcal{N}_L$ of the current pixel. Here $\mathcal{N}_L$ denotes a neighborhood of size L:

$$\mathcal{N}_L = \{(i,j)|i \in \{-L/2, ..., L/2\} \text{ and } \mathrm{j} \in \{-L/2, ..., L/2\}\}.$$

Thus, $\mathcal{B} = \{(i_0, j_0), (i_1, j_1), \ldots, (i_{a-1}, j_{a-1})\}$. This template can be found using the algorithm outlined in [43] and a this algorithm is given in Section 3.3. Then, each pattern in this template will correspond to one of the binary trees. These will also correspond to the initial 2^a tree leaves. Starting from this tree structure, we will add new tree leaves incrementally. This is done until we get sufficient number of tree leaves or we are satisfied with the inverse halftone quality. In this process the cost function will be the MSE of a specific tree structure. By this, we mean the mean squared error between the inverse halftoned images with the specific tree structure and the original images in the training set. Finding the MSE of a tree requires the contone values in the tree leaves. Given any tree, there is an optimal way to assign contone values to its leaves. In Section 4.2.4 we explain how these are calculated. Here we give an algorithm to add the 'best' tree leaf to a tree structure.

1. For each leaf t and for each pixel p in $\mathcal{N}_L$ do the following: Assume that the leaf t is split into two nodes with the additional pixel p. Calculate the MSE of this tree structure ($MSE_{t,p}$).

2. Find the leaf t_o and additional pixel p_o such that MSE_{t_o,p_o} is minimum.

3. Update the tree structure by splitting the tree leaf t_o with the additional pixel p_o.

Here the algorithm is written as above for simplicity. However, the algorithm can be improved for computational efficiency. Another simplification would be to split K leaves rather than one leaf after calculating the MSE for each tree leaf in the tree structure. Then, the second and third steps of the above algorithm should be modified as follows:

2'. Find K leaves $t_{o,1}, \ldots, t_{o,K}$ and $p_{o,1}, \ldots, p_{o,K}$ such that $MSE_{t_{o,1},p_{o,1}}, \ldots MSE_{t_{o,K},p_{o,K}}$ are the smallest K numbers of the set of all possible values of $MSE_{t,p}$.

3'. Update the tree structure by splitting the tree leaves $t_{o,1}, \ldots, t_{o,K}$ with the additional pixels $p_{o,1}, \ldots, p_{o,K}$ correspondingly.

4.2.4 Assigning contone values to tree leaves

We have to find the contone values for each tree leaf given a tree structure and additional pixel locations. Afterwards, these contone values will be assigned as inverse halftone values in the tree-structured LUT inverse halftoning algorithm (Section 4.2.2). We will use training images in this process, i.e., halftone images and corresponding contone images. First we find the tree leaves for each pixel in the training set using the inverse halftoning algorithm in Section 4.2.2. Let us denote the set of contone values of pixels which have the same tree leaf t as S_t where a_t is the size of S_t. Thus $S_t = \{x_1, x_2, \ldots, x_{a_t}\}$. Then the closest integer to the mean value of S_t will be assigned as the contone value of the leaf (c_t): $c_t = int(\sum_{i=1}^{a_t} x_i/a_t)$.

4.3 Experimental results for TLUT inverse halftoning

We will first apply the TLUT inverse halftoning algorithm on error diffused halftones to demonstrate the performance of our algorithm. Then, TLUT will be applied on clustered dither and ordered dither halftones. Note that our inverse halftoning algorithm can be applied to any type of halftoning algorithm.

4.3.1 TLUT inverse halftoning of error diffused images

First we choose the initial template using the algorithm given in [43]. In our training set we have included Lena, peppers, Barbara, mandrill, boat and Goldhill images. The template, called $TFS9$, for Floyd Steinberg error diffused images is shown in Table 4.2. In the table '0' denotes the current pixel, and the 'x' denotes the other pixels in the template. We have chosen the biggest template such that there are no nonexistent patterns in the halftone training images. Then the nodes are added incrementally using the algorithm summarized in Section 4.2.3. For the tree design algorithm, we have chosen $\mathcal{N}_3$ as our neighborhood.[1]

In Table 4.4, we have summarized the performance of tree-structured LUT inverse halftoning. The trees used differ only in the number of tree leaves. The initial template for all of these tree structures is $TFS9$ and it has nine pixels in the template. Thus all tree structures have $2^9 = 512$ initial tree leaves. The number of additional tree leaves can be seen in the second row. Storage requirements for storing the tree structure, the contone values and the additional pixel locations are also shown in third, fourth and fifth row respectively

[1] A list of acronyms used in this chapter is summarized in Table 4.3.

	x	x	x	x	
	x	x	0	x	
			x		

Table 4.2: Initial template $TFS9$ used in tree-structured LUT inverse halftoning of Floyd-Steinberg error diffused images.

$\mathcal{N}_L$	Neighborhood around the current pixel
$TFS9$	Initial template used for FS error diffused images
$FS1$, $FS2$,$FS3$,$FS4$	Different trees used in TLUT inverse halftoning of FS error diffused images
$TCD7$	Initial template used for clustered dithered images
$CD1$, $CD2$, ..., $CD7$	Different trees used in TLUT inverse halftoning of clustered dithered images
$TOD6$	Initial template used for ordered dithered images
$OD1$, $OD2$, ..., $OD7$	Different trees used in TLUT inverse halftoning of ordered dithered images

Table 4.3: Acronyms used in Chapter 4.

in terms of bytes. The total storage requirement of a tree is then reported in the sixth row. In the remaining rows, the PSNR values of the inverse halftone images (with respect to the original contone images) are shown. Note that 'x' near PSNR value denotes that the image is used in the training set.

Similarly in Table 4.5 we have summarized the inverse halftone image quality and storage requirements of 16 and 19 size LUT inverse halftoning [43], and the inverse halftoning algorithm 'fastiht2' reported in [27]. In this table 'Rect' denotes a specific 16 pixel template and '19pels' denotes a specific 19 pixel template as defined in [43].

From these tables it can be seen that the image quality achieved with tree structure $FS1$ is close to the image quality achieved with LUT halftoning using the 'Rect' template. Note that the latter needs 64KB for storage whereas $FS1$ needs less than 10K. Also, the inverse halftone image quality achieved with $FS1$ is close to fastiht2 method. Besides, the inverse halftone quality is superior for mandrill image which has a lot high frequency content. Note that, even though the airplane image is outside the training set, the inverse halftone image quality is close to the fastiht2 inverse halftone quality. Image quality for $FS1$ can be

Tree	$FS1$	$FS2$	$FS3$	$FS4$
# add. leaves	4608	8704	12800	16896
Storage(tree)	1088B	2112B	3136B	4160B
Storage(contone)	4608B	8704B	12800B	16896B
Storage(location)	4096B	8192B	12288B	16384B
Storage(total)	9792B	19008B	28224B	37440B
Lena	31.06x	31.42x	31.61x	31.76x
Peppers	30.90x	31.30x	31.51x	31.67x
Barbara	26.69x	27.19x	27.52x	27.79x
Mandrill	24.89x	25.22x	25.53x	25.81x
Goldhill	29.67x	29.93x	30.07x	30.19x
Boat	29.88x	30.22x	30.43x	30.61x
Airplane	29.82	29.99	29.95	29.65

Table 4.4: Comparison of different tree structures used in TLUT inverse halftoning. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

Method	LUT Method		fastiht2
Template	Rect	19pels	(Kite[27])
Storage	64KB	512KB	-
Lena	30.90x	31.65x	31.37
Peppers	30.61x	31.48x	31.46
Barbara	26.80x	27.60x	24.63
Mandrill	25.24x	26.41x	22.59
Goldhill	29.49x	30.05x	29.54
Boat	29.89x	30.62x	29.28
Airplane	29.63	29.92	29.86

Table 4.5: Comparison of different templates used in LUT inverse halftoning. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

Figure 4.2: Result of LUT inverse halftoning with "Rect" template.

increased by adding more tree leaves to $FS1$. Bigger trees with storage requirements are reported in second, third and fourth columns of Table 4.4. For visual comparison we now show the inverse halftone images for lena in Fig. 4.2 and Fig. 4.3. These figures can be seen better at the website [66]. In Fig. 4.2 the inverse halftone is obtained with LUT inverse halftoning using 'Rect' template. In Fig. 4.3, the tree-structured LUT inverse halftoning with tree $FS2$ is shown.

4.3.2 TLUT inverse halftoning of clustered dithered images

We have designed and trained TLUT trees on 3×6 clustered dither halftones [34]. The initial template is found using the template selection algorithm given in [43] and this template is shown in Table 4.6 (TCD7). In Table 4.7 we have listed TLUT trees with the same TCD7 initial template but with different number of tree leaves.[2] Thus, all these tree structures have $2^7 = 128$ initial tree leaves. In the table, the number of tree leaves increases from left to right. The inverse halftone images obtained using CD1 tree structure have apparent periodic structures because we did not have enough adaptive pixels. If tree structure CD2

[2]Note that the PSNR values of inverse halftoned clustered dither images are smaller than the PSNR values of inverse halftoned error diffused images (see [34],[44]). This is due to the halftone quality difference between these two halftoning methods.

Figure 4.3: Result of tree-structured LUT inverse halftoning with $FS2$.

is used in TLUT, most of the periodic structures are suppressed and the inverse halftones are almost free from blockiness. If we add leaves to the tree in an unlimited manner in order to get rid of the blockiness, then the inverse halftone images suffer from "over training." Experimentally we observed that this results in the occurence of impulsive noise. [3] However, the tree should be big enough to prevent blocking. The impulsive noise can be removed by median filtering. We used 3×3 median filtering to suppress it. Let us call the following algorithm as two step TLUT inverse halftoning: Given a halftone image, first TLUT inverse halftone the image and then apply median filtering to the result. The results of two step TLUT inverse halftoning are given in Table 4.8. If we add more tree leaves to tree structure CD3, the inverse halftone quality does not change much visually. Also PSNR values of the images outside the training set decrease as more tree leaves are added to tree structure CD3. The reason again is over-training. In Fig. 4.4 we show the two step TLUT inverse halftoned Boat400 image with CD3 tree. For comparison we show two step LUT inverse halftoned Boat400 image with 16 pixel template in Fig. 4.5. Visually, the TLUT result is better: Notice that there is a natural texture in the water in TLUT result whereas the

[3] The tree should normally depend on the halftoning method but not on the specific images in the training set as long as the training set is sufficiently rich. However, when we enter the "overtraining mode," this ceases to be the case, and we believe that the impulsive noise is caused by this phenomenon.

	x				
		0			x
x			x		
	x				

Table 4.6: Initial template $TCD7$ used in tree-structured LUT inverse halftoning of clustered dithered images.

Tree	$CD1$	$CD2$	$CD3$	$CD4$	$CD5$	$CD6$	$CD7$
additional # of leaves	354	1455	4635	8731	12827	16923	21019
Storage(tree)	105B	380B	1175B	2199B	3223B	4247B	5271B
Storage(contone)	482B	1583B	4763B	8859B	12955B	17051B	21147B
Storage(location)	708B	2910B	9270B	17462B	25654B	33846B	42038B
Storage(total)	1295B	4873B	15208B	28520B	41832B	55144B	68456B
Lena	26.00x	28.03x	29.05x	29.61x	30.01x	30.29x	30.53x
Peppers	25.95x	27.96x	29.18x	29.90x	30.34x	30.61x	30.84x
Barbara	22.33	23.18	23.51	23.53	23.46	23.38	23.30
Mandrill	19.99	20.59	20.77	20.77	20.70	20.60	20.53
Goldhill	25.16x	26.68x	27.43x	27.78x	28.03x	28.23x	28.42x
Boat	24.72x	26.20x	27.12x	27.74x	28.24x	28.64x	28.96x
Airplane	24.65	25.99	26.35	26.35	26.26	26.12	26.03
Boat400	23.60	24.63	24.92	24.90	24.75	24.64	24.42

Table 4.7: Comparison of different tree structures for TLUT inverse halftoning of clustered dithered images. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

water does not look natural in some parts in the LUT example.) Also, TLUT needs only 15KB whereas LUT needs 64KB. For comparison in Fig. 3.9 we show the inverse halftoned images with algorithm II in [34]. If this is compared with the image in Fig. 4.4, we see that the former is more smooth than the latter.

4.3.3 TLUT inverse halftoning of ordered dithered images

Here, we trained our TLUT trees on ordered dithered with 8×8 Bayer's matrix. Again the initial template is found using the template selection algorithm given in [43] and this template is shown in Table 4.9 (TOD6). We have listed TLUT trees with the same TOD6 initial template but with different number of tree leaves in Table 4.10. In this table, the number of tree leaves increases from left to right. When we add more tree leaves in order to get rid of the blockiness, impulsive noise occurs which makes it necessary to use 3×3

Tree	$CD1$	$CD2$	$CD3$	$CD4$	$CD5$	$CD6$	$CD7$
Lena	27.99x	28.76x	29.31x	29.65x	29.93x	30.14x	30.33x
Peppers	27.84x	28.87x	29.59x	30.08x	30.42x	30.65x	30.82x
Barbara	23.09	23.32	23.46	23.52	23.52	23.52	23.51
Mandrill	20.69	20.88	20.97	21.00	21.00	20.99	20.98
Goldhill	26.57x	27.23x	27.56x	27.74x	27.89x	28.01x	28.12x
Boat	26.01x	26.63x	27.15x	27.51x	27.85x	28.12x	28.34x
Airplane	25.84	26.32	26.62	26.67	26.66	26.64	26.64
Boat400	24.93	25.30	25.61	25.81	25.87	25.87	25.78

Table 4.8: Comparison of different tree structures for two-step TLUT inverse halftoning of clustered dithered images. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

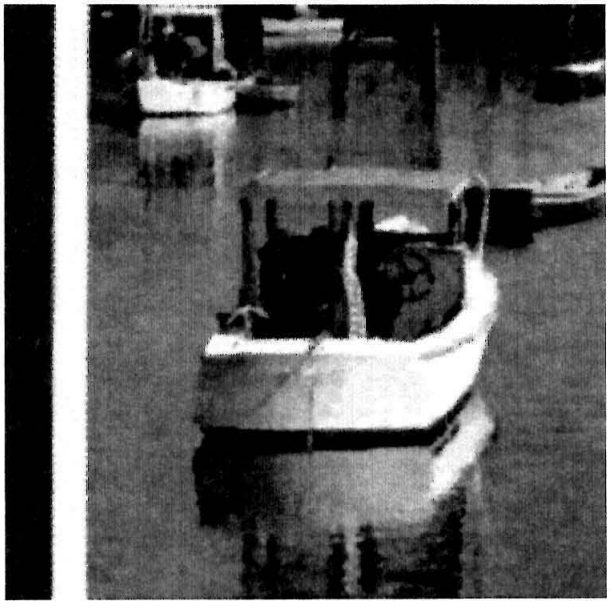

Figure 4.4: Result of two-step TLUT inverse halftone of clustered dithered Boat400 image with CD3.

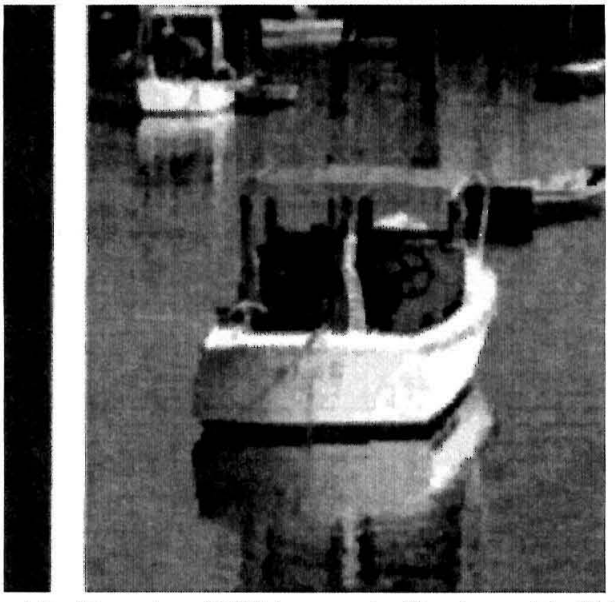

Figure 4.5: Two-step LUT inverse halftoning with "16opt" template. The halftone was a clustered dither image.

		x			
	x	x			
	x	0			
	x				

Table 4.9: Initial template $TOD6$ used in tree-structured LUT inverse halftoning of clustered dithered images.

median filtering to suppress it. Impulsive noise occurs if the tree is bigger than OD5 and most of the periodic structures go away after OD6. However, some slight periodic pattern in the water can still be seen. We have included the two step TLUT inverse halftoned Boat400 image with OD7 tree in Fig. 4.6. Notice that LUT inverse halftoning of this image even with 25 pixels could not suppress the periodic structures [44].

4.4 LUT and TLUT image halftoning

The aim of halftoning is the rendition of gray-scale images on bilevel devices. The most common algorithms for halftoning are ordered dither and error diffusion. In ordered dither a continuous-tone image is thresholded with a spatially periodic screen whereas in error diffusion halftoning, the error is 'diffused' to the unprocessed neighbor points [58].

Tree	$OD1$	$OD2$	$OD3$	$OD4$	$OD5$	$OD6$	$OD7$
additional # of leaves	184	872	3292	7388	11484	15580	19676
Storage(tree)	54B	226B	831B	1855B	2879B	3903B	4927B
Storage(contone)	248B	936B	3356B	7452B	11548B	15644B	19740B
Storage(location)	368B	1744B	6584B	14776B	22968B	31160B	39352B
Storage(total)	670B	2906B	10771B	24083B	37395B	50707B	64019B
Lena	24.64x	26.35x	27.62x	28.30x	29.00x	29.45x	29.76x
Peppers	24.75x	26.36x	27.73x	28.57x	29.24x	29.68x	30.00x
Barbara	21.83	22.76	23.25	23.45	23.52	23.53	23.49
Mandrill	20.32	20.88	21.14	21.20	21.17	21.12	21.06
Goldhill	24.15x	25.52x	26.42x	26.97x	27.38x	27.68x	27.93x
Boat	24.15x	25.47x	26.50x	27.13x	27.71x	28.15x	28.50x
Airplane	23.92	24.75	26.28	26.55	26.62	26.64	26.56
Boat400	23.92	24.26	25.09	25.44	25.54	25.43	25.38

Table 4.10: Comparison of different tree structures for TLUT inverse halftoning of 8×8 ordered dithered images. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

Tree	$OD1$	$OD2$	$OD3$	$OD4$	$OD5$	$OD6$	$OD7$
Lena	26.01x	27.58x	28.59x	29.02x	29.35x	29.60x	29.81x
Peppers	26.29x	27.72x	28.76x	29.39x	29.74x	30.02x	30.27x
Barbara	22.57	23.14	23.43	23.52	23.55	23.56	23.57
Mandrill	20.76	21.04	21.16	21.20	21.20	21.19	21.19
Goldhill	25.39x	26.48x	27.11x	27.40x	27.59x	27.74x	27.89x
Boat	25.47x	26.39x	26.99x	27.37x	27.69x	27.96x	28.18x
Airplane	24.85	25.60	26.74	26.90	26.98	27.02	27.02
Boat400	24.43	24.87	25.64	25.95	26.01	25.98	26.00

Table 4.11: Comparison of different tree structures for two-step TLUT inverse halftoning of 8×8 ordered dithered images. (Note: 'x' near PSNR value denotes that the image is used in the training set.)

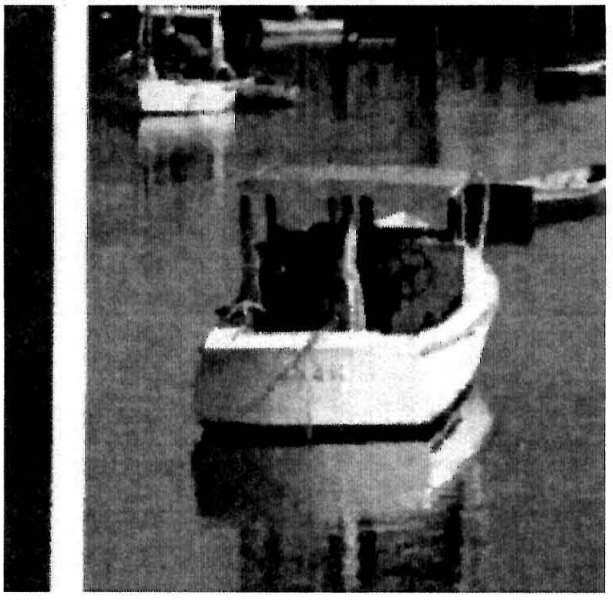

Figure 4.6: Result of TLUT inverse halftone of ordered dithered Boat400 image with OD7.

The complexities of the algorithms and the resulting image quality are different. Ordered dither requires only pointwise comparisons, and it is a parallel method. But the resulting halftones suffer from periodic patterns. Error diffused halftones do not suffer from periodicity and offer blue noise characteristic [58] which is found to be desirable. The main drawback is that error diffusion is inherently serial, and it requires some computation in the diffusion process. The blue noise mask is another way to get halftone quality similar to error diffusion [51]. The disadvantage of blue noise masks is that the resulting halftones do not have enhancement like the error diffusion inherently has.

Here, we introduce LUT based halftoning and TLUT based halftoning methods. Pixels from a causal neighborhood and the contone value of the current pixel will be included in the LUT. The LUT halftoning will require no arithmetic operations other than memory access. For any halftoning method, a sample set of images and halftones of these images will be used to construct the LUT. Even though tree-structured LUT (TLUT) halftoning is more complex than LUT halftoning, it produces better halftones and it requires much less storage than LUT halftoning. We will demonstrate how error diffusion characteristics can be achieved with this method.

	15			14	
	11	9	6	8	
13	7	3	2	4	12
10	5	1	O		

Table 4.12: LUT template used in halftoning.

There are more computation-intensive halftoning methods like Direct Binary Search (DBS) [54] which give the best halftone quality. These algorithms serve as excellent benchmarks, but are not commonly used. When TLUT is trained on DBS-like images, the halftone quality will be in between the error diffusion quality and DBS quality. Moreover, the TLUT halftone quality will get better when the size of TLUT increases.

4.5 LUT halftoning

We will process pixels one by one in raster-scan order. In order to decide the halftone value at a chosen cell (or pixel location), we will use the halftone values already decided in a carefully selected template or neighborhood of the chosen cell and the contone value of the chosen pixel. The template design phase is merely a process of deciding which neighboring cells should be involved in the prediction. We show a sample template in Table 4.12. The letter "0" denotes the cell whose halftone value is being decided and other numbers denote the cells in the template. The template selection algorithm will be described later in this section.

Let us assume that there are N pixels (excluding the pixel being predicted) in the neighborhood and they are ordered in a specific way. Let us also call the halftone values of pixels as p_0,p_1,...,p_{N-1} and the contone value of the pixel being predicted as c. Note that there are $2^N 2^8$ different patterns since $p_i \in \{0,1\}$ for $i = 0,1,...,N-1$ and $c \in \{0, 2^8 - 1\}$. Since the halftone image is a bilevel image, our LUT should return a binary value for each pattern $(p_0 p_1 ... p_{N-1} c)$: $T_{(p_0 p_1 ... p_{N-1} c)}$.

4.5.1 Design of LUT

In the design of LUT, we need training images and corresponding halftone images. Thus, we select a set of images and halftone these images with any halftoning algorithm of our

choice. We will first obtain the expected halftone value for each pattern. Then this halftone value will be assigned to the corresponding LUT position for that pattern. Let us denote the number of occurrences of pattern $(p_0p_1...p_{N-1}c)$ in the sample halftone images as $K_{p_0p_1...p_{N-1}c}$ and corresponding halftone values as

$$h_{p_0p_1...p_{N-1}c,i} \quad \text{for} \quad i = 0, 1, ..., K_{p_0p_1...p_{N-1}c} - 1.$$

If $K_{p_0p_1...p_{N-1}c} > 0$, the LUT halftone value for the pattern $(p_0p_1...p_{N-1}c)$ will be the closest quantization point to the mean of the corresponding halftone values is, i.e.,

$$T(p_0p_1...p_{N-1}c) = \begin{cases} 1 & \text{if } m_{p_0p_1...p_{N-1}c} \geq 0.5 \\ 0 & \text{if } m_{p_0p_1...p_{N-1}c} < 0.5 \end{cases}$$

where

$$m_{p_0p_1...p_{N-1}c} = \frac{\sum_{i=0}^{K_{p_0p_1...p_{N-1}c}-1} h_{p_0p_1...p_{N-1}c,i}}{K_{p_0p_1...p_{N-1}c}}.$$

4.5.2 Nonexistent pattern estimation

If $K_{p_0p_1...p_{N-1}c} = 0$, then the pattern $(p_0p_1...p_{N-1}c)$ does not exist. In this case the halftone value should be estimated in a different way. A similar problem arises in LUT inverse halftoning as well, and three methods are proposed in [42] for this. One of these called the best linear estimator, modified for LUT halftoning, works as follows: Let us number all the patterns which exist in the sample halftone images as $(p_{i,0}p_{i,1}...p_{i,N-1}c_i)$ for $i = 0, 1, ..., M-1$ where M is the number of existing patterns. Let $A(i,j) = p_{i,j}$ and $b(i) = T(p_{i,0}p_{i,1}...p_{i,N-1}c_i)$ for $i = 0, 1, ..., M-1$, $j = 0, 1, ..., N-1$. Also let $A(i,N) = c_i$ for $i = 0, 1, ..., M-1$. We are looking for an estimate of the halftone value of the nonexistent pattern of $(p_0p_1...p_{N-1}c)$ in the following form:

$$y = [p_0p_1...p_{N-1}c]\underbrace{[x_0x_1...x_{N-1}x_N]}_{\mathbf{x}}^T.$$

The best linear estimator will be the least squares solution to the overdetermined system of equations $\mathbf{Ax} = \mathbf{b}$. The solution is

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}.$$

Then for each nonexistent pattern $(p_0 p_1 ... p_{N-1} c)$, we obtain the halftone value as follows:

$$T(p_0 p_1 ... p_{N-1} c) = \begin{cases} 0, & if \quad y < 0.5, \\ 1, & if \quad y \geq 0.5. \end{cases}$$

4.5.3 Template selection

We have shown how to design the LUT for a given template. The next question is how we can choose the best template for LUT halftoning. We have proposed a method for template selection in LUT inverse halftoning in [43]. Here we will modify that algorithm for LUT halftoning.

Assume that the number of pixels to be used in the template is fixed. Our aim will be to choose the best template of size N. We will simplify the template selection problem by restricting the pixels to be in a fixed causal neighborhood. We will define our neighborhood as $\mathcal{N}_L^{'} = \{(i,j) | i \in \{-L/2, ..., 0\}, j \in \{-L/2, ..., L/2\}\} \cup \{(0,j) | \text{j} \in \{-L/2, ..., -1\}\}$. Let us call the pixel whose half–tone value is being estimated as the current pixel.

Here we give a recursive algorithm to choose the template. Let us denote a template having a pixels as $\mathcal{T}_a$. Assume that we have P images which have sizes $x_1 \times y_1$, $x_2 \times y_2$, ..., $x_P \times y_P$ in our training set. We will have both continuous tone images $D_l(n_1, n_2)$ and corresponding halftone images $H_l(n_1, n_2)$ for $l = 1, 2, ..., P$ in our training set where (n_1, n_2) denotes the pixel location in the images.

Now let us define the error in the LUT halftone images compared to LUT images for a given template $\mathcal{T}$ as follows[4]:

$$E_{\mathcal{T}}(\widehat{H}^{\mathcal{T}}, H) = \sum_{l=1}^{P} \sum_{i=1}^{x_l} \sum_{j=1}^{y_l} \left(H_l(i,j) - \widehat{H}_l^{\mathcal{T}}(i,j)\right)^2$$

where $\widehat{H}^{\mathcal{T}}$ is obtained using LUT halftoning with template $\mathcal{T}$. We can summarize our size-M template selection algorithm in two steps:

Step 0. $a = 0$. $\mathcal{T}_a = \emptyset$.

Step 1. Define (p, r) as follows:

$$(p, r) = \underset{(i,j) \in \mathcal{N}_L^{'}}{argmin} \; E_{\mathcal{T}}(\widehat{H}^{\mathcal{T}}, H).$$

[4]Note that a better error measure would be the MSE of the HVS filtered error between the halftones. However, this topic is left for further research.

Include the pixel (p, r) in the template: $\mathcal{T}_a = \mathcal{T}_{a-1} + \{(p, r)\}$.

Step 2. If Te_a does not have M elements go to step 1. Otherwise stop.

4.5.4 LUT example

We have chosen $N = 15$ and constructed our template using algorithm given in the previous section. The training set included several error diffused halftone images (see below). This template was shown in Table 4.12. In the table, "k"th cell ($k > 0$) denotes that the cell is added to the template in kth step. Note that we need $2^N 2^8 = 2^{23} Bits$ (1MBytes) to store the LUT. We have trained our LUT with images using the following images: Lena, peppers, grey ramp, boat, airplane, Zelda, grey scale ramp and two more smooth images. The halftones of these images for training are obtained with error diffusion. Afterwards we halftoned goldhill with the designed LUT. Notice that goldhill was **not in the training set**. The result is shown in Fig. 4.8. For comparison we show Goldhill halftoned with error diffusion in Fig. 4.7. Except in regions of very low and very high grey levels, the LUT halftoning method gives the same image quality as error diffusion.[5] Notice that the storage requirement of LUT halftoning is also high. In the next section we will introduce TLUT halftoning in order to overcome these problems.

4.6 Tree-structured LUT halftoning

As illustrated above, LUT halftoning has some defects in regions of very low and very high grey levels. It can be seen that the halftone pattern for $g = \frac{1}{256}$ has approximately 16×16 periodicity, and LUT halftoning with a template shown in Table 4.12 cannot capture this periodicity. This is because template shown in Table 4.12 does not have a pixel which is 16×16 pixels away from the halftone pixel being predicted. Different cells should be added to capture different halftone patterns. However, the template size cannot be increased arbitrarily because of storage problems. This problem can be solved by adding cells adaptively to the template. We will show that "*adaptive cells*" can be stored efficiently in a tree structure.

[5]The images can be found at the website [66] for better viewing.

Figure 4.7: Goldhill halftoned with FS error diffusion.

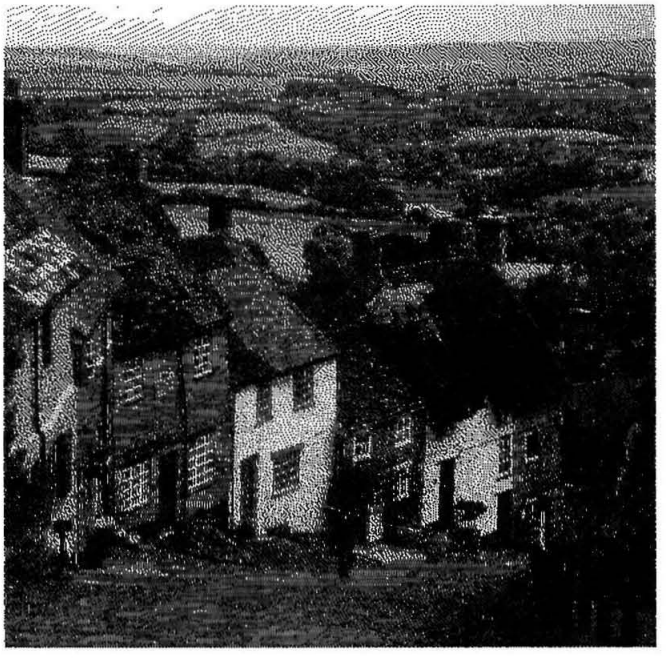

Figure 4.8: Goldhill halftoned with LUT halftoning.

4.6.1 Tree structure

Let us denote the size of the initial template used as a (this is typically small, e.g., $a = 11$). We will define 2^{a+8} binary trees corresponding to the different patterns in the template. Each tree node is either split further or it is a leaf. Tree nodes are split so that the halftone values of LUT obtained with initial template can be refined. If a node is split, then the location of additional pixel,(i, j), is stored in the node and two more nodes are attached to this node as its children. If a tree node is a tree leaf, then a halftone value is stored in the node.

In Fig. 4.1 we have illustrated a generic tree structure. The upper tree nodes are the tree roots. The black shaded nodes are the tree leaves and they store a halftone value. Unshaded tree nodes have two children and also they store the location of the additional pixel as shown.

In order to store the tree, we need to record the tree structure, additional pixel locations, and halftone values stored in the tree leaves. Let us assume that we have 2^{a+8} trees and b tree leaves. Then it can be shown that we need b bits to store the halftone values in the tree leaves, $(2b - 2^{a+8})$ bits to store the tree structure and $(b - 2^{a+8})$ memory units to store the locations of additional pixels.[6] One memory unit usually corresponds to 1 or 2 bytes and this depends on the size of $\mathcal{N}_L'$ used in the tree design step. In general one memory unit is k bits where $k < log_2(\#\ of\ elements\ in\ \mathcal{N}_L') \leq x + 1$.

4.6.2 TLUT halftoning algorithm

In TLUT halftoning, we try to find a tree leaf for each pixel in the halftone image. After finding the tree leaf, the halftone value stored in the tree leaf will be assigned as the halftone value of the pixel. To find the corresponding tree leaf for each halftone pixel location, we will do the following:

1. First look at the pattern inside the initial template $\mathcal{T}$ of size a. Each different pattern will correspond to one of the binary trees. The root of the corresponding tree is declared as the current node.

2. Each node is either split into two nodes or it is a leaf. If a node is a leaf, then the halftone value is stored in the node. This value is assigned as the halftone value at

[6]Similar calculations for storing the tree used in inverse halftoning are done in Appendix 8.2.

the pixel.

3. If a node is split into two, then the location (i, j) of the additional pixel is stored in the node. Get the halftone value of the pixel which is (i, j) away from the current pixel. If this value is 0(1), then left(right) node is assigned as the current node. Then go to step 2.

4.6.3 Designing the tree structure

First, the initial template $\mathcal{T}$ of size a should be found. The pixels in $\mathcal{T}$ are chosen from a neighborhood $\mathcal{N}_L^{'}$ of the current pixel. This template can be found using the algorithm outlined in Section 4.6.3. Then, each pattern in this template will correspond to one of the binary trees. These will also correspond to the initial 2^{a+8} tree leaves. Starting from this tree structure, we will add new tree leaves incrementally. This is done until we get sufficient number of tree leaves or we are satisfied with the halftone quality. In this process the cost function will be the MSE of a specific tree structure. By this, we mean the mean squared error between the TLUT halftoned images with the specific tree structure and the halftone images in the training set. Finding the MSE of a tree requires the halftone values in the tree leaves. Given any tree, there is an optimal way to assign halftone values to its leaves using the majority rule. Here we give an algorithm to add the 'best' tree leaf to a tree structure.

1. For each leaf t and for each pixel p in $\mathcal{N}_L$ do the following: Assume that the leaf t is split into two nodes with the additional pixel p. Calculate the MSE of this tree structure ($MSE_{t,p}$).

2. Find the leaf t_o and additional pixel p_o such that MSE_{t_o,p_o} is minimum.

3. Update the tree structure by splitting the tree leaf t_o with the additional pixel p_o.

4.6.4 Assigning halftone values to tree leaves

We have to find the halftone values for each tree leaf given a tree structure and additional pixel locations. Afterwards, these halftone values will be assigned as halftone values in the TLUT halftoning algorithm. We will use training images in this process, i.e., halftone images and corresponding contone images. First we find the tree leaves for each pixel in

Figure 4.9: Goldhill halftoned with TLUT halftoning trained on error diffused images.

the training set using the TLUT halftoning algorithm in Section 4.6.2. Let us denote the set of halftone values of pixels which have the same tree leaf t as S_t where a_t is the size of S_t. Thus $S_t = \{h_1, h_2, \ldots, h_{a_t}\}$. If there are more ones than zeros in S_t, then the halftone value of the leaf (c_t) will be one. Otherwise it will be zero.

4.6.5 TLUT example

We have chosen our initial template to consist of the first eleven elements of the template shown in Table 4.12. Then we have refined the trees with the training set as in Section 4.5.4 for LUT halftoning with $\mathcal{N}'_{17}$. In the resulting tree, we have $a = 11$, $b = 2^{a+8} + 27310 = 524288$. The TLUT halftone image is shown in Fig. 4.9. As it can be seen from the figure, the problems with very high and very low grey levels in LUT halftoned images do not occur for TLUT halftoning algorithm (see the sky in the image). The total storage cost is approximately 158KB. Note that this is much less than storage requirements of LUT halftoning which is 1 MB.

We have also trained TLUT on DBS halftones. The halftones are obtained by applying fifty steps of DBS iterations on FS error diffused halftones [54]. Our initial template consists of the first eleven elements of the template shown in Table 4.12. The parameters of the

Figure 4.10: Goldhill halftoned with TLUT halftoning trained on DBS images.

TLUT with $\mathcal{N}_{17}$ are as follows: $a = 11$, $b = 2^{a+8} + 32768 = 557056$. The TLUT halftoned image for Goldhill is shown in Fig. 4.10. Notice that the quality of the halftone image is better than error diffused halftone image (especially in the sky). The dots are more uniformly distributed in the sky in TLUT halftoned image whereas artificial worm artifacts in the sky can be noticed easily in error diffused images.[7]

4.7 Conclusion

In the first part of the chapter, we discussed tree-structured LUT inverse halftoning. This method gives good results, and the storage requirements are much smaller compared to LUT inverse halftoning. Here we have applied our algorithm to error diffused halftones, ordered dither halftones and clustered dither halftones. Other type of halftones (e.g., dot diffused images [40]) can also be inverse halftoned easily. In the second part of the chapter, a new LUT based halftoning method is discussed. The algorithm is capable of producing good quality halftones. To refine the halftones, we then proposed tree-structured LUT halftoning. We have demonstrated the performance of our algorithm by training on error

[7]The images can be found at the website [66] for better viewing.

diffused and DBS images. The complexity of TLUT halftoning is higher than the error diffusion algorithm but much lower than the DBS algorithm. Correspondingly, the halftone image quality will lie in between error diffusion and DBS.

4.8 Appendix

4.8.1 Calculation of storage requirement for tree structures used in TLUT inverse halftoning

Assume that we have 2^a trees and b tree leaves in the tree structure. Since the contone values are stored in the tree leaves, we need b bytes to store the contone values. Notice that every addition of a new pixel to the tree structure increases the number of tree leaves by one. Since the tree structure with b tree leaves is obtained by adding additional pixels to an initial tree which contains only 2^a tree leaves, tree structure contains $b - 2^a$ additional pixels and we need $b - 2^a$ memory units to store the locations of these additional pixels.

In order to explain how to store the tree structure, let us define a *tree node storage routine* as follows: In this routine, a "1" will be stored if the tree node is a leaf. If the tree node is split, then a "0" will be stored and this will be followed by applying the tree node storage routine to the right child and then to the left child. Thus, if we apply the tree node storage routine to the roots of 2^a binary trees in a specific order, we can store the tree structure. Notice that we need b bits to store the tree structure when $b = 2^a$ and each addition of an adaptive pixel to the tree structure increases the storage requirement by 2 bits. Thus we need $2b - 2^a$ bits to store the tree structure.

Chapter 5 Effects of inverse halftoning in watermarking

5.1 Introduction

Watermarking is the process of embedding a secret signal into a host signal in order to verify the ownership or authenticity of images. A recent state of the art review of watermarking methods can be found in [30].

It is sometimes necessary to watermark halftoned images. A recent scheme for this [25] is shown in Fig. 5.1. Here a contone image x is first watermarked and then halftoned, and the watermark detector works directly on the halftone image. In this chapter we consider the effect of first inverse halftoning the image before detecting the watermark (Fig. 5.2).[1] In this figure, inverse halftoning can also be thought of as another possible distortion on the watermarked image.

The first watermarking algorithm, spread spectrum watermarking method, was introduced by Cox et al.[11]. This method is a private watermarking algorithm and thus requires the original image in watermark detection. The second method, image watermarking by moment invariants, is introduced by Alghoniemy and Tewfik [1]. This is a public watermarking algorithm, and it does not require the original image in watermark detection. This method is robust to geometric affine transformations. We will perform our experiments for spread spectrum watermarking and watermarking based on image invariants.

In order to examine the effects of inverse halftoning on watermarked images, we will use **T**ree-structured **L**ook **U**p **T**able (TLUT) method for inverse halftoning of images [45],[47]. This algorithm is fast and gives good quality inverse halftone images. Basically inverse halftoning removes most of the halftoning noise. If a good quality halftoning method such as error diffusion is used, then the halftoning noise will be mostly high pass, and if the watermark is embedded in the low pass band, inverse halftoning will not affect the watermark. It may also enhance the watermark detection because it also shapes the lowpass band of

[1] In the same paper [25], an algorithm for joint halftoning and watermarking is proposed, but in this chapter we will only discuss the effects of inverse halftoning in watermark detection.

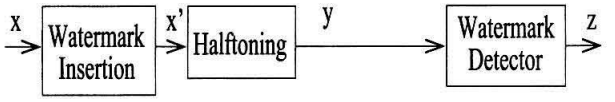

Figure 5.1: Watermark detection from the halftone directly.

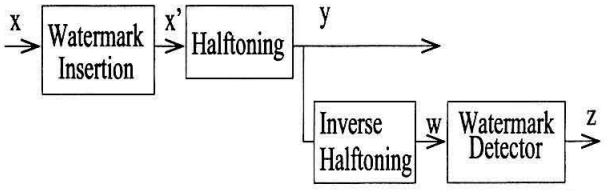

Figure 5.2: Watermark detection after inverse halftoning.

the image slightly.

Thus we could expect an enhancement in watermark detector performance if inverse halftoning is used on smooth images and we will expect a deterioration in watermark detector performance if inverse halftoning is used on high frequency images. We will experimentally show that this is indeed the case.

In Section 5.2 we review the details of these watermarking algorithms. A brief review of TLUT inverse halftoning is given in Section 5.3. Experimental results are given in Section 5.4.

5.2 Watermarking algorithms

Here we will look at two watermarking schemes. In private watermarking schemes, the original image is required for watermark extraction, whereas in public watermarking schemes original image is not needed for watermark detection. As an example of private watermarking we will describe spread spectrum watermarking and as an example of robust public watermarking we will describe image watermarking by moment invariants. Notice that Least Significant Bit (LSB) embedded watermarking is not robust against halftoning because LSB watermark is embedded in high frequency band and halftoning destroys this watermark completely.

5.2.1 Spread spectrum watermarking

Let us first summarize the spread spectrum watermarking scheme proposed in [11]. Let $\mathbf{x}$ be the original image of size $P \times Q$ and $\mathbf{f}$ be the transform domain representation of $\mathbf{x}$ obtained with a DCT. Let $v_1, \ldots v_N$ be the first N largest coefficients of $|\mathbf{f}|$ where $|.|$ stands for the point-wise absolute value operator. Watermarking $\mathbf{x}$ with the watermark $\mathbf{w} = [w_1, ..., w_N]^T$ is the process of modifying v_i's as follows: $v_i' = v_i(1 + \alpha w_i)$. Here w_i's are i.i.d. normal random variable with zero mean and unit variance and α determines the strength of watermark sequence to be embedded. To make the watermark more robust to distortions α should be made large; however, if we want the watermark to be invisible, α should be chosen below a threshold.

Let us denote the watermarked image as $\mathbf{x}'$ and the possibly distorted version of the watermarked image as $\mathbf{x}''$. In the detection process we will use normalized correlation as a performance measure. Let use assume that we want to test whether the watermark $\mathbf{w_t}$ is embedded in image $\mathbf{x}''$. Let $\mathbf{f}''$ be the DCT transform of $\mathbf{x}''$. The first N largest coefficients of $|\mathbf{f}''|$ will be $v_1'', \ldots v_N''$. Then the detected watermark is

$$w_i'' = \frac{1}{\alpha}(\frac{v_i''}{v_i} - 1) \quad \text{for} \quad i = 1, \ldots, N$$

and the normalized correlator output is[2]

$$\gamma = \frac{< \mathbf{w_t}, \mathbf{w}'' >}{\sqrt{< \mathbf{w_t}, \mathbf{w_t} >< \mathbf{w}'', \mathbf{w}'' >}}.$$

Here, we are checking whether $\mathbf{w_t}$ exists in the image. If $\mathbf{w_t} = \mathbf{w}$ then we expect γ to be close to unity; otherwise, we expect γ to be close zero.

5.2.2 Image watermarking by moment invariants

In this method, the watermark is the mean of several functions of the second and third order moments designed to be invariant to scaling and orthogonal transformations. For an image $f(x, y)$ its geometric moments are

$$m_{p,q} = \int \int x^p y^q f(x, y)$$

[2]The inner product between $\mathbf{x} = [x_1, ..., x_N]^T$ and $\mathbf{y} = [y_1, ..., y_N]^T$ is defined as $< \mathbf{x}, \mathbf{y} >= \Sigma_{i=1}^N x_i y_i$.

and central moments are

$$\mu_{p,q} = \int\int (x-\bar{x})^p (y-\overline{y})^q f(x,y)$$

where $\bar{x} = m_{1,0}/m_{0,0}$ and $\overline{y} = m_{0,1}/m_{0,0}$. Then the normalized central moments are $\eta_{p,q} = \mu_{p,q}/\mu_{0,0}^{\gamma}$ where $\gamma = (p+q+2)/2$. The following functions are found to be invariant to orthogonal transformations [23]:

$$\phi_1 = \eta_{2,0} + \eta_{0,2}.$$

$$\phi_2 = (\eta_{2,0} - \eta_{0,2})^2 + 4\eta_{1,1}^2.$$

$$\phi_3 = (\eta_{3,0} - 3\eta_{1,2})^2 + (\eta_{0,3} - 3\eta_{2,1})^2.$$

$$\phi_4 = (\eta_{3,0} + \eta_{1,2})^2 + (\eta_{0,3} + \eta_{2,1})^2.$$

$$\phi_5 = (\eta_{3,0} - 3\eta_{1,2})(\eta_{3,0} + \eta_{1,2})[(\eta_{3,0} + \eta_{1,2})^2 - 3(\eta_{0,3} + \eta_{2,1})^2]$$

$$+(\eta_{0,3} - 3\eta_{2,1})(\eta_{0,3} + \eta_{2,1})[(\eta_{0,3} + \eta_{2,1})^2 - 3(\eta_{3,0} + \eta_{1,2})^2].$$

$$\phi_6 = (\eta_{2,0} - \eta_{0,2})[(\eta_{3,0} + \eta_{1,2})^2 - (\eta_{0,3} + \eta_{2,1})^2]$$

$$+4\eta_{1,1}(\eta_{3,0} + \eta_{1,2})(\eta_{0,3} + \eta_{2,1}).$$

$$\phi_7 = (3\eta_{2,1} - \eta_{0,3})(\eta_{3,0} + \eta_{1,2})[(\eta_{3,0} + \eta_{1,2})^2 - 3(\eta_{0,3} + \eta_{2,1})^2]$$

$$+(\eta_{3,0} - 3\eta_{2,1})(\eta_{2,1} + \eta_{0,3})[(\eta_{0,3} + \eta_{2,1})^2 - 3(\eta_{3,0} + \eta_{1,2})^2].$$

Functions which are invariant to general affine transformation are also derived in [23],[18] and they are

$$\psi_1 = \eta_{2,0}\eta_{0,2} - \eta_{1,1}^2.$$

$$\psi_2 = \eta_{3,0}^2\eta_{0,3}^2 - 6\eta_{3,0}\eta_{2,1}\eta_{1,2}\eta_{0,3} + 4\eta_{2,1}^3\eta_{0,3} + 4\eta_{3,0}\eta_{1,2}^3$$

$$-3\eta_{2,1}^2\eta_{1,2}^2.$$

$$\psi_3 = \eta_{2,0}(\eta_{1,2}\eta_{0,3} - \eta_{1,2}^2) - \eta_{1,1}(\eta_{3,0}\eta_{0,3} - \eta_{2,1}\eta_{1,2})$$

$$+\eta_{0,2}(\eta_{3,0}\eta_{1,2} - \eta_{2,1}^2).$$

$$\psi_4 = \eta_{2,0}^3\eta_{0,3}^2 - 6\eta_{2,0}^2\eta_{1,1}\eta_{1,2}\eta_{0,3} - 6\eta_{2,0}^2\eta_{0,2}\eta_{2,1}\eta_{0,3}$$

$$+9\eta_{2,0}^2\eta_{0,2}\eta_{1,2}^2 + 12\eta_{2,0}\eta_{1,1}^2\eta_{2,1}\eta_{0,3} + 6\eta_{2,0}\eta_{1,1}\eta_{0,2}\eta_{3,0}\eta_{0,3}$$

$$-18\eta_{2,0}\eta_{1,1}\eta_{0,2}\eta_{2,1}\eta_{1,2} - 8\eta_{1,1}^3\eta_{3,0}\eta_{0,3} - 6\eta_{2,0}\eta_{0,2}^2\eta_{3,0}\eta_{1,2}$$

$$+9\eta_{2,0}\eta_{0,2}^2\eta_{2,1}^2 + 12\eta_{1,1}^2\eta_{0,2}\eta_{3,0}\eta_{1,2} - 6\eta_{1,1}\eta_{0,2}^2\eta_{3,0}\eta_{2,1}$$

$$+\eta_{0,2}^3\eta_{3,0}^2.$$

We will return to this issue in Section 5.4.2.

5.3 Experimental results and discussion

5.3.1 Spread spectrum watermarking

We have embedded the watermark in 0.3% of the DCT coefficients of images, i.e., $N/(PQ) = 0.003$. The strength of watermark is chosen to be $\alpha = 0.1$. We have averaged our results over 100 different watermarks. We also observed that inverse halftoning results in a slightly scaled image. We have found this scale factor to be $1/1.018$. Even though the scale parameter can be estimated from the detected watermark (see Appendix 5.4.2), the actual scale parameter is quite close to unity, and the results of scale detection were not reliable, so we have experimentally found the scale parameter. The effects of scaling on the watermark detector when $\mathbf{w_t} = \mathbf{w}$ is shown in Appendix 5.4.1. Basically γ gets scaled by $1/(\sqrt{1 + \frac{(s-1)^2}{s^2\alpha^2}})$ when the image is scaled by s.

In Fig. 5.3, we show a portion of the original, watermarked, halftoned, and inverse halftoned Lena images for visual comparison. The watermark is invisible in the watermarked image, and inverse halftoning removes most of the halftoning noise. Our results are summarized in Tables 5.1 and 5.2. The mean value of the detector output γ for 100 different watermarks is reported in the second, third and fourth columns of Table 5.1. In Table 5.2, standard deviation of the detector outputs is shown. (The mean value of the detector output when the watermarks don't match is quite small, so we report only the standard deviation of these variables.) In our experiments we use Floyd Steinberg error diffusion to get our halftones. In [45], we describe a specific TLUT, T_2 for use in inverse

Figure 5.3: Lena image, top left: original, top right: watermarked, bottom left: halftoned, bottom right: inverse halftoned. These correspond to x,x',y,w in Fig. 5.2.

halftoning. This TLUT requires 19KBytes of memory and it is used in inverse halftoning here. In both tables, the second column shows the correlator outputs after halftoning is applied to watermarked image, the third column shows the correlator outputs when inverse halftoning is applied to the halftones, and the fourth column shows the detector outputs when scaling is applied on inverse halftoned images.

	correct watermark		
Image	halftone	inverse halftone	scaled
Lena	0.9027	0.9292	0.9415
Peppers	0.7093	0.8498	0.8652
Barbara	0.8697	0.8887	0.8962
Mandrill	0.8079	0.5363	0.5531
Goldhill	0.8388	0.8981	0.9063
Airplane	0.8303	0.8561	0.8741

Table 5.1: Effects of halftoning and inverse halftoning on watermark correlator when watermarks match. Images are halftoned by Floyd-Steinberg error diffusion.

	incorrect watermark		
Image	halftone	inverse halftone	scaled
Lena	0.0343	0.0348	0.0349
Peppers	0.0329	0.0352	0.0352
Barbara	0.0312	0.0317	0.0315
Mandrill	0.0329	0.0352	0.0346
Goldhill	0.0326	0.0319	0.0320
Airplane	0.0307	0.0363	0.0365

Table 5.2: Effects of halftoning and inverse halftoning on watermark correlator when watermarks do not match. Images are halftoned by Floyd-Steinberg error diffusion.

From our experiments we have found that inverse halftoning helps in watermark detection if the original image is smooth, i.e., watermark is embedded in the low pass portion of the image. If the image has high frequency content, then the watermark will also be embedded in the high frequency band, and inverse halftoning will tend to wipe out that information. (The Mandrill image has high frequency details and the watermark correlator output deteriorates if inverse halftoning is applied to Mandrill image. On the other hand lena and peppers are smooth images and inverse halftoning increases the watermark correlator output for these images.) Notice also that halftoning and inverse halftoning do not affect the watermark correlator output when the watermark is not present. Also using the scaling parameter helps to increase the watermark correlator output.

We have also looked at the effects of clustered dot and ordered dither halftoning and inverse halftoning. We have summarized our results in Tables 5.3 and 5.4 for clustered dot and ordered dither halftoning algorithms. The watermark detector output of clustered dither or ordered dither halftones are smaller than the watermark detector output of error diffused halftones. This is expected because error diffusion produces halftones which are better than clustered dot and ordered dither algorithms. If the halftoning algorithm is clustered dither or ordered dither, then inverse halftoning makes it difficult to extract the watermark. This is again due to poor halftone quality.

	correct watermark		
Image	halftone	inverse halftone	scaled
Lena	0.6906	0.6131	0.6467
Peppers	0.7433	0.5657	0.6203
Barbara	0.6501	0.6174	0.6370
Mandrill	0.4154	0.3184	0.3340
Goldhill	0.5541	0.5101	0.5324
Airplane	0.6984	0.5747	0.6050

Table 5.3: Effects of halftoning and inverse halftoning on watermark correlator when watermarks match. Images are halftoned by clustered dot.

	correct watermark		
Image	halftone	inverse halftone	scaled
Lena	0.6841	0.6385	0.6614
Peppers	0.7351	0.6086	0.6495
Barbara	0.6312	0.6108	0.6255
Mandrill	0.4313	0.3288	0.3469
Goldhill	0.5421	0.5065	0.5329
Airplane	0.6673	0.6709	0.6741

Table 5.4: Effects of halftoning and inverse halftoning on watermark correlator when watermarks match. Images are halftoned by ordered dither.

5.3.2 Image watermarking by moment invariants

Let Φ be a vector whose elements are $\phi_1, ..., \phi_7$ which are invariant to orthogonal transformations [23] and let x^* be $|log_{10}x|$ for each variable x. We have chosen $f(\Phi^*)$ to be the mean of these $\phi_1^*, ..., \phi_7^*$ functions. The original Lena image had $f(\Phi^*) = 13.6962$ and we have modified the image as follows[3]: $g'(x,y) = g(x,y) + \beta \log g(x,y)$. We have chosen

[3]The invariants are computed after normalizing the images to have a maximum of 255.

$\beta = 8.9267$ so that the $f(\Phi^*)$ computed from the modified image is around 14.00. In order to detect the watermark correctly, we want $f(\Phi^*)$ to be close to this predefined value. We have summarized our results in Tables 5.5 and 5.6. In Table 5.6 we have shown $\psi_1^*, ..., \psi_4^*$ functions which are invariant to general affine transformations [18]. Notice that ψ_1^*, ..., ψ_4^* are invariant to aspect ratio changes whereas ϕ_1^*, ..., ϕ_7^* are not. From the tables we see that the extracted watermarks from halftoned images are close to their original value. However, clustered dot dithering and error diffusion disturb the watermark more than ordered dithering. Error diffusion affects the watermark because it enhances the edges, and edges are main parts where the watermark is embedded in the modified image. Clustered dot dithering affects the watermark, because the clustered dot halftones are worse than ordered dither halftones. Thus the watermarks extracted from clustered dot halftones are worse than the watermarks extracted from ordered dither halftones. When inverse halftoning is applied on these halftoned images, the watermark extraction becomes even harder.

	$f(\phi^*)$	ϕ_1^*	ϕ_2^*	ϕ_3^*	ϕ_4^*	ϕ_5^*	ϕ_6^*	ϕ_7^*
Original	13.696	2.892	8.203	11.976	11.193	23.837	15.125	22.648
Watermarked	14.002	2.932	8.369	12.200	11.400	24.591	15.413	23.108
Clustered dot (CD) dithered	14.033	2.935	8.381	12.202	11.406	24.780	15.424	23.105
CD dithered +inv. halftoned	14.328	2.945	8.394	12.240	11.432	26.680	15.455	23.151
Ordered dithered	14.005	2.935	8.366	12.217	11.411	24.562	15.420	23.122
Ordered dithered +inverse halftoned	14.007	2.947	8.378	12.251	11.441	24.406	15.459	23.169
Error Diffusion	13.968	2.932	8.365	12.218	11.402	24.344	15.410	23.104
Error Diffusion +inverse halftoned	14.101	2.945	8.379	12.209	11.426	25.182	15.447	23.122

Table 5.5: Effects of halftoning and inverse halftoning on moment invariants, Φ^* (used for public watermarking).

	ψ_1^*	ψ_2^*	ψ_3^*	ψ_4^*
Original	6.387	23.216	15.075	21.633
Watermarked	6.467	23.638	15.325	21.966
Clustered dot (CD) dithered	6.473	23.658	15.334	21.969
CD dithered +inverse halftoned	6.494	23.704	15.366	22.043
Ordered dithered	6.473	23.657	15.335	21.993
Ordered dithered +inverse halftoned	6.497	23.712	15.375	22.074
Error Diffusion	6.468	23.635	15.322	21.991
Error Diffusion+ inverse halftoned	6.493	23.697	15.366	22.012

Table 5.6: Effects of halftoning and inverse halftoning on moment invariants, Ψ^* (used for public watermarking).

5.4 Appendix

5.4.1 Effects of image scaling on the watermark detector

Assume that $\mathbf{f}$, $\mathbf{f}'$ and $\mathbf{f}''$ are the first N largest coefficients of the DCT of the original, watermarked and the scaled version of the watermarked images respectively.

$$f_i' = f_i(1 + \alpha w_i), \quad f_i'' = s * f_i' \quad \text{for} \quad i = 1, \ldots, N.$$

Then the elements in the detected watermark from f'' are

$$w_i'' = \frac{1}{\alpha}(\frac{f_i''}{f_i} - 1) = sw_i + \frac{s-1}{\alpha} = sw_i + \beta.$$

Now the watermark detector output would be

$$\gamma'' = \frac{< s\mathbf{w} + \beta, \mathbf{w} >}{\sqrt{< s\mathbf{w} + \beta, s\mathbf{w} + \beta >< \mathbf{w}, \mathbf{w} >}}.$$

Since the elements in $\mathbf{w}$ are zero mean normal distributed random variables,

$< s\mathbf{w} + \beta, \mathbf{w} > \approx s < \mathbf{w}, \mathbf{w} >$

$\Rightarrow < s\mathbf{w} + \beta, s\mathbf{w} + \beta > \approx s^2 < \mathbf{w}, \mathbf{w} > + \beta^2 N$

$$\Rightarrow \gamma'' \approx \frac{s < \mathbf{w}, \mathbf{w} >}{\sqrt{< \mathbf{w}, \mathbf{w} > (s^2 < \mathbf{w}, \mathbf{w} > + \beta^2 N)}}.$$

Thus, we have the following:

$$\gamma'' \approx 1/\sqrt{1 + \frac{(s-1)^2}{s^2\alpha^2}}.$$

Now, let us look at what happens to the watermark detector output when the image is scaled by s and we test whether $\mathbf{w}'$ exists in the image ($\mathbf{w}'$ and $\mathbf{w}$ are uncorrelated zero mean gaussian random variables). This implies that

$$\gamma' = \frac{< s\mathbf{w} + \beta, \mathbf{w}' >}{\sqrt{< s\mathbf{w} + \beta, s\mathbf{w} + \beta >< \mathbf{w}', \mathbf{w}' >}}.$$

Since $< \mathbf{w}, \mathbf{w}' > \approx 0$, the quantity $< s\mathbf{w} + \beta, \mathbf{w}' >$ will be close to zero.

5.4.2 Estimating the scale parameter

The scale parameter s can be derived from the watermark extracted from the scaled image. Let the detected watermark be $\mathbf{w}'' = s\mathbf{w} + \beta$. Let us evaluate

$$< \mathbf{w}'', \mathbf{w}'' > = < s\mathbf{w} + \beta, s\mathbf{w} + \beta > .$$

Since $\mathbf{w}$ is a normal random variable with unit variance and zero mean, the above expression simplifies to

$$< \mathbf{w}'', \mathbf{w}'' > \approx (s^2 + \beta^2)N = (s^2 + \frac{(s-1)^2}{\alpha^2})N.$$

Thus, we have the following:

$$s \approx \frac{\frac{1}{\alpha^2} \mp \sqrt{\frac{1}{\alpha^4} - (1 + \frac{1}{\alpha^2})(\frac{1}{\alpha^2} - \frac{<\mathbf{w}'', \mathbf{w}''>}{N})}}{1 + \frac{1}{\alpha^2}}.$$

Chapter 6 Optimal histogram modification with MSE metric and optimal codebook selection problem

6.1 Introduction

Histogram modification is a well known technique in image processing. For example, histogram equalization, a special case of histogram modification, is applied to images so that details which cannot be seen in the original image can be rendered. Recently, histogram modification has been used by Coltuc and Bolon [10] to embed watermarks in images.[1] In their work the histogram of an image is modified to a specific watermark histogram. This is achieved by defining an ordering of the pixels in the image. However, in this method, there is no direct control of how the image quality is affected due to histogram modification.

Here, we propose a histogram modification method where the mean square error (MSE) between the modified and the original image is minimized. This problem turns out to be equivalent to an integer linear programming problem. Again the histogram can be modified any way we want.

Another application of histogram modification can be in compression. We may want to modify the histogram of an image such that only some gray levels will be allowed in the modified image. By doing so, we will reduce the number of gray levels in an image so that we need less bits/pixel to store the image. Notice that further compression can be achieved on the modified image. In eliminating some of the gray levels, we will again try to find a modified image such that the MSE between the original image and the modified image is minimum and the histogram of the modified image contains only predefined gray levels. We will assume that these predefined gray levels are given.

The above problem is equivalent to quantization of a signal when the codebook is given. In fact, an easier solution to this problem than solving an equivalent linear programming problem is to find the closest codeword for every signal sample. However, the formulation

[1] A recent state of the art review of watermarking methods can be found in [30].

used in this problem helped us to formulate and solve a harder problem. This is to find the optimal codebook for a given signal histogram. Usually, this problem is solved using Lloyd's quantizer design algorithm [32],[20]. However, the global optimality of the solution is not guaranteed in this algorithm. We will define the problem of finding the optimal codebook where the codewords can come from a finite set. Then we will show that the equivalent problem turns out to be a linear integer programming problem and the solution is guaranteed to be globally optimal.

We will define our histogram modification problem in Section 6.2. The equivalent linear programming problem will be shown in Section 6.3. This will be followed by defining histogram modification for compression in Section 6.4. Then we will show our experimental results on histogram modification in Section 6.5. In Section 6.6 we will define the optimal codebook selection problem when the codewords can come from a finite set. We show the equivalent linear programming problem in Section 6.7. Then some experimental results on optimal codebook selection will be shown in Section 6.8. This will be followed by conclusion.

6.2 Histogram modification problem

Let us assume that the signal has L levels: $g_1, \ldots, g_L$. The original signal will have an histogram $h_1, \ldots, h_L$ where $h_i \geq 0$ for $i = 1, \ldots, L$. Let us denote the desired histogram as $h'_1, \ldots, h'_L$. Now we will allow a fraction of points having value h_i to become h'_j and we will denote the number of these points as $\gamma_{i,j}$. Apparently, $\gamma_{i,j}$ is a nonnegative integer. Also, from the preservation of points, for each grey level the following holds:

$$h_i - \Sigma_{j=1}^{L}\gamma_{i,j} + \Sigma_{j=1}^{L}\gamma_{j,i} = h'_i \quad \text{for} \quad i = 1, \ldots, L. \tag{6.1}$$

In order to find feasible $\gamma_{i,j}$ we have to ensure that

$$\Sigma_{i=1}^{L} h_i = \Sigma_{i=1}^{L} h'_i, \tag{6.2}$$

i.e., the equations in (6.1) are consistent. Another constraint comes from the fact that the total number of points which have signal level h_i and which become some other signal level

should not be greater than h_i:

$$\Sigma_{j=1}^{L}\gamma_{i,j} \leq h_i \quad \text{for} \quad i = 1, \ldots, L. \tag{6.3}$$

Now let us calculate the error when this type of histogram modification is applied on the signal. For simplicity let $\gamma_{i,i} = 0$. Let us assume that a signal level g_i becomes g_j in the modified signal. We denote the error due to this modifications as $f(g_i, g_j)$. Thus the sum of errors due to histogram modification will be

$$E = \Sigma_{i=1}^{L}\Sigma_{j=1}^{L}\gamma_{i,j}f(g_i, g_j). \tag{6.4}$$

The common error functions used are square error function and absolute error function:

$$\text{Square error function:} \quad f(x, y) = (x - y)^2.$$
$$\text{Absolute error function:} \quad f(x, y) = |x - y|.$$

The method described in Section 6.3 works for more general, in fact arbitrary, nonnegative error functions.

We will use square error function from now on to illustrate the power of our method. Also note that we will check whether Eqn. (6.2) holds before starting our minimization. In Eqn. (6.1) there are L equations and only $L - 1$ of them are independent. (If we sum all L equations we end up with Eqn. (6.2).)

6.3 Equivalent linear programming problem

Now, let γ be defined as follows:

$$\gamma = [\gamma_{1,2}, \ldots, \gamma_{1,L}, \gamma_{2,1}, \gamma_{2,3}, \ldots, \gamma_{2,L}, \ldots, \gamma_{L,L-1}]^T. \tag{6.5}$$

Simply, γ will contain all $\gamma_{i,j}$'s except when $i = j$. Let us also define $\mathbf{c}$:

$$\mathbf{c} = [c_{1,1}, \ldots, c_{1,L-1}, c_{2,1}, c_{2,L-1}, \ldots, c_{L,1}, \ldots, c_{L,L-1}]^T. \tag{6.6}$$

where

$$c_{i,j} = \begin{cases} (g_i - g_j)^2 & \text{if } i < j, \\ (g_i - g_{j+1})^2 & \text{if } i \geq j. \end{cases} \tag{6.7}$$

Now, it can be shown that the first $L-1$ equations of Eqn. (6.1) can be written in matrix form as

$$\mathbf{A}\gamma = \mathbf{d} \tag{6.8}$$

where $\mathbf{d} = [h_1 - h'_1, \ldots, h_{L-1} - h'_{L-1}]^T$, and $\mathbf{A} = [a_{i,j}]$ is defined to be

$$a_{i,j} = \begin{cases} 1 & \text{if } (i-1)(L-1) < j \leq i(L-1), \\ -1 & \text{if } j > i(L-1) \text{ and } \frac{j-i-i(L-1)}{L-1} \in \mathcal{Z}, \\ -1 & \text{if } j \leq (i-1)(L-1) \text{ and} \\ & \quad \frac{j-i+1-(i-2)(L-1)}{L-1} \in \mathcal{Z}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.9}$$

Similarly Eqn. (6.3) can be written as

$$\mathbf{B}\gamma \leq \mathbf{e} \tag{6.10}$$

where $\mathbf{e} = [h_1, \ldots, h_L]^T$, and $\mathbf{B} = [b_{i,j}]$ is defined to be

$$b_{i,j} = \begin{cases} 1 & \text{if } (i-1)(L-1) < j \leq i(L-1), \\ 0 & \text{otherwise.} \end{cases} \tag{6.11}$$

Now, our problem can be formulated as follows:

$$\begin{gathered} \text{minimize} \quad \mathbf{c}^T\gamma \\ \text{subject to} \quad \mathbf{A}\gamma = \mathbf{d}, \quad \mathbf{B}\gamma \leq \mathbf{e} \quad \text{and} \quad \gamma \geq 0. \end{gathered} \tag{6.12}$$

We also want the elements of γ to be integers, then the optimal solution is an integer linear programming problem of Eqn. (6.12)[19]. Since integer programming takes more time than linear programming, if we assume that the number points in the signal is huge, then the above problem can be approximated by solving the linear programming of Eqn. (12).

Example: When $L = 3$ and $g_i = i$, then the required matrices for the integer linear programming problem will be as follows:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & -1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 1 & 0 & -1 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix},$$

$$\gamma = [\ \gamma_{1,2} \quad \gamma_{1,3} \quad \gamma_{2,1} \quad \gamma_{2,3} \quad \gamma_{3,1} \quad \gamma_{3,2}\ ]^T,$$

$$\mathbf{c} = [\ 1 \quad 4 \quad 1 \quad 1 \quad 4 \quad 1\ ]^T,$$

$$\mathbf{d} = [\ h_1 - h_1' \quad h_2 - h_2'\ ]^T,\ \mathbf{e} = [\ h_1 \quad h_2 \quad h_3\ ]^T.$$

6.4 Compresion problem

In some cases we may want to reduce the number of levels in a signal for compression. In this case, some of the signal levels are not allowed in the modified histogram: $g_{m_1}, \ldots, g_{m_M}$ where $0 < m_i \leq L$ for $i = 1, \ldots M$. Here M determines how many gray levels will be in the modified image. Then, the constraints for this problem can be written as follows:

$$\begin{aligned} \Sigma_{j=1}^{L} \gamma_{m_i,j} &= h_{m_i} \quad \text{for} \quad i = 1, \ldots, M. \\ \Sigma_{j=1}^{L} \gamma_{i,j} &\leq h_i \quad \text{for} \quad i = 1, \ldots, L. \\ \gamma_{i,j} &\geq 0 \quad \text{for} \quad i, j = 1, \ldots, L. \end{aligned} \tag{6.13}$$

Thus, this problem is again an integer programming problem.

6.5 Experimental results and discussion

In this section, we will illustrate our method of histogram modification on Lena image. The original histogram of Lena image is shown in Fig. 6.1. We have chosen a raised cosine as our desired histogram as our first example. The modified histogram is shown in Fig. 6.2 (PSNR=19.42dB).[2] We can as well modify the histogram to have a constant value (histogram equalization).

Notice that we degrade the image quality (in terms of MSE) very much if we want to modify all portions of the histogram into a very different one. In histogram based watermarking, a watermark (a particular signal) is embedded in the histogram of an image

[2]PSNR values in this chapter refer to the PSNR values between the original and the modified images.

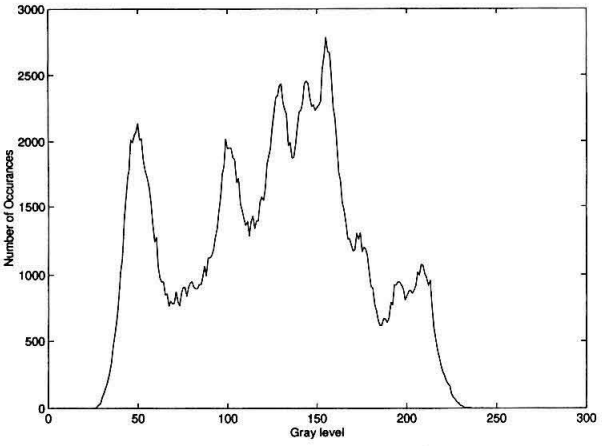

Figure 6.1: Original histogram of Lena image.

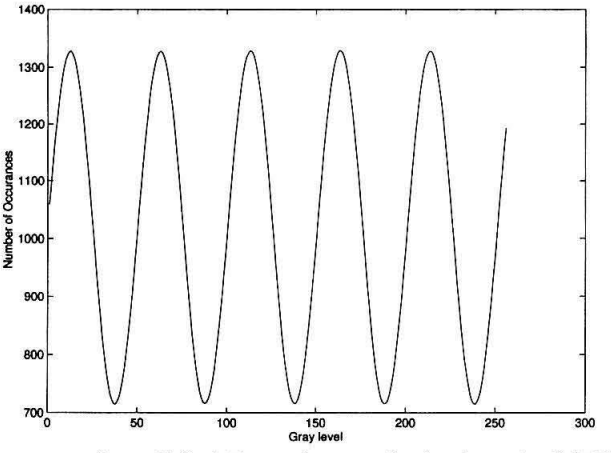

Figure 6.2: Histogram of modified Lena image (raised cosine)(PSNR=19.42dB).

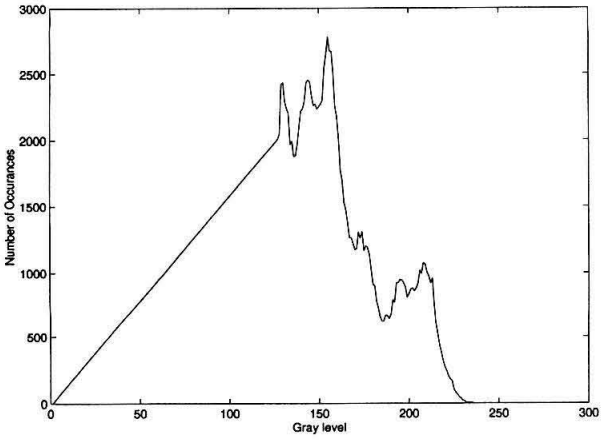

Figure 6.3: Histogram of modified Lena image (partial ramp)(PSNR=35.09dB).

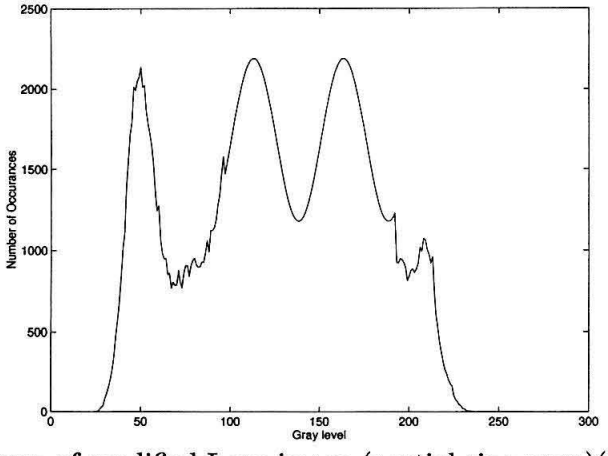

Figure 6.4: Histogram of modified Lena image (partial sine wave)(PSNR=36.05dB).

such that the watermarked and the original images are visually the same. As an example, we have made the first half of the histogram to be a ramp. The modified histogram is shown in Fig. 6.3 (PSNR=35.09dB). We observed that the histogram-modified image and the original image cannot be distinguished visually.[3] In another example, we have made a portion of histogram to look like a sine wave. This histogram is shown in Fig. 6.4 (PSNR=36.05dB). For visual comparison we show the original image in Fig. 6.5 and the histogram modified image in Fig. 6.6. The images are visually indistinguishable.

Our last example is about reducing the number of gray levels in a given image. We have chosen 13 out of 256 gray levels which can appear in the histogram modified image.

[3]All histogram-modified images can be found at [66].

Figure 6.5: Original Lena image.

Figure 6.6: Histogram modified Lena image (partial sine wave)(PSNR=36.05dB).

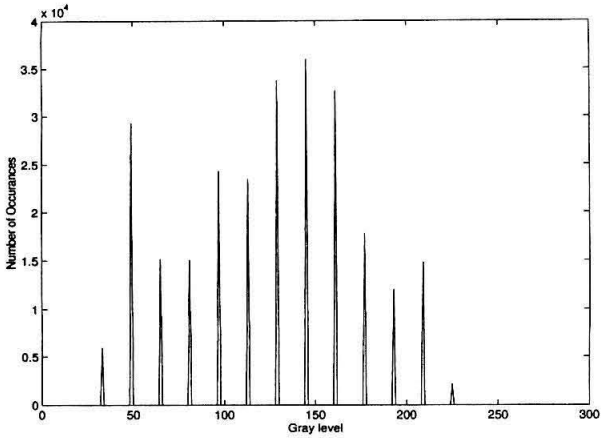

Figure 6.7: Histogram of modified Lena image (only 13 of 256 gray levels exist) (PSNR=34.97dB).

The modified histogram is shown in Fig. 6.7 (PSNR=34.97dB). Thus, we have reduced the number of gray levels in an image without degrading the image very much. Choosing the gray levels which can appear in the histogram-modified image is another problem, and this problem is addressed in the next section.

6.6 Optimal codebook selection problem

Assume that a given signal $x[n]$ can take values only from the set $\{g_i | i = 1, \ldots, L\}$ and we are given the histogram of the signal, h_i for $i = 1, \ldots, L$. We want to compress $x[n]$ by restricting the values of the quantized signal $x'[n]$ such that $x'[n]$ can take such that the error between $x[n]$ and $x'[n]$ is minimum. Let us assume that we want to choose M values out of K values of g'_i for $i = 1, \ldots, K$ such that $x'[n]$ can only take those selected K values and $x'[n]$ is close to $x[n]$. We also know the error when $x[n] = g_i$ and $x'[n] = g'_j$: $c_{i,j}$. Let us denote the number of signal points which were g_i and became g'_j as $\gamma_{i,j}$. These variables are summarized in Table 6.1. The initial signal values and possible levels in the quantized signal are depicted in Fig. 6.8. The empty circles denote the inital signal values and gray circles denote the possible levels in the quantized signal. The problem is to choose M gray circles out of K gray circles such that the error is minimized.

Let us denote the number of signal points which were g_i and became g'_j as $\gamma_{i,j}$. Then the error to minimize will be

$$\Sigma_{i=1}^{L} \Sigma_{j=1}^{K} \gamma_{i,j} c_{i,j}. \tag{6.14}$$

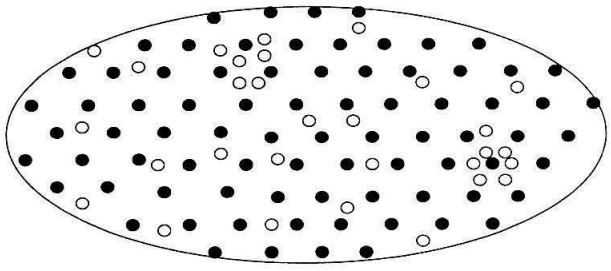

Figure 6.8: A sample diagram which shows initial signal values and possible values in the quantized signal.

Initial gray levels	g_i	for	$i = 1, \ldots, L.$
Histogram of the signal	h_i	for	$i = 1, \ldots, L.$
Set of gray levels allowed in the quantized signal	g'_i	for	$i = 1, \ldots, K.$
The error when input gray level g_i is quantized to g'_j	$c_{i,j}$	for	$i = 1, \ldots, L,$ $j = 1, \ldots, K.$
The number of allowed gray levels in quantized signal	M		

Table 6.1: Notations used in optimal codebook selection problem.

For a fixed codebook, there is a closest codeword for any signal sample. If there is more than one codeword at the closest distance to a given signal sample, equalities can be solved in favor of any of these codewords and this will not affect the overall error in Eqn. (6.14). Hence, only one $\gamma'_{i,j}$ for $j = 1, \ldots, K$ can be h_i and other ones will be zero. These are necessary conditions to minimize the defined error function. Let $\gamma'_{i,j} = \gamma_{i,j}/h_i$, and $c'_{i,j} = c_{i,j}h_i$. In terms of these new variables, we are looking for a matrix γ' such that

$$\gamma'_{i,j} \in \{0,1\} \quad \text{for} \quad i = 1, \ldots, L, \quad j = 1, \ldots, K. \tag{6.15}$$

$$\Sigma_{j=1}^{K} \gamma'_{i,j} = 1. \tag{6.16}$$

$$\gamma'_{i,m_j} = 0 \quad \text{for} \quad i = 1, \ldots, L, \quad j = 1 \quad \ldots, K - M. \tag{6.17}$$

$$\min_{m_j \quad \text{for} \quad j=1,\ldots,K-M.} \Sigma_{i=1}^{L} \Sigma_{j=1}^{K} \gamma'_{i,j} c'_{i,j}. \tag{6.18}$$

So the problem becomes selecting the correct columns of γ' which has only one "1" in

every row and which has "1"s in only selected columns to minimize the cost.

The problematic constraints are equations in (6.17). We have to express the equations in (6.17) as linear (in)equalities in terms of $\gamma'_{i,j}$'s.

Let us define n_j as follows:

$$n_j = \underset{i=1,\ldots,L}{\text{argmin}} \; c_{i,j} \quad \text{for} \quad j = 1,\ldots,K. \tag{6.19}$$

Basically, g_{n_j} is the closest gray level to g'_j. If g'_j is in the optimal codebook, then g_{n_j} can be quantized to g'_j. If g_{n_j} is quantized to g'_j, then $\gamma'_{n_j,j}$=1 and other gray levels can be also quantized to g'_j, i.e., $\gamma'_{i,j} \in \{0,1\}$ for $i = 1,\ldots,L$. If g'_j is not in the optimal codebook, then $\gamma'_{n_j,j}$=0, and $\gamma'_{i,j} = 0$ for $i = 1,\ldots,L$. Thus, the following inequalities are necessary for an optimal codebook assignment if we remember that $\gamma'_{i,j} \in \{0,1\}$:

$$\gamma'_{i,j} \leq \gamma'_{n_j,j} \quad \text{for} \quad i = 1,\ldots,L, \quad j = 1,\ldots,K. \tag{6.20}$$

But, inequalities in (6.20) are not the sufficient conditions to satisfy the equations in (6.17) because they don't restrict the number of codewords in a given codebook. Let us consider the following equation:

$$\Sigma_{j=1}^{K} \gamma'_{n_j,j} = M. \tag{6.21}$$

This equation will make sure that there are only M $\gamma'_{n_j,j}$ which are nonzero, hence it will limit the number of codewords in a codebook. On the other hand inequalities in (6.20) will make sure that no signal sample is quantized to a codeword which is not in the codebook. Thus, equations in (6.17) can be replaced with inequalities in (6.20) and equation in (6.21).

6.7 Equivalent integer linear programming problem

After converting the problematic equations in (6.17), we can express the codebook selection problem in terms of $\gamma'_{i,j}$'s as it is shown in Table 6.2. The minimization problem shown in the table is a linear integer programming problem in terms of $\gamma'_{i,j}$'s!

Thus we have converted the optimal codebook selection to an integer linear programming problem. We know that the solution to a linear programming problem can be found in finite steps and the solution is globally optimum [19]. Remember that the finite nature of the

$$
\begin{array}{c}
n_j = \underset{i=1,\ldots,L}{\text{argmin}} \; c_{i,j} \quad \text{for} \quad j = 1,\ldots,K. \\
\Sigma_{j=1}^{K} \gamma'_{n_j,j} = M \\
\gamma'_{i,j} \leq \gamma'_{n_j,j} \quad \text{for} \quad i = 1,\ldots,L, j = 1,\ldots,K. \\
0 \leq \gamma'_{i,j} \leq 1 \quad \text{for} \quad i = 1,\ldots,L, j = 1,\ldots,K. \\
\Sigma_{j=1}^{K} \gamma'_{i,j} = 1 \quad \text{for} \quad i = 1,\ldots,L. \\
\text{Minimize} \quad \Sigma_{i=1}^{L} \Sigma_{j=1}^{K} \gamma'_{i,j} c'_{i,j} \\
\gamma'_{i,j} \in \mathcal{Z}.
\end{array}
$$

Table 6.2: Equivalent linear integer programming problem.

input signal distribution and finite number of possible codewords are exploited in converting the optimal codebook selection to a linear programming problem.

6.7.1 Special case of the codebook selection problem

When $L = K$ and $g_i = g'_i$ for $i = 1,\ldots,L$, we can show that

$$n_i = i \quad \text{for} \quad i = 1,\ldots,L$$

using the equations in (6.19). The equivalent problem is shown in Table 6.3.

$$
\begin{array}{c}
\Sigma_{i=1}^{L} \gamma'_{i,i} = M \\
\gamma'_{i,j} \leq \gamma'_{i,i} \quad \text{for} \quad i = 1,\ldots,L, j = 1,\ldots,L. \\
0 \leq \gamma'_{i,j} \leq 1 \quad \text{for} \quad i = 1,\ldots,L, j = 1,\ldots,L. \\
\Sigma_{j=1}^{L} \gamma'_{i,j} = 1 \quad \text{for} \quad i = 1,\ldots,L. \\
\text{Minimize} \quad \Sigma_{i=1}^{L} \Sigma_{j=1}^{L} \gamma'_{i,j} c'_{i,j} \\
\gamma'_{i,j} \in \mathcal{Z}.
\end{array}
$$

Table 6.3: Special case of the optimal codebook selection problem when $L = K$ and $g_i = g'_i$.

This problem is equivalent to keeping only the optimum M values of allowable L values of a signal in order to compress the signal by a factor of L/M.

6.8 Experimental results in optimal codebook selection

As an example, we have taken Lena image and tried to find the best 13 out of 256 gray levels such that the error is minimized when the image samples are quantized to these gray

levels. The program took 17 minutes on a Pentium II 450 MHz computer. The linear programming used a primal-dual algorithm. The best codebook turned out to be {43, 56, 73, 88, 101, 115, 128, 141, 152, 163, 176, 194, 211}.

With this codebook, the PSNR of the quantized image is 36.02dB. Notice that, the error is smallest when another codebook of the same size is used. For example, the error is bigger for the same size codebook in Section 6.5 (PSNR=34.97dB).

For comparison, we have used Lloyd Algorithm to find the best codebook of size 13. We have started the algorithm from 100,000 random codebooks, and we have taken the best of these optimized codebooks. The best codebook was {42.47, 54.85, 72.23, 88.20, 101.26, 115.91, 128.91, 141.20, 151.72, 161.68, 175.58, 193.51, 210.13}. The PSNR of the quantized images was $36.01dB$ with this codebook. Since we want the codewords to be one of 256 initial gray levels, we found the closest codebook to the optimized codebook which satisfy this constraint: {42, 55, 72, 88, 101, 116, 129, 141, 152, 162, 176, 194, 210}. Note that we cannot impose this constraint while optimizing with Lloyd Algorithm. The PSNR of the quantized image became $35.99dB$.

We have also obtained the best codebook of size 5 with the linear programming problem and Lloyd Algorithm. They gave the same codebook: {52, 95, 130, 159, 200}. The PSNR of the quantized image with this codebook was $28.24dB$.

The computation time needed for integer programming depends on the size of the possible quantized signal values. Usually, this time is more than what Lloyd Algorithm requires to find the best codebook. Notice that if we allow bigger sizes of possible quantized signal values, our algorithm will find better codebook. However, more computation time will be needed.

Even though this algorithm may not be suitable for real time application where an optimal codebook is required, it can be used for problems where the globally optimal codebook is required and time constraint is not important. It can also serve as a benchmark for other codebook selection algorithms.

In the last sections the signal used had only one dimension, but our algorithm can be applied to signals which belong to spaces of dimension greater than one.

6.9 Conclusion

In this chapter we have proposed an optimal algorithm to modify the histogram of an image to any desired histogram. The optimality criterion is chosen to be the minimization of MSE between the modified and the original image. However, other types of error criterion can also be used in the algorithm. We have shown that the histogram modification problem is equivalent to integer linear programming problem. We have also shown examples of histogram modification which can be useful for watermarking and compression applications. In the second part of the chapter, we have shown that the optimal codebook selection problem is equivalent to linear integer programming problem when the codewords can come from a finite set and initial signal samples come from a finite set. Since linear programming finds the globally optimum solutions, we can find the best codebook whereas Lloyd's Algorithm does not guarantee the globally optimum codebook.

Chapter 7 Conclusion

In this thesis we have discussed several different image processing problems. These include halftoning, inverse halftoning, effects of inverse halftoning on watermarked images and histogram modification problems.

In the second chapter, we optimized the dot diffusion halftoning algorithm. Even though dot diffusion offers more parallelism than a popular error diffusion algorithm, it has not received much attention in the past. This is partly because the noise characteristics of error diffusion method are generally regarded as superior. We have shown that dot diffusion method with a carefully optimized class matrix is very promising. The image quality is comparable to error diffusion, and the implementation offers more parallelism than error diffusion. Since the enhancement step prior to halftoning can be objectionable in some cases, we eliminated this by making the class matrix larger. A mathematical description of dot diffusion was derived which is particularly useful in inverse halftoning of dot diffused images. We also presented a wavelet-based inverse halftoning algorithm which works very well, even though the class matrix information is not used. Then we have shown that the dot diffusion algorithm can be easily modified to have the embedding property.

In the third chapter we have introduced a novel method for inverse halftoning which produces images of very good quality. The LUT method for inverse halftoning is extremely fast (no filtering is required) and the image quality achieved is comparable to the best methods known for inverse halftoning. The method does not depend on the specific properties of the halftoning method, and can be applied to any of them. An algorithm for template selection for LUT inverse halftoning is also introduced and the LUT method has been extended to inverse halftoning of color halftones.

In the fourth chapter we have introduced tree-structured LUT (TLUT) inverse halftoning in order to reduce the storage requirements of LUT inverse halftoning. In the second part of this chapter, a new LUT based halftoning method is discussed. The algorithm is capable of producing good quality halftones. In order to refine the halftones, we proposed tree-structured LUT halftoning. We have demonstrated that any halftoning algorithm can be simulated with this method. When this algorithm is trained on computationally complex

halftoning algorithms which produce very good halftones, it will reduce the computational complexity substantially.

In the fifth chapter we discussed the image watermarking and effects of halftoning on watermarked images. We proposed methods to improve watermark detection from halftoned images.

In the last chapter, we have proposed an optimal algorithm for histogram modification, so that the modified image has any desired histogram. We have shown that the histogram modification problem is equivalent to integer linear programming problem. We have also shown examples of histogram modification which can be useful for watermarking and compression applications. In the second part of the chapter, we have shown that the optimal codebook selection problem is equivalent to linear integer programming problem when the codewords come from a finite set and the initial signal samples come from a finite set. The well-known Lloyd's Algorithm to find a good codebook does not guarantee globally optimum solutions whereas our algorithm does because the equivalent linear programming problem can be solved exactly.

We conclude by naming some further research directions for the topics discussed in this thesis. One of them is finding a stable color halftoning algorithm which uses the human visual system. The existing color halftoning algorithms are either unstable or do not use the human visual system directly. The other research direction is to improve the LUT inverse halftoning method so that gray level input images like scanned images can be input to the algorithm. In the codebook selection problem, the computational complexity of the equivalent integer programming can be very high. A faster algorithm will be of great importance. Also, watermark embedding with histogram modification can be improved so that the watermarks are more robust against compression attacks.

Bibliography

[1] M. Alghoniemy and A.H. Tewfik, *"Image watermarking by moment invariants,"* Proceedings of IEEE ICIP 2000, Vancouver, Canada.

[2] J.P. Allebach and R.N. Stradling, *"Computer-aided design of dither signals for binary display of images,"* Applied Optics, Vol. 18, pp. 2708-2713, August 1979.

[3] J.P. Allebach, *"FM screen design using DBS algorithm,"* Proceedings of ICIP, Vol. 1, pp. 549-552, Lausanne, Switzerland, 1996.

[4] M. Analoui and J. P. Allebach, *"New results on reconstruction of continuous-tone from halftone,"* IEEE Intl. Conf. Acoust. Spech Signal Process., Vol. 3, pp. 313-316, San Francisco, CA, 1992.

[5] D. Anastassiou, "Neural net based digital halftoning of images," ISCAS, Vol. 1, pp. 507-510, 1988.

[6] D. Anastassiou, and S. Kollias,"Progressive halftoning of images," Electron. Let., Vol. 24, pp. 489-490, 1988.

[7] R.B. Ash, *Real variables with basic metric space topology,* IEEE Press, 1993.

[8] B. E. Bayer, *"An optimum method for two level rendition of continuous tone pictures,"* in Conf. Rec. IEEE ICC, 1973, pp. 26-11-26-15.'

[9] P.C. Chang, C.S. Yu, and T.H. Lee, *"Hybrid LMS-MMSE inverse halftoning technique,"* IEEE Trans. Image Processing, Vol. 10, No. 1, pp. 95-103, January 2001.

[10] D. Coltuc and P. Bolon, *"Robust Watermarking by histogram specification,"* Proceedings of EUSIPCO 2000, Finland.

[11] I.J. Cox, J. Killian, F.T. Leighton, and T. Shamoon, *"Secure Spread Spectrum for Multimedia,"* IEEE Trans. on Image Processing, Vol. 6, No. 12, December 1997.

[12] S. Dally, *"Subroutine for the generation of a two dimensional human visual contrast sensitivity function,"* Eastman Kodak Tech. Rep. No. 233203.

[13] I. Daubechies, *Ten lectures on wavelets,* Philedelphia, PA: SIAM, 1992.

[14] K. Denecker, S. Assche, P. Neve, and I. Lemahieu, *"Improved lossless halftone compression using a fast adaptive context template selection scheme,"* Proc. of IEEE Data Compression Conference, 1998.

[15] B. Evans, private communications.

[16] Z. Fan, *"Retrieval of images from digital halftones,"* ISCAS, pp. 313-316, May 1992.

[17] R. Floyd and L. Steinberg, *"An adaptive algorithm for spatial greyscale,"* Proc. SID, pp. 75-77, 1976

[18] J. Flussser and T. Suk, *"Pattern Recognition by Affine Moment Invariants,"* Pattern Recognition, Vol. 26, No. 1, pp. 167-174, 1993.

[19] J. Franklin, *Methods of mathematical economics.* New York, NY: Springer-Verlag, 1980, pp. 44-45.

[20] A. Gersho and R. M. Gray, *"Vector Quantization and Signal Compression,"* Kluwer Academic Publishers, 1992.

[21] R.C. Gonzalez, R.E. Woods, *"Digital Image Processing,"* Addison Wesley, 1993.

[22] S. Hein and A. Zakhor, *"Halftone to continuous-tone conversion of error-diffusion coded images,"* IEEE Trans. Image Processing, Vol. 4, 2, pp. 208-216, February 1995.

[23] M.K, Hu, *"Visual Pattern Recognition by Moment Invariants,"* IRE Trans. on Information Theory, Vol. 8, pp. 179-187, February 1962.

[24] A. K. Jain, *"Fundamentals of digital signal processing,"* Englewood Cliffs, NJ: Prentice Hall, 1989.

[25] D. Kacker and J. Allebach *"Joint Halftoning and Watermarking,"* Proc. of IEEE ICIP, Vancouver, 2000.

[26] R. A. Vander Kam, P. A. Chou, and R. M. Gray, *"Combined halftoning and entropy constrained vector quantization,"* in SID Digest of Technical Papers, Seattle, WA, pp. 223-226, May 1993.

[27] T. Kite, N.D. Venkata, B. Evans, and A.C. Bovik, *"A high quality, fast inverse halftoning algorithm for error diffused halftones,"* Proceedings of ICIP, Chicago, IL, 1998.

[28] D. E. Knuth, *"Digital halftones by dot diffusion,"* ACM Tr. on Graphics, Vol. 6, pp. 245-273, October 1987.

[29] S. Kollias, and D. Anastassiou, *"A progressive scheme for digital image halftoning, coding of halftones, and reconstruction,"* IEEE J. Select. Areas Commun., Vol. 10, pp. 944-951, June 1992.

[30] G.C. Lagelaar, I. Setyawan, and R.L. Lagendijk *"Watermarking Digital Image and Video Data,"* IEEE SP Magazine, Vol. 17, No. 5, pp. 20-46, September 2000.

[31] D.L. Lau, G.R. Arce, and N.C. Gallagher, *"Green noise digital halftoning",* Proceedings of IEEE, Vol. 86, No. 12, pp. 2424-2444, December 1996.

[32] S. Lloyd, *" Least Squares Quantization in PCM,"* IEEE Tran. on Information Theory, Vol. 28, March 1982.

[33] D. Luenberger, *Optimization by vector spaces.* New York, NY: John Wiley and Sons, 1969, p. 52.

[34] J. Luo, R. de Queiroz, and Z. Fan, *"A robust technique for image descreening based on the wavelet transform,"* IEEE Trans. on Signal Processing, Vol. 46, No. 4, pp. 1179-1184, April 1998.

[35] S. Mallat and S. Zhong, *"Characterization of signals from multiscale edges,"* IEEE Transactions on Pattern and Machine Intelligence, Vol. 14, No. 7, July 1992.

[36] M. Meşe and P.P. Vaidyanathan, *"Image halftoning using optimized dot diffusion,"* Proceedings of EUSIPCO, Rhodes, Greece, 1998.

[37] —, *"Image halftoning and inverse halftoning for optimized dot diffusion,"* Proc. ICIP, Chicago, IL, 1998.

[38] —, *"A mathematical description of the dot diffusion algorithm in image halftoning, with application in inverse halftoning,"* Proceedings of ICASSP, Phoenix, AZ, 1999.

[39] —, *" Improved Dot Diffusion For Image Halftoning,"* Proceedings of IS&T's NIP 15: International Conference on Digital Printing Technologies, pp. 350-353, Orlando, FL, October 1999.

[40] —, *" Optimized halftoning using dot diffusion and methods for inverse halftoning,"* IEEE Transactions on Image Processing, Vol. 9, No. 4, pp. 691-709, April 2000.

[41] —, *""Properties of improved dot diffusion for Image Halftoning,"* accepted to Journal of Imaging Science and Technology.

[42] —, *"Look Up Table (LUT) inverse halftoning,"* Proc. of IEEE ISCAS, Geneva, June 2000.

[43] —, *"Template selection for LUT inverse halftoning and application to color halftones,"* Proc. of IEEE ICASSP, Istanbul, June 2000.

[44] —, *"Look up table (LUT) Method for inverse halftoning,"* to appear in IEEE Transactions on Image Processing.

[45] —, *"Tree-structured method for improved LUT inverse halftoning,"* Proceedings of EUSIPCO, 2000.

[46] —, *"Look Up Table Method for Image Halftoning,"* Proceedings of ICIP, 2000.

[47] —, *" Tree-structured Method for LUT inverse halftoning and for image halftoning,"* submitted to IEEE Transactions on Image Processing.

[48] —, *"Optimal Histogram Modification with MSE metric,"* to appear Proc. of IEEE ICASSP, Salt Lake City, May 2001.

[49] —, *"Effects of Inverse Halftoning in Watermarking,"* to appear in Proc. of fifth biennial international workshop on nonlinear signal and image processing (NSIP'01), Baltimore, 2001.

[50] —, *"Optimal histogram modification and optimal codebook selection,"* in preparation.

[51] T. Mitsa and K. J. Parker, *"Digital halftoning technique using a blue noise mask,"* J. Opt. Soc. Am. A, Vol. 9, No. 11, pp. 1920-1929, November 1992.

[52] R. Nasanen, *"Visibility of halftone dot textures,"* IEEE Transactions on Systems, Man and Cybernetics, Vol. 14, No. 6, pp. 920-924, December 1984.

[53] A. N. Netravali and E. G. Bowen, *"Display of dithered images,"* Proc. SID, Vol. 22, No. 3, pp. 185-190, 1981.

[54] M. A. Seldowitz, J. P. Allebach, and D. E. Sweeney, *"Synthesis of digital holograms by direct binary search,"* Appl. Opt Vol. 26, pp. 2788-2798, 1987.

[55] H. Stark ed., *Image recovery: Theory and application.* Orlando, FL: Academic Press, 1987.

[56] G. Strang and T. Nguyen, *Wavelets and filter banks.* Wellesley, MA: Wellesley-Cambridge Press, 1996.

[57] M. Y. Ting and E. A. Riskin, *"Error-diffused image compression using a binary-to-gray-scale decoder and predictive pruned tree-structured vector quantization,"* IEEE Trans. Image Proc., Vol. 3, pp. 854-858, 1994.

[58] R. A. Ulichney, *"Dithering with blue noise,"* Proc. of IEEE, Vol. 76, No. 1, pp. 56-79, January 1988.

[59] P.P. Vaidyanathan, *Multirate systems and filter banks.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[60] M. Vetterli and J. Kovacevic, *Wavelets and subband coding.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[61] P. Wah Wong, *"Adaptive error diffusion and its application in multiresolution rendering,"* IEEE Transactions on Image Processing, Vol. 5, No. 7, July 1996.

[62] P. W. Wong, *"Inverse halftoning and kernel estimation for error diffusion,"* IEEE Trans. Image Processing, Vol. 4, No. 4, pp. 486-498, April 1995.

[63] P. W. Wong, *"Entropy constrained halftoning using multipath tree coding,"* IEEE Trans. Image Processing, Vol. 6, No. 4, November 1997.

[64] Z. Xiong, K. Ramchandran, and M. Orchard, *"Inverse halftoning using wavelets,"* Proc. of Intl. Conf. on Image Processing, Lausanne, CH, Vol. I, pp. 569-572, 1996.

[65] D. C. Youla and H. Webb, *"Image restoration by the method of convex projections: Part I-Theory,"* IEEE Trans. Med. Imaging, Vol. MI-1, pp. 81-94, October 1982.

[66] http://www.systems.caltech.edu/mese/research/