National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

UNIVERSITY OF ALBERTA

Image Expansion Using Interpolation, Heuristics and Smoothing

by

Peng Xu

A Thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1991

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-66610-8

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: **Peng Xu**

TITLE OF THESIS: **Image Expansion Using Interpolation, Heuristics and Smoothing**

DEGREE: **Master Of Science**

YEAR THIS DEGREE GRANTED: **1991**

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) Peng Xu

Permanent Address:

Bell Northern Research Ltd.
P. O. Box 3511, Station C
Ottawa, Ontario
K1Y 4H7

Date: April 22, 1991

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Image Expansion Using Interpolation, Heuristics and Smoothing** submitted by **Peng Xu** in partial fulfillment of the requirements for the degree of **Master Of Science** in **Computing Science**.

Wayne A. Davis
Supervisor

....

Xiaobo Li

Z J Koles

Date: ....

Dedicated to:

Liqin, my parents and my younger brother.

Abstract

This thesis is concerned with the expansion of digital images. A new heuristic image expansion and smoothing algorithm is presented. This algorithm involves the classification of image features such as: edges, line or curve segments, and regions. Different methods are then applied to these features for expansion. Various interpolation techniques can be used for expansion of regions. A method using edge orientation is applied to edge expansion, and a similar method using line orientation is used for line expansion. To eliminate possible jaggies resulting from edge expansion, a heuristic smoothing method, is applied and reasonable results are obtained. Other smoothing methods are also discussed, and a comparison is presented. This new algorithm aims to produce more visually pleasing expanded images, i.e., to eliminate the blocky appearance and blurring to produce "smooth" expansions.

Acknowledgement

First of all, I would like to express my deepest thanks to my supervisor, Dr. Wayne A. Davis, for his guidance, support, and encouragement throughout this research.

I would also like to thank the member of my examining committee, Dr. Xiaobo Li, for his helpful suggestions and discussions on some of the problems encountered in the course of this research.

Thanks are also due to the other members of the committee, Dr. Barry Joe, for his patient guidance and great effort in my revision of this thesis, and Dr. Zoly Koles, for his helpful comments and suggestions.

Special thanks to my parents and my younger brother, for their love and encouragement throughout the process of writing this thesis. Without their support, the completion of this thesis would not have been possible.

Special thanks should be expressed to Miss Liqin Dong, for her great help and encouragement during my final defense and revision of this thesis.

Thanks should also go to Mr. Changsong Sun and Mr. Jiangong Zhao, for their help during this research.

This research was financially supported by the Department of Computing Science, University of Alberta, and the National Sciences and Engineering Research Council of Canada.

Table Of Contents

Chapter **Page**

1. Introduction 1
  1.1. Problem Description 1
  1.2. Fundamentals, Definitions 2
  1.3. Problem Definition: Aliasing 4
  1.4. Survey of Previous Work 6
  1.5. Overview of *Heuristic Expansion and Smoothing Algorithm* 10
  1.6. Thesis Organization 13

2. Expansion Using Interpolation 14
  2.1. Scaling 14
  2.2. Interpolation in the Frequency Domain 15
    2.2.1. Definitions, and Useful Properties of Fourier Transforms 15
    2.2.2. Interpolation and Sampling 16
  2.3. Interpolation in the Spatial Domain 19
    2.3.1. Bilinear Interpolation 20
    2.3.2. Higher Order Interpolation 24
  2.4. Summary 25

3. Heuristic Expansion Using Edge Orientation Information 26
  3.1. Edge and Line Differentiation 26
  3.2. Edge Orientation Acquisition 34
    3.2.1. Edge Orientations in Horizontal and Vertical Directions 36
    3.2.2. Edge Orientations in Diagonal Directions 38
    3.2.3. Edge Orientation Changes at the Central Pixel 40
    3.2.4. Other Cases for Edge Orientation Detection 44
  3.3. Edge Expansion 44
    3.3.1. Definition Used in the Edge Expansion Method 44
    3.3.2. Algorithm for the Edge Expansion Method 45
    3.3.3. Subroutine *full_assignment* 48
    3.3.4. Subroutine *half_assignment* 52
    3.3.5. Subroutine *quarter_assignment* 56
  3.4. Time Complexity Analysis for *Heuristic Expansion* 58
  3.5. Remaining Problems 59

4. Edge Smoothing Methods 61
  4.1. Problem Description 61
  4.2. Previous Edge Smoothing Methods 61
    4.2.1. Variable Neighborhood Averaging 62
      4.2.1.1. Description of Variable Neighborhood Averaging 62
      4.2.1.2. Time Complexity for Variable Neighborhood Averaging 62
    4.2.2. Weighted Averaging 62
      4.2.2.1. Description of Weighted Averaging 62
      4.2.2.2. Time Complexity for Weighted Averaging 64

4.2.3. Laplacian Smoothing 64
    4.2.3.1. Description of Laplacian Smoothing 64
    4.2.3.2. Time Complexity for Laplacian Smoothing 65
4.2.4. Directional Smoothing 65
    4.2.4.1. Description of Directional Smoothing 65
    4.2.4.2. Time Complexity for Directional Smoothing 66
4.2.5. Half Neighborhood Methods 67
    4.2.5.1. Description of Half Neighborhood Methods 67
    4.2.5.2. Time Complexity for Half Neighborhood Methods 68
4.2.6. Maximum Homogeneity Smoothing 68
    4.2.6.1. Description of Maximum Homogeneity Smoothing 68
    4.2.6.2. Time Complexity for Maximum Homogeneity Smoothing 69
4.2.7. Selective Neighbor Averaging 69
    4.2.7.1. Description of Selective Neighbor Averaging 69
    4.2.7.2. Time Complexity for Selective Neighbor Averaging 70
4.3. *Heuristic Smoothing* 70
    4.3.1. Description of *Heuristic Smoothing* 70
    4.3.2. Time Complexity for *Heuristic Smoothing* 73
4.4. Total Time Complexity for the Algorithm 73
4.5. Summary 74

5. Implementation, Results and Analysis **75**
    5.1. Implementation 75
    5.2. Results and Analysis 75

6. Conclusion **91**
    6.1. Overview of the Thesis 91
    6.2. Summary 91
    6.3. Possible Applications 92
    6.4. Future Research 92

Bibliography **93**

LIST OF FIGURES

Figure **Page**

1.1 The eight neighbors of p and p itself, $N_8(p)$ 3
1.2 Functional diagram of *Heuristic Expansion and Smoothing Algorithm* 12

2.1 Illustration of a linear system 16
2.2 Sampling in spatial and frequency domains 18
2.3 Illustration of bilinear interpolation 22
2.4 Pixel replication 23
2.5 (Bi)linear interpolation 23

3.1 Functional diagram for *Edge_Line_Differentiation Module* and *Edge Expansion Module* 27
3.2 A 3×3 window 28
3.3 An example for *#trans* and *epr* 31
3.4 A detected edge, where *max*(*epr*) = 5 and *#trans* = 1 32
3.5 A detected line segment, where *max*(*epr*) = 1 32
3.6 Examples for horizontal and vertical edge orientations, with *#4-neighbor*(0) = 1 and *d4-neighbor*(1) < 4 37
3.7 Examples for diagonal edge orientation change: $\pi/2$, or $3\pi/2$, with *#4-neighbor*(0) = 1 and *d4-neighbor*(1) = 4 38
3.8 Examples for diagonal edge orientations, with *#4-neighbors*(0) = 2 and *d4-neighbor*(1) > 1 39
3.9 Examples for perpendicular edge orientation change, with *#4-neighbors*(0) = 2 and *d4-neighbor*(1) = 1 40
3.10 Examples for edge orientation change: $\pi/4$, $3\pi/4$, or $5\pi/4$, with *#4-neighbors*(0) = 3 and *d4-neighbor*(1) = 1 42
3.11 Examples for diagonal edge orientation change: $\pi/2$ or $3\pi/2$, with *#4-neighbor*(0) = 3 and *d4-neighbor*(1) = 2 43
3.12 The central pixel in a patch 45
3.13 Edge expansion using *full_assignment*, when *#4-neighbor*(0) = 1 and *d4-neighbor*(1) < 4 50
3.14 Edge expansion using *full_assignment*, when *#4-neighbor*(0) = 2 and *d4-neighbor*(1) = 1 50
3.15 Edge expansion using *full_assignment* for textures 52
3.16 Edge expansion using *half_assignment*, when *#4-neighbor*(0) = 2 and *d4-neighbor*(1) > 1 54
3.17 Edge expansion using *half_assignment*, when *#4-neighbor*(0) = 3 and *d4-neighbor*(1) = 1 54

3.18 Edge expansion using *quarter_assignment*, when *#4-neighbor*(0) = 3 and *d4-neighbor*(1) = 2 57
3.19 Edge expansion using *quarter_assignment*, when *#4-neighbor*(0) = 1 and *d4-neighbor*(1) = 4 58

4.1 Mask corresponding to (4.1) 64
4.2 Mask corresponding to (4.2) 64
4.3 Pentagonal and hexagonal neighborhoods 69
4.4 Demonstration of *Heuristic Smoothing* 73

LIST OF PLATES

Plate **Page**

5.1 The Original Images: FIGURE, MOON, TEXT and BUSH 79
5.2 Results of Heuristic Expansion & Smoothing (Exp. Factor=3, 4) 80
5.3 Results of Heuristic Expansion Only (Exp. Factor=3, 4)........................ 81
5.4 Results of Bilinear Interpolation (Exp. Factor=3, 4) 82
5.5 Results of Heuristic Exp. and Var. Neighborhood Averaging (Exp. Factor=3, 4) 83
5.6 Results of Heuristic Exp. and Weighted Ave. (Exp. Factor=3, 4) 84
5.7 Results of Heuristic Exp. and Directional Smoothing (Exp. Factor=3,4) 85
5.8 Results of Heuristic Exp. and Half Neighborhood Method (Exp. Factor=3, 4) 86
5.9 Results of Heuristic Exp. and Max. Homogeneity Smoothing (Exp. Factor=3, 4) 87
5.10 Results of Heuristic Exp. and Selective Smoothing (Exp. Factor=3, 4)...... 88
5.11 Results of Pixel Replication 89
5.12 Results of Heuristic Exp. and Laplacian Smoothing (Exp. Factor=3, 4)..... 90

Chapter 1

Introduction

1.1. Problem Description

In image processing, there are many problems that still need to be solved. Image expansion is one such problem that will receive more and more attention. Usually, image expansion is regarded as the inverse of image compression. Formally, image expansion is the process of mapping an image of lower resolution to one of higher resolution, usually for display purposes and with the intent that the resulting image is visually acceptable and even pleasing. Surprisingly, there does not seem to be much work done in this area; however, a number of methods have been explored, in [Dav85] and [Atw89].

With the development of image processing technology, display devices will continuously improve in resolution. Therefore, a lot of images that were previously obtained (sampled) with devices of lower resolution will need to be displayed with new display devices of higher resolution. Furthermore it is not always feasible nor desirable to resample these images with higher resolution. Resampling requires more storage space for higher resolution images, which is not always acceptable. Even if resampling is acceptable, newer resampling devices could be too expensive to replace older resampling devices. So with image expansion and storage of lower resolution versions of images, *storage efficiency* can be achieved. And similarly, *data transfer efficiency* can be improved by transmitting lower resolution image data.

Another reason for image expansion comes from the area of browsing. Browsing is intended to offer the user the capability of examining information without having to retrieve the entire image. Actual image data in extremely high detail is often available, but for only a high cost or a relatively long retrieval time. On the other hand, the user has to know that the correct image is being requested before examining the details. Thus only images in compressed format are normally used for browsing, before the selection of a particular one for detailed retrieval. In this way, ***efficient browsing*** is possible.

For these reasons, image expansion methods are needed, preferably that do not require too much time and effort and in addition provide visually acceptable results.

1.2. Fundamentals and Definitions

In this section, the fundamentals and definitions of digital images will be introduced.

An image is considered to be a two dimensional function or array of intensity values, $f(x, y)$, where x and y denote the spatial coordinates, and the value of f at any point (x, y) is proportional to the brightness (or gray level) of the image at that point (an image refers only to a 2-dimensional grey level one, and 3-dimensional images are not considered).

The gray (or brightness) values of an analog image are real, non-negative and bounded, i.e. brightness cannot be arbitrarily large. And, the coordinates of the image function are bounded by a finite rectangular region (the boundary of the image).

A digital image is an image $f(x, y)$ which has been digitized both in spatial coordinates and in gray values.[1] The digitization partitions the analog image into compact and convex subsets (cells) of points from the analog image [Dav85]. Each such cell is then assigned a gray value which represents the intensity of the points in that cell of the analog image. So a digital image is usually represented as an $m \times n$ array, where m and n are the image sizes in horizontal and vertical directions respectively and the coordinates determine the location of each cell in the digital image and the corresponding matrix element $f(i, j)$ in the array represents the digitized value at cell (i, j). The matrix elements are called *pixels* or

---

[1] In the thesis, an image refers to a digital image, unless it specifically refers to an analog image by context.

picture elements. Usually only square grid (cells) are employed in digital images and digital images are normally square, i.e. $m = n$, which are assumed to be bounded in the range 2^6 to 2^{12}. In this thesis, it is assumed that $m = n$, thus only square images are considered. To be consistent with the convention of the imaging system on which all the algorithms in this thesis are implemented, it is also assumed that the image coordinate origin is at the upper left corner of the image. This convention is applied in the rest of the thesis.

The number of digitization levels used in digital images is usually in the range of $2^6 \sim 2^{12}$ [Dav85]. In this thesis, the number of digitization levels used is 2^8, for all images used. The choice of the number of digitization levels is an interesting research area. Please refer to [Bra78, Cas79, Gon87, Pra78, Ros82] for further discussion of the topic. In addition to the digitization of images using square grid (cells), triangular and hexagonal tessellations are discussed in [Ros82, Dan82].

p_3	p_2	p_1
p_4	p	p_0
p_5	p_6	p_7

Fig. 1.1. The eight neighbors of p and p itself, $N_8(p)$

For a square tessellation, each pixel p of the array has four horizontal and vertical neighbors, and four diagonal neighbors, except at the borders of the image. In this thesis, a neighboring pixel (or a neighbor) of a pixel p refers to a 8-neighbor of p inside the 3 × 3 window with p as the central pixel of the window. The 8-neighbors of a pixel p together with p are defined as $N_8(p) = (p, p_0, p_1, \ldots, p_7)$, where p_i, $0 \leq i \leq 7$, are located as shown in Fig.1.1. The 4-neighbors of p together with p are similarly defined as $N_4(p)=(p, p_0, p_2, p_4, p_6)$. With this notation, each neighbor p_i, $0 \leq i \leq 7$, is at the angle $45i^\circ$ measured counterclockwise from the positive x axis. A similar notation based on the spatial coordinates of p can also be used. If p is at (x, y), $N_8\{(x, y)\}=(p, p_0, p_1, \ldots, p_7)$, and $N_4\{(x,y)\}= (p, p_0, p_2, p_4, p_6)$, which will be used in the remainder of the thesis and the appropriate definition for each usage is governed by the context.

As mentioned before, the image expansion issue is closely related to the resolution of the display device. The resolution of a display screen of this type is usually defined as the number of distinct line pairs per inch. An alternative is the number of pixels per inch on a horizontal or vertical line. Although it only makes sense to describe resolution using the number of pixels per unit measure, it is convenient to use the term "number of pixels" to mean a measure for resolution and this is adopted in the rest of this thesis. Thus the values of m and n determine the resolution of the digital image.

In this thesis, the terms *resolution conversion* and *image expansion* are treated synonymously, as in [Dav85]. This is due to the possibility of spatially mapping the pixels of an appropriately expanded image onto the cells of a given higher resolution device to allow for the display of low resolution data on higher resolution devices. Thus, in this thesis, attention is focused on taking a set of gray level spatial data and producing another set that represents the original image at a desired scale by expansion methods.

In Section 1.3, an important definition, *aliasing*, is discussed, which is closely related to the topic of image expansion. In Section 1.4, a survey of previous approaches to allow better renditions of low resolution images on higher resolution display devices, including those that can resolve *aliasing*, are discussed.

1.3. Problem Definition: Aliasing

The term *aliasing* originates in the sampling of images. In order to perfectly reconstruct an image, a sufficient number of discrete points must be selected from the analog image. The sampling rate for reconstruction is determined by the sampling theorem [Dav85]. For details, refer to [Bra78, Pav82, Ros82]. Distortion occurs when an image is undersampled and this phenomenon is termed *aliasing*. For instance, if a page of text is undersampled it will appear as a sequence of white and dark bands when reconstructed [Pav82, Dav85]. In this thesis, the same term, *aliasing*, as in [Dav85], is also used to describe distortions produced by *pixel replication*, the simplest image expansion method. Thus this terminology is due to the occurrence of a blocky appearance as a consequence of either pixel replication or undersampling [Dav85]. In this thesis, *aliasing* mainly refers to distortion caused by pixel replication, unless it is specially governed by context.

When a low resolution image is displayed on a higher resolution screen, the size of the image relative to the display screen size is reduced. The simplest method to maintain the size of the displayed image is to use *pixel replication*, sometimes called *zooming*. This method replicates each pixel in both horizontal and vertical directions the proper number of times (expansion factor) before the image is displayed on the higher resolution screen. This direct replication results in a blocky appearance of the displayed image. Furthermore, the continuities of edges, lines and textures of regions in the image are interrupted. In this case, *aliasing* occurs.

Aliasing is especially noticeable when an image contains many lineal features. Mathematically, lines and curves have zero thickness; however, their discrete (digital) display representations cannot be less than a pixel thick. As a result of the discreteness of display devices and approximations involved in drawing lines or curves, slanted edges or lines can have a staircase or blocky appearance. When an edge or a line is zoomed, each pixel is mapped into a block of pixels so that jaggies are magnified.

The problem of *aliasing* can be solved by finding "smooth" expansions of the given images [Dav85]. As mentioned previously, pixel replication constitutes a piecewise constant expansion and *aliasing* arises from this approach. It is this effect that a *smooth* expansion method tries to overcome when a low resolution image, say *LRI*, is displayed on a high resolution screen. To illustrate this, let *LRI* be an image with a resolution of *r* lines/inch in each of the coordinate directions and *D2* be a display screen with *k* times the resolution of the data in both directions. If a *k* times bidirectional *smooth* expansion of *LR1* is available, then by spatially mapping each pixel in the expansion to a cell (a $k \times k$ square patch) on *D2*, a smooth rendition of *LR1* is obtained.[2] In the next section, a survey of previous work in various expansion methods is made.

1.4. Survey of Previous Work

In this section, a survey of the previous work done in image expansion is made. In [Dav85], image expansion methods are classified into the four categories:

Pixel Replication: The pixels of the original image are mapped onto the patches in the expanded image, the size of the patch being determined by the expansion factors, in the two coordinate directions, respectively. This approach is usually followed by local corrections, usually smoothing methods. In [Dav85], many smoothing methods are applied and compared.

Interpolation: Pixels of the original image are mapped onto the expanded image and then the intervening pixels are interpolated with various interpolation methods. This approach is intensively investigated and many test results are presented and analyzed in [Dav85].

Defocusing: As in the first method, pixels are first mapped onto the patches in the expanded image. The expanded image is then transformed into the frequency domain, where certain frequencies are filtered out (usually high spatial frequencies). Finally, the filtered image is transformed back into the spatial domain. There is a great amount of literature dealing with various methods for the transformation between spatial and

---

[2] This kind of expansion method is directly manipulated with the image functions, thus is called a method in *spatial domain*. Alternatively, there are some methods which are manipulated with some transformations, e.g., Fourier Transforms, which transfer the image functions into functions of frequency variables, thus are called methods in *frequency domain*.

frequency domains, such as the *Fourier Transform*. Please refer to [Dan82, Dav85, Gon87, Pra78] for more detail.

Image Analysis: This involves the identification of image features such as edges and storing the information in a scale-independent or object-oriented format. Expansion will then be easily accomplished since the information is independent of scale. The method in [Atw89] belongs to this category.

It should be noted that the methods in the first, second, and fourth categories are methods in the spatial domain. And the methods in the third category are methods in the frequency domain.

The methods in the first and second categories can be considered as mappings of the set of pixels and their adjacent relationships of the original image *LRI* to a new set of cells (patches) on the higher resolution device, by initially leaving the intervening values empty and then performing a weighted average, e.g. bilinear interpolation, or some other schemes based on the intensities in the respective neighborhood, e.g. spatial smoothing methods. This strategy is possible because each pixel is not independent of the adjacent pixels and the nature of a pixel can be estimated by the values of the pixels lying in its neighborhood. Actually, the new algorithm discussed in this thesis, which is presented in the next three chapters, is also based on this assumption.

Spatial smoothing methods that are used to reduce the presence of noise in an image represent the local correction methods (after pixel replication is applied) in the first category. Usually, smoothing operations involve evaluating a number of predicates with local information as parameters and the assigned intensity level is dependent on the outcome of these predicates. Many spatial smoothing methods are applied in [Dav85].

For the methods in the fourth category, a diversity of edge detection and segmentation techniques in [Lev82, Ros72] can be applied. As indicated before, the detected features in an image are usually stored as a structure, i.e. its data representations encapsulate the form of the features. For example, chain codes can be used to represent contours, and quadtrees can be used to store an image with a large number of regions [Sam80]. Expansion can then be easily accomplished since all the information has been translated into a scale-independent form. This approach requires extensive preprocessing, e.g. fitting curves to contours.

According to [Dav85] and [Atw89], the methods in the first three categories tend to blur edges during image expansion, and the method in the first category also tends to make a blocky appearance in the expanded image (For pixel replication followed by smoothing methods, the blurring is caused by the smoothing, and the blocky appearance resulting from pixel replication cannot be eliminated completely by the smoothing). The fourth method requires extensive preprocessing, although it shows the best promise.

[Dav85] gives a comprehensive report on the various interpolation methods in both the spatial and frequency domains and smoothing methods for expanding images, and many test results are presented. Interpolation methods, such as bilinear interpolation, can be used to obtain very smooth image expansion. However, these methods can blur image features, such as edges and lines, thus they are only appropriate for the expansion of the interior of homogeneous regions.

In [Atw89], a heuristic method is explored for image expansion. First, all crack edges[3] are detected, and sorted according to the crack edge values[4]. Then the sequences of crack edges that form connected paths through the image, called *strokes*, are set up, by repeatedly selecting the crack edge of current highest value and following its strongest neighboring crack edge, until all possible strokes are detected throughout the image. This method is based on the assumption that the strokes reveal the most obvious lineal features in the image. Next, each stroke is divided into *substrokes* and each substroke is mapped

---

[3] Because of the discreteness of digital images, a pixel has four *crack edges* as its boundaries with all of its 4-neighbor pixels. So a crack edge is the boundary between any two 4-connected pixels of sufficient difference in gray values.

[4] Suppose x and y are the coordinates of pixels that have a crack edge in common. The crack edge value is defined as $|f(x) - f(y)|$, where $f(x)$ and $f(y)$ are the gray values at x and y respectively.

onto the expanded image. The pixels on the regions are filled directly with interpolation, and the intervening pixels on the strokes are filled with a heuristic method which uses strokes as control points.

This method requires much preprocessing. The worst case of the time complexity for the entire processing is $O(k^2n^2 + n^4)$, where k and n stand for the expansion factor in x, y directions, and the sizes of the original image in x, y directions respectively [Atw89]. Furthermore, the use of crack edges to find strokes does not produce adequate results. Noticeable lineal features in the image do not seem to be followed, because diagonal measurements are not included in the crack edges [Atw89].

Other work has also been done to convert low resolution binary images for high resolution displays [Sto82]. The motivation lies in the reproduction on higher quality printers, of text data obtained from coarse scanning devices, using enlargement techniques for binary images. Techniques include logical expansion: a pixel value is based on the outcome of operations involving logical values of the neighboring pixels. For example, the decision to fill a pixel may depend on the number of painted neighboring pixels according to some majority rule: a pixel is painted white if there are more white pixels than black ones. Similarly, some directional rule can be employed in which a pixel is painted if it is in the path of painted pixels. In [Dud73, Net80], *logical smoothing* can also be performed by selecting a set of pixel configurations as paths. Other schemes apply lookup tables to fill in the pixels by matching templates with neighboring pixels.

According to [Dav85], the topic of image expansion has been pursued indirectly in signal sampling issues, as an attempt to obtain perfect interpolation of the original images, that will be discussed in more detail in Chapter 2 for completeness of the thesis. Other similar literature includes [Hou78], which has used B-splines for interpolation and studied their performance with the Fourier Transforms. [Pra78] has presented a number of interpolating filters, i.e. the Fourier Transforms of interpolating functions, and discussed their errors.

The possibility of perfect interpolation based on the discreteness of digital images, is quite remote, based on the previous work [Dav85]. Thus all previous methods surveyed in this section employ certain approximations in order to obtained better expansion effects.

1.5. Overview of *Heuristic Expansion and Smoothing Algorithm*

As mentioned earlier, most of the previous image expansion methods cause either image blurring or aliasing, i.e. blocky appearance. In this thesis, a new algorithm applied in the spatial domain, aimed at a smooth expansion while maintaining the sharpness of expanded images, called *Heuristic Expansion and Smoothing Algorithm*, is presented. Satisfactory expansion results have been yielded with this algorithm. Both blurring and aliasing have been greatly eliminated, compared with the previous expansion methods. In this section, a brief overview is given for this algorithm. To illustrate, Fig. 1.3 shows its functional diagram.

The motivation of this algorithm lies in the application of the edge and line orientation information in their expansion to avoid their discontinuities and blurring in the expanded image. This algorithm classifies pixels of the original image into two categories: pixels in the interior of regions or background, and pixels on image features, e.g. pixels on edges, abbreviated as *edge pixels*, or pixels on lines, abbreviated as *line pixels*,[5] whose expansion are termed *edge* and *line* expansion respectively. The pixels on image features also include those on textures.

---

[5] For brevity, when the term *line* is used, it also applies to curve. This convention will be applied to the remainder of the thesis, unless otherwise specified. For the definitions of an edge, line and curve, please refer to Chapter 3.

For the pixel in each category, a corresponding module in the algorithm is called for its expansion. The new algorithm consists of four modules, in each of which a corresponding method is applied.

Interpolation Module, is used to expand the interior pixels of regions, for smoothness. Bilinear interpolation is applied in this module, because of its best visual effect among all the interpolation methods. Since this method tends to blur edges or lines, as mentioned earlier, it is only applied to the interior of regions.

To differentiate between edge and line pixels, *Edge_Line_Differentiation Module* is called, in which the method *Edge_Line_Differentiation* is applied. To avoid both blurring and blocky appearance, *Edge Expansion Module* and *Line Expansion Module* are used to expand the edge and line pixels respectively. The method applied in the former module is *Heuristic Edge Expansion*, and the method applied in the latter module is *Heuristic Line Expansion*[6]. The above four modules deal with image expansion, and are thus together called *Heuristic Expansion Module*. Similarly, the above four methods are together called *Heuristic Expansion*. To smooth possible jaggies on the edges in the expanded images, *Smoothing Module* is called, in which a new smoothing method called *Heuristic Smoothing* is applied.

Next, the function of this new algorithm is briefly discussed. It should be noted that its expansion operation is *sequential* since it processes each pixel in the original image row by row, and from the upper left corner to the lower right corner of the image, using 3×3 window operation with the pixel being processed as the central pixel of the window.

If the gray level difference inside the 3×3 window does not exceed a predefined threshold, say T, the pixels inside the window are treated as the interior pixels of a region, and *Interpolation Module* is called to expand the central pixel of the window.

---

[6] This technique is not implemented in this thesis, because of the strong restrictions needed for its implementation. The issue is discussed in more details in Chapter 3.

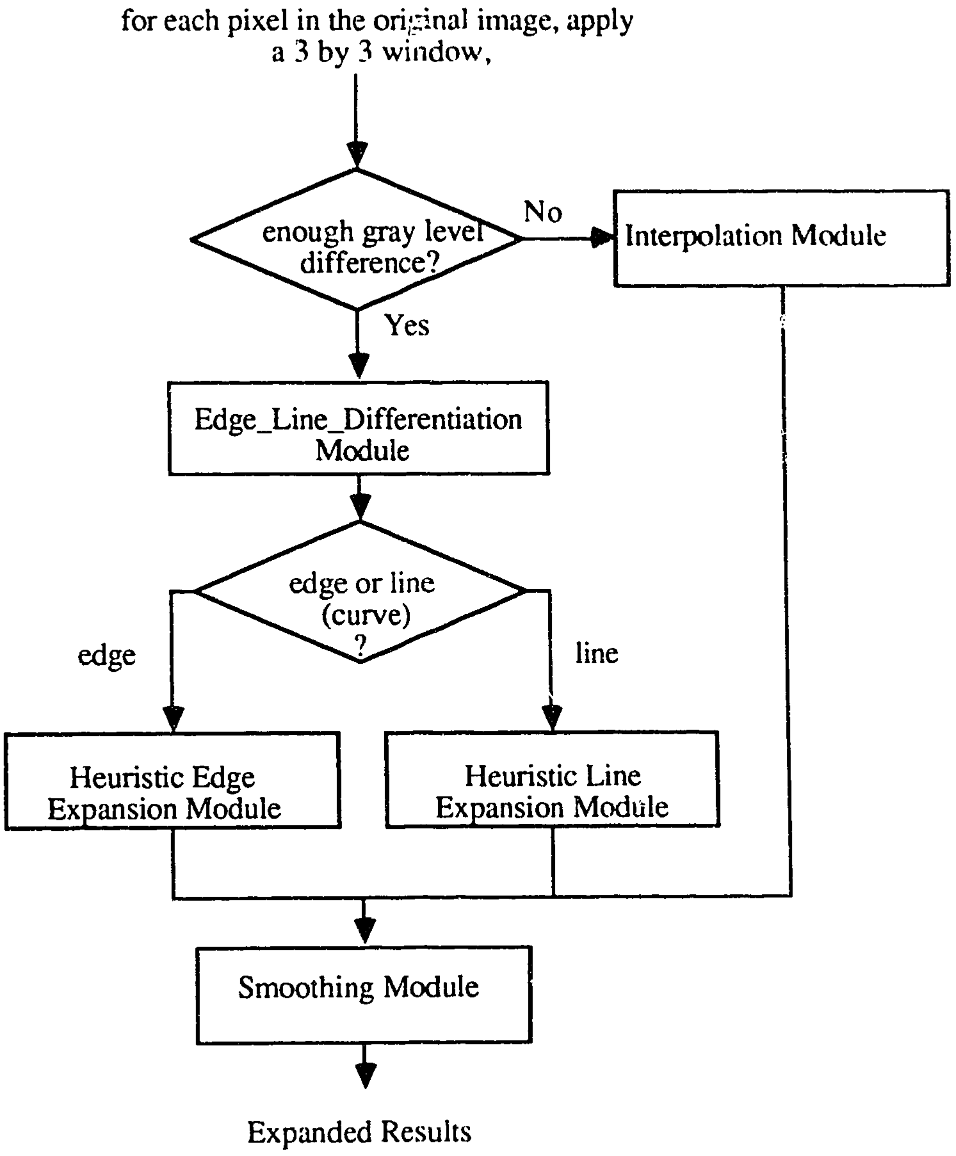

for each pixel in the original image, apply a 3 by 3 window,
enough gray level difference?
No
Interpolation Module
Yes
Edge_Line_Differentiation Module
edge or line (curve) ?
edge
line
Heuristic Edge Expansion Module
Heuristic Line Expansion Module
Smoothing Module
Expanded Results

Fig. 1.2. Functional diagram of *Heuristic Expansion and Smoothing Algorithm*

Otherwise, edges or lines exist, thus *Edge_Line_Differentiation Module* is called to differentiate between the two image features. Then either *Heuristic Edge Expansion Module* or *Heuristic Line Expansion Module* is called to obtain the expanded results, according to the differentiation. After the image expansion, *Heuristic Smoothing Module* is then called to smooth the expanded image.

It should be noted that the new algorithm is *heuristic*. It is often the case that there are no optimum solutions for some problems because of their complexity, or the optimum solutions exist but are not feasible because of the high time complexity for their implementations, thus approximations, which are usually not *analytical* but *empirical*, are applied to obtain acceptable solutions. These approximations are usually defined as *heuristic* methods.

For edge and line detection and expansion in *Heuristic Expansion*, 3 × 3 window operation is applied and only approximations of the orientation information, i.e. multiples of $\pi/4$ are obtained, for the sake of the time complexity. The approximations are based on the human empirical knowledge. For *Heuristic Smoothing*, there are also some approximations based on such knowledge. That is why this algorithm is *heuristic*.

1.6. Thesis Organization

This thesis consists of six chapters. In Chapter 2, bilinear interpolation, together with other various interpolating methods in [Dav85] are discussed. These methods can be used to obtain smooth expanded images. For completeness of this thesis, interpolation methods in the frequency domain are also briefly discussed.

Chapter 3 is concerned with the edge expansion method *Heuristic Edge Expansion*, applied in the *Edge Expansion Module*. *Heuristic Smoothing*, the method applied in the *Smoothing Module*, presented and discussed together with other smoothing methods of [Dav85] in Chapter 4, is employed to smooth possible jaggies on edges resulting from *Heuristic Expansion*.

Chapter 5 deals with implementation, results and analysis, in which the implementation issue is discussed, the test results of image expansion using the new algorithm and some methods mentioned earlier are presented, and the result analysis is made. In Chapter 6, conclusions are made and the future direction in this field is discussed.

Chapter 2

Expansion Using Interpolation

This chapter deals with expansion using interpolation, i.e., various methods which can be applied in *Interpolation Module*. First, the concept of scaling is introduced and discussed in Section 2.1, as the most basic concept for image expansion. Section 2.2 gives a brief discussion of interpolation in the frequency domain (which is not directly related to the thesis topic, but is included for completeness). To apply the best interpolation method in *Interpolation Module*, various interpolation methods presented and tested in [Dav85] for image expansion are examined in Section 2.3. Unlike in [Dav85], these methods are to be applied only to the interior pixels of the regions, in order to avoid image blurring. Edge expansion will be discussed in next chapter.

2.1. Scaling

When an object in an image is well defined (i.e. it is homogeneous), it can be described by a set of vectors, e.g. by the chain code of its contours. Expansion of such an object can be made by replicating each link in the chain code by the expansion factor [Dav85]. Similarly, regions can be expanded equally in both horizontal and vertical directions by first expanding the contours and then filling in the regions. Obviously, this is based on the assumption that the gray values of the pixels on the contours are identical. In many cases, this is not true and a gap filling algorithm is required; linear interpolation can be applied to the contours and bilinear interpolation for the interior pixels of the regions.

When the end pixels of the contours are described in horizontal and vertical coordinates, they can be scaled, relative to the origin, by S_x along the x-axis and by S_y along the y-axis into new pixels [Dav85]. Then similar schemes can be applied to the interior of the regions.

In practice, objects in an image are not always well-defined, and very often their contours can not be identified, e.g. the objects have complex textures. Interpolation offers a more general approach than the scaling techniques. This will be investigated in next two sections.

2.2. Interpolation in the Frequency Domain

To discuss the interpolation in the frequency domain, it is necessary to examine the problems connected to sampling. To facilitate the above discussion, a brief survey of the Fourier Transform is first made, from [Dav85, Bra78]. This section is used to justify the employment of the interpolation methods in the spatial domain.

2.2.1. Definitions and Useful Properties of Fourier Transforms

The Fourier transform of a one-dimensional function $f(t)$ [Bra78] is defined as

$$\mathrm{F}\{f(t)\} = F(s) = \int_{-\infty}^{\infty} f(t)e^{-i2\pi st}dt \tag{2.1}$$

The inverse Fourier transform of $F(s)$ is defined as

$$f(t) = \mathrm{F}^{-1}\{F(s)\} = \int_{-\infty}^{\infty} F(s)e^{i2\pi st}ds \tag{2.2}$$

where i is defined as $\sqrt{-1}$.

There are many properties of Fourier transform. Here only a couple of the most useful ones are presented.

The Uniqueness Theorem states that the Fourier transform $F(s)$, or the *spectrum*, for any function $f(t)$, is unique and vice-versa. The existence of the integrals in the transform equations is not considered to be a major problem in practice, since in reality, most functions satisfy the criteria for existence [Dud73].

The Convolution Theorem is the most important property [Dav85], which is as follows:

$$\mathbf{F}\{f(t) * g(t)\} = F(s)G(s), \tag{2.3}$$

$$f(t)g(t) = \mathbf{F}^{-1}\{F(s) * G(s)\} \tag{2.4}$$

where $F(s)$ and $G(s)$ are the Fourier transforms of $f(t)$ and $g(t)$ respectively, and * is a *convolution* operator. Please refer to [Dav85] for the details of *convolution*. The theorem relates two operations, one in the frequency domain, and the other in the spatial domain and implies that an expensive operation in one domain can be performed more economically in the other for the same effect.

2.2.2. Interpolation and Sampling

In this section, the theory behind sampling is reviewed. This is to justify the necessity for the approximation of interpolation techniques, in both spatial and frequency domains.

Most image processing problems are modeled as linear shift invariant systems, as shown in Fig. 2.1 [Dav85]. Please refer to [Dav85] for details about the conditions required for such systems as well as their characteristics.

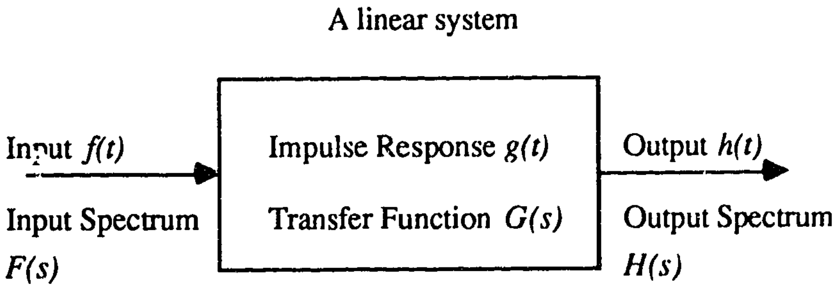


Fig. 2.1. Illustration of a linear system

In general, the Fourier transform of the impulse response $g(t)$ is called the *transfer* function $G(s)$. $F(s)$ and $H(s)$ are referred to as the *spectrum* of the input and output signals, respectively. The Convolution Theorem (2.3) and (2.4) allows such a system to be specified by either the impulse response function or the transfer function.

A function is band-limited at a frequency s_0 if its spectrum is zero in the intervals $(-\infty, -s_0]$, and $[s_0, +\infty)$, that is,

$$F(s) = 0, \text{ if } |s| \geq s_0. \tag{2.5}$$

According to [Dav85], the process of sampling $f(t)$ at equal intervals of τ can be modeled by multiplying $f(t)$ with $1/\tau III(t/\tau)$, i.e.,

$$h(t) = 1/\tau f(t) III(t/\tau) \tag{2.6}$$

where $1/\tau III(t/\tau)$ is an infinite impulse train with the interval τ and strength 1 (For more detail, please refer to [p15, Dav85]). This process zeros out $f(t)$ between the sampling points $n\tau$, with $n = 0, \ldots\ldots$, and preserves the values of $f(t)$ at the sample points [Cas79].

Using the Convolution Theorem, (2.6) implies [Dav85],

$$H(s) = F(s) * III(\tau s) \tag{2.7}$$ [1]

Since $III(\tau s)$ is a train of impulses of strength $1/\tau$ at equally spaced intervals of $1/\tau$, performing the convolution would then replicate the spectrum $F(s)$ in $H(s)$ at every $1/\tau$ interval along the s-axis from minus infinity to plus infinity [Dav85].

Thus, in order to recover the original signal (to get a smooth expansion of the original image), either $h(t)$ is interpolated for $f(t)$ in the spatial domain, or the copies of $F(s)$, except for the primary one located about the origin, are eliminated in $H(s)$ to recover $F(s)$.

One way to zero out the replicas of $F(s)$ is to multiply $H(s)$ with the rectangle function $R(s/2s_1)$, a transfer function unity in the interval $[-s_1, s_1]$ and zero everywhere else. As shown in Fig. 2.2, the value s_1 ranges from s_0 to $1/\tau - s_0$. Thus, with (2.7), the original signal is obtained by

---

[1] $III(\tau s)$ is the Fourier Transform of $1/\tau III(t/\tau)$. Please refer to [Bra78] for more detail.

$$F(s) = H(s)R(s/2s_l) \tag{2.8}$$

The transfer function $R(s/2s_l)$ is called an *interpolation filter*. Its corresponding impulse response is $2s_l sin(2\pi s_l t)/2\pi s_l t$ [Bra78]. By convention, $sin(\pi x)/\pi x$ is written as $sinc(x)$.

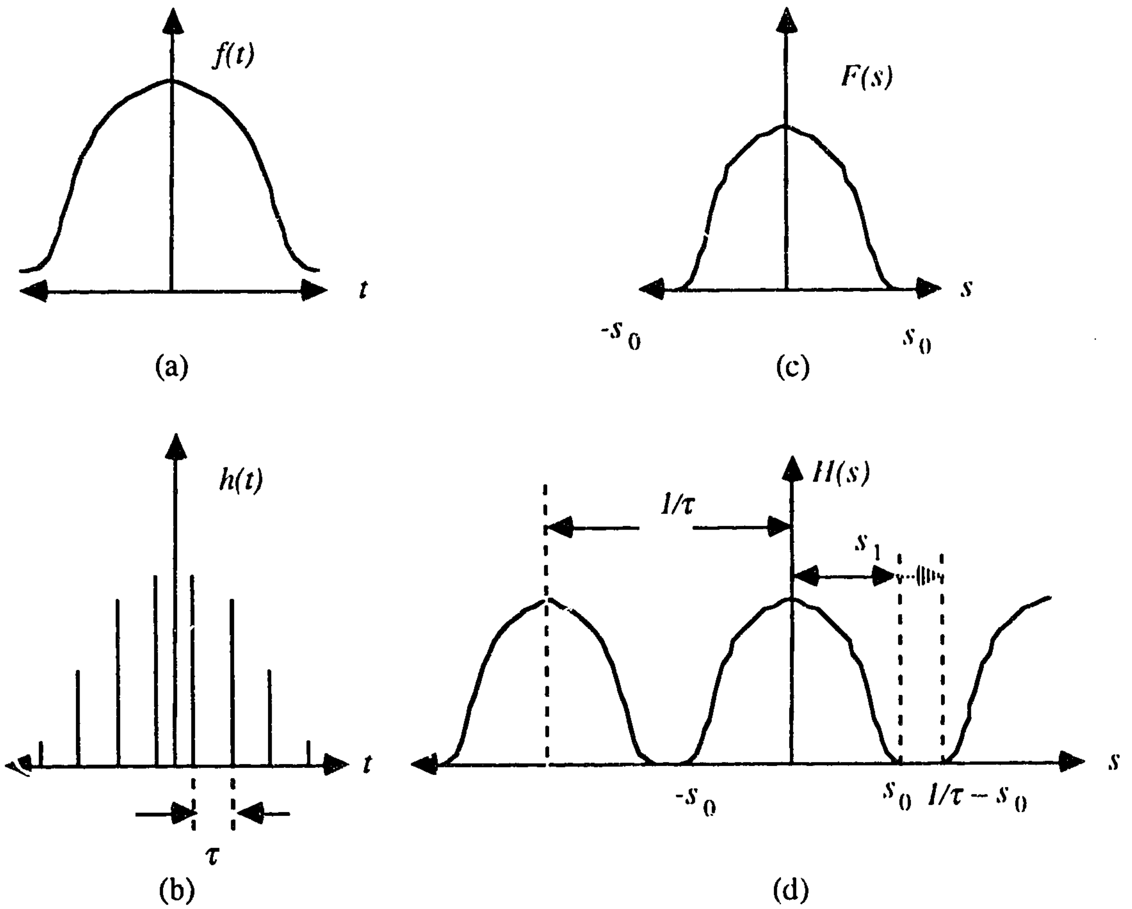


Fig. 2.2. Sampling in spatial and frequency domains

Equivalently, (2.8) implies, based on the convolution theorem,

$$\begin{aligned} f(t) &= h(t) \ * \ 2\, s_l\, sinc(s_l t) \qquad (2.9) \\ &= 2s_1 \sum_{m=-\infty}^{\infty} h(m)\, sinc(s_1(t - m)). \end{aligned}$$

This form of interpolation is often referred to as the *Cardinal Spline Interpolation* [Hou78]. There are several problems with this method, that hinder the perfect reconstruction with it from reality [Hou78]:

1. (2.9) behaves like an infinite degree polynomial whose operation is not local thus a computational problem results. If the lower and upper bounds of the summation in (2.9) are truncated, an oscillation known as *Gibb*'s phenomenon will show up in the interpolated signal [Dav85].

2. In many cases, images are not band-limited [Dav85].

Taking these into consideration, it should be noted that a perfect reconstruction by interpolation in the spatial domain is impossible [Dav85]. That is why all interpolation methods resort to approximations, which have been studied in [Dav85, Hou78, Pra78].

On the other hand, it is evident from (2.8) that $f(t)$ could be recovered by performing the inverse Fourier transform on $H(s)R(s/2s_1)$, that is, to recover $f(t)$ in the frequency domain. Whereas, it can be shown, that this is also difficult in digital image operations because of the discrete nature of digital images and the approximation employed in the calculation of the Fourier transforms $H(s)$ and $R(s/2s_1)$, and the inverse Fourier transform. Please refer to [Dav85] for more details.

These difficulties justify the pursuit of other interpolation methods in the spatial domain to recover the original signal from its samples, or more specifically, to achieve smooth expansion and to maintain continuity of the contents of the images. In the next section, these methods are examined in order to obtain the best one to be applied in *Interpolation Module.*

2.3. Interpolation in the Spatial Domain

In this section, various interpolation methods presented in [Dav85, Bra78] are discussed. Each interpolation function is modified to accommodate the discrete format of digital images.

From the point of view of numerical analysis, it is known that an increasing set of points can be interpolated by using polynomials of higher degree. The motivation of the employment of *piecewise* interpolation lies in the problems of a higher degree polynomial. The first problem is its undesired tendency to oscillate so that pinning down a polynomial at a few points for a slowly varying function may not produce a good approximation [Dav85, Bra78]. Second, since they are analytic, their global behavior is determined by their behavior in any interval; a change in the functional behavior in an interval affects everywhere. Piecewise interpolation can avoid these problems. As known, there is considerable literature on piecewise interpolation, e.g. [Bra78, Dav85, deB78, Spa74].

In this section, unlike in [Dav85], it is assumed that *a priori* knowledge of an image is known thus only the interior pixels of regions in the image are to be expanded by *Interpolation Module*, and the edges and lines in the image are not considered. This is done to avoid any edge or line blurring in the expanded images. Thus, some preprocessing of the classification between edges or lines and the interior of a region, which has been discussed in Section 1.5, is necessary before the interpolating operations.

The simplest interpolation method is bilinear interpolation, which is discussed in Section 2.3.1, along with its theoretical problem, *Mach* bands. Higher order interpolation methods are discussed in Section 2.3.2. By now the best interpolation method for digital image expansion is bilinear interpolation with regards to the visual effect and speed [Dav85]. For this reason, bilinear interpolation is applied in *Interpolation Module*.

2.3.1. Bilinear Interpolation

In [Dav85], the gray values of an image as a function of their positions are viewed as a surface so that interpolation of the surface corresponds to interpolating the gray values. The interpolation is called *piecewise* because the surface is divided into rectangular patches and the gray values in each patch are obtained by interpolating gray values of certain pixels.

In [Dav85], bilinear interpolation is defined as follows:

Suppose an image is defined to be an $m \times n$ array of gray values, $f(x, y)$, $0 \leq x < m$, $0 \leq y < n$. The expanded image $g(x, y)$, $0 \leq x < (m-1)S_x$, $0 \leq y < (n-1)S_y$, is computed as,

$$\begin{aligned} g(iS_x + k, jS_y + l) = & \left(\frac{S_x - k}{S_x}\right)\left(\frac{S_y - l}{S_y}\right) f(i, j) \\ & + \left(\frac{S_x - k}{S_x}\right)\frac{l}{S_y} f(i, j+1) \\ & + \frac{k}{S_x}\left(\frac{S_y - l}{S_y}\right) f(i+1, j) \\ & + \frac{kl}{S_x S_y} f(i+1, j+1) \end{aligned} \tag{2.10}$$

for $0 \leq i < m-1$, $0 \leq j < n-1$, $0 \leq k < S_x$, $0 \leq l < S_y$.

In this method, the expanded image is divided into $m \times n$ patches, each of which is of $S_x \times S_y$. Thus, the gray value of each pixel in the patch $\{(x, y) | x = iS_x, iS_x+1, iS_x+2, \ldots, iS_x+S_x-1, y = jS_y, jS_y+1, \ldots, jS_y+S_y-1\}$, for $0 \leq i < m-1$, and $0 \leq j < n-1$, is interpolated bilinearly.[2]

Equivalently, bilinear interpolation can be performed using linear interpolation in the x direction followed by the y direction (or vice versa) [Dav85]. In Fig. 2.3, the gray value at *P5* can be found by interpolating first for *P1* and *P3* (or *P2* and *P4*) and followed by interpolating between them.

As mentioned in Chapter 1, (bi)linear interpolation can be considered as a weighted averaging method. More specifically for the example in Fig. 2.3, the interpolation between *P1* and *P3* can be considered as a weighted average of the gray values at *P1* and *P3* with the distances between *P5*, the pixel of interpolation, and *P1* and *P3*, as the relative weights. Similarly, in the bilinear interpolation at *P5*, the distances between *P5* and (iS_x, jS_y), $((i+1)S_x, jS_y)$, $(iS_x, (j+1)S_y)$ and $((i+1)S_x, (j+1)S_y)$ act as the weights.

However, this simple approach will usually lead to blurred edges, if applied indiscriminately, as mentioned earlier, since the pixels lying between the original interior edge pixel and exterior edge pixel take on a weighted average of these two pixels [Dav85][3].

---

[2] For further details of bilinear interpolation, please refer to [Dav85].

[3] That is why in the new algorithm, only interior pixels of regions are to be expanded using interpolation.

To see this, consider Fig. 2.4, and Fig. 2.5 which depict the gray values as a function of the pixel locations of an expanded image.[4] In this case, pixels (iS_x, jS_y), $((i+1)S_x, jS_y)$, ..., $((i+8)iS_x, jS_y)$ are considered.

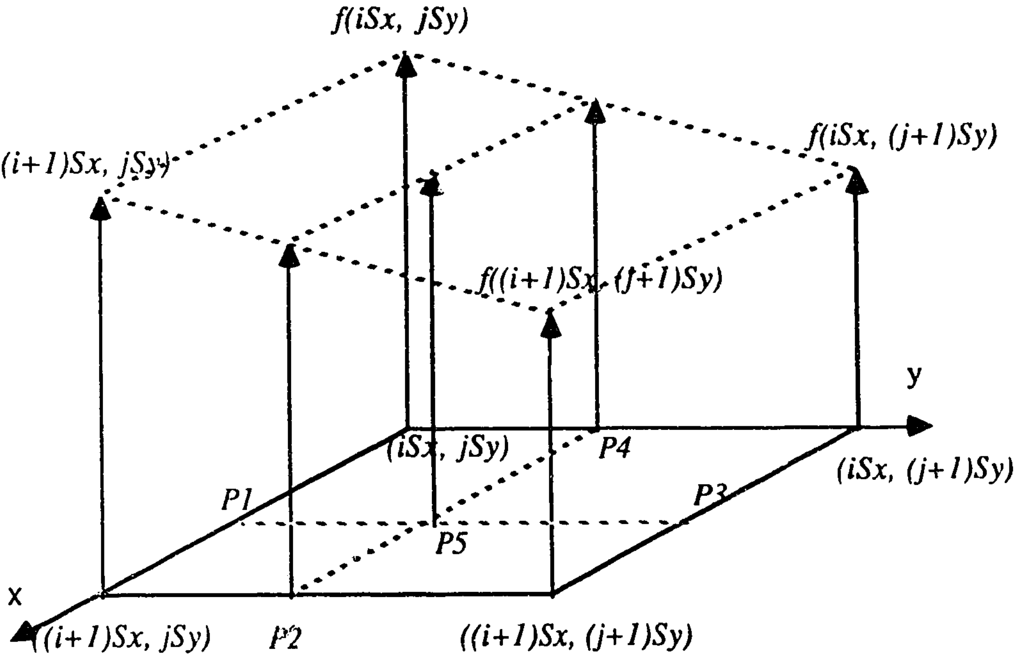


Fig. 2.3. Illustration of bilinear interpolation

Fig. 2.4 illustrates the case when expansion is performed by pixel replication. The blocky appearance is due to the discontinuity of the expanded curve. bilinear interpolation reduces the blocky appearance, as seen in Fig. 2.5.

Clearly from pixels $((i+2)S_x, jS_y)$, $((i+3)S_x, jS_y)$ and $((i+8)S_x, jS_y)$ of Fig. 2.5, the extent of the edge being blurred depends on the expansion factors, S_x and S_y, and the gray value difference in these corresponding adjacent pixels.

On the other hand, *Mach* bands, caused by the discontinuities of the first derivative (slope), manifest themselves at some intervening pixels in the expanded image, such as the positions like $((i+2)S_x, jS_y)$, $((i+5)S_x, jS_y)$, $((i+6)S_x, jS_y)$, and $((i+8)S_x, jS_y)$ [Dav85].

---

[4] This is quoted from [Dav85] by courtesy of the authors.

Since interpolation is done piecewise in patches of $S_x \times S_y$ pixels, these *Mach* bands cause the image to be slightly blocky [Fol83] [Dav85]. In Fig. 2.5, the solid line segments indicate the result of bilinear interpolation, which can be noted not to be C^1, i.e. the first derivative is not continuous. The solid curve in Fig. 2.5 indicates a higher degree interpolation (approximation) without *Mach* bands.

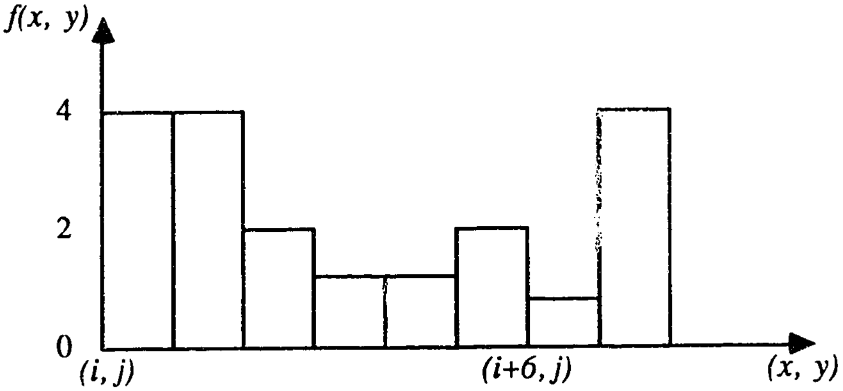


Fig. 2.4. Pixel replication

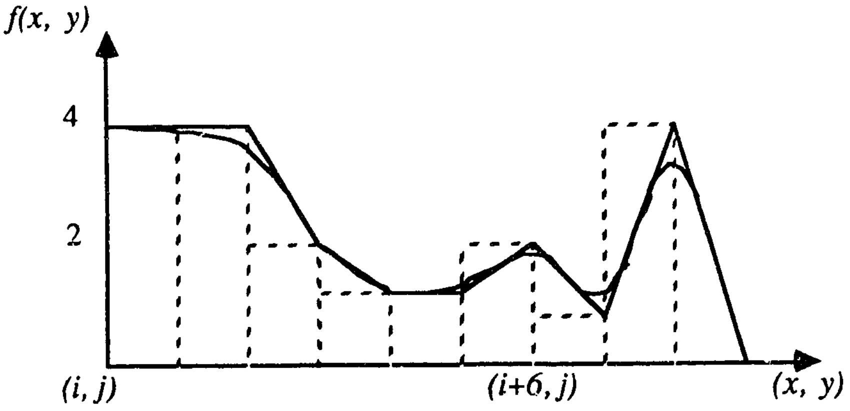


Fig. 2.5. (Bi)linear interpolation

Regardless of the above, the blocky appearance caused by *Mach* bands is usually much less noticeable than that caused by pixel replication (See Plate 5.4 and Plate 5.11 in Chapter 5 for example). A result analysis is made in [Dav85] to show the best visual effect of bilinear interpolation among all the interpolation methods. This phenomenon is due to the fact that there are still no good approximations for the derivatives necessary for the higher order interpolation methods. This issue will be discussed in next section.

2.3.2. Higher Order Interpolation

Higher order interpolation aims at providing an expanded image, the gray value function of which is at least C^1, to reduce any possible blocky appearance caused by *Mach* bands. A common choice for higher interpolating functions are the cubic polynomials. Unlike other higher degree polynomials, oscillations are controlled within a tolerable extent and no lower degree polynomial representation of curve segments provides continuity of the curve and the slope like cubic polynomials [Dav85].

Similar to bilinear interpolation, there are also some *piecewise* cubic interpolating polynomials, namely *Hermite*, *Bezier*, and *complete and natural splines*. Hermite and Bezier interpolation methods guarantee C^1, i.e. the continuity of the first derivatives, while complete and natural splines aim at even smoother expansion effect, with the gray value function being C^2, i.e. its second derivatives being continuous [Dav85].

Obviously, Hermite and Bezier cubic interpolation methods require the knowledge of the first derivatives. Unfortunately, the rate of change of gray values with regards to distance can only be estimated, due to the discreteness of the distances between pixels, thus there is considerable arbitrariness in determining the first derivatives. That is the major problem that keeps the Hermite and Bezier cubic interpolation from obtaining good visual quality of expanded images. With the similar approximation involved, the cubic spline interpolation methods have the similar problems, as Hermite and Bezier interpolation methods. Moreover, according to [Dav85], the available assumption of the boundary conditions for the second derivatives of the splines may be unrealistic. The next problem for all the above methods is that they also tend to blur the expanded images, as bilinear interpolation [Dav85].

2.4. Summary

Bilinear interpolation can be used to obtain smooth image expansion, but has the theoretical problem of *Mach* bands [Dav85]. Higher order interpolation aims at providing an expanded image, the gray value function of which is at least C^1, to reduce any possible blocky appearance caused by *Mach* bands. A common choice for higher interpolation functions are the cubic polynomials. However, the major problem for higher order interpolation is that, the limitation of the currently available approximation methods used for derivatives, as the result of the discrete nature of digital images, prevents higher order interpolation from yielding better visual effects than bilinear interpolation [Dav85]. This usually offsets their theoretical advantages over bilinear interpolation.

Although bilinear interpolation causes *Mach* bands, it is still deemed to be the best among all the interpolation methods for digital image expansion, in terms of computation speed and visual effect, unless better approximations for the derivatives mentioned earlier can be developed [Dav85]. Therefore bilinear interpolation is the choice for *Interpolation Module*.

On the other hand, most interpolation methods tend to blur edges and lines [Dav85]. As a result, interpolation is only used to expand interior pixels of regions. In next chapter, the edge expansion method, called *Heuristic Edge Expansion*, is discussed, which aims at smooth edge expansion while preserving the sharpness of edges.

Chapter 3

Heuristic Expansion Using Edge Orientation Information

This chapter deals with edge expansion. First, to facilitate edge expansion, *Edge_Line_Differentiation Module* for the differentiation between edges and lines is discussed in Section 3.1. Next, *Edge Expansion Module* for edge expansion is presented, to which *Heuristic Edge Expansion* is applied. The functional diagram of this Module, which consists of two routines of *Heuristic Edge Expansion*, is illustrated in Fig. 3.1. Section 3.2 discusses the routine *edge_orientation* for the acquisition of the edge orientation information with 3 × 3 window operation. Section 3.3 discusses *edge_expansion*, to which orientation information is applied for edge expansion without blurring and aliasing. Section 3.4 presents the time complexity analysis for *Heuristic Expansion*, the first half of the new algorithm. Section 3.5 deals with the remaining problems to be solved.

3.1. Edge and Line Differentiation

Edges and lines are most common image features. The new algorithm detects these image features with local operation, i.e. 3 × 3 window operation, to facilitate their expansion. In this thesis, the technique used for edge expansion is different from that used for line expansion. So, it is necessary to first differentiate between the two kinds of image features. Their definitions are as follows.

(1) *An edge*: it is the boundary of two adjacent, extensive regions. The gray level is relatively consistent in each of the two regions, and changes abruptly as this boundary

between the two regions is crossed. An ideal edge has a steplike cross section; a more realistic edge incorporates blurring and noise effects.

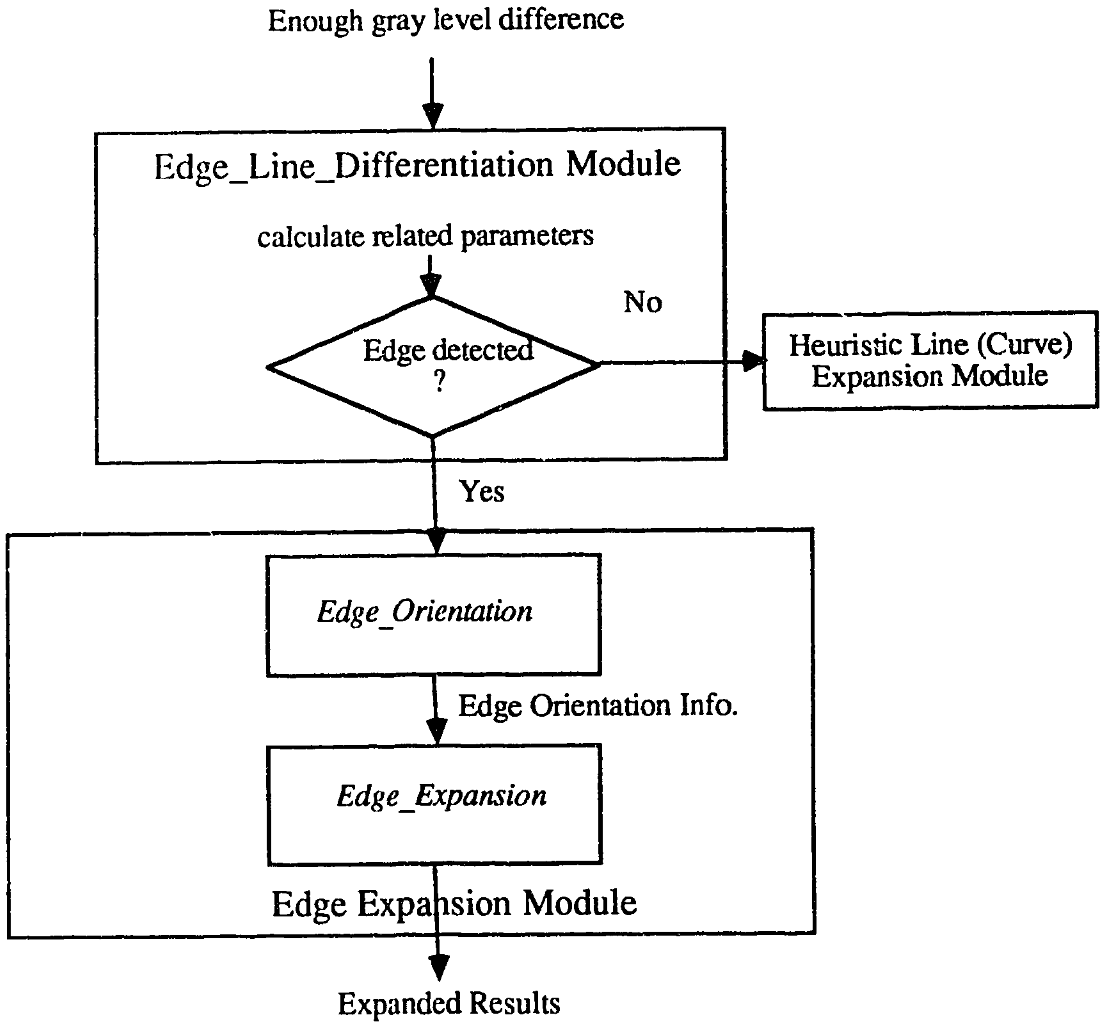

Enough gray level difference
Edge_Line_Differentiation Module
calculate related parameters
Edge detected ?
No
Heuristic Line (Curve) Expansion Module
Yes
Edge_Orientation
Edge Orientation Info.
Edge_Expansion
Edge Expansion Module
Expanded Results

Fig. 3.1. Functional diagram for *Edge_Line_Differentiation Module* and *Edge Expansion Module*

(2) *A line*: the gray level is relatively constant except along a thin strip, and this kind of feature has empty areas on both of its two sides. In cross section, this yields sharp spike. An ideal line should not have any physical width, but in digital images, widths exist for lines because of discreteness. In this thesis, all lines are assumed to be 1 pixel wide. A curve is only mathematically different from a line in that the first derivative of any point on a line is a constant, while that for a curve is a variable.

Next, the difference between a line and a line segment is discussed. A *line* means a *global* image feature, while a *line segment* is a *local* image feature detected inside the 3 × 3 window, which is a part of a line in the new algorithm. From the point of view in mathematics, a line has no ends, extending infinitely in certain two opposite directions, whereas, a line segment has two fixed ends.

In this section, the differentiation between edges and line segments using 3 × 3 window operation is discussed as follows. It should be noted that only line segments are involved instead of lines, the *global* image features, in the differentiation, since only the *local* 3 × 3 window operation is applied.

The purpose of the 3 × 3 window operation is to obtain the gray value difference information, i.e. the edge or line orientation information, if any, from the 9 pixels inside the 3 × 3 window. The information will then be used in the expansion of the current central pixel. When it is an edge or line pixel, the information is used to expand it without blurring and aliasing.

p_3	p_2	p_1
p_4	p	p_0
p_5	p_6	p_7

Fig. 3.2. A 3 × 3 window

In order to detect an edge or a line segment, the maximum and minimum gray values of the 9 pixels within the 3 × 3 window is obtained. Suppose that the 3 × 3 window is as shown in Fig. 3.2, thus its current central pixel is p. Then p's 8-neighbors together with p within the 3 × 3 window is $N_8(p) = (p, p_0, ..., p_7)$, as defined in Chapter 1. The gray values of the 9 pixels within the 3 × 3 window is defined as $N_8(f(p)) = (f(p), f(p_0), ..., f(p_7))$. For convenience in notation, $N_8(f(p))$ is abbreviated as $N_8(p)$. Thus the maximum and minimum values are denoted as $max(N_8(p))$ and $min(N_8(p))$ respectively. The first criterion for edge or line segment detection, which is used to detect the greatest gray value difference in the 3 × 3 window with a predefined threshold T, is

if $(max(N_8(p)) - min(N_8(p)) \geq T$ *then*

an edge or a line segment may exist.

This criterion is based on the assumption that an edge or a line segment can exist only if there is enough gray value difference inside the 3 × 3 window. If the above criterion is satisfied, then *Edge_Line_Differentiation Module* is called. Each of the 9 pixels in the window is classified into one of two categories: pixels on edges, lines or textures, denoted as 1, or background pixel, denoted as 0, using a 3 × 3 binary array c, i.e.,

if $f(p_i) \geq min(N_8(p)) + (max(N_8(p)) - min(N_8(p)))/2$ *then* $c(p_i) = 1 \; \{c(p_i)= 0\}$

else $c(p_i) = 0 \; \{c(p_i)= 1\}$, *for* $i = 0, ..., 7, 8$

where p_8 is referred to as p for the convenience of notation, and $f(p_i)$ denotes the gray value of p_i, and $c(p_i)$ is defined as the corresponding element in the 3 × 3 binary array. The assignments in { } are options applied when the gray values of the background pixels are higher than those of edge or line pixels. The above notation is used in the remainder of this thesis. This criterion is used to first differentiate between pixels on image features, such as edges, lines or textures, and the background pixels.

In this way, a corresponding 3 × 3 binary array is produced for the above differentiation using 1 and 0. This method is more convenient for edge or line segment detection and expansion, with regard to both notation and computational complexity. This binary 3 × 3 array is termed a 3 × 3 *binary window.*

It is noted that line expansion is different from edge expansion in that the width of a line is not changed after expansion, while usually an edge pixel is expanded in both axial directions, according to the expansion factor and edge orientation. Thus further processing is then needed to differentiate between edges and line segments, if any, inside the 3 × 3 window. The resulting 3 × 3 binary window is used for such differentiation.

It should also be noted that *a priori* knowledge of the existence of lines should be provided by the user before the differentiation, and it would be very difficult to differentiate between an edge and a line segment using 3 × 3 window operation, if any line segment were allowed to be more than one pixel wide.

However, the algorithm allows the expansion of images in which both edges and lines exist, based on the following assumptions:

1) Lines are only 1 pixel wide;

2) There are no line intersections[1];

3) No texture resembles lines, when the 3 × 3 window operation is carried out;

thus, the differentiation can be made between edges and lines, using the 3 × 3 window operation.

Before the differentiation is discussed, some terms and subroutines are introduced.

1) The number of transitions *#trans* in the binary window *c*:

The sequence of 0's and 1's determined by the eight neighboring pixels inside the binary window can be expressed in circular order as 0^+, 1^+, 0^+1^+, $0^+1^+0^+1^+$, $0^+1^+0^+1^+0^+1^+$, or 01010101, where $0^+(1^+)$ denotes a sequence of one or more 0(1). The number of transitions *#trans* is the number of 0^+1^+ subsequences in the sequence (See Fig. 3.3 for example[2], where the number of the solid or dashed arrows indicates the value of *#trans*).

2) edge pixel run *epr*:

The length of a consecutive run of 1's, measured circularly around the central pixel inside the binary window (See Fig. 3.3, where each shadowed part indicates an *epr*).

---

[1] Inside a 3 × 3 window, a line intersection may resemble some texture.

[2] In this chapter, any window in which pixels are denoted with only 1 or 0 refers to a binary window.

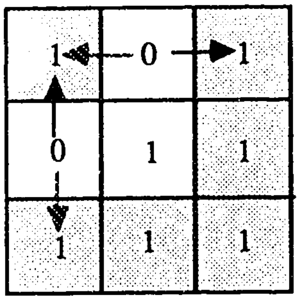
3) *max(epr)*: a routine which finds the maximum *epr* inside the 3 × 3 binary window, and then returns its value.

4) *edge_line_differentiation(p, c)*: edge and line differentiation routine;

5) *edge_orientation(p, c)*: routine for edge orientation acquisition, which also includes the texture detection;

6) *edge_expansion(p, c)*: routine for the expansion of *p*, which is an edge pixel or a pixel on some texture;

where 4) is a routine in *Edge_Line_Differentiation Module*, and 5) and 6) are routines in *Edge Expansion Module*.



#trans = 2; *max(epr)* = 5;
Fig. 3.3. An example for *#trans* and *epr*

Based on the assumptions 1) and 2), *max*(*epr*) = 1 holds whenever a line segment is detected, thus it can be assumed that an edge is possible when *max*(*epr*) ≥ 2 (See Fig. 3.4). With assumptions 1) and 2), it is also noted that the values of *max*(*epr*) and *#trans* can be used together with *a priori* knowledge of the existence of lines to detect a line segment, i.e. a line segment is detected when *max*(*epr*) = 1 and *#trans* =1 or 2 hold, and *a priori* knowledge exists (See Fig. 3.5). Since no line intersections are assumed to exist (assumption 2), *#trans* ≤ 2 always holds when a line segment is detected. So if *#trans* > 2 holds, it is assumed that part of some textures are detected, based on the assumptions 2) and 3).

The assumptions 2) and 3) are used to facilitate the detection of textures, thus to simplify the differentiation between lines and edges.

Using the above information, the differentiation routine *edge_line_differentiation* in *Edge_Line_Differentiation Module* is as follows.

0	0	0
1	1	1
1	1	1

Fig. 3.4. A detected edge, where *max*(*epr*) = 5 and *#trans* = 1

0	1	0
0	1	0
0	1	0

(a) *#trans* = 2

0	1	0
0	1	0
0	0	0

(b) *#trans* = 1

Fig. 3.5. A detected line segment, where *max*(*epr*) = 1

```
edge_line_differentiation(p, c)
p is the current central pixel inside the 3 × 3 window;
c is the corresponding 3 × 3 binary window;
{
calculate max(epr);    /* Calculate the epr of the maximum length */
calculate #trans;   /* Calculate the number of transitions */

 (1) c(p) = 0:        /* p, the central pixel, is a background pixel: no differentiation is
                         needed. */
        call the subroutine for background pixel expansion;
        /* full_assignment, a subroutine discussed in Section 3.3.3, is called for the
        expansion of the background pixel p */
(2) $c(p) = 1$ and $max(epr) \geq 2$:    /* An edge or part of some textures is detected. See Fig.
                                         3.3 and Fig. 3.4 for example */
        call edge_orientation(p, c);
        /* The subroutine calculates the edge orientation, or detects textures, if any. */

(3) $c(p) = 1$ and $max(epr) = 1$        /* The existence of line segments is possible, based
                                         on the assumption 1. */
 {
        (a)   a priori knowledge of lines exists and #trans $\leq 2$:
              /* A line segment exists (no line intersection is allowed). See Fig. 3.5
              for example. */
              call Line Expansion Module;
              /* This module is called for line expansion. */
        (b)   no such a priori knowledge or #trans $> 2$:
              /* p is on some texture. */
              call the subroutine for the expansion of the pixel on texture;
              /* full_assignment, is called for the expansion of p. */
        }  /* (3) $c(p) = 1$ and $max(epr) = 1$ */
}
/* edge_line_differentiation: classification between background pixel, edge pixel, and line
pixels */
```

In case (2), it should be noted that it is still not known at that moment whether the image feature detected inside the 3 × 3 window is an edge or part of some textures, thus further processing is needed later on, in the routine *edge_orientation*, using the value of *#trans*. It should also be pointed out that the correctness of the detection also depends on the proper selection of the predefined threshold T, thus the selection of T is very important in the edge and line differentiation and expansion.

During the thesis research, it is noted that *a priori* knowledge of line existence has to be provided for correct line detection, which is also restricted by several assumptions as mentioned previously, in order to implement the line expansion technique of the new algorithm. Furthermore, line detection is more easily subject to noise than edge detection, thus makes the differentiation between lines and edges very difficult with only local 3 × 3 window operations. These problems make the line expansion technique of the new algorithm very restrictive, thus nearly impractical for many gray level images, in which noise is usually inevitable.

Since the expansion issue of gray level images is the major concern in this thesis, and lines are much more frequent in binary images or computer graphics, details of the line expansion technique as well as its implementation are not attempted in this thesis but left for future research. *Edge_Line_Differentiation Module* is only presented in this section as a possible approach to the edge and line differentiation, necessary for edge expansion whenever both edges and lines exist in the image to be expanded.

In next two sections, *edge_orientation* and *edge_expansion* in *Edge Expansion Module* will be discussed respectively in details.

3.2. Edge Orientation Acquisition

In this section, the edge orientation acquisition issue is discussed and *edge_orientation*, the routine for the acquisition is presented. Here it is assumed that the central pixel in the current 3 × 3 window is not a background pixel, for no edge orientation information is needed for the expansion of background pixels in this new algorithm.

As known from Section 3.1, if *max(epr)* ≥ 2, it is assumed that an edge, or a part of some textures is detected. However, *epr* (*edge pixel run*) is not enough to distinguish between an edge and a part of some textures inside the 3 × 3 window (in case that *max(epr)* ≥ 2, it is also possible that part of some textures is encountered). Further detection is therefore necessary to distinguish between the two to get the possible edge orientation. For this reason, *#trans* is applied for further detection.

If *#trans* = 1, the 3 × 3 window can be divided into two parts, one belonging to a region, and the other belonging to the background, thus it can be assumed that an edge exists. In *edge_orientation*, the number of the central pixel's 4-neighbors which are background pixels, and the number of the central pixel's diagonal neighbors which are edge pixels are needed to get the edge orientation, in case that *#trans* = 1, that will be discussed in details in next three subsections. See Fig. 3.7 for example, where the dashed boxes and circles are used to indicate the above two kinds of pixels, which are called *background 4-neighbors*, and ***diagonal edge neighbors*** respectively.

To calculate the above two values, the subroutines ***#4-neighbor***(0), and ***#d4-neighbor***(1) are called respectively. Based on the two values, the corresponding criteria are applied to obtain the edge orientation. Then the corresponding expansion method is applied to the central pixel, based on the information.

If *#trans* ≠ 1, it can be assumed that irregular edges, or part of some textures exists. This will be discussed in details in Section 3.2.4.

Here two assumptions are made for edge orientation. It is first assumed that the edge orientation is measured counterclockwise with the central pixel p as the origin, starting from positive x axis, thus the edge pixels are always on the left of the edge orientation. Next, it is assumed that each edge orientation is in the range of $0 \sim 2\pi$, and is essentially a multiple of $\pi/4$, which is only an approximation of the real orientation. See Fig. 3.6 to Fig. 3.11 for example, where all cases for edge orientation are shown respectively.

As known, the new algorithm uses local 3 × 3 window operation for the edge detection and expansion for the sake of time complexity. However, with such local window operation, the real orientation is very difficult to obtain, because of the limited amount of information in the window. That is why the above approximation is applied.

Because the approximation of each edge orientation is a multiple of $\pi/4$, it guarantees the continuity of the edges in the expanded images, so that the aliasing can be greatly eliminated. This issue will be discussed in more details in Section 3.3.

In each window shown in Fig. 3.6 and Fig. 3.8, there are two neighboring pixels marked with *1(0)*, which indicate that the two pixels can belong either to an edge or to the background, but can not belong to the edge(s) (for Fig. 3.6) or to the background (for Fig.3.8) at the same time, i.e. in Fig. 3.6 both pixels with 1(0) can not be 1, and in Fig. 3.8 both can not be 0. For those allowable cases in each window, the corresponding orientation is considered to be the same. The cases that both pixels with 1(0) belong to the edge(s) or the background are presented in Fig. 3.7 and Fig. 3.9 respectively. In Fig. 3.7, the symbol "-" is used to indicate the edge orientation change, that is discussed in details in Section 3.2.3. This notation is used in the rest of this chapter.

3.2.1. Edge Orientations in Horizontal and Vertical Directions

The criterion for the detection of edge orientation in horizontal or vertical direction are as follows:

Suppose that the central pixel of the current 3×3 binary window is p, and the set of p and its 4-neighbors is $N_4(p) = (p, p_0, p_2, p_4, p_6)$.

If *#4-neighbor*(0)=1 and *#d4-neighbor*(1)<4, as shown in Fig. 3.6, the edge orientation is either in a horizontal or a vertical direction, that depends on which of the 4-neighbors is a background pixel, i.e.,

if *#4-neighbor(0)*=1 and *#d4-neighbor*(1)<4, where there is only one background pixel p_i which belongs to $N_4(p)$ and $p_i \neq p$, where $i = 0, 2, 4, 6$

then the edge orientation is either horizontal or vertical;

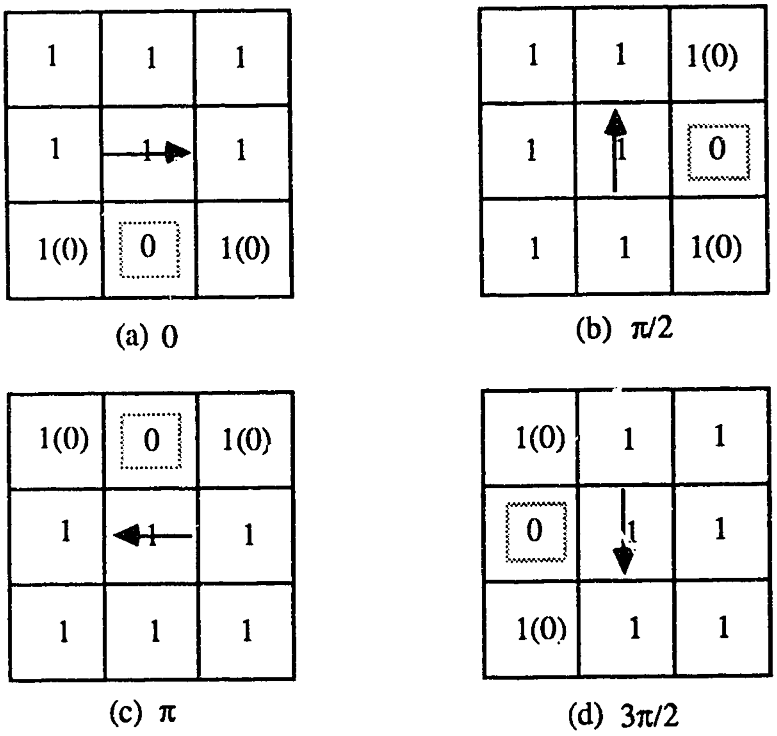
If $i = 0$ or 4, the edge orientation is vertical;

If $i = 2$ or 6, the edge orientation is horizontal;

as shown in Fig. 3.6, where the two pixels with 1(0) cannot be 1 at the same time, thus *#d4-neighbor*(1)<4.

After the horizontal or vertical edge orientation is obtained by ***edge_orientation***, the subroutine *full_assignment* in ***edge_expansion*** is called. Fig. 3.13 illustrates an example for the expanded results using ***full_assignment***, which will be discussed in details in Section 3.3.3.

If *#4-neighbor(0)*=1 and *#d4-neighbor*(1)=4, there is an edge orientation change at the central pixel *p*, i.e. two edges meet at a reflex corner, as shown in Fig. 3.7. Its detection criterion is discussed in details in Section 3.2.3, which deals with edge orientation change.



1	1	1
1	1 →	1
1(0)	0	1(0)

(a) 0

1	1	1(0)
1	1 ↑	0
1	1	1(0)

(b) $\pi/2$

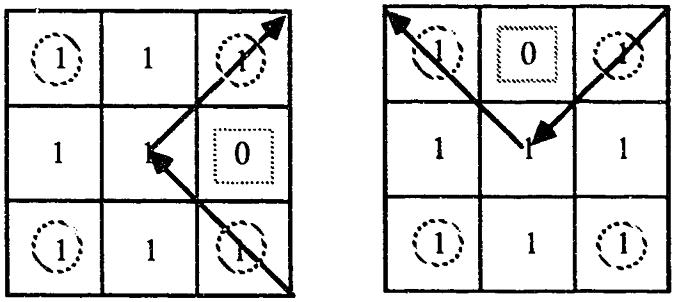
1(0)	0	1(0)
1	← 1	1
1	1	1

(c) π

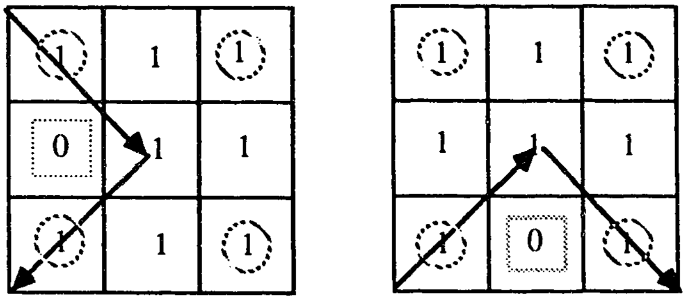
1(0)	1	1
0	1 ↓	1
1(0)	1	1

(d) $3\pi/2$

Fig. 3.6. Examples for horizontal and vertical edge orientations, with *#4-neighbor*(0) = 1 and *#d4-neighbor*(1)< 4



(a) π/4–3π/4 (b) 3π/4–5π/4



(c) 5π/4–7π/4 (d) 7π/4–π/4

Fig. 3.7. Examples for diagonal edge orientation change: π/2, or 3π/2, with *#4-neighbor*(0) = 1 and *#d4-neighbor*(1)=4

3.2.2. Edge Orientations in Diagonal Directions

This section is concerned with the criterion for the detection of the edge orientations in diagonal directions, which is discussed as follows.

If *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) > 1, it can be assumed that the edge orientation is in one of the four diagonal directions, as shown in Fig. 3.8, where the two pixels denoted as 1(0) can not be 0 at the same time, i.e. *#d4-neighbor*(1) > 1. The edge orientation is then determined by which two of the 4-neighbors are background pixels. The detection criterion is as follows:

If *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) > 1, where there are two background pixels p_i, p_j, which belong to $N_4(p) = (p, p_0, p_2, p_4, p_6)$ and $p_i \neq p$, $p_j \neq p$, and $i \neq j$, where i, j = 0, 2, 4, 6, then the edge orientation is in one of the four diagonal directions:

if $i, j = 6, 0$, the edge orientation is $\pi/4$, as shown in Fig. 3.8a;

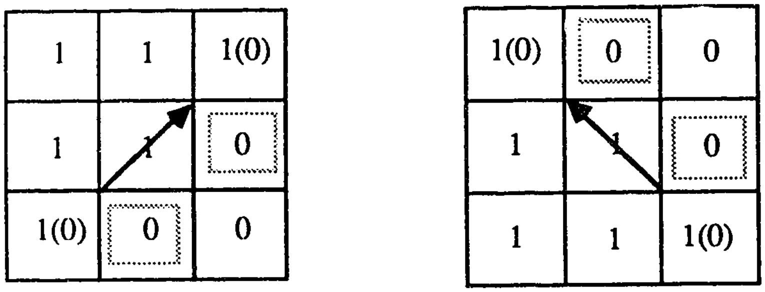
if $i, j = 0, 2$, the edge orientation is $3\pi/4$, as shown in Fig. 3.8b;

if $i, j = 2, 4$, the edge orientation is $5\pi/4$, as shown in Fig. 3.8c;

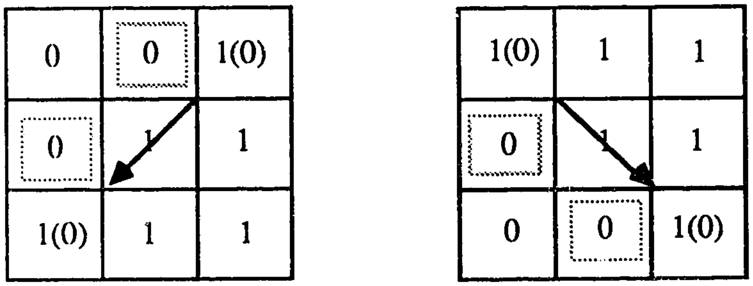
if $i, j = 4, 6$, the edge orientation is $7\pi/4$, as shown in Fig. 3.8d.

The above diagonal orientation information is used in the subroutine *half_assignment* of *edge_expansion* for the edge expansion in this case. This subroutine is discussed in Section 3.3.4.

If *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) = 1, as shown in Fig. 3.9, it is assumed that there is an edge orientation change from vertical to horizontal (horizontal to vertical) directions at p, that is discussed in next section.



(a) π/4 (b) 3π/4



(c) 5π/4 (d) 7π/4

The edge orientations = odd multiples of $\pi/4$

Fig. 3.8. Examples for diagonal edge orientations, with *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) > 1

3.2.3. Edge Orientation Changes at the Central Pixel

This section is concerned with the detection of edge orientation change at the current central pixel p inside the 3×3 window. The edge orientation changes can be classified into four cases, which will be discussed separately in this section.

The first case is shown in Fig. 3.7, where two edges meet at a reflex corner, as mentioned in Section 3.2.1. In this case, *#4-neighbor*(0) = 1 and *#d4-neighbor*(1) = 4 hold. The corresponding detection criterion is following.

If *#4-neighbor(0)*=1 and *#d4-neighbor*(1)=4, then there is an edge orientation change at the central pixel p, i.e. two edges meet at a reflex corner, as shown in Fig. 3.7. More specifically,

if $i = 0$, the orientation changes from $\pi/4$ to $3\pi/4$, denoted as $\pi/4$ - $3\pi/4$ in Fig. 3.7(a);

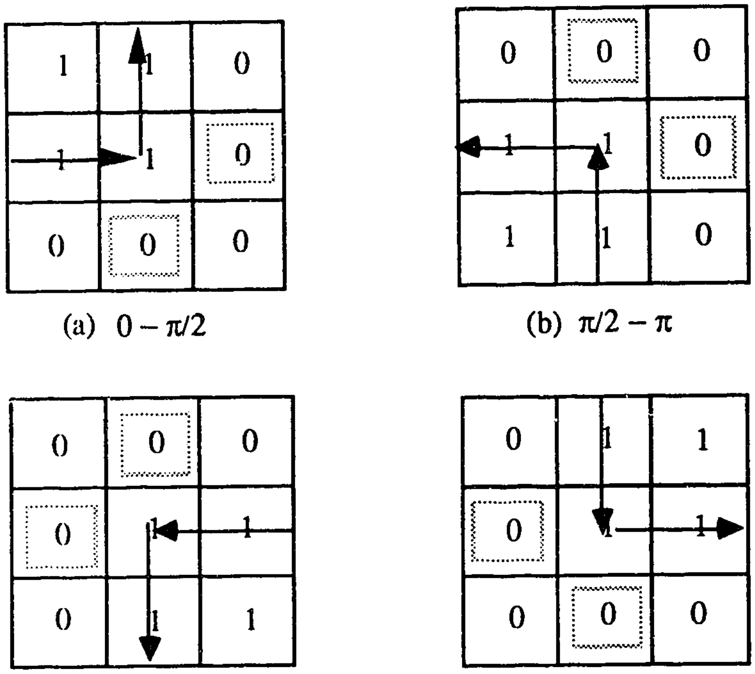
if $i = 2$, the orientation changes from $3\pi/4$ to $5\pi/4$, denoted as $3\pi/4$ - $5\pi/4$ in Fig. 3.7(b);

if $i = 4$, the orientation changes from $5\pi/4$ to $7\pi/4$, denoted as $5\pi/4$ - $7\pi/4$ in Fig. 3.7(c);

if $i = 6$, the orientation changes from $7\pi/4$ to $\pi/4$, denoted as $7\pi/4$ - $\pi/4$ in Fig. 3.7(d).

To reflect such orientation change in the expanded image, a subroutine in *edge_expansion*, called *quarter_assignment*, is applied to the corresponding patch. This subroutine is discussed in details in Section 3.3.5.

The second case is shown in Fig. 3.9, where *#4-neighbor*(0)=2 and *#d4-neighbor*(1)=1 hold. In this case, it is assumed that there is an edge orientation change from vertical to horizontal (horizontal to vertical) directions at p, i.e. two edges meet at a convex corner. The subroutine *full_assignment* is called directly for edge expansion to reflect such perpendicular orientation change in the expanded image.



(a) $0-\pi/2$ (b) $\pi/2-\pi$

(c) $\pi-3\pi/2$ (d) $0-3\pi/2$

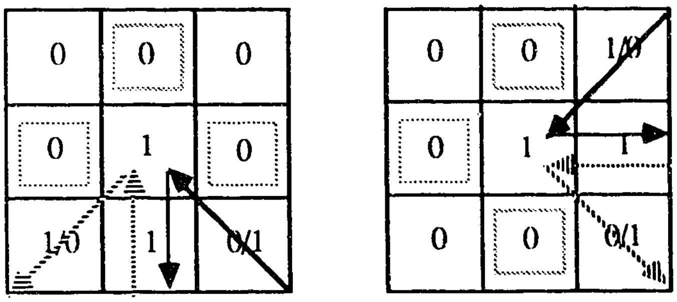
Fig. 3.9. Examples for perpendicular edge orientation change, with *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) = 1

Next, two more cases with *#4-neighbor*(0) = 3 are discussed. Examples are shown for the two cases respectively in Fig. 3.10, where *#d4-neighbor*(1) = 1, and in Fig. 3.11, where *#d4-neighbor*(1) = 2. In Fig. 3.10, one pixel is denoted as 1/0, and another is denoted as 0/1. The above denotation means that the two pixels takes 1 and 0 alternatively in the 3 × 3 binary window *c*.

With *#d4-neighbor*(1) = 1, the orientation change is $\pi/4$, $3\pi/4$ or $5\pi/4$, as shown in Fig. 3.10. When *#d4-neighbor*(1) = 2, the orientation change is $\pi/2$ or $3\pi/2$, as shown in Fig. 3.11.

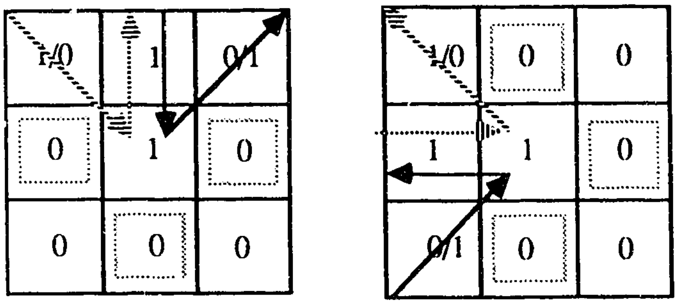
Suppose that *#4-neighbor*(0) = 3, where there are three background pixels p_i, p_j, p_k, which belong to $N_4(p) = (p, p_0, p_2, p_4, p_6)$, $p_i \neq p, p_j \neq p, p_k \neq p$, and $i \neq j \neq k$, where i, j, k belong to (0, 2, 4, 6), then there is an edge orientation change at the central pixel p.

The detection criteria for the above two cases are as follows:



(a) $3\pi/4 - 3\pi/2$ or $\pi/2 - 5\pi/4$

(b) $0 - 5\pi/4$ or $\pi - 7\pi/4$



(c) $\pi/4 - 3\pi/2$ or $\pi/2 - 7\pi/4$

(d) $\pi/4 - \pi$ or $0 - 3\pi/4$

Fig. 3.10. Examples for edge orientation change: $\pi/4$, $3\pi/4$ or $5\pi/4$, with *#4-neighbor*(0) = 3 and *#d4-neighbor*(1) = 1

1) *#d4-neighbor*(1) = 1

If i, j, k = 0, 2, 4, the edge orientation changes from,

$3\pi/4$ to $3\pi/2$, with $c(p_7)$=1 and $c(p_5)$=0,

or $\pi/2$ to $5\pi/4$, with $c(p_7)$=0 and $c(p_5)$=1,

as shown in Fig. 3.10a;

If i, j, k = 2, 4, 6, the edge orientation changes from,

0 to $5\pi/4$, with $c(p_1)$=1 and $c(p_7)$=0,

or π to $7\pi/4$, with $c(p_1)$=0 and $c(p_7)$=1,

as shown in Fig. 3.10b;

If i, j, k = 4, 6, 0, the edge orientation changes from,

$\pi/4$ to $3\pi/2$, with $c(p_1)$=1 and $c(p_3)$=0,

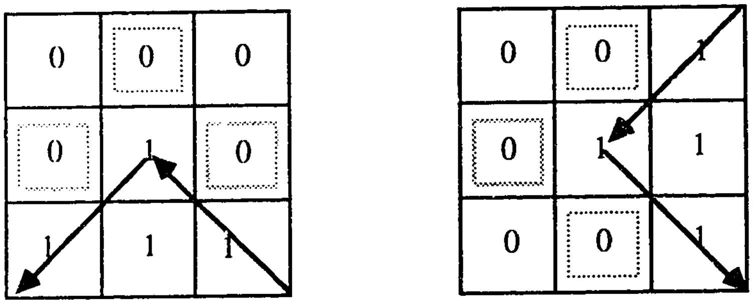
or $\pi/2$ to $7\pi/4$, with $c(p_1)$=0 and $c(p_3)$=1, as shown in Fig. 3.10c;

If i, j, k = 6, 0, 2, the edge orientation changes from,

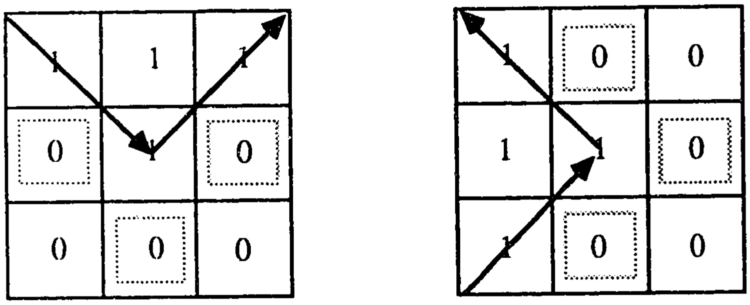
$\pi/4$ to π, with $c(p_5)$=1 and $c(p_3)$=0,

or 0 to $3\pi/4$, with $c(p_5)$=0 and $c(p_3)$=1,

as shown in Fig. 3.10d.



(a) $3\pi/4 - 5\pi/4$ (b) $5\pi/4 - 7\pi/4$



(c) $7\pi/4 - \pi/4$ (d) $\pi/4 - 3\pi/4$

Fig. 3.11. Examples for diagonal edge orientation change: $\pi/2$ or $3\pi/2$, with *#4-neighbor*(0) = 3 and *#d4-neighbor*(1) = 2

2) *#d4-neighbor*(1) = 2

If i, j, k = 0, 2, 4, the edge orientation changes from $3\pi/4$ to $5\pi/4$, as shown in Fig. 3.11a;

If i, j, k = 2, 4, 6, the edge orientation changes form $5\pi/4$ to $7\pi/4$, as shown in Fig. 3.11b;

If i, j, k = 4, 6, 0, the edge orientation changes from $7\pi/4$ to $\pi/4$, as shown in Fig. 3.11c;

If i, j, k = 6, 0, 2, the edge orientation changes from $\pi/4$ to $3\pi/4$, as shown in Fig. 3.11d.

With the orientation information obtained with the above criteria, corresponding subroutines in *edge_expansion* can be called for expansion.

For the case with *#4-neighbor*(0)=3 and *#d4-neighbor*(1) = 1, the edge orientation change is essentially an odd multiple of $\pi/4$, i.e. one of the two edge orientations detected is in a diagonal direction, so in this case, the subroutine *half_assignment* is called to indicate such diagonal orientation in the expanded image.

If *#4-neighbor*(0)=3 and *#d4-neighbor*(1)=2, the subroutine *quarter_assignment* in *edge_expansion* is called for the edge expansion, that is discussed in Section 3.3.5.

3.2.4. Other Cases for Edge Orientation Detection

If *#trans* $\neq 1$, it is assumed that part of some textures, or irregular edges are detected, thus it is impossible to detect edge orientation at the central pixel. In this case, the subroutine *full-assignment* in *edge_expansion* is called for the expansion of the central pixel p. The expansion issue is discussed in details in Section 3.3.3.

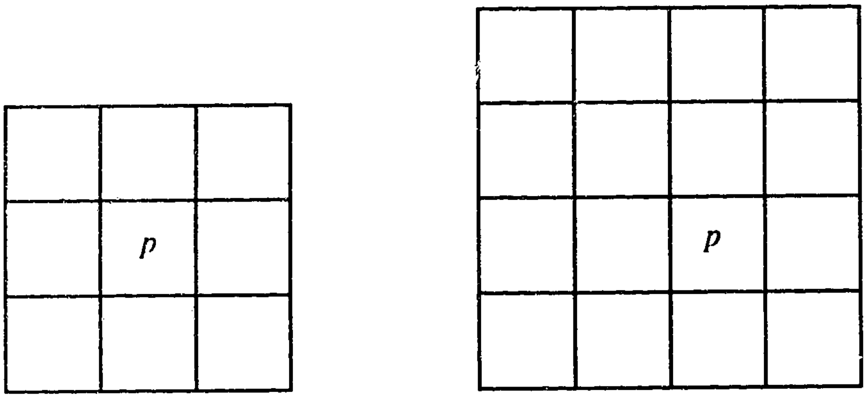
3.3. Edge Expansion

3.3.1. Definition Used in the Edge Expansion Method

Suppose that the expansion factor is k, thus, each pixel in the original image will be mapped onto a patch of size $k \times k$, in the expanded image. During the expansion, the central pixel of the patch will be processed to reflect the orientation change, if any, as shown in Fig. 3.18c. However, if k is even, there is a problem to define the central pixel in the patch of size $k \times k$. In order to facilitate the implementation, the center pixel of a $k \times k$ patch is defined as follows, no matter whether k is even or odd.

For any $k \times k$ patch, it is supposed that the coordinates at its upper left corner are (i, j). Then the central pixel of the patch is defined to be at $(i + k \text{ div } 2, j + k \text{ div } 2)$, where *div* means integer division, that truncates the fractional part of the quotient.[3] Fig. 3.12 shows the central pixels for a 3 × 3 patch and a 4 × 4 patch respectively.

With this definition, expansion will be consistent for both even and odd expansion factors.



(a) the central pixel in a 3 × 3 patch

(b) the central pixel in a 4 × 4 patch

Fig. 3.12. The central pixel in a patch

3.3.2. Algorithm for the Edge Expansion Method

With edge orientation information, its expansion is facilitated. For each case of the edge orientation information at the current central pixel p obtained with *edge_orientation*, a corresponding subroutine in *edge_expansion* is called to apply the information to the corresponding patch in the expanded image.

There are two steps for the edge expansion in each subroutine. First, the subroutine divides the patch into two parts, one for the edge, and the other for the background, if

---

[3] This is consistent with the assumption made in Sec. 1.2, that the image origin is at its upper left corner.

necessary, using the orientation. Second, pixels of each part is assigned appropriate gray values, which are determined by to which this part belongs, the edge or the background. This step also involves selective averaging operations. In this way, the edge orientation at the pixel of the original image is preserved in the patch, thus the aliasing and blurring are avoided. As mentioned before, only the approximation of the real orientation is applied in the patch, for the sake of time complexity.

In Fig. 3.13 to Fig. 3.18, except Fig. 3.15, the edge expansion results are shown respectively for the each case of edge orientations discussed earlier. In these figures, the edge gray values are indicated in bold text, while the background gray values are in plain text, for the ease of illustration. The blank boxes in these figures indicate the pixels whose gray values are not yet assigned in the expanded image at the moment, or the pixels whose gray values can not be determined using the current window information, since the expansion process is sequential. These results are obtained by calling corresponding subroutines in *edge_expansion*, based on the edge orientation information.

The routine *edge_expansion* is shown as follows:

```
edge_expansion(p, c)
p: the current central pixel in the 3 × 3 window of the original image;
c: the 3 × 3 binary window which contains the orientation information;
{   #trans: the number of transitions;
    #4-neighbor(0): the number of p's background 4-neighbors;
    #d4-neighbor(1): the number of p's diagonal edge neighbors;

    case (1) c(p) = 0:
            /* p, the current central pixel, is a background pixel: no edge orientation is
            needed for expansion */
            call full_assignment(p, c);
```

```
        /* Fill in the whole patch with the background gray values */

$(2)$  $c(p) = 1$ and $\#trans = 1$:  /* edge exists: p is an edge pixel */
{
a) #4-neighbor(0)=1 and #d4-neighbor(1) < 4
          /* As shown in Fig. 3.6: horizontal or vertical directions */
          call full_assignment(p, c);
          /* The subroutine fills in the whole patch in the expanded image with the
          edge gray values, that is discussed in Section 3.3.3. See Fig. 3.13 for
          example. */
    b) #4-neighbor(0)=1 and #d4-neighbor(1) = 4
          /* An edge orientation change at a reflex corner: see Fig. 3.7 for example */
          call quarter_assignment(p, c);
          /* To reflect such change at the reflex corner, a quarter part of the patch is
          assigned with background gray value, that is discussed in Section 3.3.5.
          See Fig. 3.19 for example. */
    c) #4-neighbor(0)=2 and #d4-neighbor(1) > 1
          /* Diagonal directions: see Fig. 3.8 for example. */
          call half_assignment(p, c);
          /* Use the diagonal edge orientation to divide the patch into two parts, fill
          each part with the corresponding gray values (of the edge or background).
          See Fig. 3.16 for example.  */
    d) #4-neighbor(0)=2 and #d4-neighbor(1)=1
          /* Orientation change of $\pi/2$ or $3\pi/2$: two edges meet at a convex corner.
          See Fig. 3.9 for example. */
          call full_assignment(p, c);
          /* Called to reflect such change in the patch. See Fig. 3.14 for example. */
```

```
            e) #4-neighbor(0)=3 and #d4-neighbor(1)=1
                /* Orientation change: $\pi/4$, $3\pi/4$ or $5\pi/4$. See Fig. 3.10 for example. */
                call half_assignment(p, c);
                /* This subroutine is used to reflect such orientation change from diagonal
                to horizontal or vertical directions. See Fig. 3.17 for example. */
            f) #4-neighbor(0)=3 and #d4-neighbor(1)=2
                /* Orientation change in two diagonal directions: $\pi/2$ or $3\pi/2$. See Fig. 3.11
                for example. */
                call quarter_assignment(p, c);
                /* Divide the corresponding patch into two parts, using the edge orientation
                change: the quarter part of the patch is filled with the edge gray values, and
                the other part is filled with the background gray values. See Fig. 3.18 for
                example. */
        }   /* (2) $c(p) = 1$ and #trans = 1 */

        (3) $c(p) = 1$ and #trans $\neq 1$:
            call full_assignment(p, c);
            /* Part of some textures or irregular edges are detected: no edge orientation can be
            obtained; fill in the corresponding patch with full_assignment(p, c). */
    } /* edge_expansion */
```

3.3.3. Subroutine *full_assignment*

It should be noted that *full_assignment*, *half_assignment*, and *quarter_assignment* as well, process the patch pixel by pixel, from left to right, and row by row.

At each pixel being processed in the patch, it is first filled (replicated) with the same gray value as its *initial gray value* (edge or background pixel, depending on whether the central pixel p is an edge or background pixel). To avoid a blocky appearance on the boundaries of adjacent patches caused by such a simple replication, an averaging operation over this pixel and its 8-neighbors are applied, and then the average result is assigned to it as its *final gray value*. Then the next pixel will be processed in the same manner. Thus, each pixel is processed sequentially.

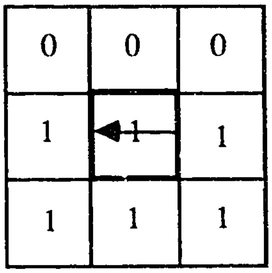
On the other hand, to avoid any edge blurring, the averaging operations should be *selective*, that is, only the same kind of neighboring pixels (background or edge pixels) as p_p, the current pixel being processed in the patch, are selected for the averaging operation. The selection criterion is based on the criterion using the predefined threshold T in the pixel classification discussed in Section 3.1. All pixels with less gray value difference with p_p than the threshold T are considered as the same *kind* of pixels as p_p. No further details are necessary for discussion, since the selection process is essentially the same as that for the pixel classification discussed in Section 3.1.

Such a *selective* averaging operation is necessary, for p_p, the current pixel being processed, and its 8-neighbors could belong to different regions. Please see Fig. 3.13c for example. In Fig. 3.13c, the four bold boxes comprises the patch being processed, with the expansion factor=2. The pixel inside the patch, indicated with dashed box is p_p, the current pixel being processed[4], which, in this case is an edge pixel, and is on the patch boundary; the other two pixels outside the patch indicated with dashed boxes are its 8-neighbors involved in the averaging operation, which in this case are also edge pixels. It should be noted that all the other 8-neighbors not involved in the operation, are either background pixels, or the pixels not set yet at the time when the current pixel is being processed (i.e. those pixels at its right or in next row inside the patch), as shown in Fig. 3.13c.

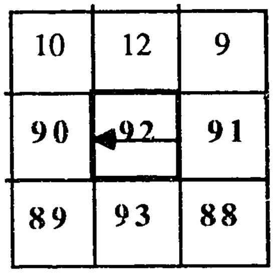
Based on the above, *full_assignment* is described as follows:

---

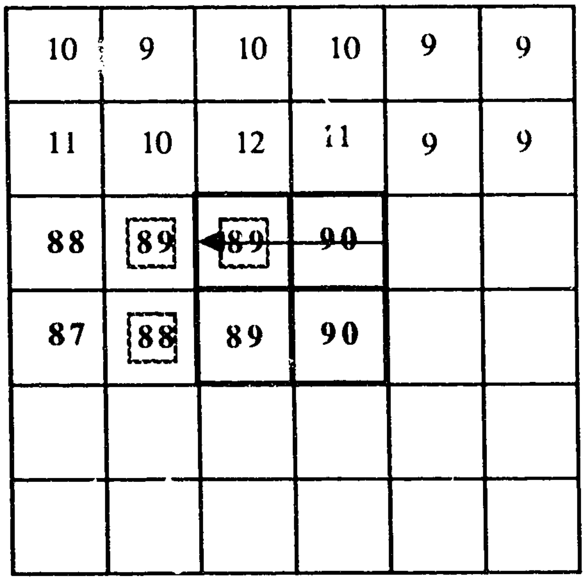
[4] Fig. 3.13, and all the other figures afterwards as well, only show the cases in which the patch has been processed, for the case of illustration.



(a) Orientation shown in the binary window

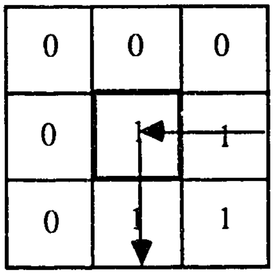


(b) Gray values in the original image

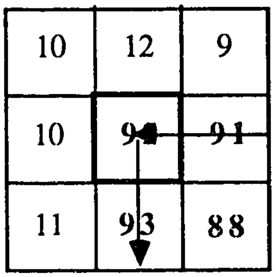


(c) Expansion results (Exp. Factor = 2)

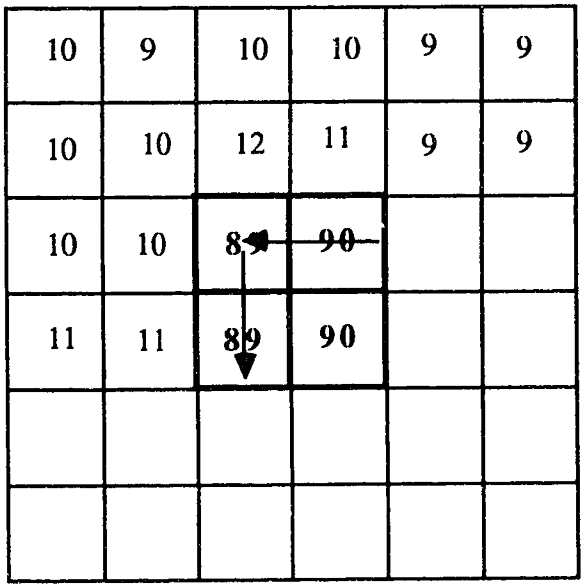
Fig. 3.13. Edge expansion using *full_assignment*, when *#4-neighbor(0)* = 1 and *#d4-neighbor(1)* < 4



(a) Orientation shown in the binary window



(b) Gray values in the original image



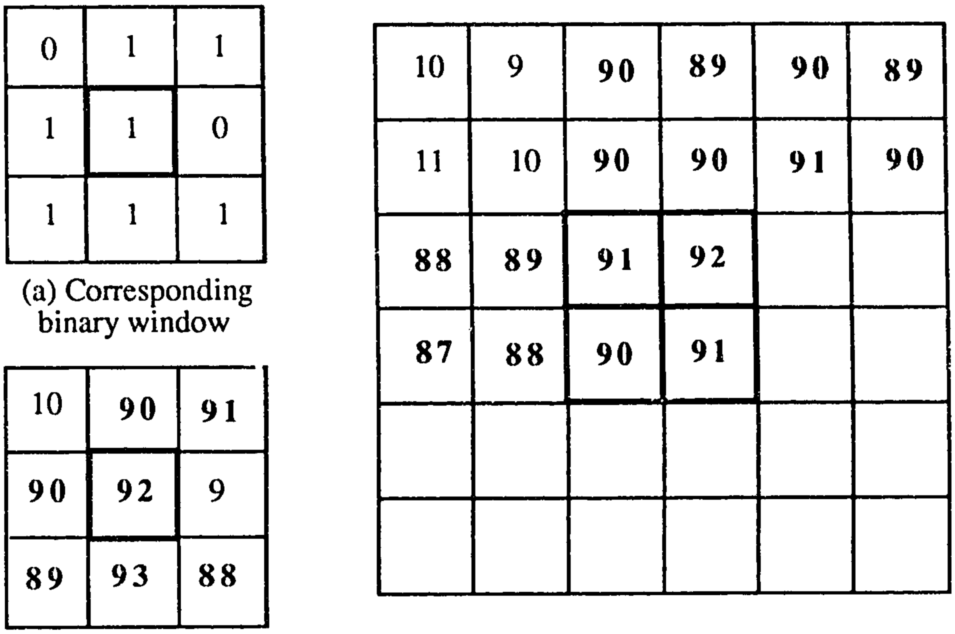
(c) Expansion results (Exp. Factor = 2)

Fig. 3.14. Edge Expansion using *full_assignment*, when *#4-neighbor(0)* = 2 and *#d4-neighbor(1)* =1

```
full_assignment(p, c)
p: the current central pixel in the $3 \times 3$ window of the original image;
c: the corresponding $3 \times 3$ binary window;
{ Process each pixel in the patch, say $p_p$ sequentially, i.e. from left to right, row by row
  and from top to bottom, using a $3 \times 3$ window as follows:
    1. Assign an initial value to $p_p$
        (a) p is an edge pixel
            replicate this edge gray value in $p_p$;
        (b) p is a background pixel
            replicate this background gray value in $p_p$;
    2. Select all its 8-neighboring pixels which are the same kind as it is (edge or
       background pixels, determined by p) and have already been set (assigned);
       Average the gray value of $p_p$ with those of all such 8-neighbors;
    3. Assign the average result to $p_p$;
/* As mentioned earlier, the above averaging operations near the patch boundaries involves
some pixels in the adjacent patches, in order to keep the expansion results "consistent"
(continuous) between these adjacent patches, to avoid any blocky appearance. */
} /* full_assignment */
```

There are four cases in which this subroutine can be applied. First, it can be applied to the expansion of background pixels, and the pixels on textures or irregular edges. In both cases, no edge orientation is required. This subroutine avoids any blurring effect for both cases and avoids the blocky effect for the background pixels as well. For the pixels on textures, however, some jaggies could result, that is determined by the nature of the textures to be expanded. This issue about such jaggies are not discussed in further details and only left for future research, since the texture expansion is not the major concern of the new algorithm. See Fig. 3.15 for an expansion example in this case.

The other two cases are shown in Fig. 3.6 and Fig. 3.9 respectively. In Fig. 3.6, the edge orientation is in a horizontal or vertical direction, while in Fig. 3.9, there is an edge orientation change from horizontal to vertical (or vertical to horizontal). For the case shown in Fig. 3.6, this subroutine reflects the smoothness of the horizontal or vertical orientation along the edge. An example for the edge expansion in this case is shown in Fig. 3.13. This subroutine can also reflect the perpendicular orientation change for the case shown in Fig. 3.9. An expansion example for this case is shown in Fig. 3.14.



0	1	1
1	1	0
1	1	1

(a) Corresponding binary window

10	90	91
90	92	9
89	93	88

(b) Gray values in the original image

10	9	90	89	90	89
11	10	90	90	91	90
88	89	91	92		
87	88	90	91		

(c) Expansion results

Fig. 3.15. Expansion using *full_assignment* for textures

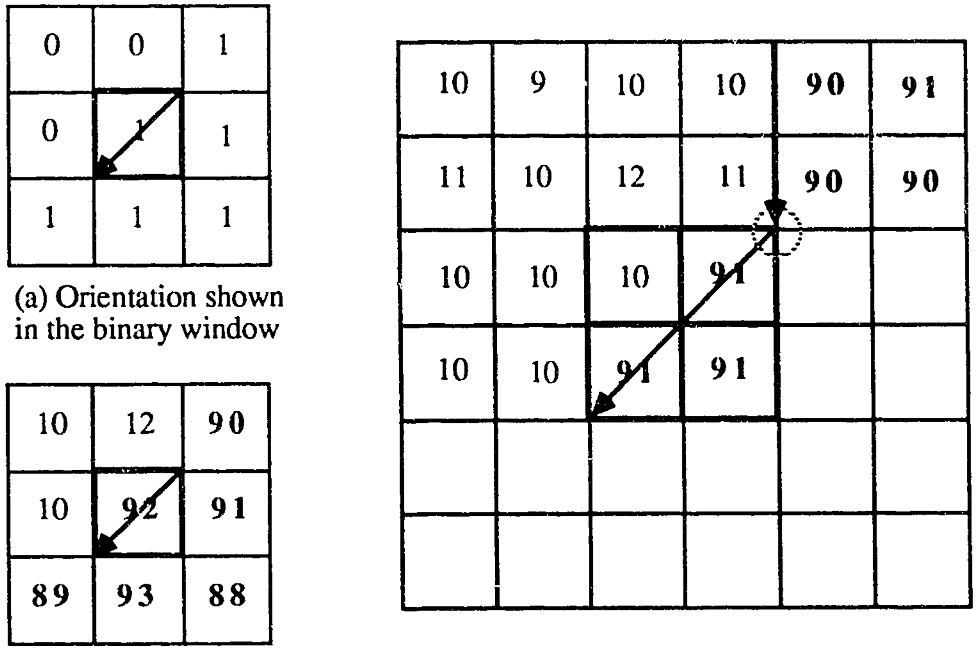
3.3.4. Subroutine *half_assignment*

The subroutine *half_assignment* is used for edge expansion when the current edge orientation is in a diagonal direction, or the orientation change is from a diagonal direction to a horizontal or vertical direction, or vice versa. To avoid any blocky appearance, *half_assignment* applies the above edge orientation in the patch being processed, thus the patch is divided into two parts by the orientation, each being assigned with appropriate gray values (of edge or background), determined by whether it belongs to an edge or background. This issue is discussed in details as follows.

There are two steps in *half_assignment*. First, since the edge orientation or orientation change dealt with in this section is an odd multiple of $\pi/4$, i.e. it is in one of the four diagonal directions, the patch is divided into two parts by this diagonal direction, the part belonging to the edge pixels and the part belonging to the background pixels, according to the edge orientation obtained, as shown in Fig. 3.16 and Fig. 3.17. For the sake of the edge continuity (consistency) along its orientation between the corresponding adjacent patches, the pixels on the diagonal of the patch are treated as edge pixels, as shown in Fig. 3.16c and Fig. 3.17c.

Second, every pixel in each part of the patch is processed sequentially, as in *full_assignment*. If the part belongs to the edge, the gray value of the central pixel p is filled in the pixel being processed as its initial value, otherwise it is filled with the background gray value obtained from the 3×3 window of the original image. Similarly, such simple replication will cause an artificial discontinuity of the gray value on the patch's boundaries with its adjacent patches. To avoid this and blurring, for each part of the patch, the same selective averaging operation as used in *full_assignment* is applied on the pixel being processed, and the average result obtained in the averaging operation is assigned to this pixel. That is, if the part of the patch belongs to an edge, the gray value of the pixel being processed is averaged with those of its 8-neighboring edge pixels, otherwise, its gray value is averaged with those of its 8-neighboring background pixels.

Since the central pixel p in the 3×3 window of the original image is itself an edge pixel in this case, the average gray value of all background pixels inside the window is used as the initial gray value for each pixel in the patch belonging to the background.



0	0	1
0	1	1
1	1	1

(a) Orientation shown in the binary window

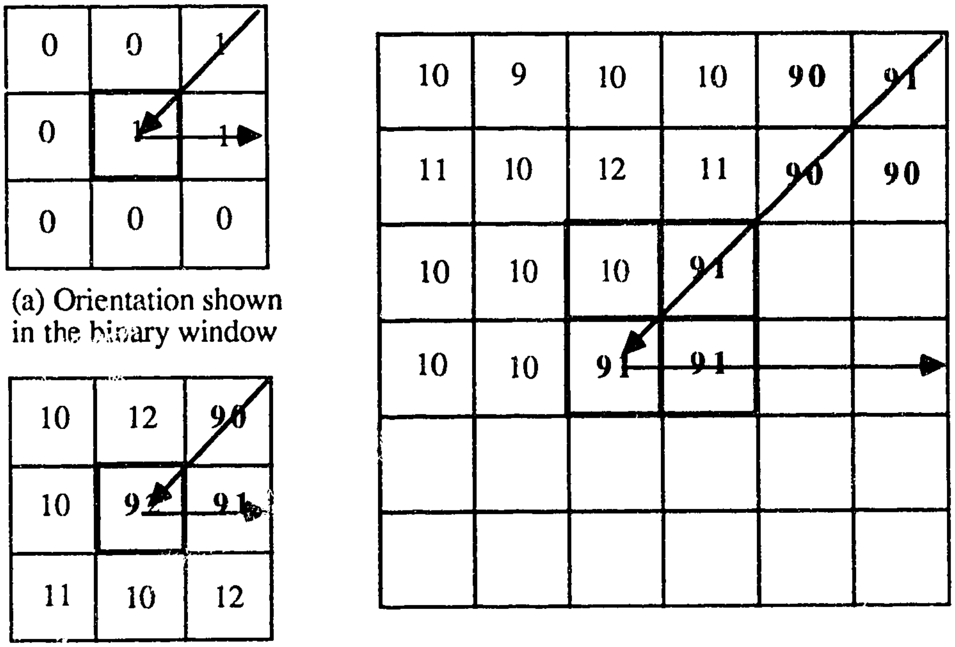
10	12	90
10	92	91
89	93	88

(b) Gray values in the original image

10	9	10	10	90	91
11	10	12	11	90	90
10	10	10	91		
10	10	91	91		

(c) Expansion results (Exp. Factor = 2)

Fig. 3.16. Edge expansion using *half_assignment*, when *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) > 1



0	0	1
0	1	1
0	0	0

(a) Orientation shown in the binary window

10	12	90
10	92	91
11	10	12

(b) Gray values in the original image

10	9	10	10	90	91
11	10	12	11	90	90
10	10	10	91		
10	10	91	91		

(c) Expansion results (Exp. Factor = 2)

Fig. 3.17. Edge expansion using *half_assignment*, when *#4-neighbor*(0) = 3 and *#d4-neighbor*(1) = 1

Thus, the subroutine *half_assignment(p, c)* is described as follows:

```
half_assignment(p, c)
p: the current central pixel in the $3 \times 3$ window of the original image;
c: the corresponding $3 \times 3$ binary window;
{
1. Divide the patch into two parts, according to the edge orientation obtained above;
    /* See Fig. 3.16 and Fig. 3.17 for example. */

2. Process each pixel inside the part of the patch which belongs to the edge, called $p_e$,
   from left to right, row by row, and from top to bottom:
        1) replicate in $p_e$ the gray value of $p$;  /* Assign initial value to $p_e$ */
        2) find all of $p_e$'s 8-neighboring edge pixels which have already been set;
        3) calculate the gray value average of $p_e$ and all such 8-neighboring pixels;
        4) assign the average result to $p_e$.
    /* See the part of the patch with gray values in bold text in Fig. 3.16c and Fig.
    3.17c for example. */

3. For each pixel inside the other part of the patch, which belongs to the background,
   called $p_b$, a similar operation is done on it:
        1) replicate first with the average gray value of all the background pixels inside
        the $3 \times 3$ window of the original image (the average of those pixels denoted
        with 0's in the corresponding $3 \times 3$ binary window c);     /* Assign initial
        value to $p_b$ */
        2) find all of $p_b$'s 8-neighboring background pixels which have already been
        set;
        3) calculate the gray value average of $p_b$ and all such 8-neighboring pixels;
        4) assign the average result to $p_b$.
```

/* For example, see the part of the patch with gray values in plain text shown in Fig. 3.16c and Fig. 3.17c. */

/* Similarly, the above averaging operations also involves some pixels near the boundaries of adjacent patches in order to keep the expansion results "consistent" (continuous) between these adjacent patches, i.e. to avoid any blocky appearance. */

}

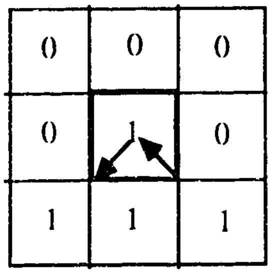
/* *half_assignment* */

As mentioned earlier, there are two cases in which *half_assignment* can be applied. One case is shown in Fig. 3.8, where *#4-neighbor*(0) = 2 and *#d4-neighbor*(1) > 1, as discussed in Section 3.2.2. An expansion example for this case is shown in Fig. 3.16. The other case is shown in Fig. 3.10, where *#4-neighbor*(0) = 3 and *#d4-neighbor*(1) = 1, as discussed in Section 3.2.3. An example for the edge expansion in this case is shown in Fig. 3.17. In both cases, the diagonal directions are detected and then applied in *half_assignment.*

3.3.5. Subroutine *quarter_assignment*

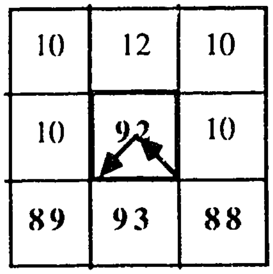
The subroutine *quarter_assignment* is used for edge expansion when there is a perpendicular edge orientation change in two diagonal directions, as shown in Fig. 3.7 and Fig. 3.11. The requirement for *quarter_assignment* is the same as that for *half_assignment,* that is, to avoid blurring and blocky appearance of the edges, and discontinuity on the boundaries of the adjacent patches. The algorithm for *quarter_assignment* is the same as that for *half_assignment,* except that the characteristics of edge orientation between the two cases are different. The same averaging operation is applied in *quarter_assignment.*

Similarly, in *quarter_assignment,* the patch to be filled in is first divided into two parts, using the information of the edge orientation change. One of the two parts is approximately a quarter the size of the patch, that is why this subroutine is called *quarter_assignment*. Whether the quarter part belongs to the edge or background is determined by the characteristics of the edge orientation change. See Fig. 3.18 and Fig. 3.19 for example. After the patch is divided, each part is filled in corresponding edge or background values and the same averaging operation is carried out, as in *half_assignment*.



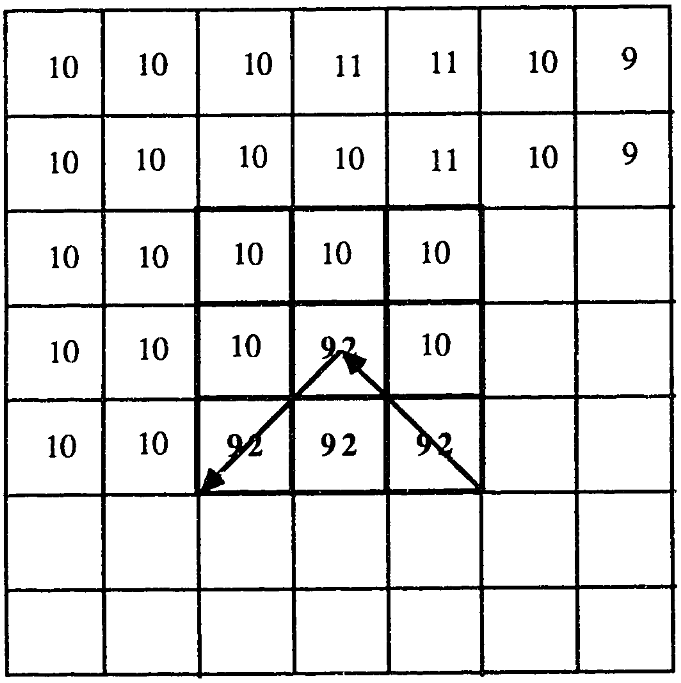
0	0	0
0	1	0
1	1	1

(a) Orientation shown in the binary window



10	12	10
10	92	10
89	93	88

(b) Gray values in the original image

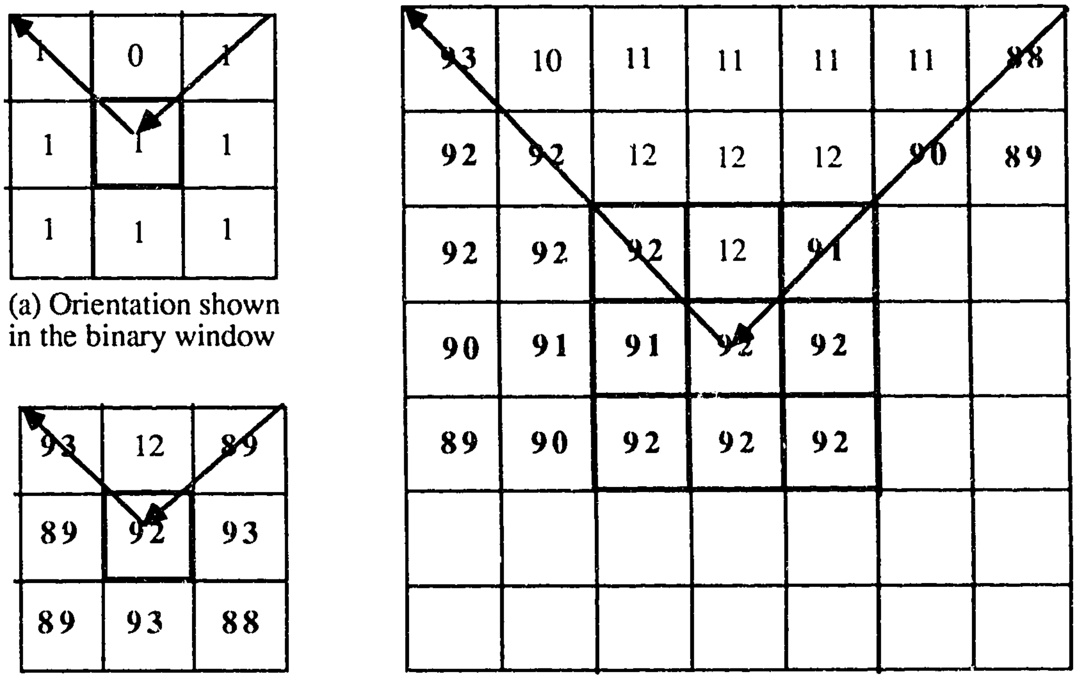


10	10	10	11	11	10	9
10	10	10	10	11	10	9
10	10	10	10	10		
10	10	10	92	10		
10	10	92	92	92		

(c) Expansion results (Exp. Factor = 3)

Fig. 3.18. Edge expansion using *quarter_assignment*, when *#4-neighbor*(0)=3 and *#d4-neighbor(1)*=2

As mentioned earlier, there are two cases in which this subroutine can be applied. One case is shown in Fig. 3.11, where *#4-neighbor*(0) = 3 and *#d4-neighbor*(1) = 2, as discussed in Section 3.2.3. In this case, an example for edge expansion is shown in Fig. 3.18, where the quarter part is assigned with edge values. The other case is shown in Fig. 3.7, where *#4-neighbor*(0) = 1, and *#d4-neighbor*(1) = 4, as discussed in Section 3.2.1. An example for edge expansion in this case is shown in Fig. 3.19. Since the current pixel to be expanded is at a reflex corner where two edges meet, as shown in Fig. 3.19a, the quarter part of the patch is assigned with background values to indicate such a reflex corner in the expanded image, as shown in Fig. 3.19c. It should be noted that the gray value of the only background pixel in the 3 × 3 window of the original image (as shown in Fig. 3.19b) is used as the initial gray value to be assigned to each pixel in the quarter part, since the current pixel to be expanded is an edge pixel in this case.



(a) Orientation shown in the binary window

(b) Gray values in the original image

(c) Expansion results (Exp. Factor = 3)

Fig. 3.19. Edge expansion using *quarter_assignment* when *#4-neighbor*(0)=1 and *#d4-neighbor*(1)=4

3.4. Time Complexity Analysis for *Heuristic Expansion*

Suppose that the expansion factor is k, in both x, and y directions. For edge and line segment differentiation, in each 3 × 3 window, the acquisition of the maximum and minimum values needs 3 × 3 time units. And since there is overlapping in the 3 × 3 window operation throughout the image in both x and y directions, the number of such operations needs $18n^2$ time units, thus the time complexity for this process is $O(n^2)$, where $n \times n$ is the size of the image to be expanded.

The next step is the acquisition of edge or line orientation. The time complexity for *max(epr)* and *#trans* inside the 3×3 window is 3×3 time units. To consider the overlapping mentioned above, the worst case for the total number of the operations for the acquisition of edge or line orientation is $18n^2$, thus the time complexity for this step is $O(n^2)$.

The third step is image expansion. For *bilinear interpolation*, the expansion of an $n \times n$ image by k in each dimension requires $O(k^2n^2)$ multiplications and additions since there are $k^2n^2 - n^2$ pixels to be interpolated. As for bicubic interpolation, e.g., *piecewise Hermite interpolation* and *bicubic spline interpolation*, the time complexity is also $O(k^2n^2)$, but with a higher constant.

The number of additions for *Heuristic Expansion* in edge and line expansion is $9k^2n^2$, and number of divisions is k^2n^2, since in the worst case there is an average operation over each pixel in each $k \times k$ patch, and all of its 8 neighbors. So based on the time complexity analysis for the first three steps, the time complexity for the worst case for the *Heuristic Expansion* in image expansion is $O(k^2n^2)$, with a constant larger than that of bilinear interpolation.

3.5. Remaining Problems

Heuristic Expansion, the technique for edge expansion introduced in this chapter can not avoid jaggies completely, due to the approximation of edge orientation employed in this technique and noise present in the original images.

It is possible that there are still some jaggies between adjacent patches along the edge orientation (See Fig. 3.16c for example, in which the dashed circle is where a jaggy would occur). The orientation changes inside patches could be in fact jaggies for the cases shown in Fig. 3.18 and Fig. 3.19, possibly due to noise in original image at the corresponding pixels being expanded. This scenario could be more noticeable when expansion factor is larger. Jaggies can be seen in Plate 7.3 of Chapter 5, where some expanded results of this technique are shown.

To solve this problem, an edge smoothing approach, called *Heuristic Smoothing* in *Smoothing Module* is applied to the expanded image as postprocessing. In Chapter 4, the issue is discussed in details.

Chapter 4

Edge Smoothing Methods

4.1. Problem Description

As discussed in Chapter 3, there are still jaggies in the image expanded by *Heuristic Expansion*. These jaggies are due to the fact that some approximation is made for edge orientation acquisition using a 3 × 3 window operation. As known, with the 3 × 3 window operation in *Heuristic Expansion*, the edge orientation is approximated as a multiple of $\pi/4$. The approximation is manifested in image expansion, thus jaggies result. On the other hand, the jaggies may come from the noise in the original image, as mentioned in last chapter. To eliminate these jaggies, some edge smoothing methods should be applied to the expanded image. There are numerous edge smoothing methods in [Dav85], some of which can be applied directly.

In Section 4.2, seven smoothing methods used in [Dav85] are discussed. There are some problems with these methods. Some of them blur the image during the smoothing. Others do not have noticeable effect on jaggy removing. So, in Section 4.3, a new edge smoothing method, called *Heuristic Smoothing* is discussed, which can remove considerable jaggies while preserving the sharpness of edges. This method is part of *Smoothing Module* in the new algorithm.

4.2. Previous Edge Smoothing Methods

There are numerous smoothing methods introduced in [Dav85]. In this section, seven of these methods to be applied to the expanded images are discussed, which are those that either have lower time complexity or better smoothing effect than the others not discussed here.

4.2.1. Variable Neighborhood Averaging

4.2.1.1. Description of Variable Neighborhood Averaging

Instead of using a neighborhood of a fixed size for the smoothing operation at each pixel in the expanded image, a variable neighborhood size can be used [Dav85].

This method is to use growing neighborhoods around the pixel until the number of gray values in the neighborhood that are substantially different from the pixel reaches a threshold. So the neighborhood will be large if the pixel is in a uniform region and small if the region has a high gray level variability. This method is based on the assumption that the uniformity of a surrounding region determines the degree of confidence that a pixel belongs to this region. If the region is uniform, this smoothing method based on a larger neighborhood will blend the pixel into the region. Otherwise, i.e, the local area is highly non-uniform, it is unlikely for the pixel to be in the interior of a region and smoothing is kept to a minimum region to avoid effects of blurring.

4.2.1.2. Time Complexity for Variable Neighborhood Averaging

For this method, it is difficult to analyze the time complexity since the size of the neighborhood is dependent on the threshold and the nature of the image to be expanded. It should also be noted that the extent of the blur present in the resulting images depends on the threshold choice.

4.2.2. Weighted Averaging

4.2.2.1. Description of Weighted Averaging

As known, the most straightforward smoothing method is local averaging, i.e. averaging over a neighborhood of a fixed size. For a uniform region, local averaging preserves the local average and reduces the variability. Thus isolated noise pixels can be suppressed; however, this operation has a blurring effect. If applied indiscriminately, sharp details will be blurred at the same time that noise is weakened. For instance, lines or edges are smoothed out by averaging across them.

Thus, local averaging over a fixed neighborhood as described is only effective for a homogeneous image, i.e. an image with very few local variations. The blurring effect is proportional to the chosen size of the fixed neighborhood. Thus small neighborhoods should be used to keep the blurring to a tolerable level.

To alleviate blurring, weighted averaging based on the pixels in the neighborhood can be used for image smoothing. In [Dav85], two common weighted averaging schemes are applied as follows. Suppose that p is the pixel at (x, y),

$$g_1(x, y) = \frac{1}{10}[\sum_{(i, j) \in N_8(p) - p} f(i, j) + 2f(x, y)] \qquad (4.1)$$

$$g_2(x, y) = \frac{1}{16}[\sum_{(i, j) \in N_4(p) - p} 2f(i, j) + \sum_{(i, j) \in ND(p)} f(i, j) + 4 f(x, y)] \qquad (4.2)$$

where $ND(p)$ is $N_8(p) - N_4(p)$.

(4.1) represents a mask of 3 × 3 pixels in Fig. 4.1 and (4.2) corresponds to the mask in Fig. 4.2. Averaging can be carried out across the image by centering the respective mask over each pixel in the image and multiplying each weight in the mask with the underlying pixel. Weights lying outside the mask are assumed to be 0's so that the sum of the pixels is the weighted sum. The divisor in each average is used as a normalization factor for the mask. The first mask assumes that each of the eight neighbors has an equal effect, but the second mask takes into consideration the difference in the distances between p and its vertical or horizontal neighbors and the distances between p and its diagonal neighbors, thus more weight is given to its vertical or horizontal neighbors than to its diagonal neighbors.

1	1	1
1	2	1
1	1	1

Fig. 4.1. Mask corresponding to (4.1)

1	2	1
2	4	2
1	2	1

Fig. 4.2. Mask corresponding to (4.2)

4.2.2.2. Time Complexity for Weighted Averaging

The number of multiplications in 3 × 3 window operation throughout the expanded image is $9k^2n^2$, thus the time complexity for the use of a 3 × 3 mask as in (4.1) or (4.2) is $O(k^2n^2)$ [Dav85].

4.2.3. Laplacian Smoothing

4.2.3.1. Description of Laplacian Smoothing

The Laplacian is a rotation-invariant second order difference that has zero response to linear ramps, but responds to the top and bottom of the ramp, where there is a change in the rate of gray value change [Ros82]. In [Dav85], the discrete approximation for the Laplacian is defined as follows:

$$Lap\, f(x, y) = 5f(x, y) - [f(x-1, y) + f(x+1, y) + f(x, y+1) + f(x, y-1)], \qquad (4.3)$$

where it can be noted that the approximation of the Laplacian of a gray level image f is proportional to the difference between f and an average of f.

Similar to gradient smoothing [Dav85], a pixel can be smoothed with the k neighbors with the Laplacian values closest to that of p. If p is in a homogeneous region, the response is small and therefore it is averaged with pixels in the same homogeneous region. When p is near an edge, its Laplacian value is large, and the neighbors on the same side of edge as p have Laplacian values closest to that of p, thus the average of p and those pixels are then assigned to p. In the thesis, k is chosen as 5 for this method. It should be noted that the Laplacian has an advantage over the gradient in that the Laplacian is rotation-invariant, and the polarity of a pixel determines the location of the pixel relative to an edge. For the gradient, its direction has to be evaluated in addition to its magnitude.

4.2.3.2. Time Complexity for Laplacian Smoothing

The time complexity for this method depends mainly on the calculation of the Laplacian value at each pixel, and finding those pixels whose Laplacian values are close to that of p. The above operation also involves the sorting of the Laplacian values of these pixels so that the pixels with their Laplacian values closest to that of p can be easily selected. The cost of sorting is a constant, $c1$, that depends on the kind of sorting scheme applied. The size chosen for each neighborhood is 3×3 and the cost for the calculation for the Laplacian at each pixel is a constant, $c2$. Hence, the time complexity for this method is $O(k^2n^2)$. It should also be noted that the magnitude of constant $c2$ depends on the Laplacian approximation schemes.

4.2.4. Directional Smoothing

4.2.4.1. Description of Directional Smoothing

Instead of using the gradient or Laplacian to indicate edge information in the vicinity of a pixel at (x, y), Rosenfield and Kak [Ros82] propose four measures, d_1, d_2, d_3, and d_4, to indicate edge information. Two methods are proposed in [Ros82]. In the first method, the first two measures, d_1 and d_2 are defined as in (4.4) and (4.5). If $max(d_1, d_2)$ exceeds some threshold T_1, then the average of the two pixels in the direction $tan^{-1}d_1/d_2$ and inside the 3 × 3 neighborhood of (x, y) is evaluated to the nearest $\pi/4$, and used to replace $f(x, y)$. Otherwise, the average of all the pixels in the 3 × 3 neighborhood is used to replace $f(x, y)$. It should be noted that since $tan^{-1}d_1/d_2$ approximates the gradient at (x, y), it indicates the edge orientation at (x, y).

Alternatively, d_3 and d_4 defined in (4.6) and (4.7) respectively can be used for directional smoothing in addition to d_1 and d_2. If $max\{d_1, d_2, d_3, d_4\} - min\{d_1, d_2, d_3, d_4\}$ exceeds the predefined threshold T_2, the two pixels in the direction of the minimum absolute gray level difference, i.e. in the direction determined by $min\{d_1, d_2, d_3, d_4\}$, is used. The following formulas were proposed in [Ros82]:

$$d_1 = |(f(x-1, y+1) + f(x, y+1) + f(x+1, y+1)) - (f(x-1, y-1) + f(x, y-1) + f(x+1, y-1))| / 3 \tag{4.4}$$

$$d_2 = |(f(x-1, y-1) + f(x-1, y) + f(x-1, y+1)) - (f(x+1, y-1) + f(x+1, y) + f(x+1, y+1))| / 3 \tag{4.5}$$

$$d_3 = |(f(x-1, y) + f(x-1, y+1) + f(x, y+1)) - (f(x, y-1) + f(x+1, y-1) + f(x+1, y))| / 3 \tag{4.6}$$

$$d_4 = |(f(x, y-1) + f(x-1, y-1) + f(x-1, y)) - (f(x+1, y) + f(x+1, y+1) + f(x, y+1))| / 3. \tag{4.7}$$

In this thesis, the latter scheme is applied to smooth the expanded images.

4.2.4.2. Time Complexity for Directional Smoothing

In the worst case, when the threshold is exceeded for each pixel, the time complexity for each scheme presented above is $O(k^2n^2)$ [Dav85]. However, the second scheme will require a constant number of extra operations per pixel more than the first scheme. By choosing the proper thresholds, the amount of blurring can be controlled.

4.2.5. Half Neighborhood Methods

4.2.5.1. Description of Half Neighborhood Methods

In [Sch80], an alternative to using the gradient and Laplacian at each pixel p, the possibility of an edge passing through a neighborhood of p can be used for edge smoothing. If there is sufficient evidence of such an edge, the pixel p is averaged with the pixels on the same side of the edge.

The 8 neighbors of p are used to determine the edge information. These eight neighbors are divided into two groups of consecutive pixels. Let $N_1(x_i)$ be the group with 5 pixels x_i, $x_{(i+1)mod8}$, ..., $x_{(i+4)mod8}$, and $N_2(x_{(i+5)mod8})$ with 3 pixels, $x_{(i+5)mod8}$, $x_{(i+6)mod8}$, $x_{(i+7)mod8}$, for $i = 0, 1, ..., 7$, where the tessellation of the 8 neighbors are the same as defined in Chapters 1 and 3. The region containing p is the group of $N_1(x_i)$ with the nearest average gray value to that of p, for some i in $[0, 7]$. The presence of an edge is then decided by the magnitude of $|ave(N_1(x_i)) - ave(N_2(x_{(i+5)mod8}))|$ where the value for $ave(N_j(x_i))$ is the average gray value for $N_j(x_i)$ for $j = 1, 2$ and $0 \leq i \leq 7$.

Another method is also presented in [Sch80]. For some predefined threshold T, if $|ave(N_1(x_i)) - ave(N_2(x_{(i+5)mod8}))| > T$, it is then assumed that an edge passes through the neighborhood of p and p is most likely to be in a region together with $N_1(x_i)$. Thus, p is smoothed with the pixels in $N_1(x_i)$. Otherwise $|ave(N_1(x_i)) - ave(N_2(x_{(i+5)mod8}))| \leq T$, and p is inside a bigger region and the average gray value of $N_1(x_i) \cup N_2(x_{(i+5)mod8}) \cup p$ for some $i = 0, 1, ..., 7$ is used.

A different criterion to determine the presence of an edge is to use $|ave(N_1(x_i)) - ave(N_2(x_{(i+5)mod8}))| > T$ and then choose the largest $|ave(N_1(x_i)) - ave(N_2(x_{(i+5)mod8}))|$ to determine the presence of an edge. Thus the strongest possible edge is chosen, and p is then given the average of the gray values of the pixels in $N_1(x_i) \cup p$. Otherwise, p is given the average of the gray values of all pixels inside the current 3 × 3 window. In this thesis, the last criterion is applied for image smoothing.

4.2.5.2. Time Complexity for Half Neighborhood Methods

The first scheme requires the determination of the half neighborhood $N_1(x_i)$ with $ave(N_1(x_i))$ nearest to the gray value of the pixel that is being processed. For each half neighborhood four additions and a division is needed. The nearest gray value can be found by determining the smallest absolute difference between the average gray values of the half neighborhoods and the pixel that is being processed. Hence, locating the proper half neighborhood will require 8 divisions and 40 additions/subtractions. The number of operations needed to process each pixel in the image is the same. Therefore, $O(k^2n^2)$ operations are needed. Again, the amount of smoothing can vary, depending on the value of the selected thresholds.

The last scheme locates the largest difference, $|ave(N_1(x_i) - ave(N_2(x_{(i+5)mod8}))|$ and checks to see if a given threshold is exceeded. In this case, averaging is performed using the pixels in $N_1(x_i)$. Since $ave(N_2(x_{(i+5)mod8}))$ is calculated for each pixel, this scheme will require a constant number of additional operations per pixel more than the first scheme when the worst cases are considered. The time complexity is therefore $O(k^2n^2)$, but with a larger constant than in the first scheme [Dav85].

4.2.6. Maximum Homogeneity Smoothing

4.2.6.1 Description of Maximum Homogeneity Smoothing

Instead of using edge information for image smoothing, an alternative approach examines a set of fixed neighborhoods around a pixel and the average gray value of the region with the least variability in gray value is given to that pixel. The motivation is due to the degree of confidence that the pixel is in that region.

To find the most homogeneous region around a pixel p, [Nag77] applies pentagonal and hexagonal masks for smoothing as shown in Fig. 4.3. In Fig. 4.3, a pentagonal T and a hexagonal X region is shown. The average gray value and gray value variance of each pentagonal or hexagonal region defined by the corresponding mask are calculated and p is then given the average gray value of the region with the lowest variance. With this smoothing method, no independent regions with false gray levels can arise.

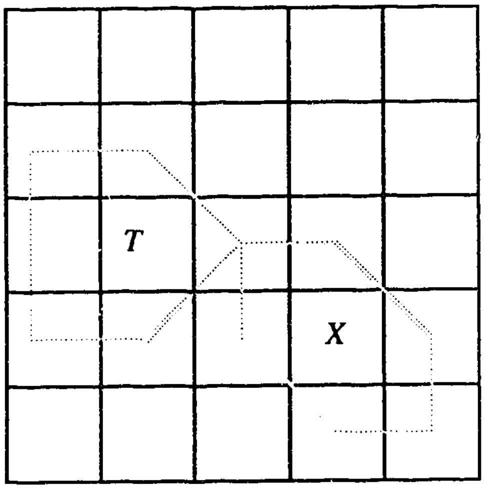


Fig. 4.3. Pentagonal and hexagonal neighborhoods

4.2.6.2. Time Complexity for Maximum Homogeneity Smoothing

For this method, the mean value and standard deviation should be calculated to determine the neighborhood with the least gray level variability. The operation of mean and standard deviation for each of the 8 neighborhoods (hexagonal or pentagonal region) is needed, for each pixel to be processed. Each mean needs 1 division, and each standard deviation needs 7 multiplications (for 7 pixels in each hexagonal or pentagonal region). The number of operations for each pixel to be processed is 8 × (7 + 1) multiplications/divisions, thus for the whole resulting image, the time complexity is $O(k^2n^2)$.

4.2.7. Selective Neighbor Averaging

4.2.7.1. Description of Selective Neighbor Averaging

An alternative to variable neighborhood averaging is selective neighborhood averaging, i.e. instead of using the size of the neighborhood of a pixel $f(x, y)$ to effect the amount of smoothing, the m pixels with gray levels nearest to $f(x, y)$ in a fixed size neighborhood can also be employed [Dav85]. Usually the size of the neighborhood is chosen to be 3 × 3 and $m = 2, 4, 6$ are commonly used. In particular, when $m = 2$, edges and lines passing through a 3 × 3 neighborhood are preserved best since the two pixels adjacent to $f(x, y)$ have the two closest gray values [Dav85]. In this thesis, the selective neighbor averaging with $m = 2$ is applied to the expanded images resulting from *Heuristic Expansion*.

4.2.7.2. Time Complexity for Selective Neighbor Averaging

Again, the number of additions involved when m nearest gray level pixels as the processed pixel are used is mk^2n^2 and the number of divisions is k^2n^2. Before these m pixels can be selected, however, the pixels in each neighborhood must be sorted. The number of comparisons needed in the worst case when *quicksort* is used is 3^2c for some $c > 0$, for 3 × 3 neighborhoods [How78]. So, the time complexity for this smoothing method is also $O(k^2n^2)$, the constant magnitude of which depends on the choice of m and the sorting method [illegible] applied [Dav85].

4.3. *Heuristic Smoothing*

4.3.1. Description of *Heuristic Smoothing*

The algorithm for *Heuristic Smoothing* aims to remove any jaggies resulting from edge expansion while preserving the sharpness of edges.

Suppose that the expansion factor in both x and y directions is k. A $k \times k$ window operation[1] is applied to the expanded image, starting from the upper left corner to the lower right corner of the image.

---

[1] When k is even, a $(k+1) \times (k+1)$ window is applied.

First, the minimum and maximum gray values inside the window are calculated. If the difference between the minimum (*min*) and maximum (*max*) gray value is less than a predefined threshold T, it is assumed that a homogeneous region is detected. Thus only unweighted averaging is done to the central pixel of the window, using the gray values of all the pixels inside the window. Otherwise, the largest homogeneous region inside the window is obtained, and the average of the gray values of all the pixels inside the largest homogeneous region is calculated, and the average is then assigned to the central pixel in the window.

To simplify the calculation complexity of the largest homogeneous region, the classification of pixels can be done first. Similar to the criterion used in edge or line detection, *min+(max-min)/2* can be applied for the pixel classification, for which an assumption is made that the histogram for the window covering the boundary of any two regions is usually *bimodal*, since the two regions are assumed to be homogeneous, and there are considerable gray value difference between them. Thus, *min+(max-min)/2* approximates the position of the *valley* in the histogram, where there are least pixels belonging to either region, so least errors for classification will occur.

Based on the above, the pixel classification is as follows. Any pixel value is greater than *min+(max-min)/2* is classified to one region, and the other pixels are classified to another region, just as in the edge or line segment detection. This assumes that there are, at most, two regions inside the window. In comparison, there are assumed to be two kinds of pixels in the 3×3 window used in edge or line detection, i.e. background pixels and edge or line pixels. This assumption is reasonable, because the size of the window in each direction applied for smoothing is approximately the same as the expansion factor, so it is most probable that at most two regions are detected inside the window applied[2]. There is

---

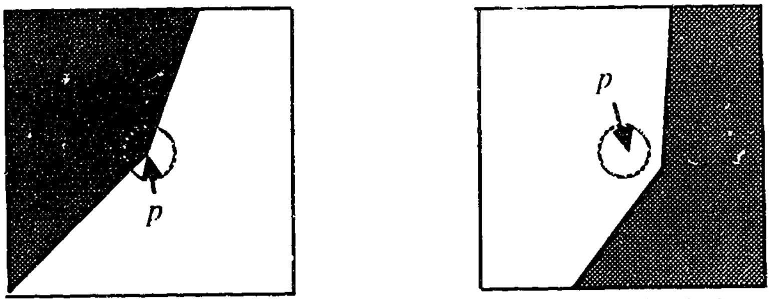
[2] Suppose that k is the expansion factor in both x and y directions. Then in the expanded image, there are at most 2 different regions in a $k \times k$ patch, to which a pixel of the original image is mapped.

still some possibility that the window covers the boundaries for more than two regions, but this case is far less probable than the former, thus it is not considered in this method for the sake of time complexity.

The region with the larger number of pixels is the larger region (inside the window). The average operation is done over the pixels inside the larger region, and then assigned to the central pixel in the window.

Heuristic Smoothing is based on a *heuristic* majority rule: it is more likely and reasonable that the central pixel of the window belongs to the larger region, in which there are more pixels of the same kind in terms of gray value uniformity, as long as both regions inside the window are homogeneous. Besides, averaging using the pixels of the same region (selective averaging), is least likely to cause blurring. So it usually has much less blurring effect than weighted averaging, which does not discriminate between pixels of different regions.

On the other hand, it is noted that a jaggy resulting from *Heuristic Expansion* usually causes a sharp edge orientation change, as shown in Fig. 4.1. Whenever each pixel near where the orientation change occurs is to be processed (as the current central pixel in the window), it will be assigned the average gray value of the larger region. If the pixel belongs to the smaller region, as shown in Fig. 4.1a (which is usually the case for smaller region, because of the geometrical feature of a jaggy), it will be merged into the larger region, thus the jaggy is greatly eliminated. If the pixel belongs to the larger region, as shown in Fig. 4.1b, its gray value is averaged with those of all the other pixels in the larger region, thus it still belongs to the larger region. So it is noted that there is a *bias* towards the larger region in smoothing. Furthermore, among all the smoothing methods discussed in this thesis, only *Heuristic Smoothing* accounts for the expansion factor, i.e. the window size is determined by the expansion factor, thus the amount of information of image features which can be held in the window is proportional to the expansion factor, whereas none of the other smoothing methods take the expansion factor into account This is why *Heuristic Smoothing* has the best smoothing effect, with the least amount of blurring and the greatest elimination of jaggies, among all the smoothing methods discussed in this thesis, for images with homogeneous regions.



(a) central pixel p belongs to smaller (shadowed) region

(b) central pixel p belongs to the larger (white) region

Notes: in (a), p is merged into the larger region

Fig. 4.4. Demonstration of *Heuristic Smoothing*

4.3.2. Time Complexity for *Heuristic Smoothing*

With *Heuristic Smoothing*, the time complexity is $O(s^2k^2n^2)$, where s^2 is the size of the window used for the smoothing. This includes the number of operations for the calculation of the maximum and minimum values inside the window, $2s^2k^2n^2$, and that for the calculation of the average value, $s^2k^2n^2$.

4.4. Total Time Complexity of the Algorithm

As discussed earlier, the time complexity for each smoothing method is either $O(k^2n^2)$, or $O(s^2k^2n^2)$ (for *Heuristic Smoothing* only), except for the variable neighborhood averaging, which is difficult to determine, as mentioned in Section 4.2.1.2.

Thus, the total time complexity for the whole algorithm, i.e. *Heuristic Expansion and Smoothing Algorithm*, at the worst case, is $O((s^2k^2+k^2)n^2)$, while that for the algorithm introduced in [Atw89] is $O(k^2n^2 + n^4)$. When the size of the image to be expanded is large relative to the expansion factor k and the window size s applied (that is usually true), the algorithm introduced in this thesis is better than that in [Atw89], with regards to time complexity. For the interpolation techniques presented in Chapter 2, the time complexity for each is $O(k^2n^2)$, while the *piecewise Hermite interpolation* and *bicubic spline interpolation* have much larger constants than bilinear interpolation [Dav85]. So, *Heuristic Expansion and Smoothing Algorithm* has the same magnitude of time complexity as the interpolation techniques.

4.5. Summary

In this chapter, several smoothing methods in [Dav85] are first discussed as possible approaches to the jaggy elimination. These methods suffer either from image blurring or unnoticeable jaggy elimination effect. So a part of the new algorithm, called *Heuristic Smoothing* is presented and discussed, which has the best smoothing effect for images with homogeneous regions among all the smoothing methods presented in this thesis because it greatly eliminates jaggies with the least amount of blurring results.

Chapter 5

Implementation, Results and Analysis

5.1. Implementation

The image expansion and smoothing methods discussed in the previous chapters, including *Heuristic Expansion and Smoothing Algorithm*, are implemented on an *IIS*, i.e., *International Imaging System Model* 75, an image processing system connected to a Vax 11-780.

Images are digitized as 2-D arrays. The gray level scale is in the range 0 to 255. Short integers are used for implementation efficiency and storage considerations. Four digitized images stored in the system are used as test data, representing four different types of images, as shown in Plate 5.1. Among the four images are: an image of text, called *TEXT*, an image of a lady's face, called *FIGURE*, an image of the moon, called *MOON*, and an image taken from an aerial photograph of some trees, called *BUSH*.

5.2. Results and Analysis

In this section, the expansion and smoothing results of the above four images using the methods presented in the previous chapters are discussed. An analysis is then made of the results obtained.

The four images are first expanded with *Heuristic Expansion*, and then the expanded images are smoothed with *Heuristic Smoothing* and other smoothing methods discussed in the last two chapters respectively, as shown from Plate 5.2 to Plate 5.10. In Plate 5.11 and

Plate 5.12, the images expanded with bilinear interpolation and pixel replication are also shown for comparison with all the other results.

In order to achieve the best possible results, the threshold values are carefully chosen for *Heuristic Expansion*, *Heuristic Smoothing*, and the other smoothing methods to which the values are applicable. The threshold selection are usually based on the histograms of the images to be expanded, especially for *Heuristic Expansion*, and *Heuristic Smoothing*, thus the threshold value is usually different for each image to be expanded. So the threshold selection is both *heuristic* and *ad hoc*.

From the above, it should be noted that the solution to image expansion by *Heuristic Expansion and Heuristic Smoothing Algorithm*, as well as the other smoothing methods in [Dav85], like all the previous techniques, is *ad hoc*. This conclusion is not only restricted to this area of image processing [Dav85]. As known, there are three reasons for the lack of universality in current solutions. First, no single model can be used to describe image enhancement; the actual models used are dependent on the nature of the processes by which the image was obtained. Second, the computational considerations usually require approximations, e.g. the gray values in digital images are discrete instead of continuous. Third, there is no universally agreed criteria by which the quality of the processed image can be judged, thus it is difficult to find proper numerical measures to indicate that one processed image is better than the other, with regards to the visual quality. The fourth reason is that the expansion and smoothing methods introduced here do not consider the textures for the sake of time complexity. Only gray value difference is used to distinguish the different regions in the images to be expanded. Thus, the expansion approach yields better visual effects for images with homogeneous regions (i.e., with less complex contents). As an example, the expanded result of *TEXT* has better visual effects than that of *FIGURE*, as shown in Plate 5.2.

For result analysis, the numerical evaluation measures in [Dav85], as well as the human perception, i.e. subjective evaluation are applied to the expanded images, and comparison between numerical measures and subjective evaluation is made, during the thesis research.

The numerical measures in [Dav85] are based on some kinds of pixel gray value difference between the original and expanded images. Because of this, a smaller resulting measure for an expansion technique should indicate a closer resemblance of the processed image to the original one, according to the common sense. Thus, for a measure to be consistent with human visual evaluation, its values for the most of the expansion methods should be smaller than that for pixel replication. However, it is found that none of the measures is consistent with the human visual evaluation. Instead, *Heuristic Expansion and Smoothing Algorithm* has the best visual effect among all the other methods for *TEXT*, the image with homogeneous regions, according to human visual evaluation, but this conclusion cannot be drawn from the numerical measures. Thus it is concluded that better visual effect of the processed image does not mean closer resemblance of the expanded image to the original one with regard to pixel gray value difference.

Assuming, however, that each measure is consistent for all expansion methods, it is possible to compare the performance of an expansion method for different image contents using each of the measures, which assumes that a smaller value of the measure indicates the better performance of a corresponding expansion method. Based on the above, it is shown that *Heuristic Expansion and Smoothing Algorithm* has better expansion effect on images with homogeneous regions, e.g. *TEXT*, than the images with textures, e.g. *BUSH*.

From the above and the previous work concerned, it is noted that there is no numerical measure that is consistent with human visual evaluation to judge the effect of image expansion. Actually this is a major problem that is closely related to psychology and needs to be solved for performance evaluation of image processing [Dav85]. Thus, the human visual evaluation is a very important factor in the quality judgment of image expansion discussed in this thesis. The numerical measures, however, are still useful references for the evaluation on the performance of an expansion method on images of different natures, as mentioned above.

Based on the above, in the quality judgement of image expansion, new numerical measures should be developed and applied together with the human visual evaluation, in order to find an expansion method which can effect the least deviation of the expanded images from the original images, for the sake of image fidelity, while good visual effect is maintained; these will involve the development of a more suited model for human vision. This is an interesting topic and it is expected that considerable work can be done for it.

Plate 5.1. The Original Images: FIGURE, MOON, TEXT and BUSH

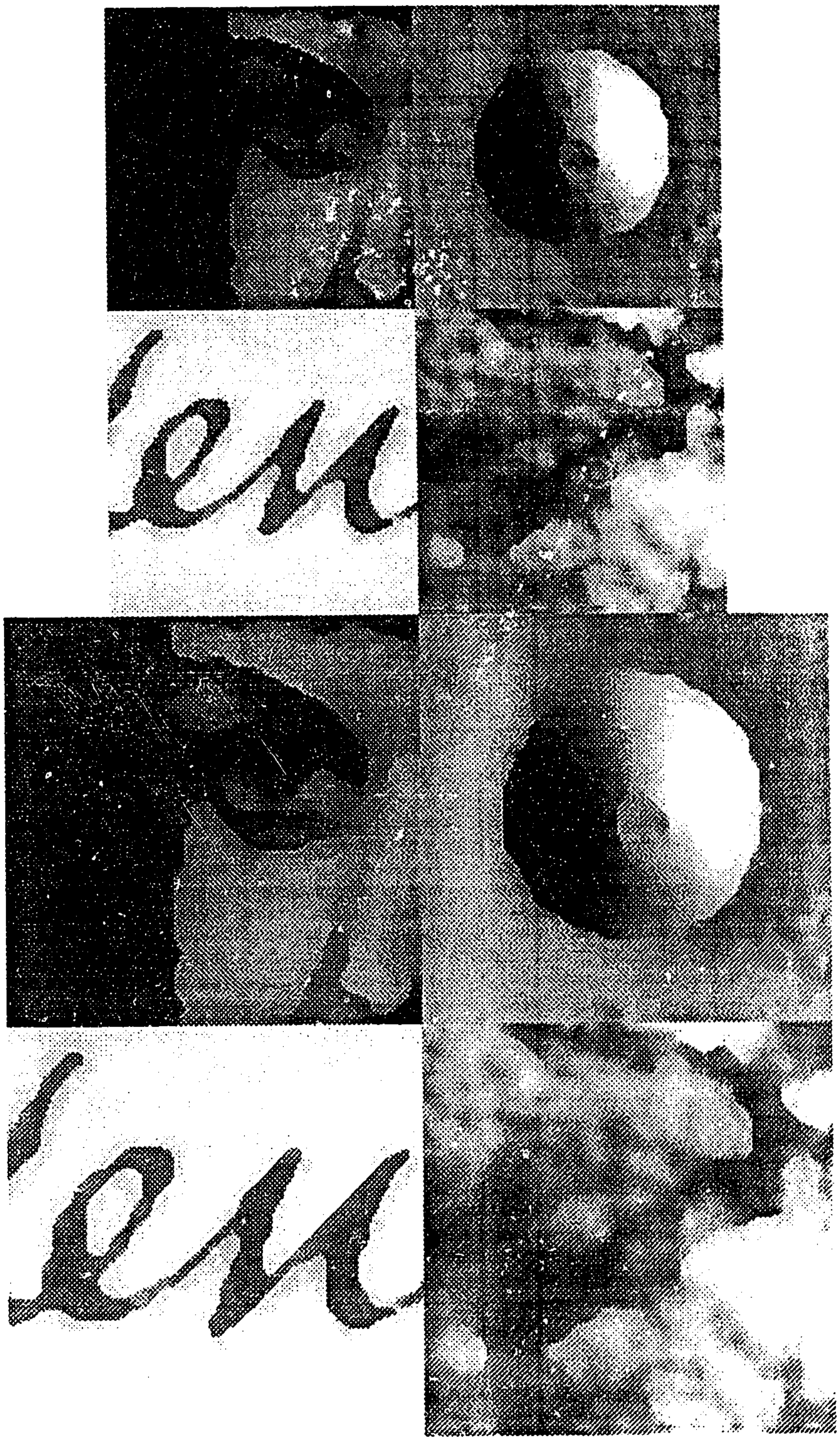


Plate 5.2. Results of Heuristic Expansion and Smoothing (Exp.Factor= 3,4)

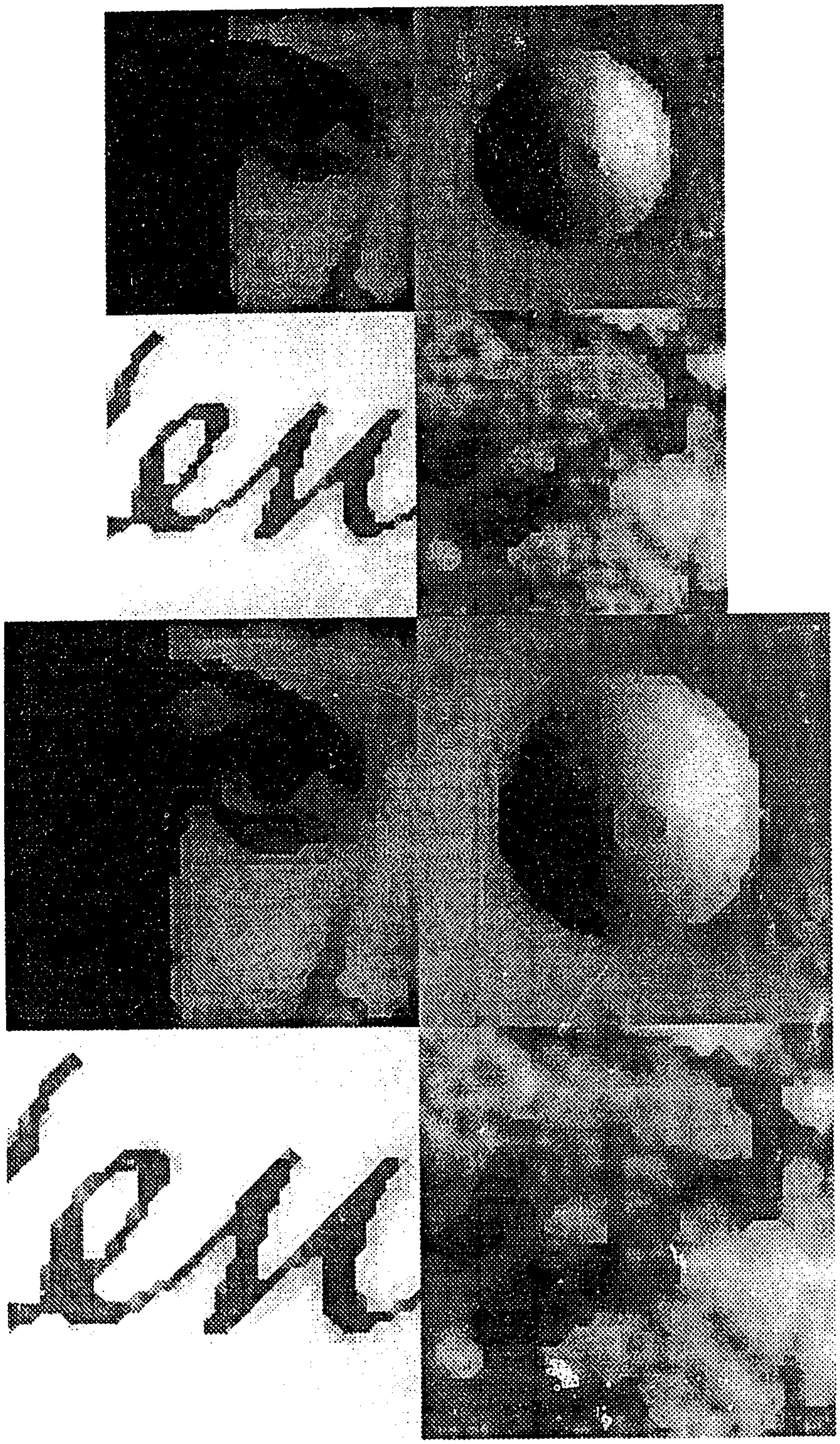


Plate 5.3. Results of Heuristic Expansion Only (Exp. Factor=3, 4)

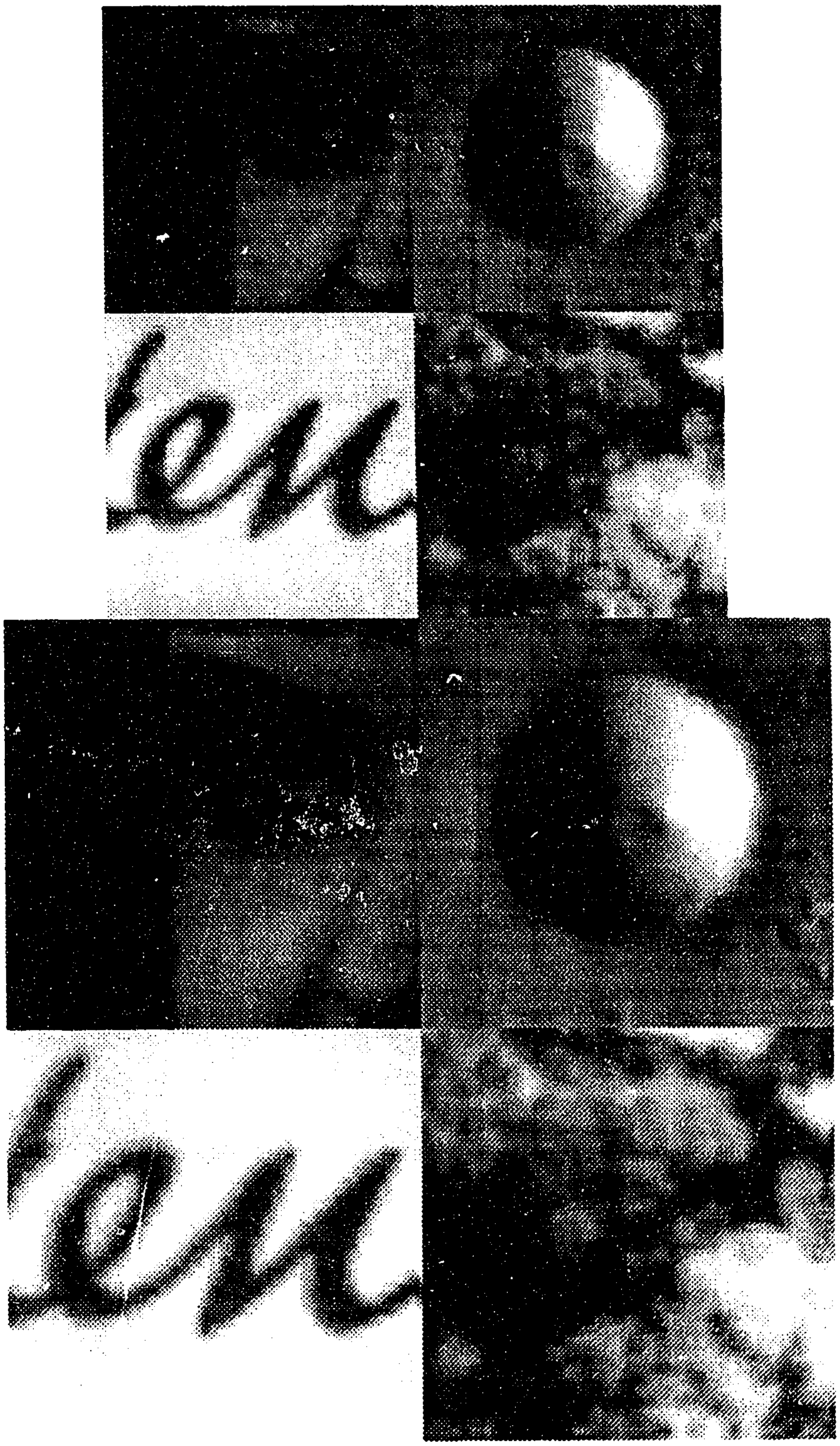


Plate 5.4. Results of Bilinear Interpolation (Exp. Factor=3,4)

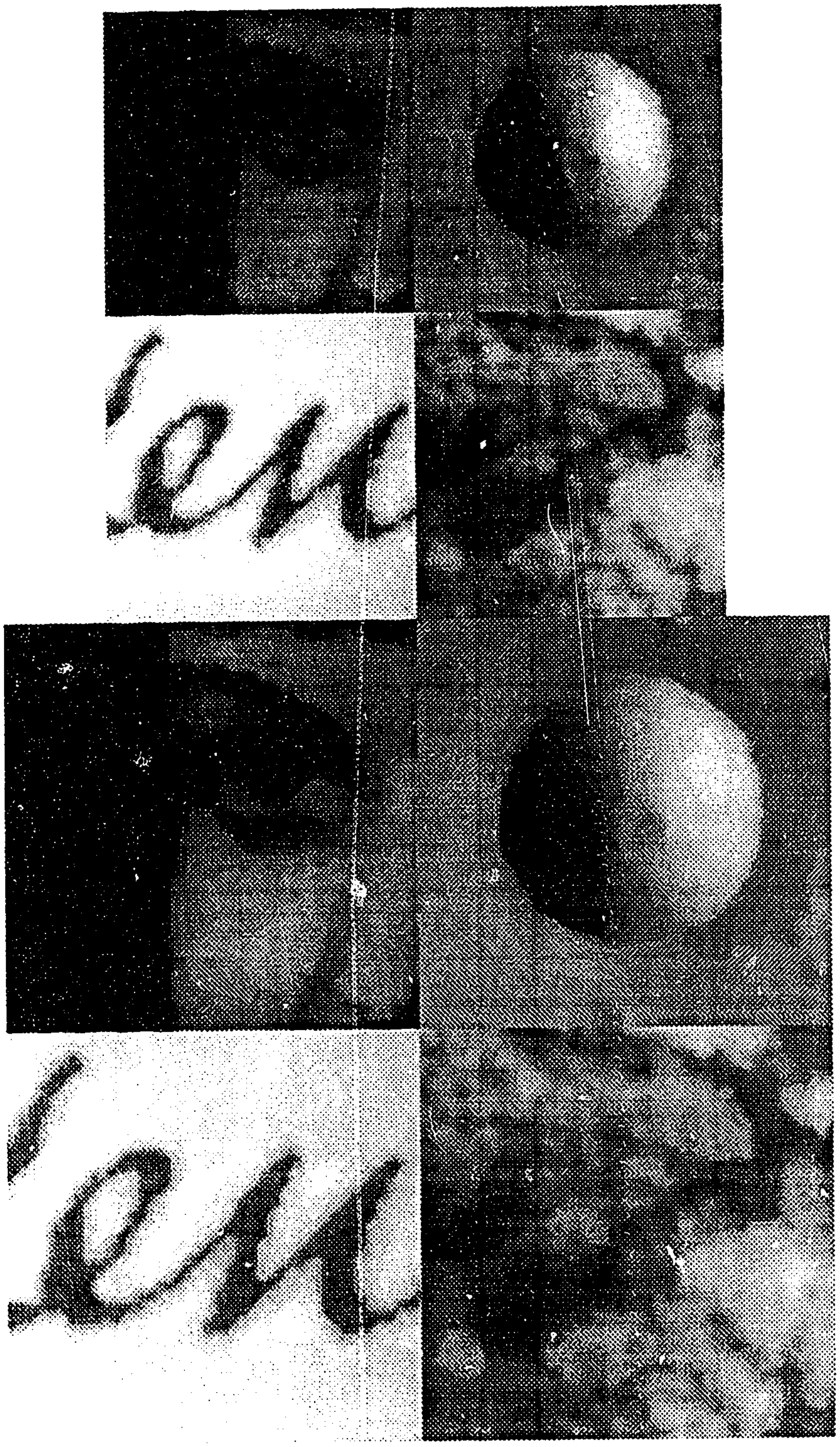


Plate 5.5. Results of Heuristic Exp. & Var. Neighborhood Ave. (Exp. Factor=3,4)

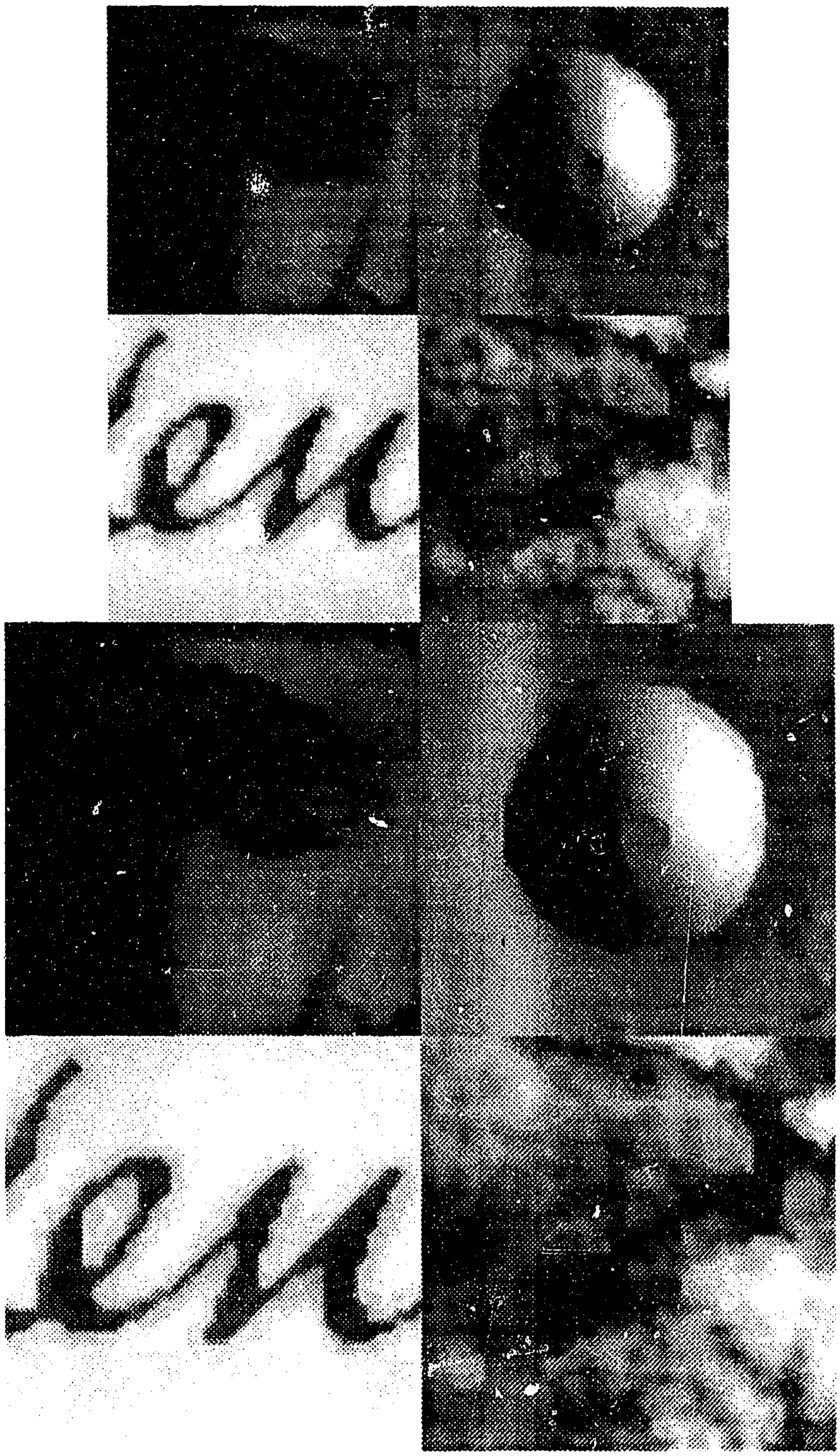
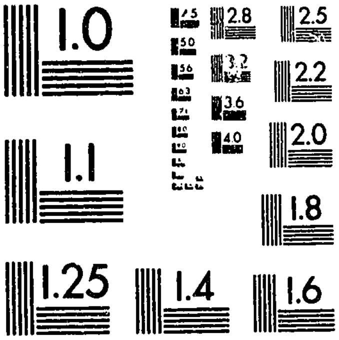


Plate 5.6. Results of Heuristic Exp. & Weighted Ave. (Exp. Factor=3,4)

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT



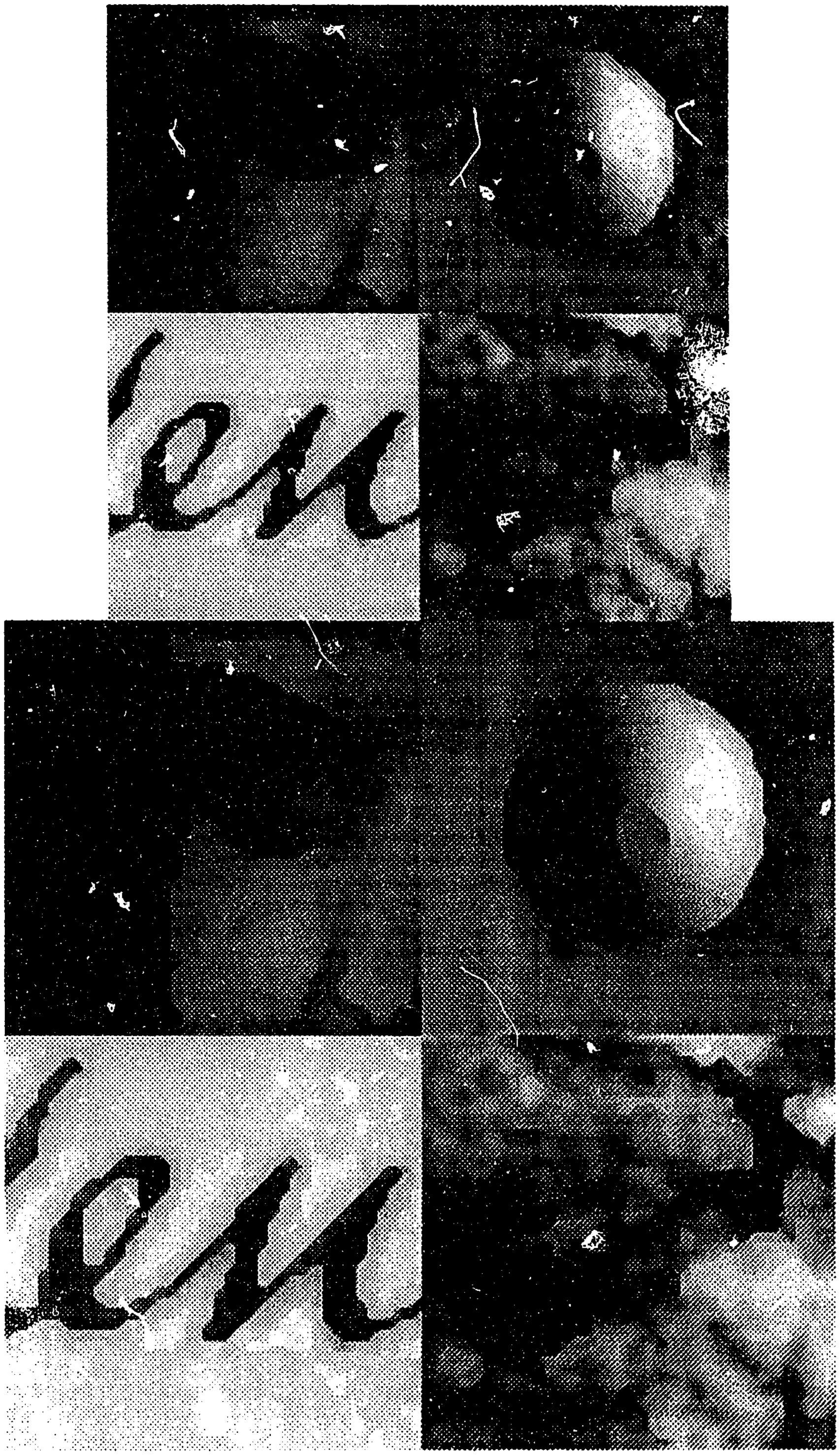


Plate 5.7. Results of Heuristic Exp. & Directional Smoothing(Exp. Factor=3,4)

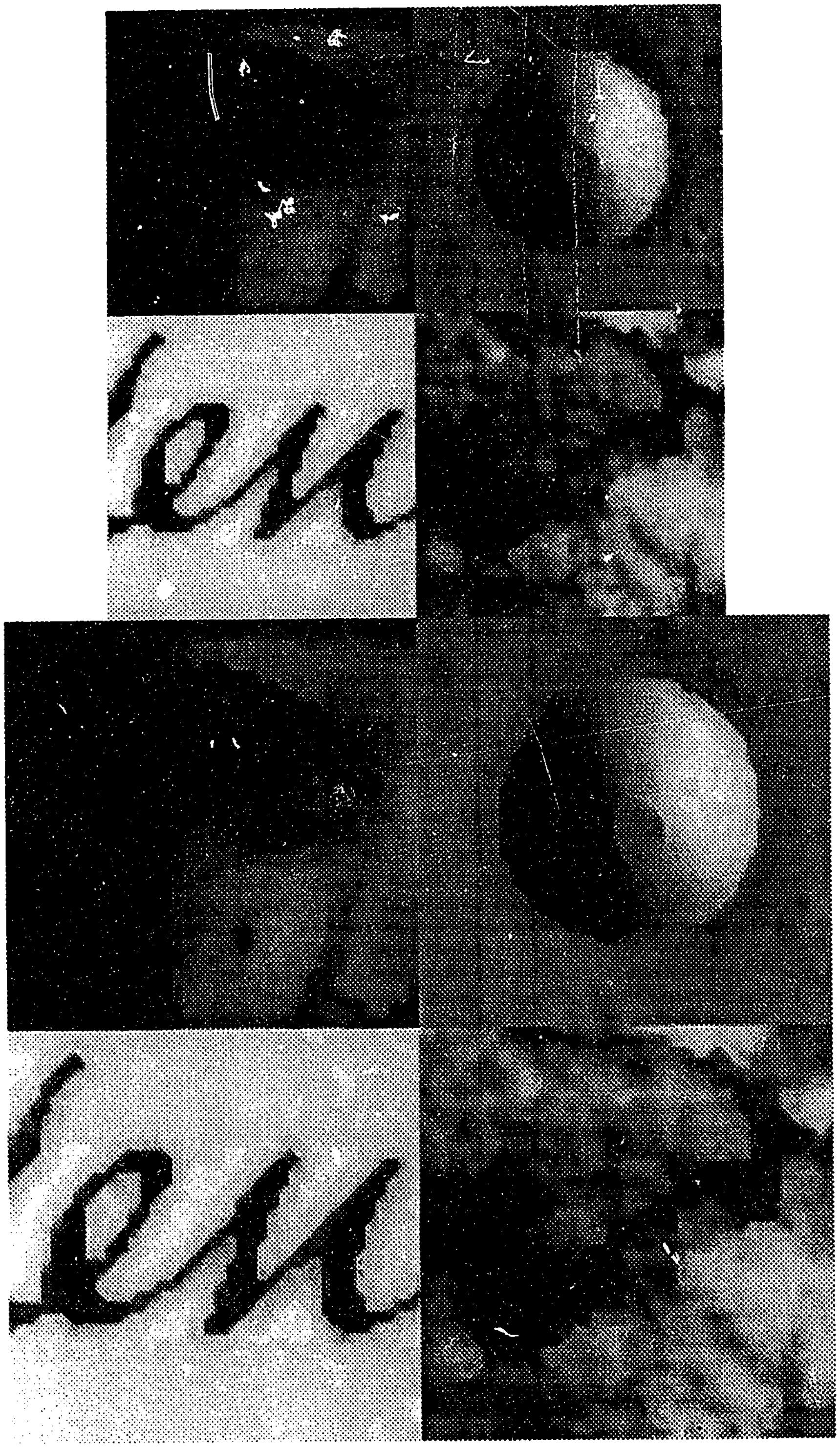


Plate 5.8. Heuristic Expansion and Half Neighborhood Method(Exp. Factor=3,4)

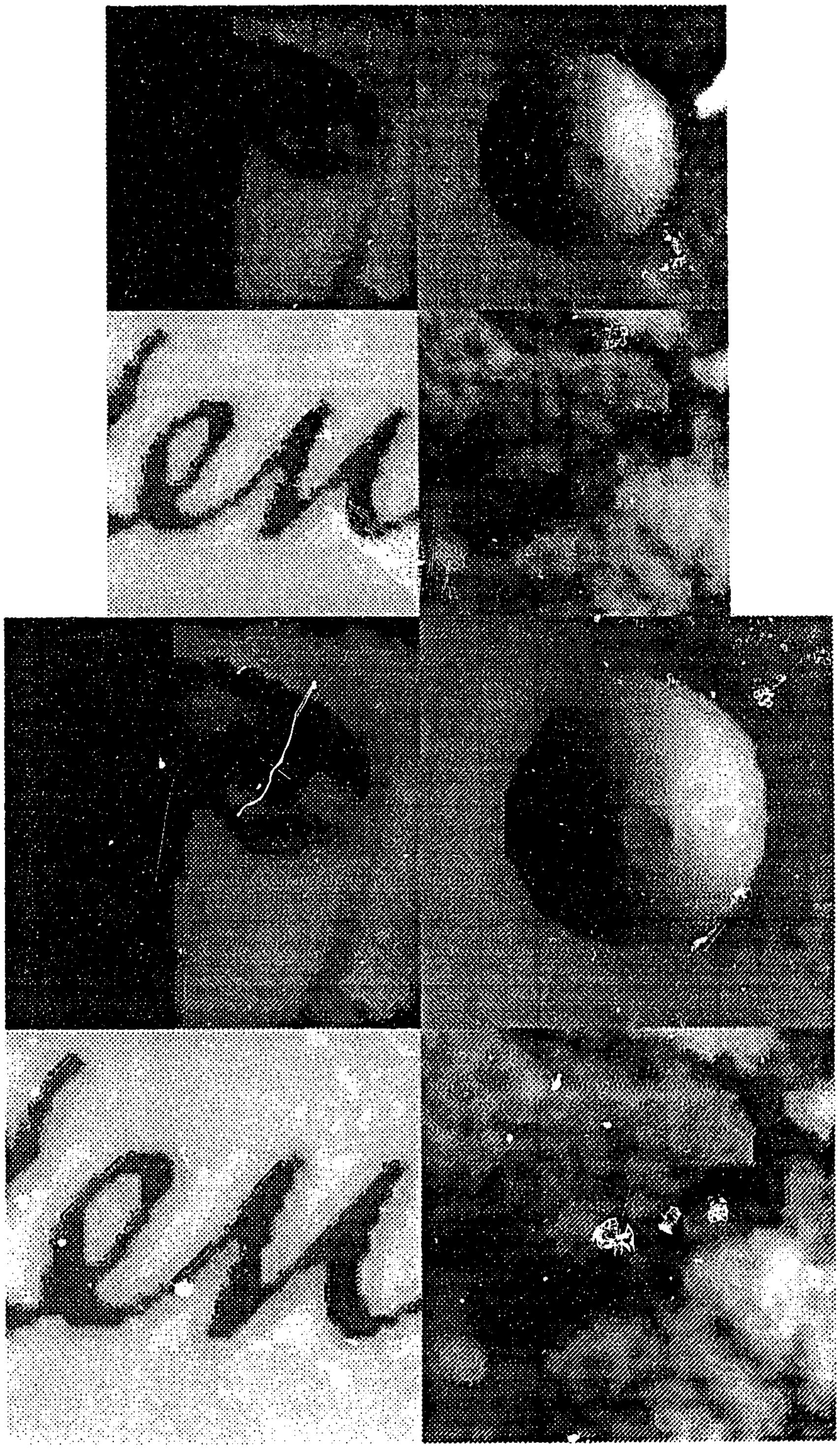


Plate 5.9. Heuristic Expansion and Max. Homogenity Smoothing(Exp. Factor=3,4)

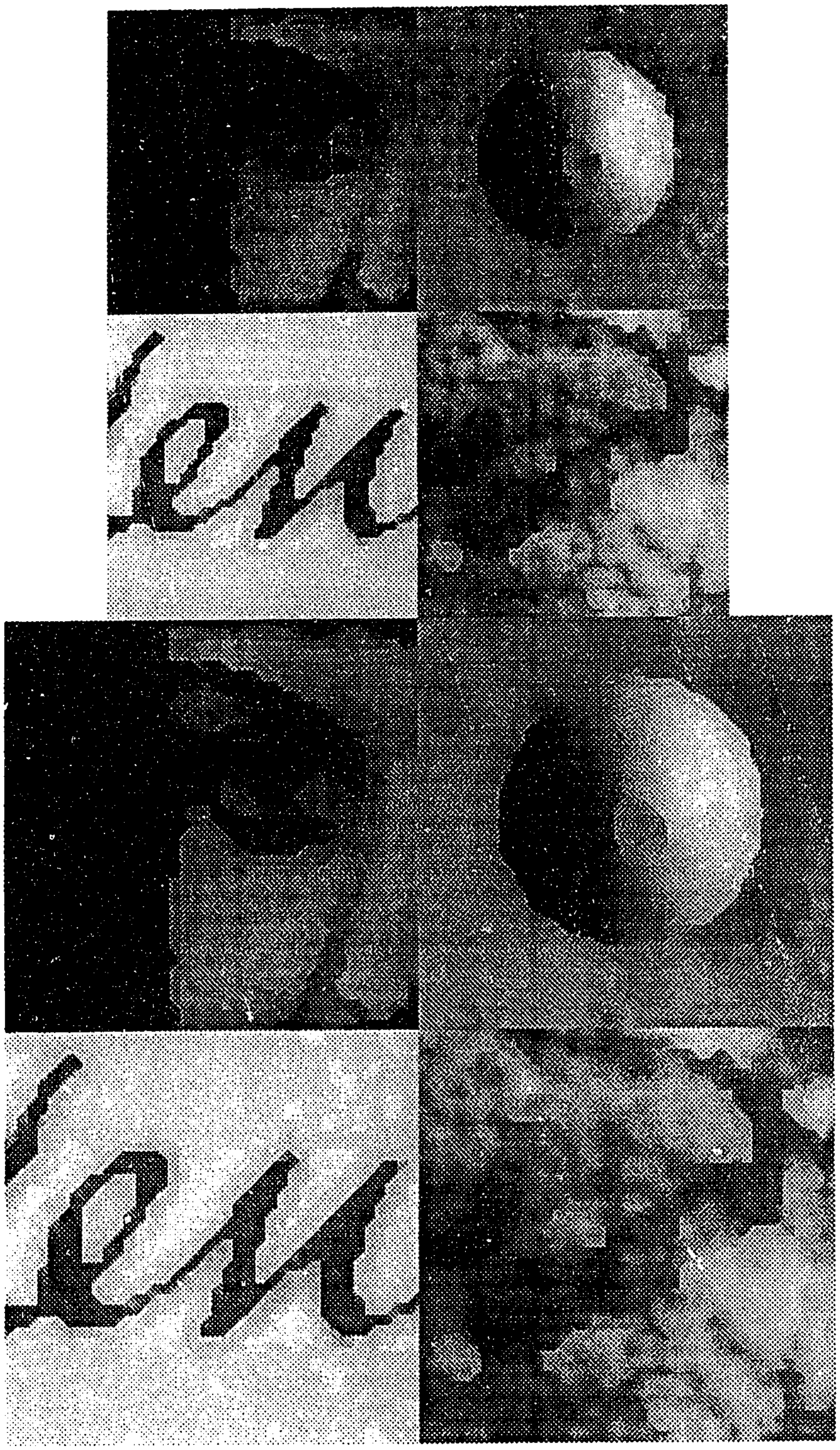


Plate 5.10. Heuristic Exp. and Selective Neighbor Smoothing(Exp. Factor=3,4)

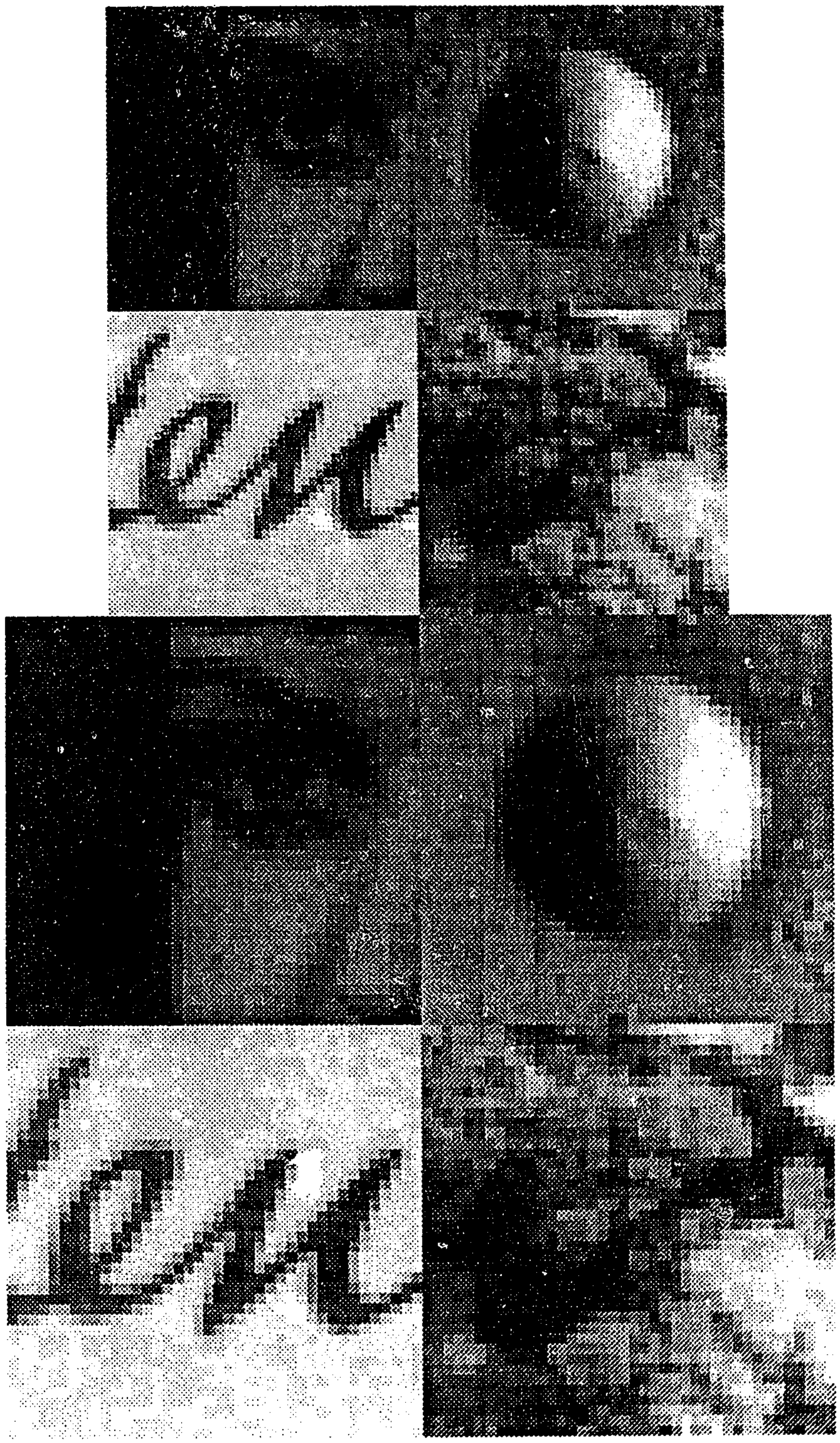


Plate 5.11. Results of Pixel Replication(Exp. Factor=3,4)

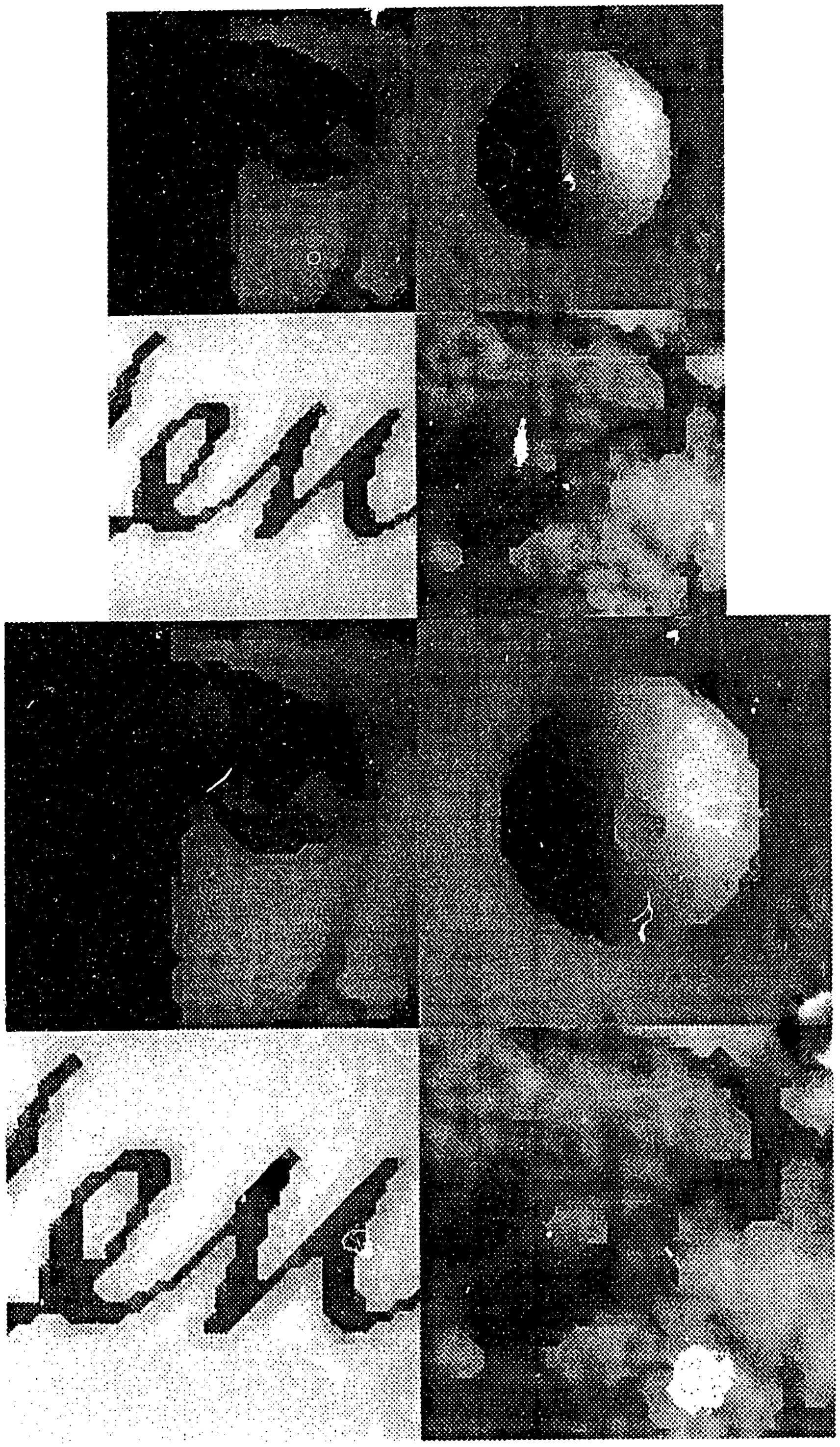


Plate 5.12. Heuristic Expansion & Laplacian Smoothing(Exp. Factor=3,4)

Chapter 6

Conclusion

6.1. Overview of the Thesis

The thesis investigates the employment of a new heuristic method for image expansion, called *Heuristic Expansion*, and a new heuristic smoothing method, called *Heuristic Smoothing*, to yield visually satisfactory image expansion. *Heuristic Expansion* is composed of four techniques, bilinear interpolation, *Edge_Line_Differentiation*, *Heuristic Edge Expansion*, and *Heuristic Line Expansion*. The above two methods comprise a new image expansion algorithm, called *Heuristic Expansion and Smoothing Algorithm*.

6.2. Summary

The *Heuristic Expansion and Smoothing Algorithm* presented in this thesis yields visually satisfactory results, compared with those obtained with previous techniques, such as the interpolation methods in [Dav85] and the heuristic edge following method in [Atw89]. The edge blurring effect is eliminated by *Heuristic Expansion*, and *Heuristic Smoothing* reduces most of jaggies caused by *Heuristic Expansion*, while causing the least amount of blurring, compared with the other smoothing methods. The total time complexity for this algorithm is better than that for the method in [Atw89], but a little worse than that for bilinear interpolation in [Dav85], in that the time complexity has a larger constant than that of bilinear interpolation.

6.3. Possible Applications

The new algorithm discussed in this thesis provides a good approach for faster and better high resolution display of low resolution image data. And it also provides good support for efficient image storage and transmission. So the prospects for the application of this method is promising.

6.4. Future Research

It is obvious that considerable work is still needed to get more visually acceptable expanded images, and yet faster display to accommodate more applications.

Here are some possible future research directions:

1. The pursuit of a more proper model for human vision, which will benefit the expansion methods.
2. Better interpolation methods or better approximations for the available interpolation methods.
3. Adaptive thresholding in different parts of an image to be expanded.
4. Partition of the general expansion problem into more specific sub-classes, e.g. binary images, so that more effective heuristics can be employed.
5. With the availability of cheaper hardware and the parallel nature of the image expansion methods, parallel processing adaptations of these methods should be investigated to reduce the time complexity, in order to make their real-time applications feasible.

Bibliography

[Atw89] Gordon H. Atwood, and Wayne A. Davis. Image Expansion Using Interpolation & Heuristic Edge Following. *3rd International Conference on Image Processing*, 18-20 July 1989, Warwick, England.

[Ado87] Adobe Systems Inc. *PostScript Language Tutorial and Cookbook*. Addison-Wesley, 1987.

[Bra78] R. Bracewell. *The Fourier Transform and its Applications*. McGraw Hill, 1978.

[Cas79] K. R. Castleman. *Digital Image Processing*. Prentice Hall, 1979.

[Dan82] Dana H. Ballard, et.al. *Computer Vision*. Prentice Hall, Inc., 1982.

[Dav85] Wayne A. Davis and Kok Wai Chan. *Image Expansion Using Interpolation and Noise-Cleaning Methods*. Technical Report TR 85-15, Department of Computing Science, University of Alberta, Nov. 1985.

[Dave79] J. P. Davenport. *A Comparison of Noise-Cleaning Methods*. Computing Science Technical Report 689, University of Maryland, College Park, Maryland, 1979.

[deB78] C. deBoor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[Dud73] R. O. Duda and D. E. Hart. *Pattern Recognition and Scene Analysis*. Wiley, 1973.

[Fol83] J. D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1983.

[Gon87] Gonzalez & Wintz. *Digital Image Processing*. Addison-Wisley, 1987.

[Hou78] H. S. Hou and H. S. Andrews. Cubic Splines for Interpolation and Filtering. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-26:508-517, 1978.

[How78] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms, Computer Science Press*, 1978.

[Lev82] A. Lev, S. W. Zucker and A. Rosenfeld, Interactive Enhancement of Noisy Images. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-7(7), 1977.

[Nag77] Nagao and Matsuyama. Edge Preserving Smoothing. *Computer Graphics and Image Processing*, 9:394-407, 1977.

[Net80] A. Netravali and F. W. Mounts. Ordering Techniques for Facsimile Coding: A Review. *Proc. IEEE*, 68-7:796-807.

[Pav82] Theo Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

[Pra78] William K. Pratt. *Digital Image Processing*. John-Wiley and Sons, 1978.

[Ros72] Azriel Rosenfeld, M. Thurston and Y. H. Lee. Edge and Curve Detection: Further Experiments. *IEEE Transactions on Computers*, c-21:77-715, 1972.

[Ros82] Azriel Rosenfeld, et. al. *Digital Picture Processing*. 2nd Edition, Vol. 1 & 2, Computer and Applied Mathematics, 1982.

[Sam80] H. Samet. Region Representation: Quadtrees from Boundary Codes. *Comm. ACM*, 23-3:163-170, 1980.

[Sch80] A. Scher, F. R. Velasco and A. Rosenfeld. Some New Image Smoothing Techniques. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-10:153-158, 1980.

[Spa74] H. Spath. *Spline Algorithms for Curves and Surfaces*. Unitas Mathematica Publ., 1974.

[Sto82] J. C. Stoffel. *Graphical and Binary Image Processing and Applications*. Artech House, 1982.