



**This electronic thesis or dissertation has been  
downloaded from Explore Bristol Research,  
<http://research-information.bristol.ac.uk>**

*Author:*  
**Smith, R. W**

*Title:*  
**The extraction and recognition of text from multimedia document images**

**General rights**

The copyright of this thesis rests with the author, unless otherwise identified in the body of the thesis, and no quotation from it or information derived from it may be published without proper acknowledgement. It is permitted to use and duplicate this work only for personal and non-commercial research, study or criticism/review. You must obtain prior written consent from the author for any other use. It is not permitted to supply the whole or part of this thesis to any other person or to post the same on any website or other online location without the prior written consent of the author.

**Take down policy**

Some pages of this thesis may have been removed for copyright restrictions prior to it having been deposited in Explore Bristol Research. However, if you have discovered material within the thesis that you believe is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please contact: [open-access@bristol.ac.uk](mailto:open-access@bristol.ac.uk) and include the following information in your message:

- Your contact details
- Bibliographic details for the item, including a URL
- An outline of the nature of the complaint

On receipt of your message the Open Access team will immediately investigate your claim, make an initial judgement of the validity of the claim, and withdraw the item in question from public view.

**THE EXTRACTION AND RECOGNITION OF TEXT  
FROM MULTIMEDIA DOCUMENT IMAGES**

**Raymond Wensley Smith**

**A thesis submitted to the University of Bristol in accordance with the  
requirements for the degree of Ph.D. in the Department of Computer Science,  
Faculty of Science.**

**November 1987.**

## ABSTRACT

Almost all the current commercial OCR machines employ matrix matching, resulting in high speed and accuracy, but a severely restrictive range of recognized fonts. Published algorithms conversely, concentrate on feature extraction for font independence, yet they have previously been too slow for commercial use. Current algorithms also fail to distinguish between text and non-text images. This thesis presents a new approach to the automatic extraction of text from multimedia printed documents.

An edge detection algorithm, which is capable of extracting the outlines of text from a grey level image, is used to obtain a high level of discrimination between text and non-text. An additional benefit is that text of any colour can be read from almost any background, provided that the contrast is reasonable. The outlines are approximated by polygons using a fast two-stage algorithm.

A feature extraction approach to font independent character recognition is described, which uses these outline polygons. It is shown that highly accurate and fast recognition can be achieved using a remarkably small number of carefully chosen features. The results show that after training on only seven quite similar fonts, the recognition algorithm provides greater than 95% accuracy on fonts different to the training set.

A more complex edge extraction algorithm is also described. This is capable of extracting text and line graphics from an arbitrary page. Although not essential for character recognition, this algorithm is useful for the interpretation of engineering drawings. As a further contribution to this problem, a thinning algorithm is defined, which is non-iterative and uses the polygonal approximated outlines from the edge extractor.

## ACKNOWLEDGEMENTS

My University supervisor, Dr. Eric Lewis has been of great assistance in listening to each new idea, and providing helpful suggestions, as well as proof reading my thesis.

I would also like to thank Steve Lieske, my initial Hewlett Packard supervisor, now returned to the USA, for having the insight to sponsor me and initiate a project which was almost completely unrelated to any other work in the Bristol division at that time.

Finally, I would like to thank my wife Tracey, for her continual support and encouragement.

AUTHOR'S DECLARATION

I declare that, except where otherwise stated, all the work described herein is entirely my own.

R W Smith

9-Nov-1987

## CONTENTS

Title Page	1
Abstract	2
Acknowledgements	3
Author's Declaration	4
Contents	5
List of Tables	9
List of Figures	10
Chapter 1. INTRODUCTION	16
1.1. Applications	16
1.2. The Two Main OCR Methods	17
1.2.1. Matrix Matching	17
1.2.2. Feature Extraction	19
1.3. Recently Published OCR Research	21
1.3.1. Extraction of Features From Outlines	21
1.3.2. Extraction of Features From Skeletons	24
1.4. Currently Available Machines	26
1.4.1. Low-End Matrix Matching Machines	26
1.4.2. Feature Extraction Systems	27
1.5. Unsolved Problems in OCR	28
1.5.1. Recognition Speed	28
1.5.2. Recognition Accuracy	29
1.5.3. Binary and Grey Level Images	29
Chapter 2. A NEW APPROACH TO CHARACTER RECOGNITION	33
2.1. The Scanner	33
2.2. Halftone Processing	34
2.3. Edge Extraction	35
2.4. Polygonal Approximation	36
2.5. Feature Extraction	37
2.6. Text Ordering	38
2.7. Final Classification	39
2.8. Text Removal	39
2.9. Image Compression	40
2.10. Matrix Matching	40
Chapter 3. EDGE EXTRACTION	41
3.1. Some Edge Detection Operators	41

3.2.	A Description of the New Edge Extractor	43
3.2.1.	Definition of the Edge Operator	43
3.2.2.	The Edge Following Algorithm	44
3.2.3.	The Choice of Edge Path	45
3.2.4.	A Simple Edge Path Algorithm	47
3.2.5.	Association of Related Edges	48
Chapter 4.	POLYGONAL APPROXIMATION	51
4.1.	The State of the Art in Polygonal Approximation	51
4.1.1.	Approximation Using Wedges	51
4.1.2.	Approximation Using Chain Codes	52
4.2.	Improving the Sklansky and Gonzalez Algorithm	53
4.2.1.	Using Cross Products to Find the Wedge Intersections	53
4.2.2.	A Fast Approximation for the Tangents	55
4.3.	A Simple Polygonal Approximation Algorithm	57
4.4.	A Combination of Polygonal Approximations	58
Chapter 5.	FEATURE EXTRACTION	60
5.1.	The Work of Shillman et. al.	60
5.1.1.	Ambiguity is the Key to Recognition	60
5.1.2.	Shillman's Functional Attributes	62
5.1.3.	The Physical to Functional Rules	65
5.2.	A Modified Set of Functional Attributes	66
5.2.1	Shillman's Attributes Applied to Outlines	66
5.2.2.	A Reduced Set of Attributes	68
5.3.	Extraction of the Features From Outlines	68
5.3.1.	Concavities	69
5.3.2.	Closures	72
5.3.3.	Axes	73
5.3.4.	Lines	74
5.3.5.	Symmetry	75
5.3.6.	Calculation of Area and Centroids	76
5.3.7.	Ordering of Features	77
5.4.	Generation of the Standard Templates	78
5.4.1.	Rationale	79
5.4.2.	Clustering	80
5.4.3.	Partitioning Classes	81
5.4.4.	Merging Ambiguous Classes	81
5.4.5.	Assigning Weights to Features	82

5.5.	Recognition of Features	83
5.5.1.	Flexible Comparison	83
5.5.1.1.	Matching Individual Features	84
5.5.1.2.	Matching Lists of Similar Features	85
5.5.2.	Reducing the Search Size	86
5.6.	Summary	87
Chapter 6.	A MORE GENERAL EDGE EXTRACTOR	89
6.1.	The Deficiencies of the Simple Edge Extractor	89
6.1.1.	Nesting Trees	89
6.1.2.	Memory Limitations	91
6.2.	Creating the Nesting Tree	92
6.2.1	Marking Leading/Trailing Edge Points	93
6.2.2.	Orientation of Edges	95
6.3.	The Choice of Edge Path	97
6.3.1.	Avoiding Collisions	97
6.3.2.	Locating Fork Points	98
6.3.2.1.	Fork Analysis: Intersection of Edge Sets	98
6.3.2.2.	Fork Analysis: Backtracking	99
6.3.2.3.	Fork Analysis: Reverse Edge Operator	99
6.3.3.	Re-tracing Failed Edges	100
6.4.	Some Problems Caused by Halftoned Images	101
6.4.1.	Faint Edges	101
6.4.2.	Self-Intersecting Edge Paths	103
6.4.3.	A Full Stack-Based Solution	103
6.5.	Summary	104
Chapter 7.	THINNING	106
7.1.	Previous Thinning Algorithms	106
7.1.1.	The Iterative Thinning Algorithms	106
7.1.2.	Faster Thinning Algorithms	109
7.2.	A New Thinning Algorithm	111
7.3.	Nearest Points	113
7.4.	Endpoints	114
7.4.1.	Testing For Endpoints	114
7.4.2.	The Radius of Curvature Test	116
7.4.3.	Coalescing Endpoints into Line Endings	118
7.5.	Constructing the Skeleton	120
7.5.1.	Tracking Lines From Endpoints	120



7.5.2.	Other Continuous Regions	120
7.6.	Line Junctions	121
7.6.1.	Creation of Junctions	122
7.6.2.	Closing Junctions	123
7.6.3.	Adjacent Junctions	124
7.7	Connecting Junctions	125
7.7.1.	Partitioning Junctions	125
7.7.2.	Connecting 4-Line Junctions	127
7.7.3.	Extrapolation Vectors	127
7.7.4.	Connecting Junctions with Extrapolation Vectors	129
7.8.	A Note on Optimization	129
7.8.1.	Optimizing the Nearest Point Tests	130
7.8.2.	Intersecting Nearest Lines	131
7.9.	Summary	131
Chapter 8.	RESULTS AND CONCLUSIONS	132
8.1.	Edge Extraction and Polygonal Approximation	132
8.2.	Feature Extraction	134
8.3.	Conclusion: The Current Problems Resolved	139
8.3.1.	Recognition Speed	139
8.3.2.	Accuracy	139
8.3.3.	Separation of Text and Pictures	140
8.4.	Further Work	141
8.4.1.	Halftone Processing	141
8.4.2.	Edge Extraction	141
8.4.3.	Feature Extraction	141
8.4.4.	Text Ordering	141
8.4.5.	Final ASCII Classification	142
8.4.6.	Text Removal	143
8.4.7.	Font Storage/Matching	143
8.4.8.	Construction of Comprehensive Test Data	144
8.4.9.	Image Compression	144
8.5.	Longer Term Research	144
8.5.1.	Parallel Processors for Higher Speed	144
8.5.2.	Capture of Engineering Drawings	145
8.6.	Summary	146
References		148
Referenced Machines		156

## LIST OF TABLES

6.1	The nesting relation between the edges in Fig. 6.1(b).	90
7.1	The conditions for deleting a black pixel.	108
8.1	The present ambiguities, their grade and solution.	137

## LIST OF FIGURES

1.1	(a)	The original template for J.	18
	(b)	The reduced template.	
1.2		The effect of (a) horizontal and (b) vertical position of the recognition rate.	19
1.3		An example of a letter "R" segmented into straight and curved sections.	20
1.4	(a)	A typical binary image of a character.	21
	(b)	The outline of (a).	
	(c)	A thinned version of (a).	
1.5	(a)	The outermost points on the outline of Fig. 1.4(b).	22
	(b)	The segmentation of the outline into convex and concave regions.	
1.6		Strokes from samples of the letter "P" illustrating clustering in the feature space.	25
1.7		A difficult problem for any OCR machine.	30
1.8		The image of Fig. 1.7 thresholded at 5.5.	31
1.9		The image of Fig. 1.7 thresholded at 10.5	31
1.10		The result of applying an edge detector to Fig. 1.7.	32
2.1		A system block diagram.	33
2.2		A grey level image of an "a" and its hexadecimal values.	34
2.3		The word "Both" printed on a halftoned background.	35
2.4	(a)	The edge path of the image in Fig. 2.2.	36
	(b)	A polygonal approximation of the outline in (a).	
2.5		The outline of Fig. 2.4(b) with a closure and 3 concavities identified.	38
3.1		The Roberts edge operator in window notation.	41
3.2		A pair of edge operators defined by Prewitt.	42
3.3		The 4-neighbour operators of Chaudhuri and Chanda.	42
3.4	(a)	The 3 by 3 neighbourhood of X.	43
	(b)	The edge operator for chain code 0.	
	(c)	The edge operator for chain code 7.	
3.5	(a)	A part of the "a" in Fig. 2.2.	44
	(b)	Edge values at the circled point.	
	(c)	Edge values at the rounded square.	

	(d)	Edge values at the diamond.	44
3.6	(a)	Part of the "a" in Fig. 2.2.	46
	(b)	Edge values at the circled point.	
	(c,d)	Edge values from the alternatives in (b).	
3.7	(a)	A correctly extracted "ti" pair.	47
	(b)	Merging by incorrect path choice.	
	(c)	Both characters lost by the loop closing incorrectly.	
	(d)	The "i" collides with the correct "t".	
3.8		An extra edge operator for 90 <sup>0</sup> bends.	48
	(a)	Diagonal bends.	
	(b)	Upright bends.	
3.9	(a)	An example of the failure of bucket sorting.	50
	(b)	A simple rotation transformation solves the problem.	
4.1		Sklansky and Gonzalez polygonal approximation.	52
	(a)	Processing point S.	
	(b)	Processing point T.	
4.2	(a)	A current wedge with a new wedge under test.	54
	(b)	The allowed ranges of Q <sub>a</sub> and Q <sub>b</sub> .	
4.3		An example of approximating the tangents.	55
4.4		Calculating the approximations to the tangents.	56
4.5		Some examples of digitized straight lines, showing repeated patterns of chain codes.	57
4.6		Examples of the simple polygonal approximation algorithm.	58
	(a)	A and B differ by more than 1.	
	(b)	B is repeated.	
	(c)	The run-length of A changes.	
	(d)	A third chain code is encountered.	
4.7		An example of two stage approximation.	59
	(a)	An original noisy outline.	
	(b)	The result of the simple approximation of section 4.3.	
	(c)	Reapproximation by the algorithm of section 4.2.	
4.8	(a)	The polygon of Fig. 4.7(b) with fixed points marked "+".	59
	(b)	The final result of selective reapproximation.	
5.1		The levels of the attribute leg.	61
5.2	(a)	An ambiguous shape.	62

	(b)	The shape of the "P" makes a "Y" more likely.	62
	(c)	The shape of the "P" makes a "V" more likely.	
	(d)	Lexical context has a strong effect on the ambiguity threshold.	
5.3		Line-like attributes.	63
	(a)	Shaft.	
	(b)	Leg.	
	(c)	Arm.	
5.4		Concave attributes.	63
	(a)	Bay.	
	(b)	Inlet.	
	(c)	Notch.	
	(d)	Hook.	
5.5		Miscellaneous features.	64
	(a)	Closure.	
	(b)	Symmetry.	
5.6		Junction type features.	64
	(a)	Weld.	
	(b)	Crossing.	
	(c)	Marker.	
5.7		Physical to functional rules.	65
	(a)	An example of a leg.	
	(b)	An example of a closure open at the top.	
	(c)	An example of a closure open at the right.	
5.8		Straight segments on the outlines of the characters in Fig. 5.3.	66
5.9		Concave regions on the outlines of the characters in Fig. 5.4.	67
5.10		Replacing junction attributes with bays.	67
	(a)	Weld.	
	(b)	Crossing.	
	(c)	Marker.	
5.11	(a)	An example of a concavity.	70
	(b)	The position of the ends after the concavity is expanded.	
5.12	(a)	A concavity with a large angle at $P_j$ .	70
	(b)	Contracting the concavity to reduce the angle.	
5.13		An example of the final concavity end adjustment.	71

5.14	(a)	Calculating the minor axis.	73
	(b)	Adjusting the major axis.	
5.15	(a)	Unacceptable lines.	74
	(b)	Accepted lines.	
5.16	(a)	An italic "B".	75
	(b)	The inverse sheared "B" with its reflection.	
5.17		Calculation of the centroid of a closed region.	76
5.18	(a)	A "+" with the origin at the top.	78
	(b)	A "+" with the origin at the left.	
5.19		Some of the possible variations in an "E".	79
	(a)	Shape.	
	(b)	Line width.	
	(c)	Angle.	
	(d)	Embellishments.	
5.20	(a)	The concavities of an "H".	81
	(b)	The final template for the class "H".	
5.21		The result of the flexible matching process.	84
5.22	(a)	Having matched the first pair of features, recursion evaluates the remainder.	85
	(b,c)	When testing against subsequent standard features, skipped features are regarded as missing.	
	(d)	The first test feature is skipped and assumed extra.	
6.1	(a)	A simple page.	90
	(b)	The outlines of (a).	
6.2		The nesting tree of the page in Fig. 6.1(a).	90
6.3		The outline of an "a" with chequered leading edge points and solid trailing edge points. The edge path is also shown.	92
6.4	(a)	An example of the rule for $C_1=5$ .	93
	(b)	All possible 2-step paths which include $C_1=5$ in the first step.	
6.5	(a,b)	Examples of an edge path containing a step with chain code 7.	94
	(c)	Derivation of the rule for chain code 7.	
6.6	(a)	Edges found in one buffer.	95
	(b)	The nesting tree for (a).	
6.7	(a)	The amount of object visible in the second buffer.	95
	(b)	The corresponding nesting tree.	

6.8	(a)	The completed object.	96
	(b)	The final nesting tree.	
6.9		A tree of possible edge paths.	97
6.10		Backtracking to resolve forks and the result with two different positions of the real edges.	99
6.11		The reverse edge operator.	100
	(a)	For chain code 0.	
	(b)	For chain code 7.	
6.12	(a)	A correctly traced dot.	101
	(b)	A dot started at the bottom.	
6.13	(a)	The top point must be convex.	102
	(b)	An anticlockwise convex point.	
6.14		A self-intersecting object.	103
7.1		The 8-neighbours of P.	107
7.2	(a)	A binary image of a "Q".	108
	(b)	The thinned version of (a).	
7.3		An example of a nucleic acid molecule and its skeleton.	109
7.4		The equal diagonal algorithm of Shapiro, Pisa and Sklansky.	110
7.5		An example of the failure to detect a junction.	110
7.6	(a)	A simple page.	112
	(b)	The nesting tree of (a).	
7.7		An alternative page to produce the nesting tree of Fig. 7.6(b).	112
7.8	(a)	An example of nearest point test (a).	113
	(b)	An example of nearest point test (b).	
7.9		Some actual outlines with nearest points marked by arrows.	114
7.10	(a)	An example of sharply convex points which are not endpoints.	115
	(b)	Sharply convex points which are endpoints under test (b).	
7.11		Outlines with single endpoints marked by asterisks and endpoint pairs joined by thick lines.	115
7.12	(a)	Radius of a circular arc.	116
	(b)	A discrete arc.	
7.13	(a)	A scaled version of Fig. 7.12(b)	117

(b)	The angular approximation.	117
7.14	Constructing the skeletal line at a line ending.	118
7.15	The result after completing line endings.	119
7.16	(a,b) Examples of continuity.	120
(c)	An example of discontinuity.	
7.17	The skeleton present after processing regions of continuity.	121
7.18	The components of a junction pair.	122
7.19	(a) The first line ends at a junction.	123
(b)	The second line ends at the junction.	
(c)	The junction is complete.	
7.20	A non-closed junction.	123
7.21	Two adjacent junctions.	124
7.22	Closed junctions are shown dashed.	124
7.23	A 5-line junction with a partition.	125
7.24	(a) A 4-line junction with the possible partitions.	126
(b)	The resulting pair of 3-line junctions	
7.25	Partitioning the 4-point junction in Fig. 7.22.	126
7.26	A 4-line junction with perpendicular diagonals.	127
7.27	The extrapolation vectors of a junction.	128
7.28	A case of an extrapolation vector which crosses the outline.	129
7.29	The final result of thinning.	129
8.1	A document containing text and pictures.	132
8.2	The output of the edge extractor when applied to Fig. 8.1.	133
8.3	The outlines produced from the page in Fig. 1.7.	134
8.4	The training set.	134
8.5	The recognized characters from Fig. 8.2.	135
8.6	The rejected characters from Fig. 8.5.	136
8.7	The characters recognized from the outlines in Fig. 8.3.	138
8.8	The rejected characters from Fig. 8.7.	138
8.9	The result of thinning the outlines in Fig. 8.3.	145



## 1. INTRODUCTION

The use of a machine to read text from a printed page has been a subject of research for many years. *Optical Character Recognition (OCR)* is the term most frequently used to describe this process. This introduction will first discuss some of the important applications of OCR and will then examine two of the main methods as described by early papers on the subject. Subsequently, there will be a description of the state of the art in terms of recently published research and also some currently available machines. The remainder of the introduction will focus on the current problems in OCR.

### 1.1. APPLICATIONS

The applications of OCR are presently limited by the state of the art. When OCR machines are capable of reading almost any print style, the range of applications will increase significantly.

The most common application is in office automation. Many large commercial organisations now have extensive electronic document processing and mail facilities, yet the majority of incoming mail is still in paper form. OCR could be used at several different levels to assist with the processing of these documents.

Desktop publishing is a relatively new application. With recent reductions in the cost of high quality laser printers, complete publishing systems are now widespread. These systems provide the facility to lay out text and pictures on a page and re-set the text in a selection of possibly 1000 fonts. Text from common word processors can be used in the document, but text that exists only in printed form has to be re-typed. OCR could be used to overcome this problem.

If all the printed material in a library could be scanned and the images stored on-line, it would be possible to read or borrow items using only a remote terminal. Presently, this idea is totally unrealistic, since the storage capacity required for even well compressed images is so enormous that holding all the material in just a small library would be astronomically expensive. Even the physical space required is likely to be larger than that required by the original paper.

The compression ratios possible by using OCR reduce the necessary storage capacity sufficiently to make on-line libraries more likely. The other advantage is

that once the material at the library is in machine-usable form and on-line, the terminal used to access it need not be a display screen. It may be a speech synthesis unit or a Braille terminal enabling blind people to borrow and read books without even having to leave their home.

Stand-alone reading machines for the blind have already been built and sold. Existing reading machines however, have the same font limitations as other OCR machines.

## 1.2. THE TWO MAIN OCR METHODS

There were several attempts at OCR by mechanical means, before the invention of the digital computer. The approaches to character recognition are generally split into two very different methods. These are *matrix matching*, in which the bit mapped image of a character is matched directly with a set of templates; and *feature extraction*, where some higher level structure of the character is found and compared with a standard.

Feature extraction techniques can be sub-classified in many ways since the term covers several radically different approaches. This section will concentrate on illustrating the two main approaches to OCR as described by early papers published around 1959.

### 1.2.1. MATRIX MATCHING

Wada et. al.<sup>(1)</sup> describe an electronic reading machine which is based on a diode matrix. This machine and others like it form the basic principle used in the majority of current commercial OCR systems. (See section 1.4.1.) The principle is to fix a character cell to contain an array of (in this case) 10 by 12 square pixels. A sample of each character is presented to the machine and each pixel is digitized using simple thresholding to black or white. The pixels positioned on a boundary between white and black regions are set to grey. Grey pixels are used to provide flexibility by regarding them as matching both black and white pixels. These samples form templates against which all unknown characters are tested.

When the machine is presented with an unknown character, each pixel is thresholded to black or white. Corresponding pixels are then compared in every template. The number of pixels matching black for black or white for white are counted. Pixels that are grey in the template are ignored. The unknown character is then recognized as being the one with the highest match value.

Certain pixel positions in the templates help to distinguish between characters better than others. For instance, some pixel positions may be white in all templates. In order to reduce the complexity of the machine, a large proportion of pixels in the templates are changed to grey. Once all the templates have been created, each is compared against all the others. Those pixels which have the greatest discriminating power are called "Important" pixels. As many as possible of the least important pixels, i.e. those with least discriminating power, are set to grey in all templates subject to the constraint that all the templates remain different.

In this way, only those points that are capable of distinguishing between characters are retained. This reduction results in an average of only 11.3 pixels being tested for each character. An example of this process is shown in Fig. 1.1.

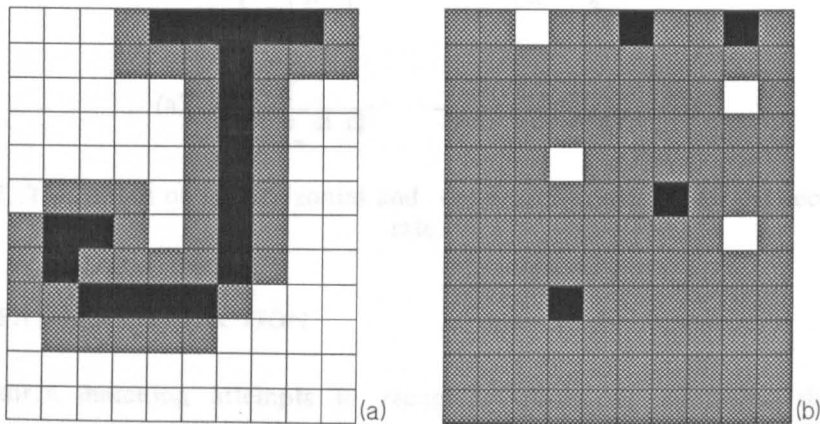


Fig. 1.1. (a) The original template for J. (b) The reduced template.

Using state of the art hardware technology in 1959, this machine achieved a recognition time of about 15 milliseconds per character. This is well beyond the capabilities of the programmed computers of the time and shows the outstanding advantage of matrix matching - it is extremely simple to implement in hardware or software and fast at recognition.

The outstanding disadvantage of matrix matching is evident in Fig. 1.2. This figure is reproduced from the paper by Wada et. al.<sup>(1)</sup> and shows the effect of horizontal and vertical deviation of the position of characters within the cell on the recognition rate. Clearly different fonts have different aspect ratios, thickness of lines, sizes of serifs etc. All these characteristics contribute to a movement of different parts of each character relative to one another and thus to a severe reduction in the recognition rate.

A slight change in the size of the character would cause a similar effect. There are many different type sizes in common use and matrix matching is incapable of managing a drastic change in pattern size without multiplying the number of templates by the number of sizes to be accepted.

Furthermore, the majority of *printed* text, as opposed to *typewritten* text, is proportionally spaced. Not only does this make it difficult to locate the character cell on the page, but also, the character cell is no longer of a fixed size. All the above problems together constrain the application of matrix matching to recognition of a small range of fonts in a fixed size.

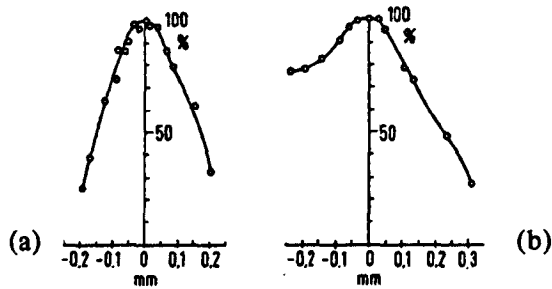


Fig. 1.2. The effect of (a) horizontal and (b) vertical position on the recognition rate.

### 1.2.2. FEATURE EXTRACTION

Matrix matching attempts to recognize characters by using the lowest possible level of information. *Feature extraction* is a general term which covers many different techniques for extracting higher level information about a character, in an effort to achieve font independence.

Grimsdale et. al.<sup>(2)</sup> describe an OCR system which analyses the strokes comprising each character. A line finding algorithm segments the character into straight and curved line sections. An example of the segmentation of a character is shown in Fig. 1.3. The length, slope and curvature of each segment are combined with information on the connectivity with other segments into a 40 bit representation. The set of these 40 bit representations which make each character is called the "statement" for the character.

The statement of the unknown character is compared with the statements for the standard characters on which the system was trained. The character is recognized as being the one corresponding to the statement which best matches the statement of the unknown.

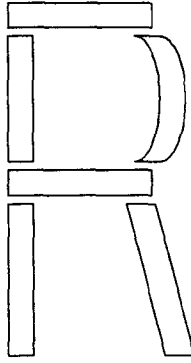


Fig. 1.3. An example of a letter "R" segmented into straight and curved sections.

The interesting attribute of this particular approach is that characters may be recognized in any orientation. As long as the slope for all sections deviate from the standard by the same amount, the character can still be recognized.

Using a Manchester Mark I computer, with 4000 machine instructions, this system recognized characters at an average rate of one per 60 seconds. Understandably, there are no detailed test results in the paper. The system was trained and tested with carefully hand painted figures in white drawn on a black background. Consequently the figures used had no adornments or serifs and it is difficult to tell what effect they would have had on the recognition rate.

The principal advantage of feature extraction is that if the features are chosen well enough, recognition is independent of position, size, orientation and font. In this particular example the independence of size and orientation are there, although independence of font is unlikely.

The main disadvantage is also dramatically shown when the recognition time is compared with that of Wada et. al.<sup>(1)</sup> from section 1.2.1. The authors anticipate at least a thousandfold reduction in the recognition time with advances in computer technology. Nevertheless, the steps involved in recognition by feature extraction are so complex that the process has to be software based, whereas matrix matching can be performed in hardware. Even when performed by software, matrix matching is still an order of magnitude faster than feature extraction.

### 1.3. RECENTLY PUBLISHED OCR RESEARCH

There have been several surveys of character recognition.<sup>(3-6)</sup> A detailed survey is not attempted here; this section will instead concentrate on describing two recent but highly contrasting approaches. They differ not only in the features that are used in recognition, but also in the way in which the features are obtained from the bit-mapped image.

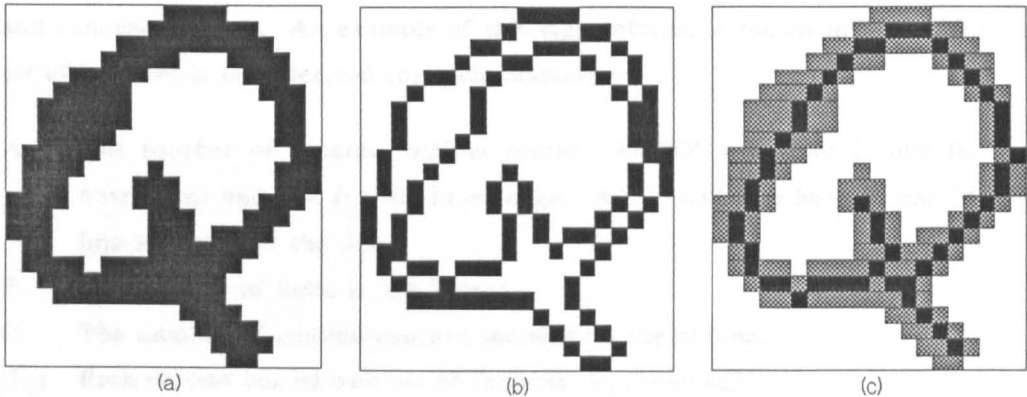


Fig. 1.4. (a) A typical binary image of a character.  
(b) The outline of (a). (c) A thinned version of (a).

Fig. 1.4(a) shows a typical example of a binary image of a character. All previous work on OCR, except the work by Brady et. al.<sup>(25-27)</sup>, has operated on a binary image, i.e. each pixel is taken to be either white or black. From the binary image, the features have to be extracted in some way. Two possible approaches are as follows:

- (1) Find the outline of the character as in Fig. 1.4(b). The result is still a bit-map, but the outline may be approximated by a polygon to make it easier to manipulate.
- (2) Thin the lines to produce a skeleton as in Fig. 1.4(c). As in (1), the result is still a bit-map, but the features may be easier to find. The skeleton may be converted to vectors and stored in a graph representation.

The above alternatives will now be considered in more detail.

#### 1.3.1. EXTRACTION OF FEATURES FROM OUTLINES

Yamamoto and Mori<sup>(7)</sup> describe a method by which outermost points on the outline of a character are found and used to extract convex and concave sections of the outline. An outermost point is defined as follows:

Let the outline consist of  $n$  points with coordinate vectors  $E_i$  where  $i$  ranges from 1 to  $n$ . The vectors may have an arbitrary origin, for instance the bottom left corner of the character cell. The outermost point in the direction of vector  $A$  is then  $E_i$  such that  $A \cdot E_i$  is a maximum over  $i$  from 1 to  $n$ .

The outermost point on the object is found in each of a fixed set of 16 directions. The outermost points on the outer edge of the "Q" in Fig. 1.4(b) are shown in Fig. 1.5(a). These points are used to segment the outline into convex and concave sections. An example of this segmentation is shown in Fig. 1.5(b). A set of features is then derived for each character:

- A The number of separate outline pieces. An "O" will have 2, one for the outer edge and one for the inner edge. An "i" will also have 2, one for the line and one for the dot.
- B The number of holes in the object.
- C The number of convex/concave sections on the outline.
- { $F_i$ } Each section has its own set of features,  $F_i$  ( $1 \leq i \leq C$ ):
  - (1) Whether the section is convex, concave, or a hole.
  - (2) The "centre point" of the line section. There is no definition of the centre point in the paper.
  - (3) The length of the section along the original outline.
  - (4) The straight line distance between the ends of the section.
  - (5) The perpendicular distance of the centre point from the straight line joining the ends of the section.
  - (6) The orientation of the straight line joining the ends of the section.
  - (7) The ratio of (4) to (3).
  - (8) The difference between (4) and (3).
  - (9) The angular distribution of edge elements in the section.

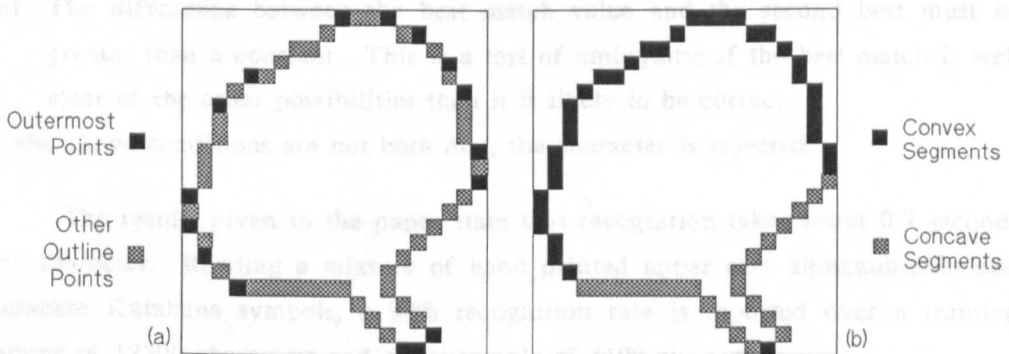


Fig. 1.5. (a) The outermost points on the outline of Fig. 1.4(b).  
 (b) The segmentation of the outline into convex and concave regions.

Let the  $j^{\text{th}}$  feature in the set  $F_i$  be denoted by  $F_{i,j}$  ( $1 \leq j \leq 9$ ). Thus each character will have the values A, B, C, and a set  $F_i$  containing the values  $F_{i,1}$  to  $F_{i,9}$  for each different section of the outline. Rather than compare these values against the entire known character set, the set is split into groups according to the values A, B, C and the values  $F_{i,6}$ , the direction of each section. Within each group, the features are then matched against every character in the group.

By splitting the templates into groups in this way, recognition time is reduced. Also, the matching process is simplified by having C the same for all the templates in each group. There can then be a direct comparison of the features  $F_{i,j}$  with those of the corresponding  $F_{i,j}$  in the template.

In the templates, each feature has associated with it an allowed range, given by an upper limit  $U_{i,j}$  and a lower limit  $L_{i,j}$ . The range is expanded by an allowed deviation  $dU_{i,j}$  and  $dL_{i,j}$ . The match value for the feature  $F_{i,j}$  in the test character is given by:

$$M_{i,j} = \begin{cases} W_{i,j}(L_{i,j} - F_{i,j}) & \text{If } L_{i,j} - dL_{i,j} \leq F_{i,j} < L_{i,j} \\ 0 & \text{If } L_{i,j} \leq F_{i,j} \leq U_{i,j} \\ W_{i,j}(F_{i,j} - U_{i,j}) & \text{If } U_{i,j} < F_{i,j} \leq U_{i,j} + dU_{i,j} \\ \text{Infinity} & \text{If } F_{i,j} < L_{i,j} - dL_{i,j} \text{ or } F_{i,j} > U_{i,j} + dU_{i,j} \end{cases}$$

$W_{i,j}$  is a weight for that particular feature. The match rating of the test character against the template is thus the summation of  $M_{i,j}$  over the 9 features in each of the C line sections. The best match is given by the lowest match value.

The character is accepted and recognized if the match rating meets the following conditions:

- (a) The best match value is below a fixed limit. This ensures that the character is reasonably close to the ideal shape.
- (b) The difference between the best match value and the second best must be greater than a constant. This is a test of ambiguity; if the best match is well clear of the other possibilities then it is likely to be correct.

If the above conditions are not both met, the character is rejected.

The results given in the paper state that recognition takes about 0.7 seconds per character. Reading a mixture of hand printed upper case alphanumeric and Japanese Katakana symbols, a 99% recognition rate is reported over a training sample of 12300 characters and a test sample of 4100 characters.



In a recent US patent application, Yamamoto and Saito<sup>(8)</sup> present a hardware implementation of a modified version of the outermost point algorithm. The modification causes generation of a polygonal approximation to the outline, instead of the rather inaccurate convex hull produced before. Unfortunately, the patent application describes only the polygonal approximation/outermost point algorithm and contains no information on the recognition method.

### 1.3.2. EXTRACTION OF FEATURES FROM SKELETONS

Kahan, Pavlidis and Baird<sup>(9)</sup> describe an OCR system designed to recognize printed text of any font and size. Processing starts with the run-length encoded binary image of the entire document. (Run-length coding simply replaces each run of black or white pixels with a number indicating the length of the run.) Firstly, the Line Adjacency Graph<sup>(10)</sup> (LAG) is constructed. The LAG is a representation of dark run lengths in the image and indicates their connectivity with other dark run lengths on adjacent lines. Each connected component of the LAG is thus a single character, except in the case of a deliberately broken character such as "!".

Thin, straight line segments are extracted directly from the LAG<sup>(11)</sup>. Thinning by this route has a considerable speed advantage over the conventional pixel-based iterative thinning methods. The features used in the recognition process are generated from the output of the LAG based thinning step. They are: strokes, holes, concavities, crossings of strokes and endpoints in the vertical direction.

Kahan Pavlidis and Baird give an example of how the strokes are used in recognition. The size, position and orientation of the strokes are normalized with respect to the rectangular bounding box of the character. These variables are then transformed into a 4-dimensional feature space,  $\langle x, y, r, i \rangle$ .  $\langle x, y \rangle$  gives the normalized position of the centre of the stroke.  $\langle r, i \rangle$  is a representation of the orientation and length of the stroke.  $\langle r, i \rangle$  is a complex number such that the modulus is equal to the normalized length of the stroke and the argument twice the angle with the x-axis.

Analysis of a large number of samples of the same character reveals that the transformed strokes form clusters in the feature space. An example of strokes and the corresponding clusters from the letter "P" is shown in Fig. 1.6.

Counting the individual stroke clusters for all the character set gives about 500, but many of these are very similar. By merging clusters the number can be reduced to 100 with a negligible effect on ambiguity. The number of clusters for all the features can be likewise reduced from 1500 to 300. The 300 different clusters are used as the binary features in a Bayesian classifier<sup>(12)</sup>.

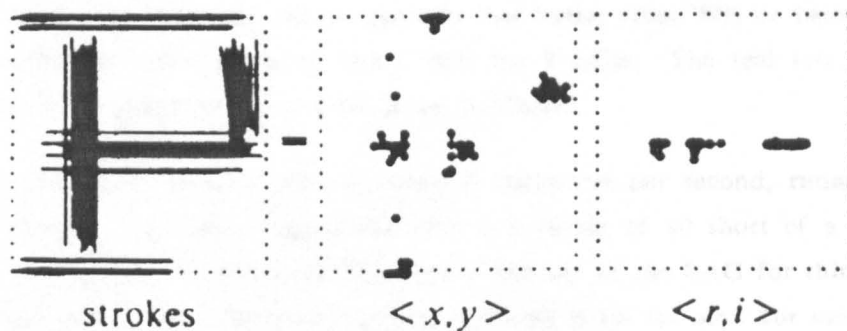


Fig. 1.6. Strokes from samples of the letter "P" illustrating clustering in the feature space.

The features listed above are inadequate to distinguish all characters, so contour analysis is used to resolve the ambiguities. The groups of ambiguous characters are quite revealing:

- (1) a, e, s, g, 8, B.
- (2) O, o, 0, D, Q.
- (3) 6, b.
- (4) b, h.
- (5) f, t.

In group (1), a, e and s are only confused in the case of a small boldface type where the concave region is fully closed. Such closure is due to a combination of the scanner resolution and the light threshold used to prevent broken characters.

One would expect the left vertical stroke of the B to distinguish it from the 8 but this is not so since the LAG traversal manages to find a similar line in the 8. A similar comment applies to O/D in group (2) and 6/b in group (3).

In group (4), b and h are confused when the h has large serifs. Confusion of f/t in group (5) is understandable due to the small physical difference.

After contour analysis for ambiguous characters, words are tested with a spelling checker. Where the word is rejected, alternatives are tried according to

decreasing probability, until the word is accepted, or the probability falls below a set level.

Test results are given for a large data set of 196000 characters. The published results only cover tests conducted using *the same fonts as the training set*. Thus font independence is not shown convincingly. One test covers training on six dissimilar fonts and the recognition rate varies from 97% or better for 14 point or bigger print, down to below 90% for 8 point. The real test of using different fonts from the training set is not published.

Recognition speed is approximately 5 characters per second, running on a VAX 11/750. The authors agree that this is a factor of 10 short of a practical figure. An important feature of this work is the use of the LAG for thinning. It is now accepted that pixel based iterative thinning is far too slow for use in OCR on a single conventional processor system<sup>(13)</sup>. This result is confirmed by Kahan et. al. who state that their LAG based thinning takes under 3 minutes per page of text versus over 20 minutes for pixel based thinning. For a detailed description of one of the iterative thinning algorithms see chapter 7.

#### 1.4. CURRENTLY AVAILABLE MACHINES

The applications of section 1.1 are not at present feasible, because of the profuse variety of type fonts used in modern printing. Most publishers now offer several hundred styles in a wide range of sizes. The majority of currently available OCR machines use matrix matching and are consequently incapable of reading this abundance of fonts. Only a small fraction of current machines use feature extraction and these tend to be much more expensive than the matrix matching machines.

##### 1.4.1. LOW-END MATRIX MATCHING MACHINES

Most present day OCR machines can read only a small fixed set of fonts. Consider for example, the DEST Workless Station<sup>(M1)</sup>. This machine can cope with 8 popular monospace timesteps plus 4 proportionally spaced timesteps. Obviously, this represents only a very small fraction of all printed matter.

The Workless Station can process an A4 page in as little as 30 seconds with a substitution error rate of 1 in 300000. A substitution error occurs when the machine claims to recognize a character correctly, but is in fact incorrect. The rejection ratio is rather higher than the substitution error rate; it is of the order of 1 in 500. Rejected characters are those which did not match any single character

particularly well. Rather than make a substitution error, they are rejected and highlighted for the operator to correct.

A problem with this type of OCR device is that it cannot cope with any document containing non-text. Even a simple coloured box can confuse it completely. The Workless Station is thus extremely limited in its application, but within those limits it has excellent performance. A survey of some similar machines is given in Ref.(14).

#### **1.4.2. FEATURE EXTRACTION SYSTEMS**

The Kurzweil reading machine<sup>(M2)</sup> is a good example of a machine designed to recognize a reasonable range of fonts. The cost in manual intervention however, is extremely large. For each new font encountered it is necessary to train the machine, either confirming its choice of character as correct, or supplying the appropriate correction. This process is many times slower than retyping the section on which training takes place.

The training process is also unreliable, since some area of the document must be used as a sample and very few documents print the entire character set in a separate box especially for training! Thus those characters not present in the sample may be recognized incorrectly when they are encountered later in the document.

Once the Kurzweil reader is trained for a font, the problems are not solved because the machine must be told which training set to use to read the various portions of a document. A digitizing pad is used to indicate regions containing each different font. Hence it is necessary for the operator to be able to recognize fonts by sight and know the name of each.

The difficulty with processing documents containing non-text also occurs with the Kurzweil reader. When using the digitizing pad to indicate regions of each font, areas of non-text must be avoided. In common with the matrix matching machines, the Kurzweil reader is limited to reading black text on light coloured paper.

## 1.5. UNSOLVED PROBLEMS IN OCR

The OCR market is presently poised for a dramatic change. Currently, almost every commercial machine is based on matrix matching, yet for nearly thirty years, the emphasis in the literature has been on feature extraction. Kahan, et. al.<sup>(9)</sup> state in their introduction that to the best of their knowledge, there is no commercial implementation of feature based OCR that is capable of reading a wide range of print styles. The only possible exceptions to this statement according to Kahan et. al. are the PRODATA TO-3000<sup>(M3)</sup> which uses the Yamamoto-Mori<sup>(7)</sup> method, and the Palantir Compound Document Processor which became available in the UK in 1987 through FormScan<sup>(M4)</sup>.

The Compound Document Processor in fact employs an advanced matrix matching algorithm<sup>(15)</sup>, which scales each character before matching, enabling the recognition of a range of sizes with only one set of templates for each font. The marketing literature from Kurzweil<sup>(M2)</sup> seems to indicate a feature extraction approach, but the machine still needs to be trained for each new font encountered.

The probable reasons for this discrepancy between commercial machines and the published algorithms are:

- (1) Published feature based OCR methods have insufficient accuracy.
- (2) Feature extraction cannot yet compete with the speed of matrix matching.

The market demands an accuracy of at least 99.9% correct with all errors being rejections rather than substitution errors. (See section 1.4.1.) An acceptable recognition speed is 50-100 characters per second for a machine costing under (US)\$5000.

Some of the currently unsolved problems in OCR by feature extraction will now be explained. The remainder of this thesis will concentrate on a description of some new solutions to these problems.

### 1.5.1. RECOGNITION SPEED

Working backwards from an acceptable speed of 100 characters per second gives 10 milliseconds per character. Divide this by the number of pixels in a typical character cell (1200 for a 30 by 40 pixel cell) and there are only 8 microseconds per pixel. Clearly, to process an image at that rate in software needs some extremely fast algorithms for extracting the features.

The most time consuming step is usually the pre-processing needed to assist the extraction of the features. For example, in thinning based feature extraction, thinning itself is the rate determining step. Times of 250 milliseconds per character are typical. Even the LAG based thinning algorithm<sup>(11)</sup> is slow at around 40 milliseconds. Either a new faster thinning algorithm must be found, or the step must be eliminated completely.

### 1.5.2. RECOGNITION ACCURACY

The principle of feature extraction is that it is fundamentally more general than matrix matching and hence better able to read many different fonts. In reality, even the most recent methods<sup>(9)</sup> have a recognition rate which tails off as the number of fonts increases. The problem lies with the features themselves and the way they are chosen. In the majority of papers on OCR by feature extraction, there is no reason given at all for the choice of features.

This lack of methodology is reflected in the number of papers that have been published on feature selection<sup>(16,17)</sup> and the elimination of redundant features<sup>(18)</sup>. Psychological tests have been performed<sup>(34-41)</sup> to determine those features responsible for distinguishing a range of ambiguities and perhaps this is the way in which better font independence and accuracy can be achieved.

### 1.5.3. BINARY AND GREY LEVEL IMAGES

Fig. 1.7 shows a portion of a typical document which would be impossible for any existing OCR machine to read. The Workless Station would struggle with even the black text, since it is proportionally spaced and unlikely to be in one of its standard fonts. The white text would be lost completely and the photograph may cause rejection of the entire page.

The Kurzweil reader would be able to read the black text, after training, but is incapable of reading the white text. Neither machine can automatically distinguish between the text and the picture.

**For documents conventional OCR can't handle, Kurzweil offers a better approach: ICR.**



For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerised systems. Yet in all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

Fig. 1.7. A difficult document for any OCR machine.

The common reason for all these failings is that both machines will threshold the image before they look for text. Fig. 1.7 was scanned with 16 shades of grey (level 0 is black, level 15 is white) and printed using *error diffusion*.<sup>(19)</sup> Error diffusion preserves some of the grey level information on a printer capable of printing only dots of uniform size and colour. The use of grey levels provides better quality pictures. More importantly, grey levels provide a clear distinction between text and non-text.

As an example of the power of grey levels over binary images, Fig. 1.8 shows the same image of Fig. 1.7 after applying an adaptive thresholding algorithm. Adaptive thresholding adds a fraction of the local average to the threshold. The main effect of the adaptive algorithm is to enhance edges in the images by pushing the threshold nearer to white in a light region and nearer to black in a dark region. Hence grey pixels in a white region are more likely to be mapped to black and show up than they are to be mapped to white and disappear. Fig 1.8 shows the result of thresholding at 5.5, plus a quarter of the local 3 by 3 average. Fig. 1.9 shows the result of thresholding at 10.5, again plus by a quarter of the local average.

Fig. 1.9 The image of Fig. 1.7 unthresholded at 10.5.

Consider now Fig. 1.10. This shows the image from Fig. 1.7 after applying a simple edge detection operator, as described in section 3.1. The pixels printed

For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerised systems. Yet in all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.



Fig. 1.8 The image of Fig. 1.7 thresholded at 5.5.

Both the Kurzweil reader and the Workless Station will have images something like either Fig. 1.8 or Fig. 1.9. Looking at the thresholded images, it becomes immediately obvious why neither machine could read *both* the black and the white text, *even if they had the capability to look for white marks on a black background*. It also gives a good idea as to why they would both struggle with the picture; it has little to distinguish it, other than size, from the rest of the text.

**For documents conventional OCR can't handle, Kurzweil offers a better approach: ICR.**

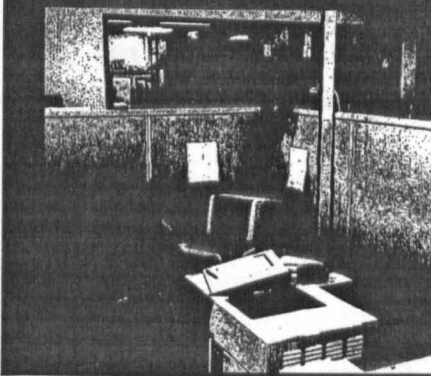


Fig. 1.9 The image of Fig. 1.7 thresholded at 10.5.

Consider now Fig. 1.10. This shows the image from Fig. 1.7 after applying a simple edge detection operator, as described in section 3.1. The pixels printed



in black in Fig. 1.10 are those with an edge strength greater than 2. See section 3.1 for a definition of the edge strength.

This example shows clearly that starting with shades of grey, rather than a binary image, would provide an OCR system with tremendous benefits in terms of greater power to extract the interesting parts from an image. It would also make it much simpler to ignore the parts which do not contain text.

For documents conventional OCR can't handle, Kurzweil offers a better approach: ICR.



For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerised systems. Yet in all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

Fig. 1.10. The result of applying an edge detector to Fig. 1.7.

The problem that must be overcome before grey scale can be used is that at 16 grey levels for instance, four times as much data is generated per page. It is highly likely that almost any algorithm will take about four times as long to process a page under these circumstances. Therefore, there needs to be an extremely fast first step which reduces the volume of data greatly. Otherwise, the problem of the speed of feature extraction will be aggravated.

## 3.1 THE SCANNER

The scanner is an HP/RTA Scanjet, a flat-bed scanner capable of producing an image at a resolution of 300 pixels per inch in 16 grey levels. At this resolution, each A4 page can be read in approximately 30 seconds. The resulting image occupies over 4 Mbytes.

## 2. A NEW APPROACH TO CHARACTER RECOGNITION

The approach to character recognition taken in this thesis is different to any that has been used before. This chapter gives an overview of the complete system of which Fig. 2.1 is a block diagram. All the components of Fig. 2.1 are described in outline, while edge extraction, polygonal approximation and feature extraction will be described in detail in chapters 3 to 6.

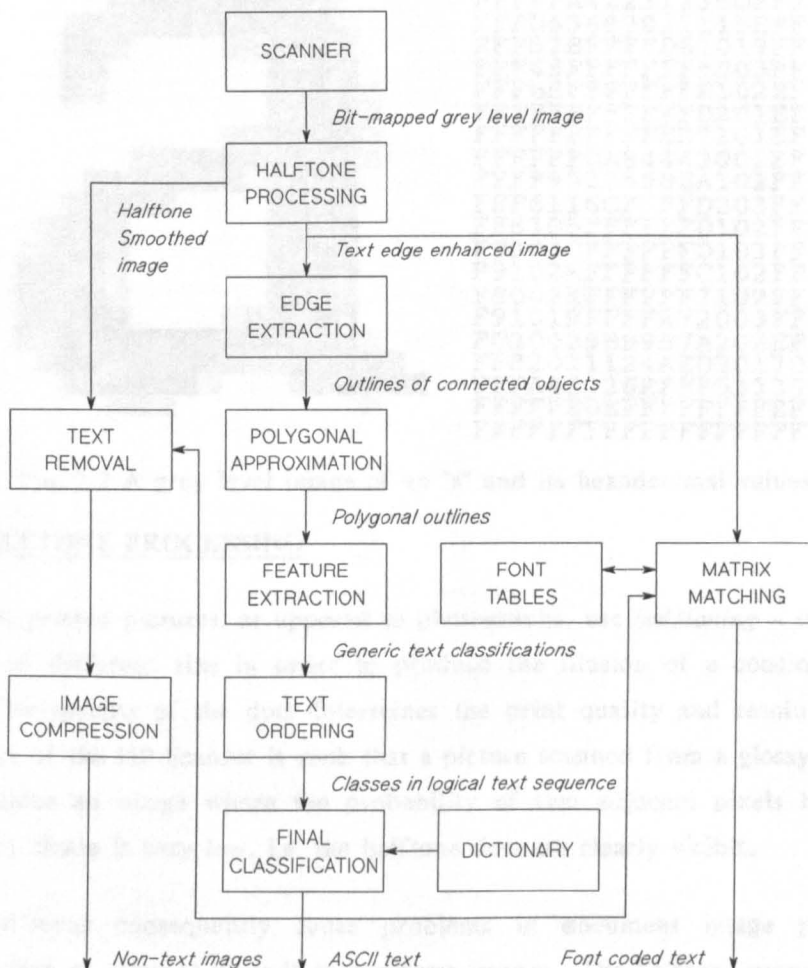


Fig. 2.1. A system block diagram.

### 2.1. THE SCANNER

The scanner is an HP9190A Scanjet; a flat-bed scanner capable of producing an image at a resolution of 300 pixels per inch in 16 grey levels. At this resolution, each A4 page can be read in approximately 30 seconds. The resulting image occupies over 4 Megabytes.

The mechanism is now becoming commonplace. It operates on the same principle as the scanning half of a photocopier. A linear Charge Coupled Device (CCD) array is the light sensing element. In the ScanJet, both the paper and the CCD array remain stationary, while a system of mirrors is moved vertically down the page. Fig. 2.2 shows the grey level image of a letter "a" both in image form and as hexadecimal digits, with 0 and F representing black and white respectively.

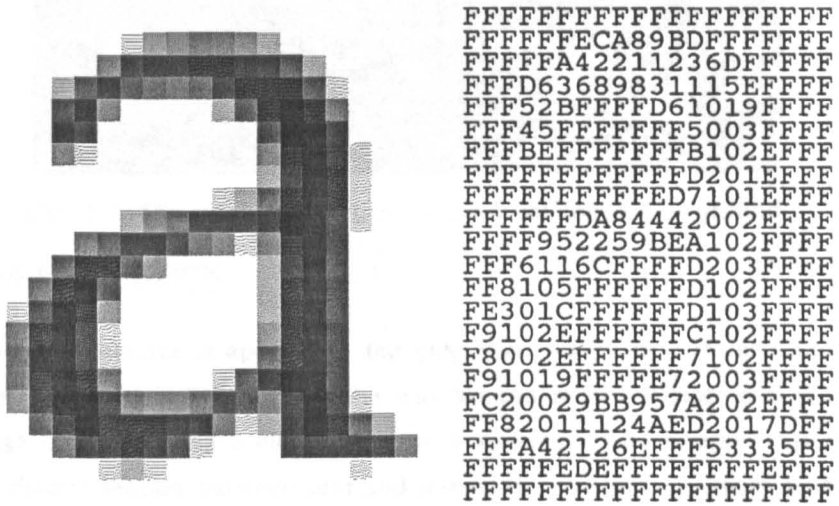


Fig. 2.2 A grey level image of an "a" and its hexadecimal values.

## 2.2. HALFTONE PROCESSING

All *printed* pictures, as opposed to photographs, use *halftoning* - ink printed in dots of different size in order to produce the illusion of a continuous grey scale. The spacing of the dots determines the print quality and resolution. The resolution of the HP ScanJet is such that a picture scanned from a glossy magazine will produce an image where the probability of two adjacent pixels having the same grey shade is very low, i.e. the halftone dots are clearly visible.

Halftones consequently cause problems in document image processing, mainly when an attempt is made to compress images. The halftone processing step performs two separate functions. Firstly, halftones which would normally make an image difficult to compress are smoothed by local averaging. The smoothed image can then be compressed much more easily by relatively simple methods.

Text is occasionally printed on a halftoned or shaded background. Where the contrast is high, the edge extractor can still find the outlines. When the contrast is low however, the edge breaks up and disappears completely. It would be desirable to be able read such text, even though the situation occurs rarely. The second task of the halftone processing step is thus to enhance the appearance

of text when printed on a halftoned background. Fig. 2.3 shows a particularly difficult example.

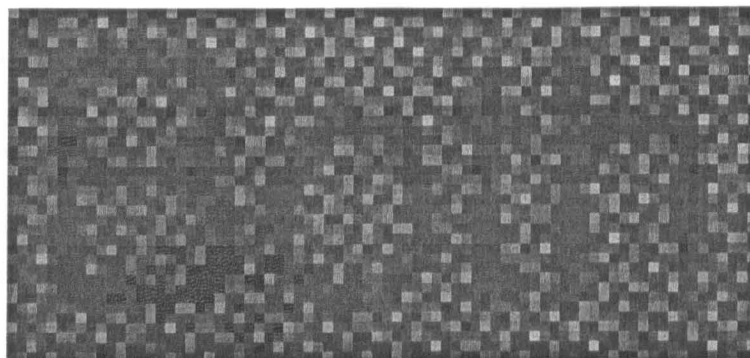


Fig. 2.3. The word "Both" printed on a halftoned background.

### 2.3. EDGE EXTRACTION

An edge detector is applied to the grey level image in a raster scan. When an edge is found, it is followed all the way around until a closed loop is formed. If the edge does not form a closed loop, it is ignored. This operation provides a powerful discrimination between text and non-text. The use of a grey level image and an edge extractor is the major difference between this approach to OCR and any previous method. The important advantages are:

- (a) Looking for edges rather than black regions provides the ability to read any colour of text on a background of any colour, provided that the contrast is sufficiently great. This is simply impossible to do by thresholding without manually selecting for positive or negative print.
- (b) When an image is thresholded, non-text portions of the image form the same kind of black regions on a white background as text. This causes the character cell location algorithm to be confused. With a grey level image, the vast majority of edges in the non-text region do not form closed loops and are therefore rejected. Any edges which do form closed loops will either not look like any known character, or will fail to form a reasonable text line. Hence, grey level images provide discrimination between text and non-text.
- (c) Apart from a simple raster scan of the entire image, the edge extractor examines only the edges of characters. The number of pixels in the image that are analysed in detail is thus reduced by about 90%, making it possible to process a page in a reasonable time.

## 2.4. POLYGONAL APPROXIMATION

The edge extractor follows edges from pixel to pixel around the outline. The output is simply a long sequence of steps between neighbouring pixels. Polygonal approximation is a process whereby outlines are approximated by sequences of straight line segments, i.e. polygons.

There are two reasons why it is better to approximate the outlines before passing them on to feature extraction:

- (a) Both polygonal approximation and the processes used in feature extraction consume linear time with respect to the number of points on the input polygon. The constant of proportionality however, is much larger in the case of feature extraction. Consequently, polygonal approximation results in a considerable time saving.
- (b) One of the most important features used in recognition is the presence of concave regions in the outline. Polygonal approximation straightens out the zig-zags in the outline which would otherwise produce false concavities and confuse the recognition process.

The edge extractor reduces the volume of data in an image by about 90%. The quantity remaining however, is still extremely large and the polygonal approximation algorithm must be chosen carefully to suit the available processing time.

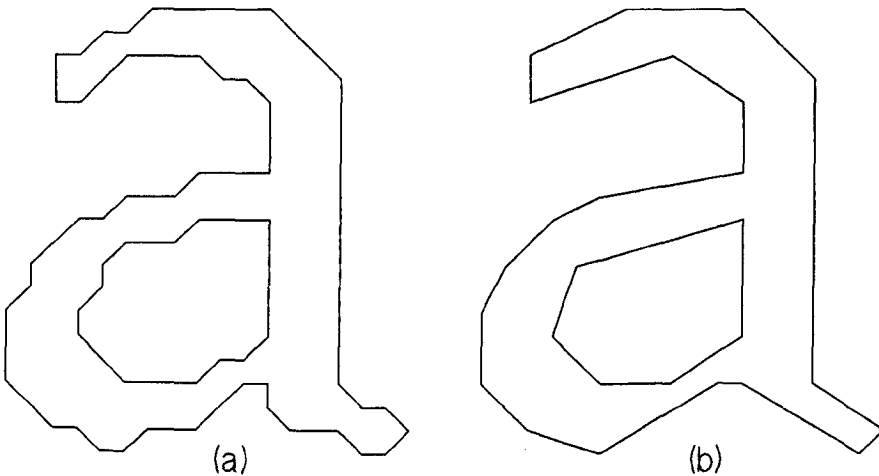


Fig. 2.4. (a) The edge path of the image in Fig. 2.2.  
(b) A polygonal approximation of the outline in (a).

Fig. 2.4(a) shows the outline path traced by the edge extractor for the image of Fig. 2.2. Fig. 2.4(b) shows the result of polygonal approximation when applied to Fig. 2.4(a). The number of points on the outline is thus reduced from 98 to 26.

## 2.5. FEATURE EXTRACTION

From the outlines of characters, fundamental features are extracted. There is some psychological evidence<sup>(34-41)</sup> that the particular features used are important to humans in the recognition of text. The features used are:

- (a) **Concavities.** A concavity is more complex to define than a set of adjacent concave points; it is an area that can be filled, which may include some convex points. This difference is illustrated by Fig. 2.5, which contains three concavities shown as dot filled discs.
- (b) **Closure.** A closure is a region of the background which is completely enclosed by the character. In certain fonts, characters such as "P" and "e" are printed with an incomplete closure. It is the presence of *functional* closure rather than *physical* closure which must be detected. These terms are defined in section 5.1.1.
- (c) **Lines.** A line is a straight part of the outline. Line detection is complicated by the polygonal approximation, since curves are converted into a small number of straight lines. They are thus rendered less distinct from actual straight lines.
- (d) **Axes.** An axis is used for objects which have no concavity or closure. The axis feature measures the ratio of the lengths of the major and minor axes of a convex object. It is used to distinguish between characters such as dot and a straight comma.
- (d) **Symmetry.** Symmetry is measured by comparing parts of the character on either side of an axis. Measurement of symmetry is difficult in the case of italicised text, where the axis of symmetry is not perpendicular to the direction of measurement.

In Fig. 2.5, the outline of Fig. 2.4(b) is repeated with the closure and three concavities shown by discs. The closure is illustrated by a dashed disc centred on the centroid of the closed polygon. Its area is equal to the area of the polygon. The dotted discs are similarly centred on the centroids of the concavities with their areas being equal to those of the respective regions. Note that the position

of the centroid is shown only to the nearest integer and that that it is unclear where the ends of the uppermost concave region may be.

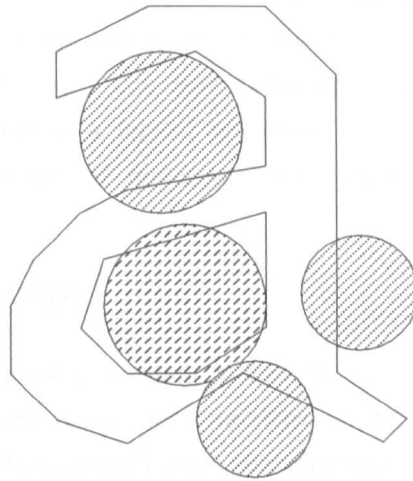


Fig. 2.5. The outline of Fig. 2.4(b) with a closure and 3 concavities identified.

Using these features, an unknown character is matched with a generic character class. The mapping of generic character classes to ASCII characters is not one to one. A small number of characters appear in more than one fundamental shape, for example "g". Such characters must have more than one class which map to the same ASCII character.

A more common case is a single class which can map to one of several characters. In certain fonts for instance, the characters 1(digit one), l(letter ell) and I(capital i) are identical. The only way to distinguish them is by context. At the feature extraction stage, there will be several generic classes to cover the various physical shapes of these three characters. Most of these classes will map to one of two or three ASCII characters, depending on the particular shape.

## 2.6. TEXT ORDERING

The edge extractor locates individual connected regions of the image. Certain characters are deliberately printed in 2 or more pieces, such as "!". These are considered separate entities by the edge extractor, but are re-associated by the text ordering stage, so that the dot can be used to distinguish the exclamation mark from "I".

In the previous section, it was stated that some generic classes can map to "1", "l" and "I". In fact, until broken characters are re-associated, the same class can also map to "!", "i" and occasionally "".

Another problem caused by the edge extraction process is that the raster scan will not find characters in an ideal order. Tall characters on a line such as "l" and "t" will all be found first. The smaller characters such as "e" and "o" will be found several scan lines down the page. A logical text line will thus be broken into several sorted partitions. The text ordering phase uses a bucket sort to quickly put the characters in a logical sequence. This apparently trivial task is complicated by the possibility of multiple text columns in different font sizes on the same page.

## **2.7. FINAL CLASSIFICATION**

Since the system aims at reading almost any printed font of any size, all characters are scaled. Capital and small versions of some letters are thus rendered indistinguishable. The final classification step discriminates upper and lower case by using contextual information. Neighbouring characters of known case provide the expected size and relative position of the unknown character. The final classification step also uses context to resolve more difficult ambiguities such as l/1/I which, as stated in section 2.6, will map to the same class in certain variations.

Contextual resolution of ambiguities can be assisted by a spelling dictionary. If one or more characters in a word have two or more possibilities each, combinations of these choices can be given to the spelling checker, in order of decreasing probability, until it accepts one of the alternatives.

## **2.8. TEXT REMOVAL**

When text has been recognized it can be removed from the image. Any interesting non-text images on the document can then be compressed and stored for use in later documents. This facility will be of use not only for office automation applications, but also for on-line libraries, where the main aim is to achieve maximal compression of any kind of document.

In the trivial case of black text on a white background, removal of the text can be achieved by simply writing white to the rectangular bounding box of each character. This can be extended to removal of text on any plain background by filling with the grey level surrounding the rectangular bounding box.

When the text is printed on top of a patterned or halftoned background, the task becomes very much more difficult. When text is printed over a picture, text removal is virtually impossible without leaving some visible mark.



## 2.9. IMAGE COMPRESSION

Many techniques have been developed for image compression.<sup>(20)</sup> A significant proportion of them rely on some knowledge of the image to be compressed. The algorithm required here is not expected to deal with text; the main requirement is to compress a blank image and continuous grey levels.

Simple run-length coding, replacing runs of the same grey level with a level and a length, would perform adequately for documents with very few or simple images, and could be executed in software. More complex algorithms which make use of the 2-dimensional nature of the data<sup>(20)</sup> would yield higher compression ratios for use with large images.

## 2.10. MATRIX MATCHING

Once a character has been recognized as an "A" for instance, the bit map of that letter can be compared with bit maps of other versions of the same letter. If the bit map matches a character in the font table, then a font code could be output with the ASCII value of the character, which would enable the document to be reproduced exactly. If no match is found, then the new character could be added to the font table.

This approach is useful for document storage, but if it is necessary to edit the document and print a new copy, a difficulty arises. If a section of the document contained a previously unknown font, there is likely to be only a limited portion of the full character set stored in the font table. The editing process could introduce characters of the same font for which there was no sample in the original document.

In order to satisfy the request for characters which have no defined bit map, it is necessary to be able to construct realistic predictions based on a small set of examples. Automatic font generation of this nature is possible, but the discussion is beyond the scope of this thesis.

### 3. EDGE EXTRACTION

*Edge detection* is a term generally applied to the process of finding steep changes of intensity in a grey level image. Because OCR systems operate with binary images, edge detection has not previously been applied to character recognition, except by Brady et. al.<sup>(25-27)</sup> In their work, edge detection was used to construct a set of disjoint, straight edges quantized in 8 directions. Further processing then used the individual lines.

This chapter will describe an algorithm for the extraction of closed loop edge paths from grey level images. Closed loops are more useful than disjoint straight lines, since they indicate the shape of the entire object and do not obscure curves. Firstly, some typical edge operators will be examined.

#### 3.1. SOME EDGE DETECTION OPERATORS

There are many different approaches to edge detection. Levialdi<sup>(21)</sup> gives a comprehensive survey of the field. The most common technique is to apply a window operator to a neighbourhood of each pixel in the image. The operator calculates a weighted difference of the grey levels in the neighbourhood.

Several such window operators have been suggested. Most of them are derived by considering the image to be the values of a function of two variables. Discrete approximations to the gradient operator on the function provide the window operators. Rather than use function notation however, the operators are clearer if they are presented as windows. A simple example is the Roberts<sup>(22)</sup> operator in Fig. 3.1. There are two windows, and each is applied to all positions in the image. The grey value at each pixel is multiplied by the number in the corresponding window cell. The results are summed, with each window giving the strength of the edge in a particular direction.

0	-1
1	0

-1	0
0	1

Fig. 3.1. The Roberts edge operators in window notation.

In order to derive a single positive number which represents the strength of the edge at each window position, it is necessary to combine the results of the two

windows. Ideally, they should be squared and added but, in reality, this operation is too time consuming. In practice, the choice is between adding the absolute values and taking the maximum of the absolute values. In either case, the resulting edge strength is dependent on the orientation of the edge.

1	0	-1
1	0	-1
1	0	-1

-1	-1	-1
0	0	0
1	1	1

Fig. 3.2. A pair of edge operators defined by Prewitt.

A similar pair of operators shown in Fig. 3.2 are due to Prewitt<sup>(23)</sup> and use a larger window. The difficulty with the larger window is that edges at the limit of resolution of the image are blurred or lost completely by the operator. Depending on the situation, this may be regarded as elimination of noise. In some fonts however, a full stop "." may be only two pixels square and faint. The large size of the operator would spread and weaken the edges to an intolerable degree.

Chaudhuri and Chanda<sup>(24)</sup> suggest a 4-neighbour gradient operator shown in Fig. 3.3 which overcomes some of these losses by considering fewer points. This is the operator which was used to produce the example edge map in Fig. 1.10, since it is similar to the tracking operator described in section 3.2.1.

0	-1	0
0	0	0
0	1	0

0	0	0
-1	0	1
0	0	0

Fig. 3.3. The 4-neighbour operators of Chaudhuri and Chanda.

The usual application of edge detectors is in the field of computer vision. An image would typically contain a face, or some boxes, or an industrial component. The images concerned are usually from a TV camera and are consequently small, (256 by 256 pixels) with as many as 256 grey levels and the objects are large with wide edges. Furthermore, the output edges are not generally expected to be continuous or form closed loops, and they are left in the

form of a bit map. In contrast, an A4 document produces images of approximately 2400 by 3300 pixels, in which objects may be a single pixel in width. It is required to find continuous closed loops on the image, representing the outlines of the text characters, while rejecting edges associated with non-text images.

### 3.2. A DESCRIPTION OF THE NEW EDGE EXTRACTOR

The edge extractor uses a simple new edge operator to follow edges around outlines until they form a closed loop. The version described in this chapter has not yet been implemented. It is however, a reasonable simplification of the edge extractor described in chapter 6 which has been implemented and tested extensively.

#### 3.2.1. DEFINITION OF THE EDGE OPERATOR

The edge operator is designed to find the edge strength perpendicular to the direction of travel between neighbouring pixels. Fig. 3.4(a) shows the 3 by 3 neighbourhood of X. The *chain code* of a point in the neighbourhood relative to X is given by the number in the box corresponding to the point. If X is the present position then the *edge value* at a candidate next point with chain code n is the difference between the grey levels at points with chain codes (n+1) (modulo 8) and (n-1) (modulo 8). Thus the edge value at the point with chain code 0 is found by subtracting the grey level at 7 from the grey level at 1. The subtraction is illustrated in Fig. 3.4(b) in the style of the previous window operators. A similar representation of the operator for chain code 7 is shown in Fig. 3.4(c). Note that there is only a single operator for each direction, and that the result is signed.

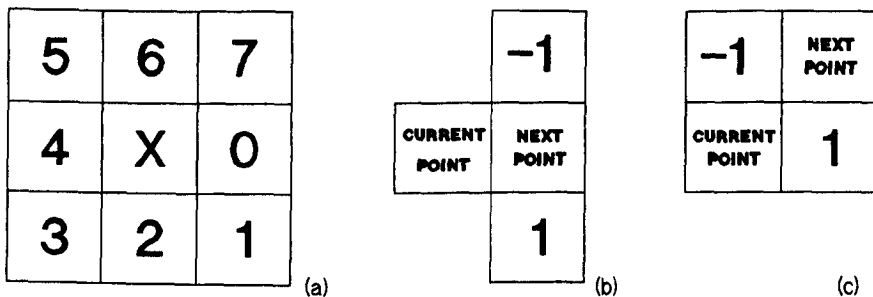


Fig. 3.4 (a) The 3 by 3 neighbourhood of X. (b) The edge operator for chain code 0. (c) The edge operator for chain code 7.

The edge operator was derived experimentally from real images. Any larger operator would get confused at very sharp corners and any smaller operator would be less symmetric. This operator is difficult to apply in the sense of the window

operators of section 3.1, i.e. to produce a bit map of the edge strengths, but for chain codes 0, 2, 4 and 6 it is equivalent to the 4-neighbour operator of Chaudhuri and Chanda.<sup>(24)</sup>

### 3.2.2. THE EDGE FOLLOWING ALGORITHM

The main process is to apply the edge operator corresponding to chain code 4, in a raster scan of the image. The raster scan thereby locates the top edge of a new object. When a point is found with an edge value which is greater than a specific starting threshold and of either sign, the raster scan repeats the check one pixel lower in the image. If the edge value there is of the same sign and also greater than the starting threshold, raster scanning is suspended and the line tracking process is started from the point with the higher edge value. The double check is to ensure that the edge is significant and that tracking starts at the strongest point.

The line tracking process commences by tracking left from the starting point, i.e. with chain code 4, aiming to complete an anticlockwise traversal of the outline. Given the chain code  $n$  of the last direction taken, edge values are found at each of the 5 points with chain codes  $n-2$ ,  $n-1$ ,  $n$ ,  $n+1$  and  $n+2$  (modulo 8) from the current position. Call the 5 or less points which have the correct sign and are above the threshold, the *edge set* corresponding to this position and direction. From the edge set, the point with the strongest edge value is found. The choice of next point is non-trivial, but in most cases it will be the point with the strongest edge value.

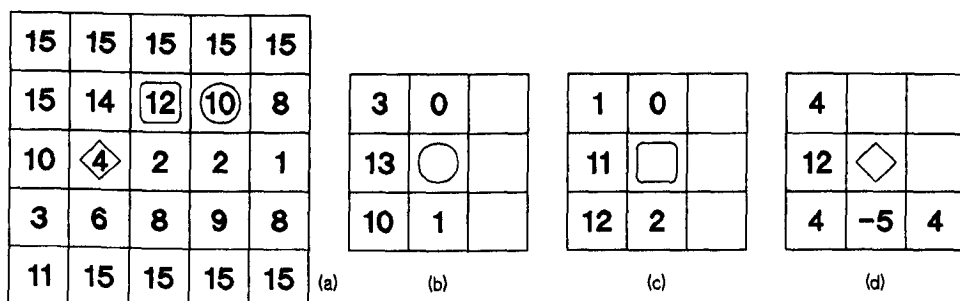


Fig. 3.5. (a) A part of the "a" in Fig. 2.2. (b) Edge values at the circled point. (c) Edge values at the rounded square. (d) Edge values at the diamond.

An example of the tracking process is illustrated in Fig. 3.5. The values in the pixels in Fig. 3.5(a) are the actual grey level values for part of the "a" in Fig. 2.2. Assuming that the current point is circled, and the last move was in direction 4, then Fig. 3.5(b) shows the edge values which would be calculated at the circle.

The next step would be to the rounded square, corresponding to edge value 13. The edge values of Fig. 3.5(c) would be calculated at the square. Fig. 3.5(d) demonstrates how the calculated edge values change with the last chain code and represents the edge values at the diamond.

It can be seen in Fig. 3.5, that at each step there are several useful edge values. While edge following proceeds, points on the edge path are marked *current* to indicate that they form part of the latest outline to be processed. Clearly if only one point is marked at each step, the raster scan will later find unused points on the same edge and attempt to follow the outline again. Therefore, as many points as possible, (up to a limit of three) points are marked, subject to the following constraints:

- (a) The strongest edge point is always marked. (Equal strongest edge values are discussed in section 3.2.3.)
- (b) All points to be marked must be in the edge set and have edge values over the tracking threshold.
- (c) All the marked points must be adjacent.

Thus the points marked in Fig. 3.5 will be 13, 10 in (b), 11, 12 in (c), and 12 in (d), assuming a threshold of 4.

If at any time no points are found over the tracking threshold, then the edge is considered to have faded, the object is discarded and the whole edge is re-marked as *deleted*. The tracking process continues until the edge fades, a closed loop is completed, or the edge coincides with a previously completed outline. When an edge closes a satisfactory loop, all the points on the loop are re-marked *used*. The reason for changing all the marks when an edge is complete is to enable the discrimination between collisions with other lines and closing of the loop.

### 3.2.3. THE CHOICE OF EDGE PATH

In the example of Fig. 3.5 it is clear that at each step there is not necessarily an obvious choice of edge path. For instance, at the rounded square in Fig. 3.5(c) the highest edge values are 12 and 11. In the majority of circumstances, the choice has an insignificant impact on the result, except perhaps to make the outline more or less jagged. There are several cases however, where it is important to select a specific direction:

- (a) In many fonts, lines are of variable width. At the thinnest part of a line, it may be possible to follow an edge path across the line and thus incorrectly break up the character.
- (b) Some characters may be printed in very close juxtaposition. At the point where the two characters almost touch, there is a two-way decision to be made when traversing either character. Both decisions must be correct, otherwise the outline of one will collide with the other, resulting in the loss of one or both characters.
- (c) Where an edge is blurred, it can be necessary to take a path corresponding to a weaker edge value in order to avoid the edge fading at a subsequent step.

Fig. 3.6 exemplifies problem (a). Fig 3.6(a) is a portion of the "a" in Fig. 2.2 at the point where the top of the loop meets the stem. The darkest pixels are shaded. Fig. 3.6(b) gives the edge values at the circled point, showing two points with edge value 11. The edge sets from each of these points are shown in Fig. 3.6(c) and (d), with the possible destinations indicated by a rounded square and a diamond in Fig. 3.6(a). The rounded square represents the correct path and the diamond is on the wrong side of the loop.

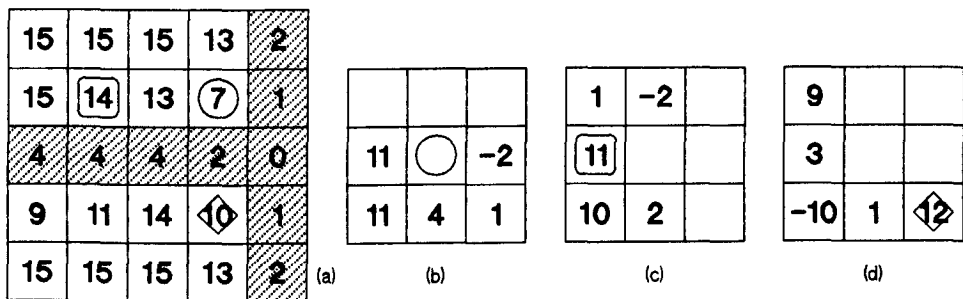


Fig. 3.6. (a) Part of the "a" of Fig. 2.2. (b) Edge values at the circled point. (c), (d) Edge values from the alternatives in (b).

Problem (b) is illustrated by Fig. 3.7. Fig 3.7(a) shows the correct outlines of a pair of closely spaced characters. The starting point on each outline is marked "+". In Fig. 3.7(b), a different choice was made while tracing the "t", resulting in merging of the characters. The same path was taken in Fig. 3.7(c), but with a different decision on the second pass through the difficult region. The result is that the "i" is completed, but the edge did not close at the start of the loop, therefore both characters are lost. Finally, in Fig. 3.7(d), the "t" was negotiated correctly, but the same route was taken for the "i" as in Fig. 3.7(b), so it collided with the completed "t" and was deleted.

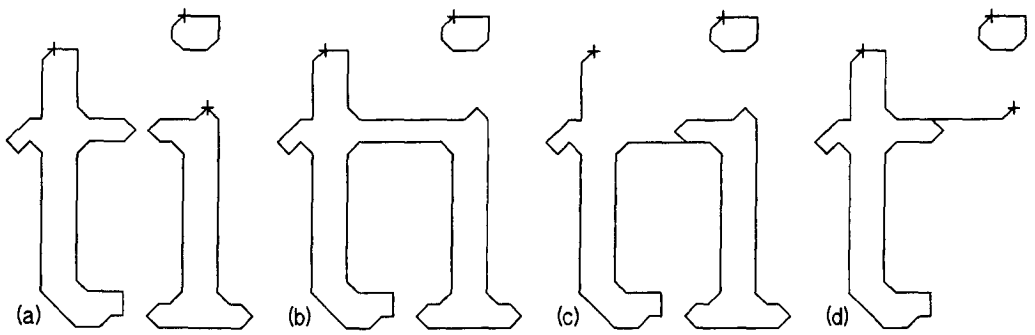


Fig. 3.7. (a) A correctly extracted "ti" pair. (b) Merging by incorrect path choice. (c) Both characters lost by the loop closing incorrectly. (d) The "i" collides with the correct "t".

The most comprehensive single solution to the above problems is to look ahead several steps to edge values beyond the current point. By comparing the position resulting from taking the leftmost choice at each step, with the position resulting from taking the rightmost choice at each step and measuring the distance between, possible forks in the edge path can be detected. Tracing the edge in the reverse direction from each of these points solves problems (a) and (b) by determining the viability of each alternative. If only one path can be traced backwards successfully then that path is selected.

Look ahead also enables the detection of fading edges and those paths which collide with a completed line. Alternative paths can be chosen in order to avoid (c) above. The only difficulty with this approach is that the amount of time consumed by searching a tree of possible paths to a reasonable depth is unacceptably expensive.

There are many different algorithms for selecting the most appropriate edge path in a reasonable time. A rather complex solution is discussed in section 6.3. The next section describes a simple alternative, which is based on the failure modes described above.

### 3.2.4. A SIMPLE EDGE PATH ALGORITHM

The algorithm for selecting between several alternative edge paths needs to be fast, since there is some choice at almost every point on most outlines. The number of cases where the choice is vital however, is only a small fraction of the total. The most important consideration is to avoid losing characters as in Fig. 3.7(c) and (d). Merging adjacent characters is a relatively insignificant problem,



since the range of print quality which an OCR system is expected to handle includes both merged and disintegrated characters.

In Fig. 2.2, it is clear that there is a good dark line where the loop meets the stem, but it is a single pixel wide. The problem characterized by Fig. 3.6 is due to a failing of the edge operator. A simple addition to the edge operator will solve this particular problem, illustrated in Fig. 3.8. The extra operator is applied only in the event of a 90° bend and acts as a qualifier to exclude the possibility of crossing a unit width line. An example of the diagonal case is given in Fig. 3.8(a) and the upright version in Fig. 3.8(b). Any edge value associated with a right angle bend is replaced by the minimum of the usual edge value and the value obtained from the additional operator in Fig. 3.8.

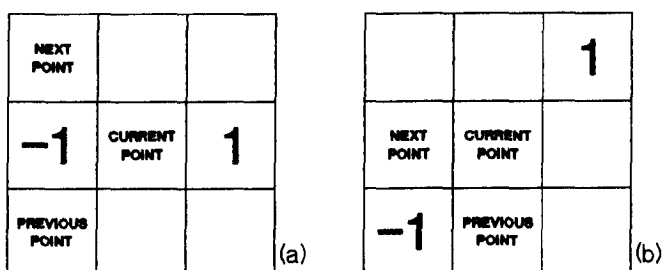


Fig. 3.8. An extra edge operator for 90° bends. (a) Diagonal bends. (b) Upright bends.

The requirement of a reasonable level of look-ahead, (3 steps) is inevitable to prevent edges fading and collisions of the type in Fig. 3.7(d). In order to save time however, the simple edge path algorithm looks ahead down the strongest edge path only. Alternatives are considered whenever the strongest edge path fades, or collides with another edge.

If an edge collides with itself, away from the start of the line, as in Fig. 3.7(c), the line is not discarded. The closed part is accepted and the rest of the line is continued, after backtracking to a point where there is an alternative edge path. This approach will solve many of the present failure modes of the complex edge path algorithm described in section 6.3. It will also act as a backup for when the extra edge operator in Fig. 3.8 fails to resolve the problem of Fig. 3.6.

### 3.2.5. ASSOCIATION OF RELATED EDGES

All edge lines are detected and traced independently, yet some edges are related. Consider for instance the inside and outside edges of a letter "O". The edges are both part of the "O" and need to be related in the output structure in

order for the features to be extracted correctly. It is necessary to record that the inner edge is enclosed entirely by the outer edge.

In the edge extractor of chapter 6, a *nesting tree* is described as a suitable representation of the relationship between the edges. Construction of the nesting tree is however, a complex process. Given that the edge extractor is only required to locate text, the requirement of the nesting tree can be eliminated.

Consider the minimal upright rectangular bounding box of a closed loop  $C_1$ . Let the coordinates of the bottom left hand corner of the box be  $(x_{min_1}, y_{min_1})$ . Similarly, let the coordinates of the top right hand corner of the box be  $(x_{max_1}, y_{max_1})$ . If the coordinates of the points on  $C_1$  are given by  $(x_i, y_i)$ ,  $i=1, n$ , then:

$$x_{min_1} = \min(x_i), \quad x_{max_1} = \max(x_i) \quad \text{over } i=1, n.$$

$$y_{min_1} = \min(y_i), \quad y_{max_1} = \max(y_i) \quad \text{over } i=1, n.$$

Clearly, given a second closed loop  $C_2$ , which lies entirely within  $C_1$ , the rectangular bounding box of  $C_2$  lies within the bounding box of  $C_1$ , i.e.

$$x_{min_1} < x_{min_2}, \quad x_{max_1} > x_{max_2}, \quad y_{min_1} < y_{min_2}, \quad \text{and} \quad y_{max_1} > y_{max_2}.$$

Given that only text is to be processed, the bounding boxes of all objects in the page are sorted using a 2-dimensional bucket sort. The bucket size is set to the minimum acceptable character size so that the number of buckets in a page is considerably less than the number of pixels. As each outline is completed, it is inserted into the relevant bucket, according to the top-left corner of the bounding box. When the page is completed, a simple linear search of the buckets is used to locate all completed outlines. For each outline, the small local area of buckets covered by the bounding box is searched. Any other outlines found which are enclosed are recovered and recorded as holes in the character.

The processing time consumed by this bucket sorting procedure is minimal, since the ratio of buckets to pixels is very small, (around 1 in 256 would be adequate) yet it guarantees to find the nesting relations for all characters in the ASCII character set. There is however, one character in the ASCII set for which the bucket sort can produce incorrect results. It is possible, although highly unlikely, for the percent symbol "%" to be printed such that the circles appear to be enclosed by the stroke.

This situation is illustrated in Fig 3.9(a). Since this is a single exception, there is a simple solution. If the  $(x,y)$  space is transformed to the  $(x+y,x-y)$  space, and bounding boxes are also calculated for the rotated space, then testing the enclosure condition for *both* spaces ensures a correct result. The result of this simple rotation transformation is illustrated in Fig. 3.9(b).

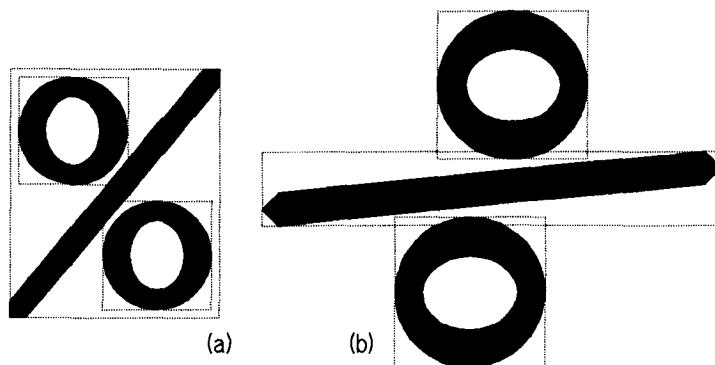


Fig. 3.9. (a) An example of the failure of bucket sorting. (b) A simple rotation transformation solves the problem.

Note that the sole objective of the bucket sort is to obtain the nesting information of characters with holes such as "O". As a side-effect, for a wide range of page layouts, the operation provides a successful sort of the characters into logical text order.

## 4. POLYGONAL APPROXIMATION

The edge extractor of chapter 3 produces chain coded closed loops. This chapter describes how these chain coded loops are replaced by polygons made up of straight line segments. Firstly, the state of the art will be reviewed. This will be followed by descriptions of a faster version of one of the reviewed methods and a new polygonal approximation algorithm.

### 4.1. THE STATE OF THE ART IN POLYGONAL APPROXIMATION

Several algorithms for polygonal approximation<sup>(28-32)</sup> have been proposed. Many of these have been too slow to be considered in this particular application. Two algorithms have recently been published which are worthy of consideration. These will now be described.

#### 4.1.1. APPROXIMATION USING WEDGES

Sklansky and Gonzalez<sup>(31)</sup> describe a polygonal approximation algorithm which has a precisely controlled error margin. The input points need not be regularly spaced and the processing time is linear in the number of input points.

Suppose the required approximation error distance is  $d$ , (typically taken to be 1 pixel unit) i.e. every source point must lie within distance  $d$  from the final line. In Fig. 4.1(a),  $P$  is the start of the line and  $Q, R, S$  etc. are the subsequent points. The circles are of radius  $d$ , therefore any approximating line must lie within all the circles. The algorithm steps along the line to find the next approximation point. In Fig. 4.1(a)  $R$  is already accepted and  $S$  is under consideration. A wedge is drawn from  $P$  forming tangents to the circle around  $R$ . For  $S$  to be accepted, a similar wedge drawn to  $S$  must overlap the current wedge. The wedge of intersection is used in the next step.

In Fig. 4.1(b) a wedge is drawn to  $T$  and this is found to overlap the current wedge. The wedge for  $U$  however, does not overlap the updated current wedge, so there the process stops.

Note that when  $T$  was processed, its wedge overlapped the current wedge but  $T$  itself was not within the current wedge. If the approximating line were to be drawn to  $T$ , the maximum approximation error would be greater than  $d$ . If a point subsequent to  $T$  is found which does lie within the updated wedge, then that can be used and the result will be correct.

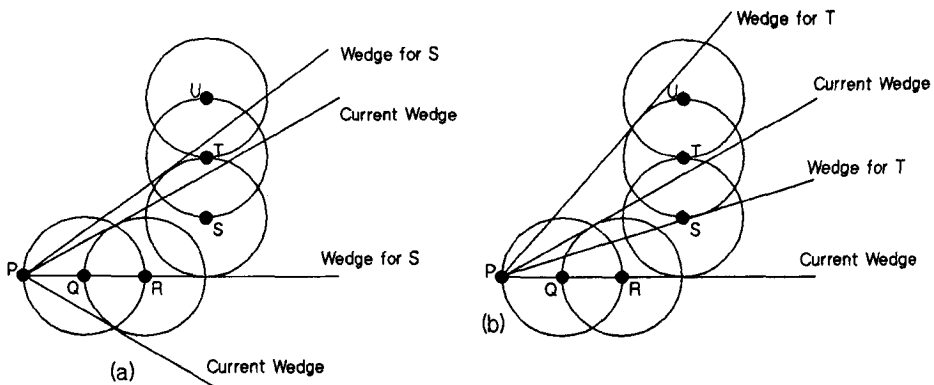


Fig. 4.1. Sklansky and Gonzalez polygonal approximation.  
 (a) Processing point S. (b) Processing point T.

Therefore, when there is no overlap of the new wedge and the current wedge, as with U in Fig. 4.1(b), the last point to fall within the wedge when it was tested will be used in the approximation, in this case S. The approximation process will then restart at S.

In the paper by Sklansky and Gonzalez,<sup>(31)</sup> all the testing of wedges is done using angles from the positive x-axis. This approach requires floating point arithmetic and inverse trigonometric functions to calculate the angles. Finding the tangents also requires trigonometric functions. Such use of floating point arithmetic makes this algorithm too slow. A fast integer version has been designed however, which increases the speed to a point where it is almost viable. For a discussion of this new algorithm, see section 4.2.

#### 4.1.2. APPROXIMATION USING CHAIN CODES

Hung and Kasvand<sup>(32)</sup> describe an algorithm which uses chain codes to obtain an approximation. Differences between adjacent chain codes are calculated. Non-zero difference codes indicate a local change of direction of the line. Sequences of pixels with non-zero difference codes are called *non-zero segments* (NZS). Seven rules for generating output points are then applied to each NZS. Some of these rules are extremely complex and are thus not repeated here.

The important aspect of this algorithm is that no floating point arithmetic is required. This makes it extremely fast and probably faster than the integer version of the Sklansky and Gonzalez algorithm described in section 4.2.

The complex rules governing the generation of output points are explained by example, but there is little indication of their derivation, nor of how they will perform given any input line. The main discussion is centred around the fact that

if a straight line is digitized on a square raster, it produces either a sequence of one repeated chain code, or regular zig-zags. If these straight lines can be detected, then the remaining changes of direction are true bends in the line.

The use of difference codes assists this process. Where the difference codes are non-zero, adjacent pairs are summed. Zero pair sums represent a cancellation of two local changes in direction, exactly the case of a straight line being digitized into regular zig-zags. This idea has been used by Chaudhuri and Kundu<sup>(33)</sup> for exact compression of chain codes, and leads on to the simple polygonal approximation algorithm described in section 4.3.

## **4.2. IMPROVING THE SKLANSKY AND GONZALEZ ALGORITHM**

The algorithm of section 4.1.1 is linear with respect to the number of source points, but the operations at each point are complex. Square root and inverse trigonometric functions are required to find the tangents and the angles from the x-axis. All these functions and the need for floating point arithmetic can be eliminated completely with an integer approximation to the tangents and finding the wedge intersections by using cross products of vectors.

The most complex function remaining is integer multiplication. This reduction in complexity is obtained at the cost of changing the statement about the approximation error from:

"The maximum approximation error is  $d$ " to:

"The maximum approximation error is no more than  $d$ ."

In other words, there may be a slight increase in the number of output approximation points, due to a variable but small reduction in  $d$ , the error circle radius.

### **4.2.1. USING CROSS PRODUCTS TO FIND THE WEDGE INTERSECTIONS**

The cross product is normally defined for 3-dimensional vectors, where the result is perpendicular to the other two. In this thesis, the vector product of 2-dimensional vectors is considered to be a scalar value.

If  $\mathbf{a}$  and  $\mathbf{b}$  are 2-dimensional vectors with respective components  $(x_1, y_1)$  and  $(x_2, y_2)$ , then the cross product of  $\mathbf{a}$  and  $\mathbf{b}$  is denoted by:

$$\mathbf{a} \times \mathbf{b} = x_1 y_2 - x_2 y_1 = a b \sin A,$$

where  $a, b$  are the Euclidean lengths of  $\mathbf{a}, \mathbf{b}$  respectively and  $A$  is the angle between the vectors taken anti-clockwise from  $\mathbf{a}$  to  $\mathbf{b}$ .

In Fig. 4.2(a), let the directions of the tangents to  $Q$  be given by the vectors  $Q\mathbf{a}$  for the left (anticlockwise) side and  $Q\mathbf{b}$  for the right (clockwise) side. Similarly, let the current wedge be the area between the vectors  $\mathbf{a}$  and  $\mathbf{b}$ . The wedges do not overlap and a new output point needs to be generated if:

$$\mathbf{b} \times Q\mathbf{a} \leq 0 \text{ or } Q\mathbf{b} \times \mathbf{a} \leq 0 \quad (1)$$

Otherwise, the new wedge is given by:

$$\mathbf{a} = \begin{cases} Q\mathbf{a} & \text{if } Q\mathbf{a} \times \mathbf{a} > 0 \\ \mathbf{a} & \text{otherwise.} \end{cases} \quad (2)$$

$$\mathbf{b} = \begin{cases} Q\mathbf{b} & \text{if } Q\mathbf{b} \times \mathbf{b} < 0 \\ \mathbf{b} & \text{otherwise.} \end{cases} \quad (3)$$

Also, if the vector from  $P$  to  $Q$  is  $\mathbf{v}$ , then  $Q$  is within the resulting wedge if:

$$\mathbf{v} \times \mathbf{a} \geq 0 \text{ and } \mathbf{v} \times \mathbf{b} \leq 0 \quad (4)$$

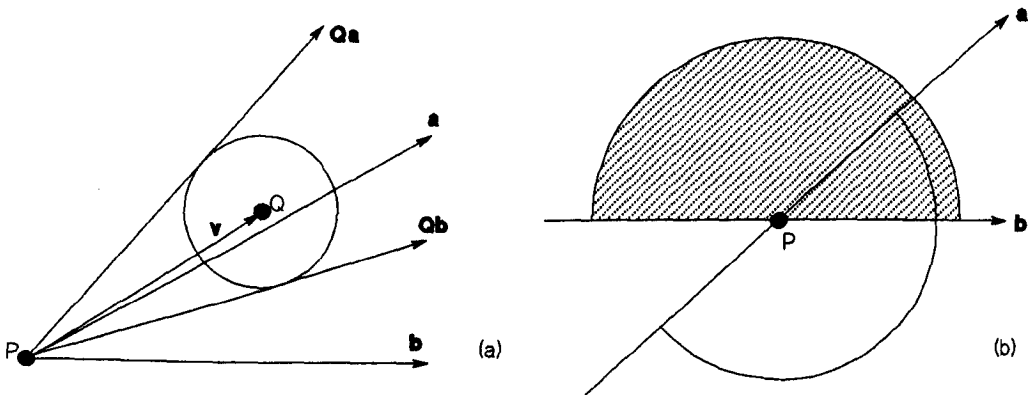


Fig. 4.2. (a) A current wedge with a new wedge under test.  
(b) The allowed ranges of  $Q\mathbf{a}$  and  $Q\mathbf{b}$ .

Tests (1) to (3) are guaranteed to work assuming that no wedge can subtend an angle of more than 90 degrees. If the approximation to the tangents examined in section 4.2.2 is used, then this is a valid assumption.

In Fig. 4.2(b)  $\mathbf{a}$  and  $\mathbf{b}$  are the current wedge. A new wedge will be rejected by test (1) unless  $Q\mathbf{a}$  lies somewhere within the dot filled semicircle and  $Q\mathbf{b}$  lies

within the plain semicircle. Clearly, no valid overlapping wedge could be rejected by test (1) unless it subtends an angle of more than 90 degrees.

Similarly no non-overlapping wedge that subtends an angle of 90 degrees or less can be constructed such that  $Qa$  lies in the dot filled semicircle and  $Qb$  lies in the plain semicircle.

Test (2) simply updates  $a$  if  $Qa$  lies in the region between  $a$  and  $b$ . Likewise, test (3) updates  $b$  if  $Qb$  lies in the region between  $a$  and  $b$ . Test (4) checks that  $v$  lies within the intersection of the semicircles.

These simple tests replace the angles from the  $x$ -axis used by Sklansky and Gonzalez<sup>(31)</sup>. It is important to note that the tests are independent of the lengths of the vectors. The next section will describe a simple way of finding the tangents using only integer arithmetic. Since all source coordinates are integers, and the tangents are expressed as integer vectors, the cross products used above require only integer multiplication for their calculation.

#### 4.2.2. A FAST APPROXIMATION FOR THE TANGENTS

The floating point arithmetic can be eliminated completely by approximating the tangents. In Fig. 4.3 there is a circle of radius  $d$  and centre  $Q(x,y)$  with a point  $P$ . The tangents will be replaced by an appropriate choice of 2 out of the 4 points  $(x+d,y)$ ,  $(x-d,y)$ ,  $(x,y+d)$ ,  $(x,y-d)$ . The choice is made on the basis of least angular error.

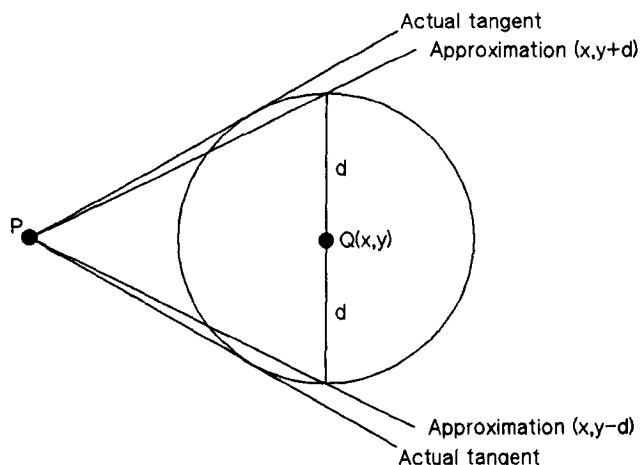


Fig. 4.3. An example of approximating the tangents.



An example of approximating the tangents is shown in Fig. 4.3. Tangents are drawn from P to the circle and there are also lines passing through the points  $(x,y+d)$  and  $(x,y-d)$ . It can be seen that in all cases, the approximations to the tangents will result in a slight reduction in the size of the wedge, and thus a slight decrease in the approximation error.

The approximations to the tangents can be calculated without even any multiplication. The choice of vectors depends on the position of the point relative to the current starting point. In the graph in Fig. 4.4, the current starting point, P, is taken to be the origin. The location  $(x,y)$  on the graph of the point under test, Q, defines the wedge vectors  $Qa$  and  $Qb$  by the quadrant in which it lies. The 4 quadrants are defined by the lines  $y=x$  and  $y=-x$ . There are 8 different pairs of values of the wedge vectors, one for each quadrant and one each for Q situated on the dividing line.

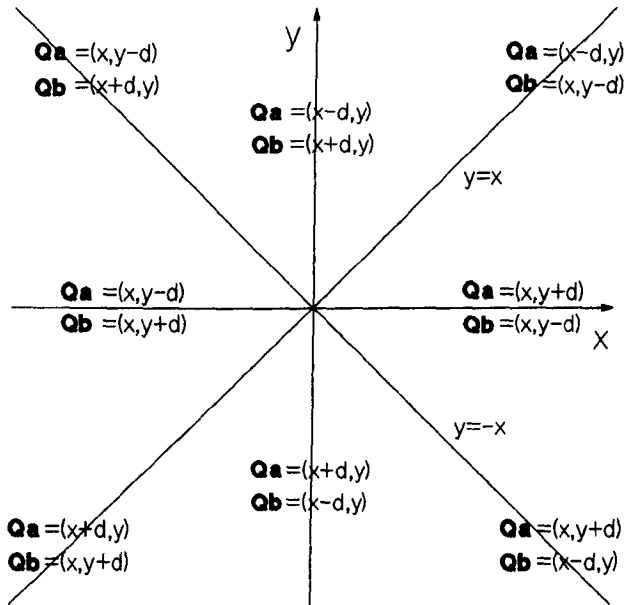


Fig. 4.4. Calculating the approximations to the tangents.

Any rational value of  $d$  can be used, say  $p/q$ , as the radius of error, without having to abandon integer arithmetic. In this case  $p$  is used as the value of  $d$  and all coordinates are multiplied by  $q$  on input. The only constraint on the value of  $d$  is that no wedge may subtend an angle of more than 90 degrees. Clearly, if  $d > 1$ , some extra treatment is needed for the case of points that are less than distance  $d$  from the origin point, since even with the original definition of the Sklansky and Gonzalez algorithm, it is impossible to construct tangents in the way intended.

### 4.3. A SIMPLE POLYGONAL APPROXIMATION ALGORITHM

This algorithm is based on the way that straight lines are represented on a square raster. Any straight line can be constructed using pixels joined by at most 2 chain codes. In the case where 2 chain codes are used, say A and B, the line will consist of repeated units of  $nA+B$  where  $n$  is a fixed integer.

Some examples of digitized lines are shown in Fig. 4.5. Each box represents a single pixel. The number contained within the box is the chain code to the next pixel. All lines are considered to start at the top of the figure. On the upper right hand line for example,  $A=0$ ,  $B=1$  and  $n=2$ .

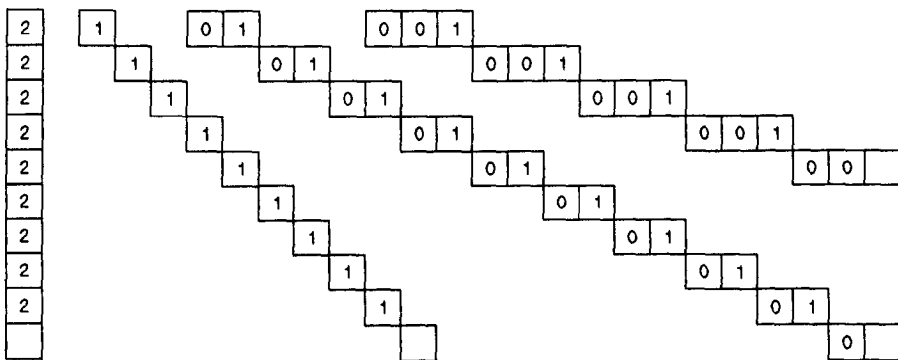


Fig. 4.5. Some examples of digitized straight lines, showing repeated patterns of chain codes.

Given a sequence of chain codes representing an outline, the algorithm assigns the first two distinct chain codes to A and B. A and B are exchanged if necessary when one of them occurs twice in succession to ensure that the repeated code is A. An approximation point is output when any of the following conditions is met:

- A and B do not differ by 1 (modulo 8).
- B occurs twice in succession.
- A consecutive sequence of A has a length different to the previous sequence.
- A third distinct chain code is encountered.

The above rules are illustrated in the corresponding parts of Fig. 4.6. In Fig. 4.6, the output approximation point is shown by a dot filled pixel. All lines start at the left hand end of the figure.

These rules are very simple, yet they produce an output point at the end of each line of the form  $nA+B$ . The approximation is thus guaranteed to be accurate

to within one pixel everywhere, and most importantly, it is extremely fast. The only disadvantage is that the number of output points is rather greater than would be produced by the algorithm of section 4.1.1, although similar to the output of section 4.1.2.

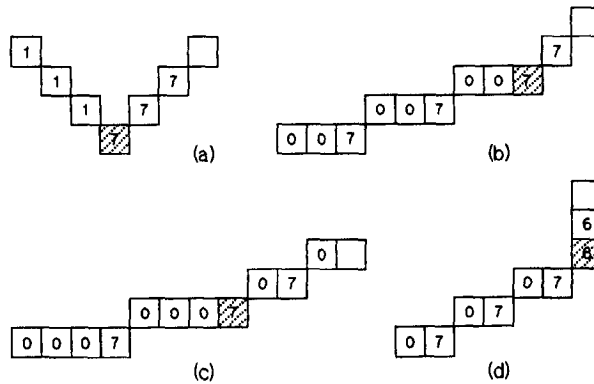


Fig. 4.6. Examples of the simple polygonal approximation algorithm. (a) A and B differ by more than 1. (b) B is repeated. (c) The run-length of A changes. (d) A third chain code is encountered.

#### 4.4. A COMBINATION OF POLYGONAL APPROXIMATIONS

The edge extractor of chapter 3 poses a particularly difficult problem to the polygonal approximation algorithm; the variability of edge strengths around the outline can result in a rather jagged edge. Feature extraction depends on reliably locating concave regions on the outlines and would be confused by the presence of spurious concavities. Therefore, the kinks must be removed by polygonal approximation. The simple algorithm of section 4.3 is by far the fastest polygonal approximation algorithm available. Unfortunately, due to its high accuracy it does not remove many of the kinks in the outline.

The speed of the algorithm of section 4.3 is such that it can be used as a first step in approximation, the result being reapproximated by a more complex algorithm. One possibility is to use the integer version of the Sklansky and Gonzalez algorithm described in section 4.2. This is unsatisfactory however, since most of the arc curvature information is lost.

An example of a noisy outline is shown in Fig. 4.7(a) with the result of the fast approximation in Fig. 4.7(b) and the reapproximation in Fig. 4.7(c). It can be seen that the kinks have been eliminated. The arc curvature problem is clearly visible however, in that it is difficult to tell whether the lines should be regarded as straight or curved. Such a loss of information can result in unnecessary ambiguities at a later stage, for instance between the pairs O/D and (/.

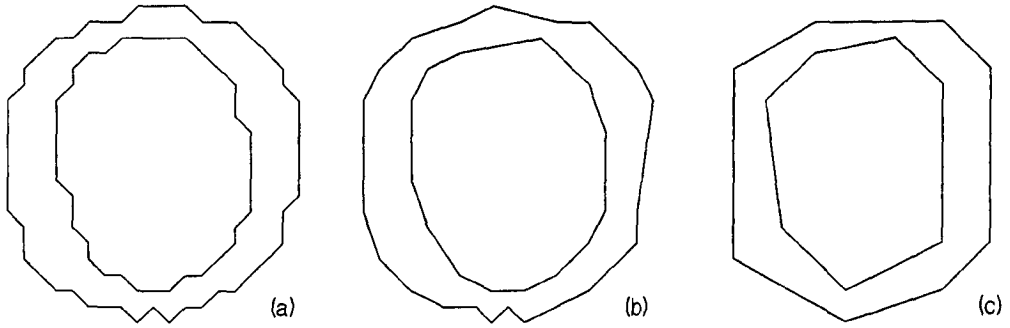


Fig. 4.7. An example of two stage approximation. (a) An original noisy outline. (b) The result of the simple approximation of section 4.3. (c) Reapproximation by the algorithm of section 4.2.

In order to specifically iron out kinks, while not applying a further approximation to smooth arcs, the reapproximation algorithm is only applied to certain parts of the polygon. Points that are already smooth remain unchanged, and reapproximation is applied to the remainder.

A smooth point is defined as being a point which is at the centre of three consecutive convex or concave points. Points satisfying this definition are illustrated in Fig. 4.8(a). Any concavity which contains three consecutive concave points is almost certain to be of significance to the feature extraction process, therefore it should remain. Similarly, any consecutive set of three or more convex points will remain. If a line which was originally straight has not already been reduced to a single line, then the points in the approximation would, in all probability, be alternately convex and concave. Such points would be subject to reapproximation. Smooth convex points therefore, are likely to be on arcs, and it is beneficial to retain them all to maintain the difference between straight lines and curves. The result of the selective reapproximation is in Fig. 4.8(b).

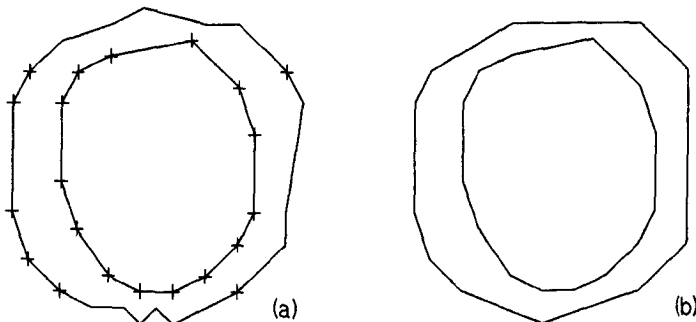


Fig. 4.8. (a) The polygon of Fig. 4.7(b) with fixed points marked "+". (b) The final result of selective reapproximation.

## 5. FEATURE EXTRACTION

This chapter describes the features that are used in character recognition and explains how they are extracted from the outline polygons. The chapter commences by describing some fundamental work from which the present features were derived.

### 5.1. THE WORK OF SHILLMAN et. al.

In section 1.5.2, it was stated that OCR by feature extraction is not sufficiently accurate to satisfy the demands of the market. The reason given is that there is a lack of methodology in the way in which the features are chosen. Between 1974 and 1977, a PhD. thesis<sup>(34)</sup> and several papers<sup>(35-41)</sup> were published by members of a group working on the psychological basis of character recognition. That work forms the foundation of the feature extraction method described in this chapter. This section presents a concise description of the key aspects of this work.

Shillman considers the recognition of isolated handprinted characters and asserts that this is a fundamentally different problem to the recognition of machine printed characters. This is reasonable in the sense that matrix matching algorithms will fail in the recognition of handprint, as they fail with multiple fonts. Conversely however, any algorithm which attempts to recognize handprint must also cope with machine printed characters, since the variations in handprinting are far greater than between machine fonts.

#### **5.1.1. AMBIGUITY IS THE KEY TO RECOGNITION**

The fundamental theme is that ambiguity is the key to recognition. Given that there exists an ideal set of features which can be derived from a physical character shape, there is a feature space into which physical shapes can be transformed. In the feature space, different physical versions of the same character form tight clusters. An ambiguous character is one that does not lie close to the centre of one of the clusters; because it is near to two or more cluster centres, it could be either character. It is argued that this is how humans recognize isolated handwritten characters, and that if machines had the same features, then they could be similarly accurate.

An intensive study was made of a collection of deliberately created ambiguous capital letters. As a result of this study, three different levels of

feature were identified. These are: *physical*, *perceptual* and *functional*. Because of these different levels, the term *attribute* is used by Shillman, where others use the term *feature*. This enables use of the word feature to describe physical features without confusion.

Fig. 5.1 defines the difference between these levels for the attribute *leg*. A *physical* leg will be found at pixel level by an OCR machine, but will be missed by the human observer. A *perceptual* leg is large enough to be seen, but too small to be considered significant in changing the character. A *functional* leg is clear enough to tell the difference between a "Y" and a "V"; it is the presence or lack of the functional leg that must be determined in order to recognize the character correctly. The state of transition between the functional leg being present and absent gives a clue as to the rule needed to test its presence.

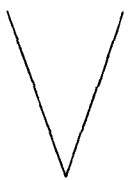
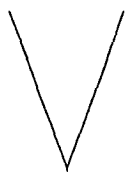
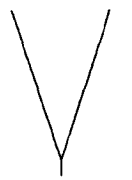

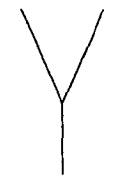
LETTER SHAPE					
PHYSICAL LEG	NO	YES	YES	YES	YES
PERCEPTUAL LEG	NO	NO	YES	YES	YES
FUNCTIONAL LEG	NO	NO	NO	?	YES
LETTER LABEL	V	V	V	?	Y

Fig. 5.1. The levels of the attribute leg.

Twelve functional attributes were derived by studying the collection of ambiguous characters and concentrating on the attribute in transition. The functional attributes themselves are useless without rules for determining their presence from physical measurements which can be made on the character. Much of the published work<sup>(34,36-39)</sup> describes various psychological tests that were used to find some of these rules. Details of the tests will not be repeated here, since it is the results that are most important to the present discussion. The set of functional attributes will be described in section 5.1.2.

The only functional attributes for which there are published physical to functional rules are closure and leg. A later paper<sup>(40)</sup> discusses results for a specific ambiguity between "U" and "V". As stated by Shillman<sup>(34)</sup>, the given set of attributes is not necessarily complete, unique or minimal. Another problem

with the functional attributes is that the physical to functional rules are extremely complex because they change according to context.

Fig. 5.2(a) is an ambiguous shape that could be either a "V" or a "Y". Fig. 5.2(b) and (c) show the same shape in two different situations that tend to bias the decision in opposite directions. In Fig. 5.2(b) the shape is more like a "Y" because the adjacent "P" has a similar compression of the bottom half. The bottom is extended in Fig. 5.2(c) making the shape more like a "V".

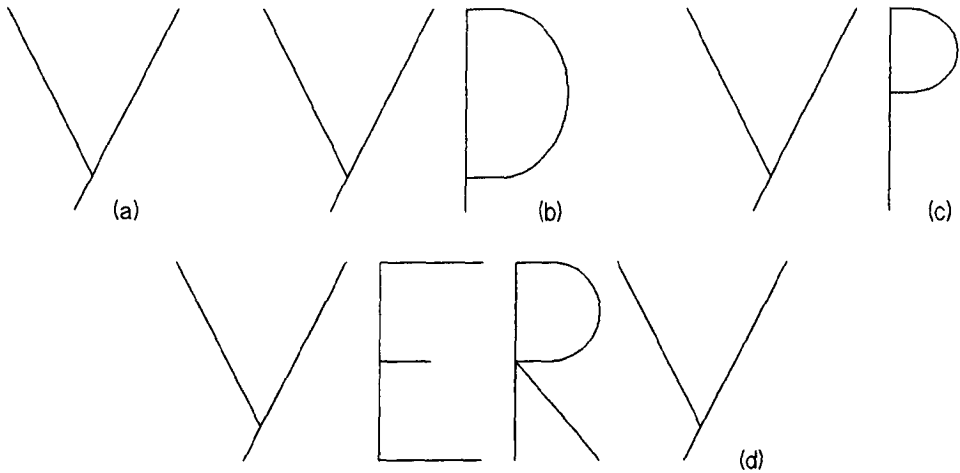


Fig. 5.2. (a) An ambiguous shape. (b) The shape of the "P" makes a "Y" more likely. (c) The shape of the "P" makes a "V" more likely. (d) Lexical context has a strong effect on the ambiguity threshold.

Word level context is demonstrated in Fig. 5.2(d). Since the word "VERY" is more likely than "YERY" or "VERV", the shape is read as a "V" at the beginning of the word and as a "Y" at the end. Hence there are at least two different levels of context operating: *physical*, where the shape of surrounding characters can change the rules, and *lexical*, where impossible characters can be ruled out.

The inevitable conclusion is that if ambiguity is the key to recognition, context provides the resolution of ambiguities. Consequently, the physical to functional rules cannot be regarded as exact.

### 5.1.2. SHILLMAN'S FUNCTIONAL ATTRIBUTES

The functional attributes are each given a set of modifiers, i.e. sub-attributes, which indicate their location, orientation, segmentation and concatenation. Different attributes have different sets of permitted modifiers. For example, the functional attribute leg may be located at the left, right or

middle of the character and be ascending, descending or horizontal. The modifiers are also functional and therefore have no exact physical rules to test them. For this reason the modifiers will not be detailed for any of the attributes.

Physical examples of the twelve functional attributes are shown in Figs. 5.3 to 5.6 and are described below. Note that these are only physical examples and that exact physical to functional rules are specified only for leg and closure.

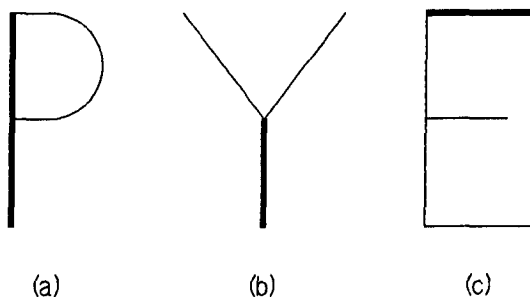


Fig. 5.3. Line-like attributes. (a) Shaft. (b) Leg. (c) Arm.

Three line-like features are shown in Fig. 5.3. A *shaft*, in Fig. 5.3(a), is a straight line segment which covers the entire height or width of the character, depending on its orientation. A *leg*, in Fig. 5.3(b), is a line with one end connected to the lower part of a character, and the other end unconnected. An *arm*, in Fig. 5.3(c), is the same as a leg except that it is connected to the upper part of a character. Clearly, arm and leg could be combined into a single attribute.

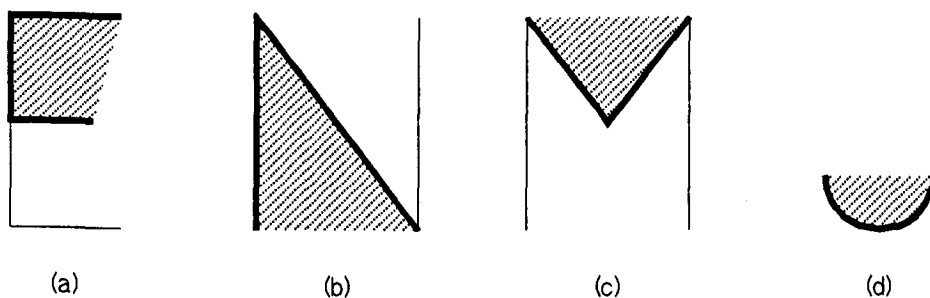


Fig. 5.4. Concave attributes. (a) Bay. (b) Inlet. (c) Notch. (d) Hook.

Bay, inlet, notch and hook are all attributes associated with concave parts of the character. Bay, inlet and notch, shown shaded in Fig. 5.4(a), (b) and (c) respectively, are differentiated by the number of connected ends. A *bay* is a concavity with both ends unconnected. It may be attached somewhere along the inside however. An *inlet* has one end attached to the rest of the character, and a



*notch* has both ends connected. A *hook* is a small inlet in that it is connected at one end only and occupies less than one half of the height of the character. The main purpose of a hook is to distinguish the pairs I/J and C/G.

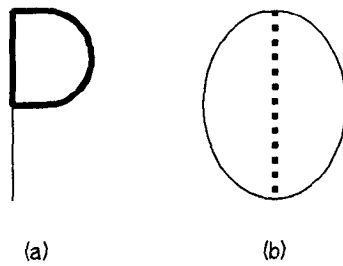


Fig. 5.5 Miscellaneous features. (a) Closure. (b) Symmetry.

Fig. 5.5(a) illustrates a closure. A *closure* is a region of the background that is, functionally speaking, completely surrounded by the character. As with the other attributes, the figure is a physical example only, and in some cases, a character may be deliberately printed without physical closure where a functional closure is intended.

Fig. 5.5(b) gives an example of *symmetry*. This is a feature that is used in only a small fraction of the upper case alphabet, mainly to distinguish "O" and "D".

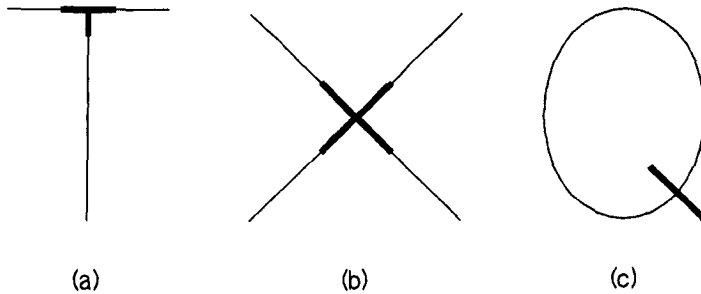


Fig. 5.6. Junction type features. (a) Weld. (b) Crossing. (c) Marker.

Fig. 5.6(a) shows an example of a *weld*. This is best thought of as the end of one straight line joining onto another. A *crossing*, as in Fig. 5.6(b), is simply two strokes which cross one another. A *marker*, illustrated in Fig. 5.6(c) is used only to distinguish "O" and "Q". There is little reason why crossing and/or weld could not be used for the purpose, except that the variability of the markers used on "Q" is extraordinary.

### 5.1.3. THE PHYSICAL TO FUNCTIONAL RULES

As stated in section 5.1.1, exact physical to functional rules have been published for only two functional attributes. These will now be given. The rule for leg of the type illustrated in Fig. 5.7(a) is given by:

A leg is present if  $m/l > 0.20$ , otherwise, no leg is present.

This value is useful only if the angle at the "V" is between  $42^\circ$  and  $90^\circ$ . For values below  $42^\circ$ , the threshold value for  $m/l$  increases, due to the fact that it becomes difficult to visually locate the exact location of the intersection.

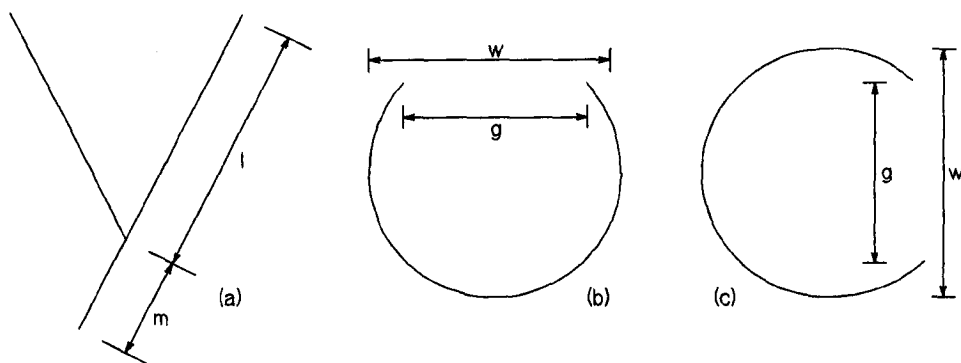


Fig. 5.7. Physical to functional rules. (a) An example of a leg. (b) An example of a closure open at the top. (c) An example of a closure open at the right.

The rule for closure is more complex, since it depends on the orientation of the opening and the shape of the closure. For a circular character with a gap at the top, as in Fig. 5.7(b), the rule is given by:

The gap is functionally closed if  $g/w < 0.56$

This rule could be used in the case of an ambiguity between "O" and "U". If the ambiguity is between "O" and "C", i.e. the gap is centred at the right, as in Fig. 5.7(c), the rule becomes:

The gap is functionally closed if  $g/w < 0.23$ .

No result is given for a gap centred on the left, but presumably, since there is no valid character with such a shape, the threshold is lower still.

In conclusion, the physical to functional rules are, in reality, far more complex than might at first be imagined. This complexity provides a good explanation for the failure of past attempts at feature extraction to attain high accuracy.

## 5.2. A MODIFIED SET OF FUNCTIONAL ATTRIBUTES

The functional attributes of section 5.1.2 are defined in terms of the strokes that make up each character. The thinning process required to extract such strokes from digitized character images is computationally demanding however, and should be avoided if at all possible. On closer inspection of the attributes, it can be seen that most of them can be determined from the outline of a character with equal accuracy. The attributes were derived in consideration of the 26 upper case letters only. In order to extend the character set to say, the ASCII standard, additional attributes may be necessary.

### 5.2.1. SHILLMAN'S ATTRIBUTES APPLIED TO OUTLINES

A straight line in the skeleton cannot be detected by simply locating a straight part of the outline, because in certain circumstances a line may be duplicated by the outline, and in others it may not. In any given character however, the number and type of straight segments will be almost constant. Fig. 5.8 illustrates the lines which may be detected on outlines of the example characters from Fig. 5.3.

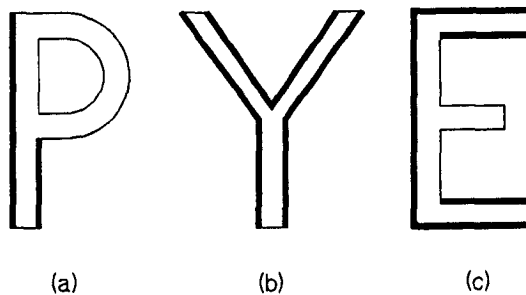


Fig. 5.8. Straight segments on the outlines of the characters in Fig. 5.3.

Since the polygon is constructed from straight line segments, there needs to be some condition which can distinguish a straight line from an arc. One possibility is to specify a minimum length relative to the size of the character. Another is to assume that lines must end in a significant bend. In any case, the attributes shaft, arm and leg are not simple to locate, but it is possible to detect less specific instances of line-like objects.

Bay, inlet, notch and hook can all be detected on an outline polygon, except that they can no longer be distinguished. This is because the concept of the end of a line is difficult to define sufficiently well to make it easily detected. The problem is aggravated by the presence of serifs on line endings. Fig. 5.9

demonstrates the similarity between the concavities of Fig. 5.4 when applied to outlines.

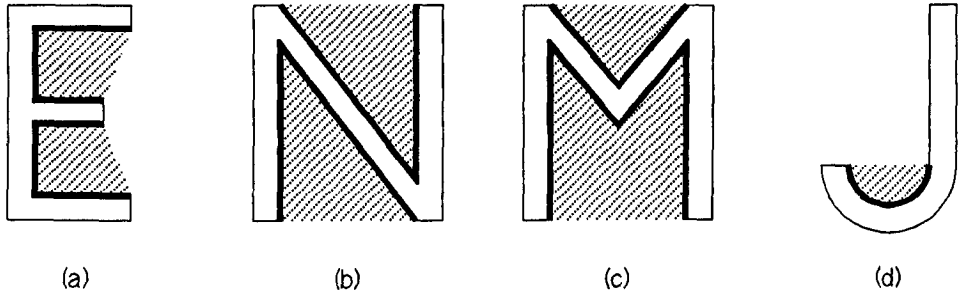


Fig. 5.9. Concave regions on the outlines of the characters from Fig. 5.4.

The detection of physical closure on outlines is trivial, since each closure is indicated by the presence of an additional polygon. Each concavity must be tested for functional closure using a physical to functional rule similar to those defined in section 5.1.3.

Symmetry is no more difficult to detect on outlines than it is on stick figures. In both cases, the difficulty is provided by italicised characters where the axis of symmetry is sheared in the horizontal direction.

The attributes weld, crossing and marker are extremely difficult to locate on outlines, since they are features of the interior, rather than the outlines themselves. It is apparent however, that these attributes all result in small concavities in the outline. If the concept of a bay is extended to include such concavities, weld, crossing and marker become redundant. Fig. 5.10 illustrates how the examples in Fig. 5.6 produce bays in the outline.

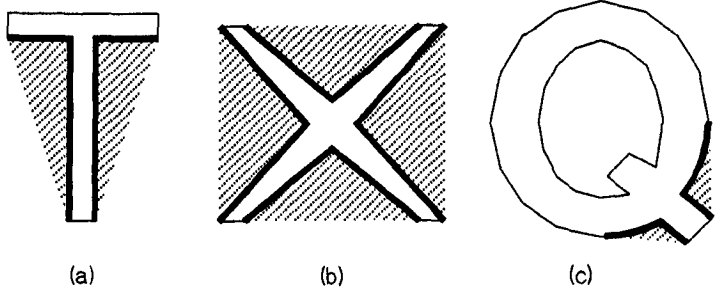


Fig. 5.10. Replacing junction attributes with bays. (a) Weld. (b) Crossing. (c) Marker.

### 5.2.2. A REDUCED SET OF ATTRIBUTES

The original set of twelve functional attributes can be reduced to only four generic attributes when processing character outlines rather than stick figures. Consideration of the remaining 70 characters in the ASCII set requires the addition of one more attribute. The reduced set of five functional attributes will now be described in order of decreasing importance for recognition:

- (a) **CONCAVITY.** A *concavity* is defined to be *any* concave part of the outline. This definition includes bay, inlet, notch and hook described by Shillman, plus the small concavities which result from crossing, weld and marker. All the shaded concave regions in Fig. 5.9 and Fig. 5.10 are concavities by this definition.
- (b) **CLOSURE.** Functional closure retains the definition used by Shillman.
- (c) **LINE.** A *line* is a straight part of the outline which results from a shaft, arm or leg.
- (d) **AXIS.** An *axis* is defined only for characters which do not have a concavity or a closure. The axis indicates the direction of the major axis of the object and measures the ratio of the lengths of the major and minor axes. It resolves ambiguities within the ASCII character set which are not covered by Shillman's attributes, for instance between a dot and a straight quote or comma.
- (e) **SYMMETRY.** Symmetry can be measured in either a vertical or a horizontal axis.

### 5.3. EXTRACTION OF THE FEATURES FROM OUTLINES

This section describes in detail how the features are extracted from a polygonal outline approximation and the numerical representation of each. Before going into detail, a concise notation for points and vectors on the outline will be described. This notation is also used throughout chapter 7.

The object to be thinned consists of one or more closed loops, formed by a total of  $n$  points with integer coordinates. If the  $n$  points are connected in sequence by straight line segments, and the last point is joined back to the first, then a left oriented closed loop is formed; i.e. the interior of the character is always on the left when moving from a point to its successor. Thus, the outside

loop is stored in anticlockwise sequence and holes are stored in clockwise sequence.

$P_i$  is the coordinate pair  $(x_i, y_i)$  of the  $i$ th point.

$P_{i+1}$  represents the coordinate of the *successor point* to  $P_i$ . Note that this is the next point around the closed loop containing  $P_i$  and not necessarily the  $(i+1)$ th point.

$P_{i-1}$  similarly is the *predecessor point* to  $P_i$ .

$P_i P_j$  is a vector from the point  $P_i$  to the point  $P_j$ .  $P_i$  and  $P_j$  may be on different closed loops.

$P_{i+}$  is an abbreviation for  $P_i P_{i+1}$ .

$P_{i-}$  is an abbreviation for  $P_i P_{i-1}$ .

$a \cdot b = a_x b_x + a_y b_y$  is a scalar product of two vectors.

$a \times b = a_x b_y - a_y b_x$  is a cross product of two vectors, as defined in section 4.2.1. The result is a scalar value.

$P_i$  Is *convex* if  $P_{i+} \times P_{i-} \geq 0$ , otherwise it is *concave*.

$|P_i P_j|^2 = P_i P_j \cdot P_i P_j = (x_j - x_i)^2 + (y_j - y_i)^2$  is the squared Euclidean length of the vector  $P_i P_j$ , i.e. the distance between  $P_i$  and  $P_j$ . In the remainder of this thesis, the terms *distance*<sup>2</sup>, *length*<sup>2</sup> etc. will refer to this measure.

$|P_i P_j|$  is the square root of  $|P_i P_j|^2$ . This value cannot be expressed exactly using integer arithmetic and is therefore used less frequently than the *distance*<sup>2</sup>.

### 5.3.1. CONCAVITIES

A concavity must contain at least one concave point. Extraction of a concavity therefore commences with locating one or more consecutive concave points. Fig. 5.11(a) shows such a sequence of points.  $P_i$  and  $P_j$  represent the *ends* of the concavity. The concavity in Fig. 5.11(a) is expanded, i.e.  $P_j$  is advanced, to include some of the convex points while

$$P_{j+} \times P_i P_j > 0$$

which is true while  $P_i P_j$  passes entirely outside the object. Expansion stops if at any step the ratio

$$\frac{A}{|P_i P_j|^2}$$

is reduced when compared with the current value, where  $A$  is the area enclosed by the concavity and the line  $P_iP_j$ .  $P_i$  may also be moved in a similar way. Fig. 5.11(b) shows the final position of  $P_i$  and  $P_j$ .

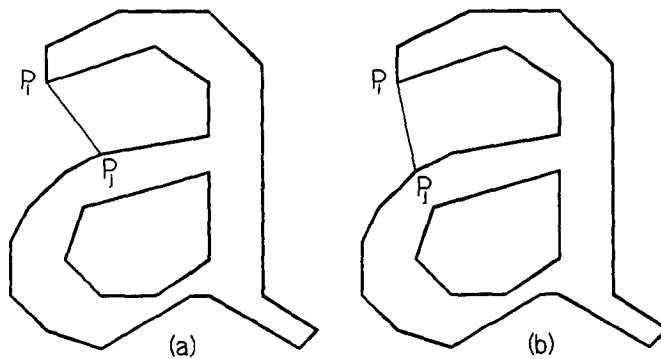


Fig. 5.11. (a) An example of a concavity. (b) The position of the ends after the concavity is expanded.

An additional constraint is placed on the ends of the concavity for when the anticlockwise angle between  $P_iP_j$  and  $P_{i+}$  or between  $P_{j-}$  and  $P_jP_i$  is more than  $180^\circ$ . The opposite end to the one with the large angle is moved inwards, while the contraction does not reduce the angle below  $180^\circ$ . Thus the concavity in Fig. 5.12(a) will be contracted to Fig. 5.12(b).

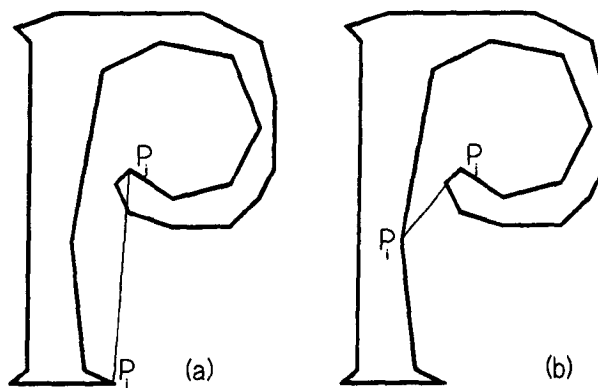


Fig. 5.12. (a) A concavity with a large angle at  $P_j$ . (b) Contracting the concavity to reduce the angle.

Once the ends of the concavity have been found, a numerical representation is calculated, which will be used in the recognition process. Firstly, if the total turn angle in the concavity is more than  $180^\circ$ , one of the ends may be moved slightly to obtain an accurate representation of the concavity direction.

If the perpendicular from  $P_j$  to the line  $P_iP_{i+1}$  intersects between  $P_i$  and  $P_{i+1}$ , then  $P_j$  is moved to the point of intersection. Similarly, if the perpendicular

from  $P_i$  to the line  $P_jP_{j-1}$  intersects between  $P_j$  and  $P_{j-1}$ , then  $P_j$  is moved to the point of intersection. This slight adjustment of the ends ensures that the vector  $P_iP_j$  gives a true indication of the concavity direction and is not confused by long lines in the the polygonal approximation. An example of a situation where this adjustment is beneficial is given in Fig. 5.13.

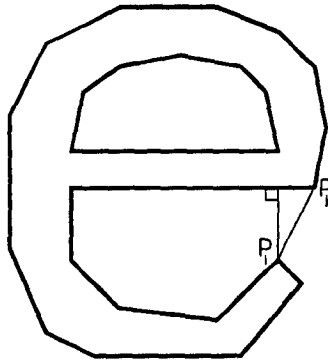


Fig. 5.13. An example of the final concavity end adjustment.

The numerical representation of a concavity contains the following components:

- (a) The normalised relative coordinates of the centroid of the concavity. The  $(x,y)$  coordinates of the centroid are calculated relative to the bottom left corner of the minimal upright rectangular bounding box of the character. The relative coordinates are then normalised independently to the range  $[0,63]$  such that  $(63,63)$  is the top right corner of the bounding box.
- (b) Let  $c$  be the vector from the midpoint of  $P_iP_j$  to the centroid of the concavity. If  $|c|^2 < |P_iP_j|^2$ , then the direction of  $P_iP_j$  is quantized to the range  $[0,63]$ . Otherwise the direction of the concavity is given by the direction of  $c$  rotated clockwise through  $90^\circ$ . Direction is quantized by dividing a circle into 64 equal sectors which are numbered 0 to 63. The number of the sector into which a vector falls gives the quantized direction.
- (c) The *depth* of a concavity is defined to be the maximum perpendicular distance of any point in the concavity between  $P_i$  and  $P_j$  from the line  $P_iP_j$ . A measure of the size of the concavity is given by the ratio of the depth to the longest side of the rectangular bounding box of the character. This measure is also scaled to the range  $[0,63]$ .



- (d) A measure of the *shape* of the concavity is given by the ratio of the perpendicular distance of the centroid from the line  $P_iP_j$  to the depth of the concavity. The shape ratio is normalised to the usual [0,63] range.

Values (a) and (b) above are adequate for discrimination of the majority of the ASCII character set. The measures of depth and shape are necessary to distinguish characters such as { C, <, (, [ }.

### 5.3.2. CLOSURES

A closure is generated for each physical closure. As each concavity is generated, it is tested for functional closure. The test depends on the direction. For a right-facing concavity, experimentation has revealed that the threshold value stated by Shillman, given in section 5.1.3, is too large. In order to prevent certain occurrences of "G" from being functionally closed, the threshold has been reduced from  $g/w = 0.23$  to  $g/w = 0.17$ , where  $g = |P_iP_j|$  and  $w$  is the maximum width of the concavity in the same direction as  $P_iP_j$ .

Different values are used for other directions. For a downward or left facing concavity, the threshold is currently set at 0.29. For an upward facing concavity, the threshold is set at 0.38. For all directions, an additional constraint has been added:

$$\frac{gd^2}{wA} < 0.45,$$

where  $d$  is the depth of the concavity as described above, and  $A$  is the area of the concavity. This constraint effectively reduces the  $g/w$  threshold for deep concavities, making it less likely that characters such as "e" are closed.

The following values form the numerical representation of a closure:

- (a) The normalised relative coordinates of the centroid of the closed region. The calculation of the coordinates of the centroid is identical to the method used for concavities and is described in section 5.3.6.
- (b) The scaled area of the closure as a fraction of the area of the rectangular bounding box of the character. The fraction is scaled to the usual [0,63] range.

### 5.3.3 AXES

The axis feature is generated only if the character has no concavity or closure. Rather than perform a squared order calculation to find the greatest distance between any pair of outline points, a linear algorithm is used.

Firstly, the centroid  $C$  of the object is calculated. The point  $P_i$  on the outline is found such that  $|P_i C|^2$  is a maximum. The maximum perpendicular distance of any point on the outline from the line  $P_i C$  is calculated independently for each side of the line. The results are summed to give the width of the object. Fig. 5.14(a) illustrates how the width of the object is calculated.

The direction of the major axis of the object is given by the vector  $P_i C$ , unless a smaller width result is obtained by moving  $P_i$  to the midpoint of  $P_{i+}$  or the midpoint of  $P_{i-}$ . Moving  $P_i$  to one of the adjacent midpoints provides an adjustment of the direction in the case of the object in Fig. 5.14(b), where the new position of  $P_i$  is shown as  $P_i'$ .

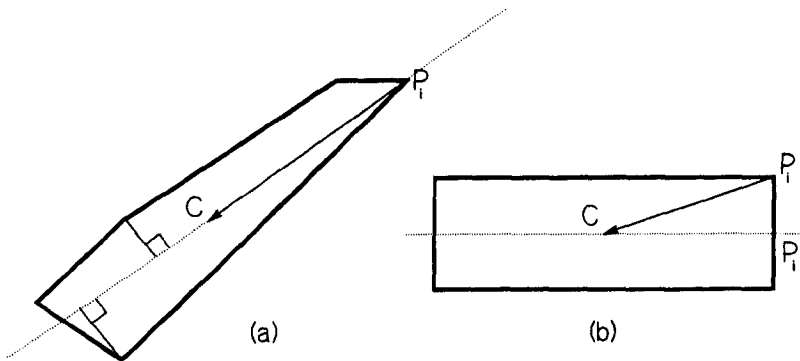


Fig. 5.14. (a) Calculating the minor axis. (b) Adjusting the major axis.

Having established the direction of the major axis, the length is obtained by applying the same width calculation to a vector perpendicular to  $P_i C$ .

The numerical representation of an axis consists of the following items:

- (a) The ratio of the area of the object to the length<sup>2</sup> of the major axis. The width to length ratio is thus calculated without requiring a square root, with the additional advantage that the result is less subject to any inaccuracy which could be caused by the polygonal approximation. The ratio is scaled to the usual [0,63] range.

- (b) The direction of the major axis is quantized to the [0,63] range. Unlike the other features which include a direction, the axis has no sense. Therefore it may differ by 32 units (modulo 64) from another in the same direction.

The direction is clearly meaningless when the width to length ratio is high, (i.e. close to 1,) since a dot, for instance, can have no direction.

#### 5.3.4. LINES

Lines are presently used only when an unknown character closely matches two of the standard templates, as measured with concavities, closures and axes. In the detection of lines there exists a trade-off between reliably locating all useful lines and finding false lines in approximations to curves. The present approach is by no means perfect, but it is reasonably successful.

A line is currently defined to consist of one or more segments of the polygonal approximation which satisfy a minimum length condition. If the line consists of more than one segment, the maximum perpendicular deviation of any point along the line, from the straight line joining the ends, must be below a certain limit. A long and sufficiently straight line must also have a significant bend near both ends before it is regarded as a line feature.

The conditions serve to reject lines which are part of a curve. Fig. 5.15(a) illustrates some examples of possible lines which do not meet all the criteria. The bold parts of the outline in Fig. 5.15(b) indicate accepted lines.

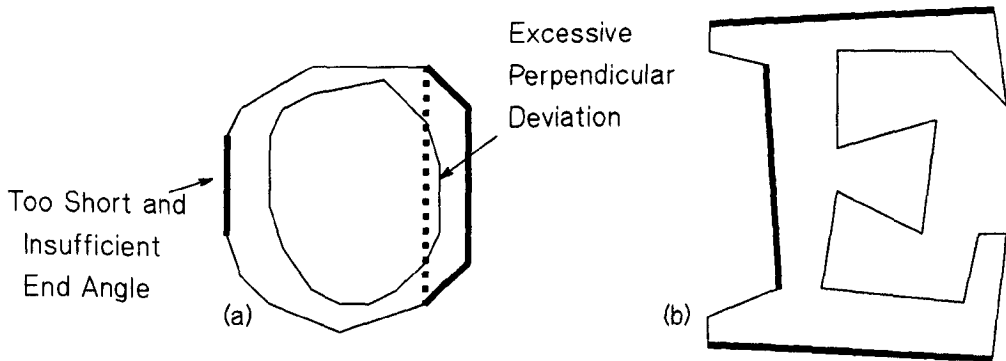


Fig. 5.15. (a) Unacceptable lines. (b) Accepted lines.

A line is identified by the following numerical values:

- (a) The normalised relative position of the centre of the line.
- (b) The quantized direction of the line.

- (c) The scaled length of the line as a fraction of the size of the rectangular bounding box in the same direction as the line.

### 5.3.5. SYMMETRY

Symmetry is not at present used. It is a useful measure however, for discrimination of the following pairs of characters: C/G, j/), j/], T/1, ]/7, B/8, O/D. The main difficulty is to locate the axis of symmetry. For vertical text, the axis is simply a vertical line through the centre of the upright bounding box of the character. For italic text however, the axis is rotated slightly and the exact position is difficult to locate, especially with characters such as "O".

Both vertical and horizontal symmetry will be used in the recognition process. Vertical symmetry is measured by consideration of each point  $P_i$  on the outline. If the axis is not vertical, the coordinates of  $P_i$  are transformed back from the sheared coordinates to rectangular coordinates. The least distance<sup>2</sup> between the reflection of  $P_i$  in the axis and the transformed outline is measured. A measure of the vertical symmetry is given by the maximum such distance for all points on the outline.

Horizontal symmetry is measured similarly using a horizontal axis. The same transformed outline as was constructed for measurement of vertical symmetry is used. Fig. 5.16(a) illustrates the outermost outline of an italic "B". In Fig. 5.16(b), the outline has been inverse sheared back to the vertical, and the reflected points are shown as "+" joined by dotted lines. Some of the "+" marks are clearly a significant distance from any part of the solid outline.

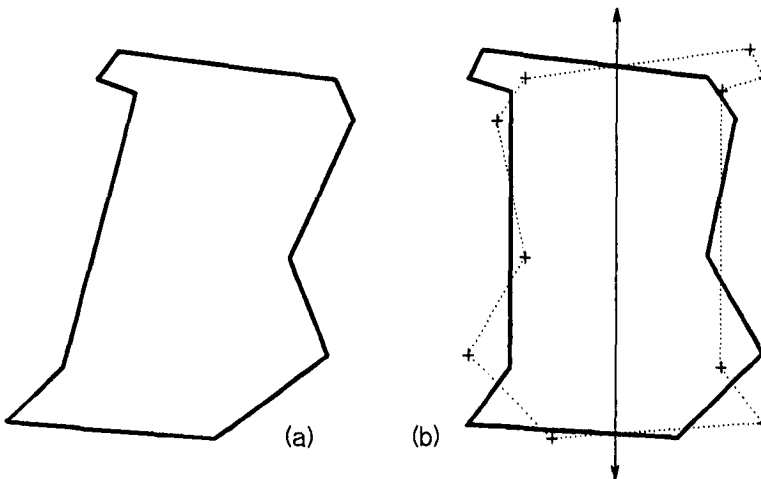


Fig. 5.16. (a) An italic "B". (b) The inverse sheared "B" with its reflection.

Location of the nearest outline point corresponding to all reflected points could conceivably consume squared order time. This can be avoided by commencing at a point where the axis intersects the outline and working in opposite directions simultaneously. If the object is to be at all symmetric, then the point nearest a reflected point must be in the locality of the current point on the opposite side of the object. Such an approach arguably reduces the time to being linear in the number of source outline points.

Measurement of symmetry will be applied only in the case of an unknown character matching a particular pair of templates. Only a single direction will be measured in most cases, according to the pair in question. The result is a distance<sup>2</sup> which must be scaled to a fraction of some measure of the size of the object, for instance, the area of the rectangular bounding box.

**5.3.6. CALCULATION OF AREA AND CENTROIDS**

A simple procedure is used to simultaneously calculate the area and centroid of a closed region. In Fig. 5.17, there is a closed region bounded by a polygonal approximation of  $n$  points,  $P_0, P_1, \dots, P_{n-1}$ . The closed region is divided into  $n-2$  triangles,  $T_1, T_2, \dots, T_{n-2}$ . Let the triangle  $T_i$  be bounded by the points  $P_0P_iP_{i+1}$ . Then the area  $A_i$  of triangle  $T_i$  is given by:

$$A_i = (P_0P_{i+1} \times P_0P_i)/2$$

Let the vector From  $P_0$  to the centroid of  $T_i$  be  $c_i$ . Then

$$c_i = (P_0P_i + P_0P_{i+1})/3.$$

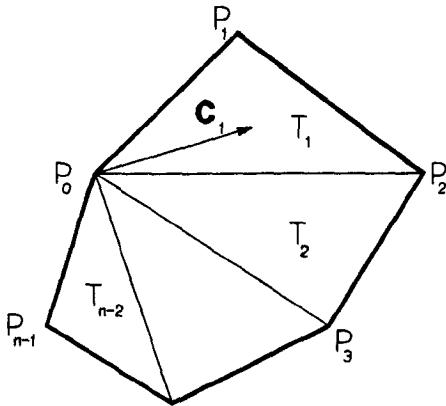


Fig. 5.17. Calculation of the centroid of a closed region.

The vector from  $P_0$  to the centroid of the entire region is given by:

$$\frac{\sum_{i=1}^{n-2} A_i c_i}{\sum_{i=1}^{n-2} A_i}$$

It can be shown that the above formulae for the total area and centroid are not affected by the object being concave, since the cross product yields negative results where necessary.

### 5.3.7. ORDERING OF FEATURES

The different types of features are regarded as being totally separate and therefore adjacency relations between a concavity and a line, for instance, are not recorded. This separation simplifies the recognition process and has been experimentally determined to actually improve recognition accuracy. There are two reasons why complex adjacency information reduces rather than increases the recognition rate:

- (a) Simple positional information is adequate for uniqueness in the ASCII character set. As an example, it is difficult to construct an outline which contains a closure at the top, a downwards facing concavity at the bottom and oblique strokes at the right and left without producing an "A". Removal of the adjacency information therefore does not reduce accuracy.
- (b) A line may be located within a concavity. Occasionally, one end of the line may extend beyond the limits of the concavity, thus changing the apparent ordering of the features. Using the adjacency information can therefore reduce accuracy.

Multiple occurrences of the same type of feature however, are ordered, since this also simplifies comparison. The features of each individual type are recorded in the order in which they are located on the outline. This necessitates a reliable definition of origin.

An obvious solution is to maximise a function of the coordinates of a point taken over all the outline. The function must be reasonably simple to calculate, and yet the contours must be such that no character has two distant points on the maximum contour. As a simple example, consider the function  $f(x,y)=y-x$ , which

selects the top left corner of the object. The character "+" however, has two distant points which lie on the maximal contour of  $f$ . Slight variations in size or aspect ratio may result in a dramatic shift in the location of the origin, which would cause different representations of the same character.

The problem is exemplified by Fig. 5.18. The dotted line indicates the maximum value of  $f(x,y)=y-x$  in each case. The "+" in Fig. 5.18(a) has its origin at the top, whereas the "+" in Fig. 5.18(b) has its origin at the left. The present function for selecting the origin is  $f(x,y)=4y-x$ . This function was chosen to allow a reasonable skew of the page, whilst selecting a reliable point on every ASCII character with the possible exception of "#".

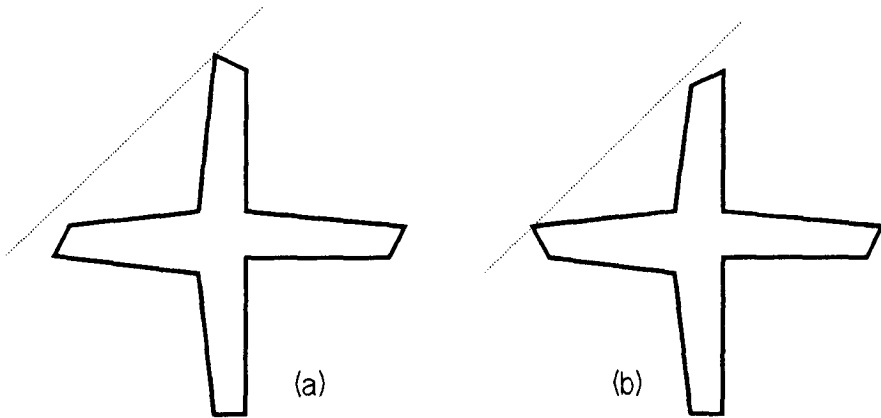


Fig. 5.18. (a) A "+" with the origin at the top. (b) A "+" with the origin at the left.

#### 5.4. GENERATION OF THE STANDARD TEMPLATES

The system requires a set of standard templates with which to compare an unknown character. The most appropriate way to generate the templates is by analysis of samples covering a wide range of fonts. It is intended that the completed system will recognize a very wide range of printed fonts without any training or intervention from the user. The learning process must therefore, provide a sufficiently wide range of fonts such that no further training is required.

Publishing companies provide books containing extensive selections of fonts. Another possible approach is to manually construct widely varying samples of each character to be recognized, by combining the possible variations. Examples of the possible extreme variations of a letter "E" are given in Fig. 5.19. Note that the variations shown must be combined to give a full range.

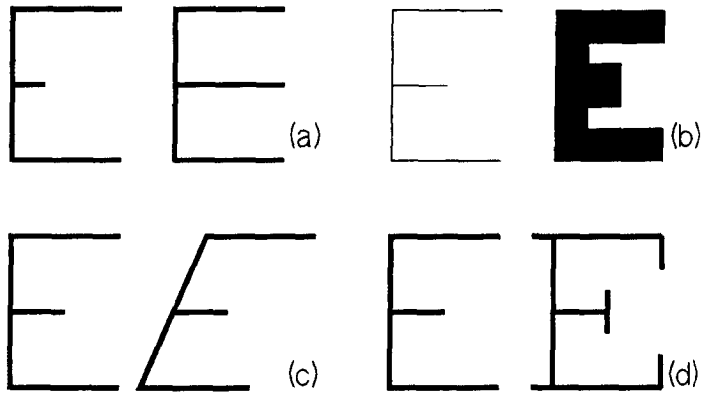


Fig. 5.19. Some of the possible variations in an "E". (a) Shape. (b) Line width. (c) Angle. (d) Embellishments.

#### 5.4.1. RATIONALE

The learning process involves generating the features for a set of character outlines with known ASCII codes. Several samples of each character in as wide a range of fonts as possible must be supplied, so as to learn the expected variation in the numerical representation of each feature.

The result of the learning process is a set of *character classes*. A character class represents a set of one or more characters which have an indistinguishable set of features. An example of a pair of characters which occupy the same class is C/c, where the only difference is one of size relative to neighbouring characters.

Conversely, one ASCII character may appear in several different guises, such as a/a. A more subtle variation occurs in punctuation marks. The comma, which appears as the same shape in a semi-colon, quote and double quote, has two fundamentally different shapes: straight as in " and curly as in '. Two different classes will result, each of which contains all the aforementioned characters.

In order to construct the standard templates automatically, it is necessary to detect the differences between two versions of the same character, and similarities between different characters. The process therefore includes the following steps:

- (a) Clustering. All the similar features from different samples of the same character are grouped together into clusters. The extreme feature values within each cluster give the limits on the acceptance range. The number of samples which contribute to a cluster indicates the importance of the corresponding feature in recognition.



- (b) Partitioning of classes. Those classes which do not have a minimum number of consistent features, i.e. those for which a cluster is generated by all samples, are partitioned.
- (c) Merging of ambiguities. Ambiguous classes are merged where there is some physical distinction between the characters which can be used at a later stage to distinguish them.

#### 5.4.2. CLUSTERING

A standard clustering algorithm<sup>(43,44)</sup> could be used to detect similar features in different samples of the same character. The nature of the data however, simplifies the clustering problem.

A sample character may have several features of the same type, each of which must lie in a separate cluster. If most samples of the same character contain largely the same features, as is the case if the class is not to be partitioned, then it is reasonably simple to match corresponding features and create new clusters for those which do not match.

The clustering algorithm copies the first sample character to the standard template. Each subsequent sample of the same character is compared with the standard template using a laxer version of the flexible matching algorithm described in section 5.5.1. In the comparison, only the most important numerical values are considered, for instance the position and direction of a concavity. A much greater tolerance of error is also included. For features which match a feature in the standard template, the acceptance limits are expanded. Those features which do not have a match in the standard template are inserted in the relevant position, using the order from the origin.

Fig. 5.20(a) shows an example of the first sample of a letter "H". The square indicates the limits of the standard [0,63] range on the (x,y) coordinates of the centroid of the concavities. The "+" marks indicate the positions of the two concavities in the first sample. Fig. 5.20(b) shows the final template for the concavities after a total of 7 samples have been analysed. The clusters on the left and right of the square are caused by serifs on the top and bottom of the "H". A dotted "H" is superimposed on Fig. 5.20(b) to show the relation between the character and the concavities.

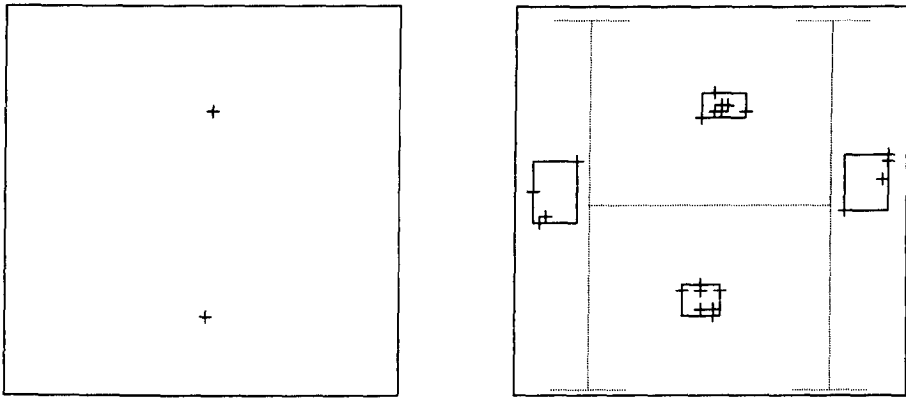


Fig. 5.20. (a) The concavities of an "H". (b) The final template for the class "H".

### 5.4.3. PARTITIONING CLASSES

A *consistent feature* is one which exists in all the sample characters which were used to construct a template. When all the samples of a character have been read and compacted by the clustering algorithm, a count is made of the consistent features. If the number is below a set limit, the class is partitioned. The limit is set individually for each character in a database to either one or two.

Partitioning is based on the modal feature. All samples which possess the modal feature are placed in one partition, and the remainder go in another. If the minimum number of consistent features for the character is two, then only those samples which possess the two most common features are placed in the first partition.

If the second partition does not contain the required number of consistent features, then that is sub-divided until all partitions satisfy the minimum consistent feature criterion. A partition can fail to satisfy the minimum consistent feature criterion if it contains a sample with less features than the minimum. There must clearly be an exception to the rule, to allow a partition to contain too few consistent features if there is no feature which is not consistent.

### 5.4.4. MERGING AMBIGUOUS CLASSES

The less ambiguity that exists between the final templates, the higher the accuracy of the final classification. In order to obtain a low ambiguity level, it is inevitable that some classes must contain more than one ASCII character. Merging is acceptable only if the characters can be distinguished by some physical measure which is unavailable at the feature extraction stage. Examples of such measurements are:

- (a) **Relative height.** Used to distinguish capital and lower case letters.
- (b) **Position on the text line.** Pairs such as **./'** and **-/\_** can only be discriminated by position on the line.
- (c) **Neighbouring marks.** The set { **!, ", %, =, ;, :, ?, i, j** } are all distinguished from other characters by the presence of neighbouring marks. These characters result in several classes, each containing a different set of characters.
- (d) **Context.** Certain characters appear identical in some fonts. The usual problem sets are {**O, 0**} and {**l, 1, I**}. The only guaranteed way to determine the correct character is by context. The classes can therefore be merged. Certain other pairs such as {**S, 5**} are difficult but not impossible to distinguish. Context will be helpful in making the correct decision in these cases, but it is not acceptable to merge the classes.

In order to automatically merge the correct characters into a reasonable set of standard templates, use is made of a knowledge base. The knowledge base consists of a sparse  $m$  by  $m$  array, where  $m$  is the number of characters in the known character set, 94 for ASCII. Entries are made where the system detects an ambiguity between a pair of character templates.

Each entry in the array indicates how an ambiguity between the pair of characters may be resolved. It may indicate that one of the physical measurements above may be used and that therefore the classes may be merged. Alternatively, an entry may nominate a specific test which may be performed to resolve an ambiguity. No such specific tests have yet been implemented, but a possibility is to investigate a particular feature such as a straight line or a corner for {**5, S**}, or a concavity for {**f, t**}.

#### **5.4.5. ASSIGNING WEIGHTS TO FEATURES**

When the final classes have been determined, a weight is assigned to each feature within each class. The weight determines the matching error which will be generated if the feature is absent in an unknown character.

The ideal way to generate the weights would be to try all reasonable combinations of values to minimise the ambiguity across the entire character set. This is obviously impractical, due to the enormous number of combinations of reasonable values. It would be possible however, to use an iterative evolutionary

process<sup>(45)</sup> to refine the weights randomly and maximise the distance between different classes. Such a process would identify those features which are important in distinguishing almost ambiguous characters, such as the closure in the pair (\*, #).

The present algorithm for assigning weights to features is simple but effective. For consistent features, the weight is inversely proportional to the number of consistent features in the class. A "C" therefore, with a single consistent feature, will have a high weight for the concavity. Conversely, "#" will have a low weight for each feature, since loss of any one, apart from the closure, is relatively insignificant and common. Features which occur in more than 75% of the samples have weights assigned similarly, but with a lower constant of proportionality. All remaining features are considered embellishments and are assigned zero weight.

## **5.5. RECOGNITION OF FEATURES**

Once the features have been extracted from an unknown character, an attempt is made to match the features against one of the standard class templates. One of the most important aspects of the recognition process is that it must be flexible. Different fonts cause variations in the features that are detected. It must be possible therefore to recognize a character which has one or two additional features or missing features.

Simultaneously, it is desirable to test the unknown character against as few templates as possible, in order to increase the recognition speed. Speed and flexibility are, to some extent, mutually exclusive.

### **5.5.1. FLEXIBLE COMPARISON**

As discussed in section 5.3.7, each individual type of feature is considered separately. For each type however, the features are ordered in the sequence in which they occur from the origin. The comparison task therefore is to match two ordered lists, each of which may contain elements which do not occur in the other. Fig. 5.21 illustrates the result of the flexible matching process. A list of test features  $\langle T_1, T_2, T_3, T_4 \rangle$  is to be matched against the list of standard features  $\langle S_1, S_2, S_3, S_4 \rangle$ . Arrows indicate matches between individual features.

A match error is generated for the entire list, which is the sum of individual match errors between the features connected by arrows, and errors generated for the missing and additional features.

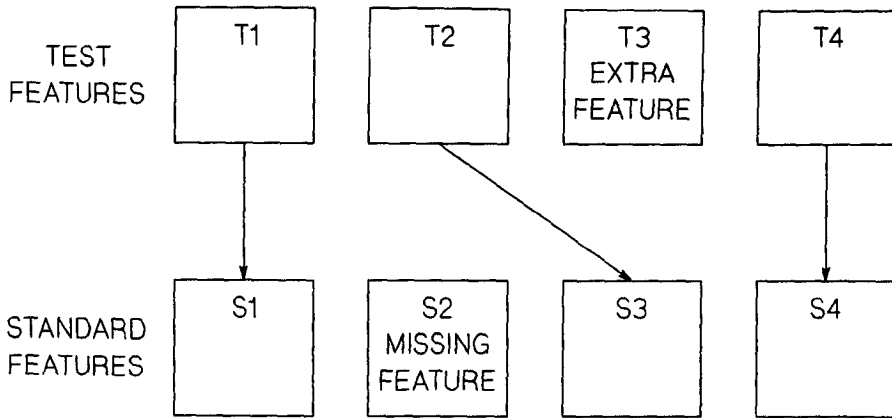


Fig. 5.21. The result of the flexible matching process.

#### 5.5.1.1. Matching Individual Features

A direct comparison between two single features involves testing all the elements of the numerical representation of the feature. Consider for example, a closure. Let the limits on the  $x$ -coordinate of the centroid of the closure be  $X_{min}$  and  $X_{max}$ . If the  $x$ -coordinate of the centroid of the closure in the test feature is  $X$ , then a match error,  $e$ , is generated as follows:

$$e = \begin{cases} |X - (X_{min} + X_{max})/2| & \text{If } X_{min} \leq X \leq X_{max}, \\ K(X_{min} - X) & \text{If } X_{min} - d < X < X_{min}, \\ K(X - X_{max}) & \text{If } X_{max} < X < X_{max} + d, \\ Kd & \text{If } X \geq X_{max} + d \text{ or } X \leq X_{min} - d, \end{cases}$$

where  $d$  is a maximum allowed deviation from the acceptance range, and  $K$  is a large constant.  $K$  is used to make out of range errors significantly greater than any possible within range error.

The match errors for all the elements of the feature are summed, with weights on each element. If the total is greater than  $M_{max}$ , it is set to  $M_{max}$ . This ensures that all failed matches have the same value, whatever the type of the feature. Let the match error between a test feature  $T_i$  and a standard feature  $S_j$  be denoted by  $M(T_i, S_j)$ .

This approach to matching features is similar to the one used by Yamamoto and Mori<sup>(7)</sup>, with the important difference that values which are well outside the acceptance range produce a fixed maximum error, rather than an infinite error. The fixed maximum provides flexibility by still allowing a match when small parts of a limited number of features are out of range.

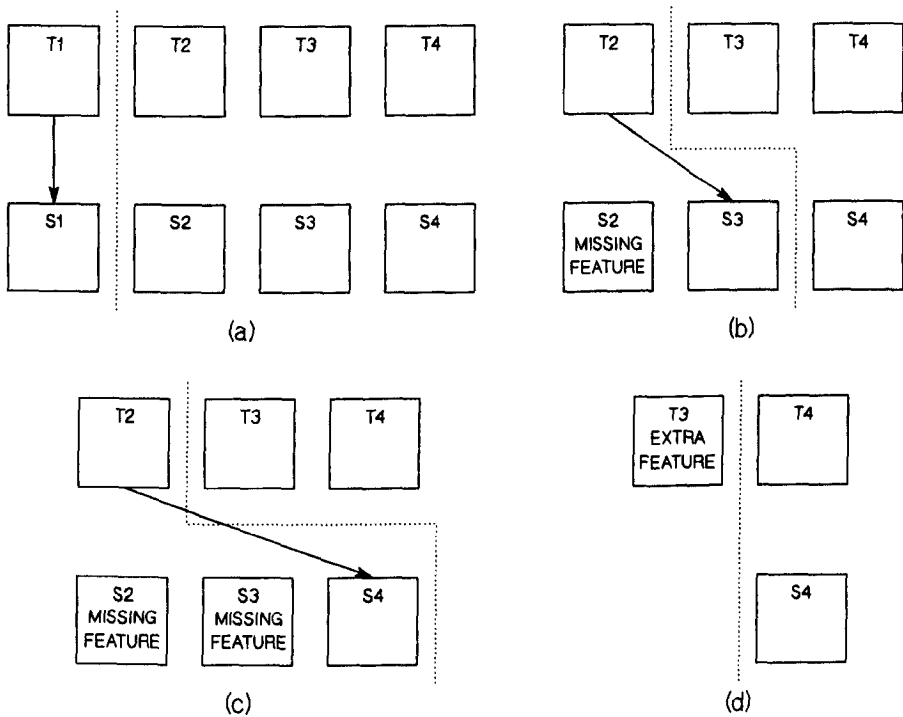
If the value is within the acceptance range, then a small match error is generated, rather than zero, indicating the distance from the centre of the range. This small error is to provide some difference between intrinsically ambiguous characters such as "u" and "v", where otherwise a character may have a match error of zero for both classes. The large value of K, presently 256, ensures that any out of range errors are significant in comparison.

**5.5.1.2. Matching Lists of Similar Features**

A recursive procedure finds the best match between two ordered lists of features by performing a depth-first search of the tree of all possible ordered matches. The first operation is illustrated in Fig. 5.22(a). If

$$M(T_1, S_1) < M_{max},$$

then the match rating for the entire string is equal to  $M(T_1, S_1)$  plus the best match rating for the remainder of the lists, generated by recursion.



**Fig. 5.22.** (a) Having matched the first pair of features, recursion evaluates the remainder. (b), (c) When testing against subsequent standard features, skipped features are regarded as missing. (d) The first test feature is skipped and assumed extra.

The first test feature is then matched against each of the remaining standard features in turn, summing the errors for each skipped standard feature. This process is illustrated at the second level of recursion by Fig. 5.22(b), where S<sub>2</sub> has

been skipped and  $T_2$  is matched against  $S_3$ , and in Fig. 5.22(c) where both  $S_2$  and  $S_3$  have been skipped. The error generated for skipping  $S_2$  is governed by its weight. Note that recursion only occurs when there is a match less than  $M_{\max}$ .

The final operation is to assume that the first test feature is an extra feature. This is skipped, generating a fixed level of error, and recursion is used to evaluate the match rating of the rest of the list. Fig. 5.22(d) illustrates skipping the first test feature at the third level of recursion.

The match rating for the list is defined to be the lowest result obtained from the depth-first search. An obvious reduction in match time is obtained by truncating the search where the accumulated error exceeds the lowest full depth error so far.

### 5.5.2. REDUCING THE SEARCH SIZE

Section 5.5.1 described the algorithm for matching an unknown character against a standard template. Testing each character against the entire set of templates would guarantee that all possible matches are detected. The time involved however, is unacceptably great. It is therefore necessary to reduce the number of standard templates tested, while ensuring that the correct match will be obtained.

The method adopted is to use a look-up table to select a small subset of the full character set. Each feature of a test character is used to locate those standard classes which include the same features.

For every possible x-coordinate in the usual [0,63] range, a bit array is constructed, with one bit per character class. The bit for a class is set if the class has one or more concavities which include the x-coordinate in their acceptance ranges. The acceptance range in this respect includes the maximum error allowance,  $d$ , made during comparison of two features in section 5.5.1.1.

A similar array of bit arrays is constructed for the possible y-coordinates and directions of a concavity. The concavities of an unknown character will be used to look-up bit arrays corresponding to the position and direction of the concavity. The three arrays are then logically ANDed, resulting in an array of bits indicating all the classes which will accept the concavity.

A similar operation on all the other features of the unknown character results in a set of bit arrays, one for each feature. The arrays are used to

construct a single array, which gives the number of ones in each bit position. If the unknown character has  $n$  features, and it is a perfect match with one of the standard templates, then the highest bit count will be  $n$ , in the position of the matching class. Comparison of the unknown character against all the classes which have a bit count of  $n$  will guarantee to find the correct class.

If the unknown character is not a perfect match to the relevant template, then it may have one or more extra features and/or missing features. Missing features will still result in a perfect response with the correct class, but an increased number of other classes will also give a perfect response, resulting in an increased search time.

If sufficient extra features are present, then it is possible that the correct template will not even be tested, since some other class may have more matching features, yielding a higher bit count. In this case, the correct class can still be found by reducing the target bit count when no classes are found with a sufficiently low match error.

In conclusion, the look-up tables can be used to obtain a significant reduction in the number of templates which need to be compared with an unknown character, without any danger of missing the correct class. The typical reduction obtained in practice is of the order of 80-85%. This figure would be considerably higher if there were fewer classes which have only one significant feature, i.e. when symmetry is added.

## 5.6. SUMMARY

The twelve features or attributes of Shillman<sup>(34)</sup> are, to some extent, psychologically tested, but they are based on stick figures. Consideration of how the attributes may be detected in character outlines reduces the number of different attributes. Those attributes which cannot be extracted directly from outlines may be inferred by the presence of other attributes.

A reduced set of five features is adequate for recognition of the entire ASCII character set. These features are: concavity, closure, line, axis and symmetry. Each of the above features can be detected in a time linearly related to the number of outline points in a character. The need to perform any kind of thinning operation is thus completely eliminated.

A training set of characters is used to produce the standard templates by a split and merge procedure. Firstly, classes are split where the same character has



disjoint sets of features, such as with "a" and "ɑ". Ambiguous characters are then merged into single classes where a simple operation may be used to distinguish them at a later stage of processing.

A flexible matching process is used to match unknown characters to class templates even when one or more features are absent, additional or out of the allowable acceptance range. A look-up table is used to reduce the number of templates against which an unknown character is tested.

## 6. A MORE GENERAL EDGE EXTRACTOR

The edge extractor described in chapter 3 is adequate for extracting text from documents which may contain any other kind of object, such as photographs and line graphics. For some applications however, a more general edge extractor may be required. For instance, there presently exist many thousands of engineering drawings which were produced before the invention of CAD (computer aided design). It would be highly beneficial to be able to convert these drawings to a form readable by current CAD systems. This chapter describes a more general edge extractor which is capable of extracting both text and arbitrary line or block graphics from a grey level image.

### 6.1. THE DEFICIENCIES OF THE SIMPLE EDGE EXTRACTOR

The edge extractor of chapter 3 makes some simplifying assumptions which enable it to be considerably faster than the edge extractor described here. These are all originated from the knowledge that only the text is required:

- (a) Nesting information can be obtained by applying a simple bucket sorting procedure to the rectangular bounding boxes of edges. This simplification is not applicable to line drawings.
- (b) An arbitrary maximum character size can be set, to reduce the amount of image which needs to be stored in memory during the extraction of any one outline. Clearly a continuous line can cover the entire page, whereas a character could be limited to occupying an area of say, one inch square.

In order to explain the problem of nesting more clearly, a *nesting tree* will now be defined.

#### 6.1.1. NESTING TREES

The output from the edge extractor is a tree structure, the shape of which corresponds directly to the nesting of edges within each other in the image. This representation as a tree is standard for this kind of problem.<sup>(46)</sup>

In Fig. 6.1(a) there is a document containing two characters, and a third within an outlined box. Fig. 6.1(b) illustrates the outlines which would be extracted from the page. Each outline has a numeric label.

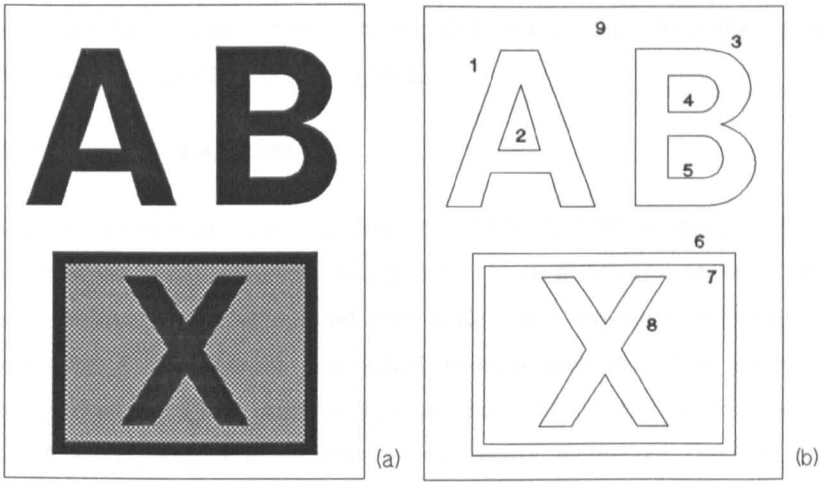


Fig. 6.1. (a) A simple page. (b) The outlines of (a).

Table 6.1 shows the nesting relation between the outlines in Fig. 6.1(b). For example, outline 1 is the outline of the letter A, it is inside outline 9 and contains outline 2.

OUTLINE	DESCRIPTION	OUTER EDGE	INNER EDGE
1	Outside of letter A	9	2
2	Inside of letter A	1	-
3	Outside of letter B	9	4,5
4	Top hole in letter B	3	-
5	Bottom hole in letter B	3	-
6	Outside of outline of box	9	7
7	Inside of outline of box	6	8
8	Outline of letter X	7	-
9	Outline of page	-	1,3,6

Table 6.1. The nesting relation between the edges in Fig. 6.1(b).

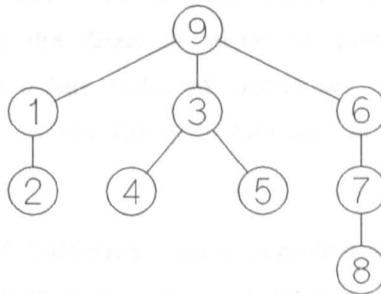


Fig. 6.2. The nesting tree of the page in Fig. 6.1(a).

The tree structure in Fig. 6.2 is derived from the table, commencing with the outline of the page. This becomes the root of the tree i.e. the top. All objects contained inside the root are its children and are placed immediately below it in

the tree. Each of the children is then taken in turn. The objects within each child are placed immediately below that child in the tree. The tree is thus built downwards until all the objects are recorded.

### 6.1.2. MEMORY LIMITATIONS

On an A4 page there are approximately 2400 X 3400 pixels at 300 pixels per inch. In order to construct the nesting tree, it is necessary to mark each pixel with a label unique to the associated tree node. It is possible to construct a page with more than  $2^{16}$  different individual objects and therefore each label will occupy more than 16 bits. Adding this to the 4 bits required to store the grey level of each pixel, and 4 more bits for the other necessary flags at each pixel, gives a result of more than 24 bits per image pixel. Thus, the total requirement for the image alone exceeds 24 Megabytes. Consequently, the image must be split into sections to be processed. The problem then becomes one of how to follow the edges of objects which are too large to fit into memory.

The image is split into buffers consisting of the full width of the page and a small number of scan lines. If the line tracker follows an object out of the buffer when the raster scan is more than half way through the buffer, the contents of the buffer from the start to the current raster scan position are discarded, the remainder is moved to the start of the buffer, and a new section is read in. If the raster scan was in the first half of the buffer, then the line tracker saves all the information about the line and puts it in a list of lines which ran out of the buffer. It then restarts at the beginning of the line and tracks in the opposite direction, i.e. beginning with chain code 0.

The new line is marked with the same label, because it is part of the same object, but an extra flag, the *direction*, marks all pixels as being part of a line followed anticlockwise or a line followed clockwise. Whenever a new buffer is read, tracing of the lines in the list of unfinished lines is continued before the raster scan restarts.

The introduction of buffering causes complications in relation to creating the nesting tree. Also, when a line now collides with pixels which form part of another line, it may have legitimately hit either the other end of the same edge or the end of a line belonging to another object.

## 6.2. CREATING THE NESTING TREE

The use of a unique label for each line obviates the need to mark a line current while it is being traced and re-mark all points on the line when it is complete. The pixels are still marked, but the two states are now applied to creating the nesting tree. Each pixel is instead marked as *leading* or *trailing*, defined as follows:

Any point  $(x,y)$  on an edge line such that the point  $(x+1,y)$  is outside the closed loop and is not on the same edge is marked *trailing*. All other used points are marked *leading*. Fig. 6.3 shows the edge points on the "a" of Fig. 2.2, with leading edge pixels shown chequered and trailing edge pixels solid. The edge path is shown superimposed in thick lines.

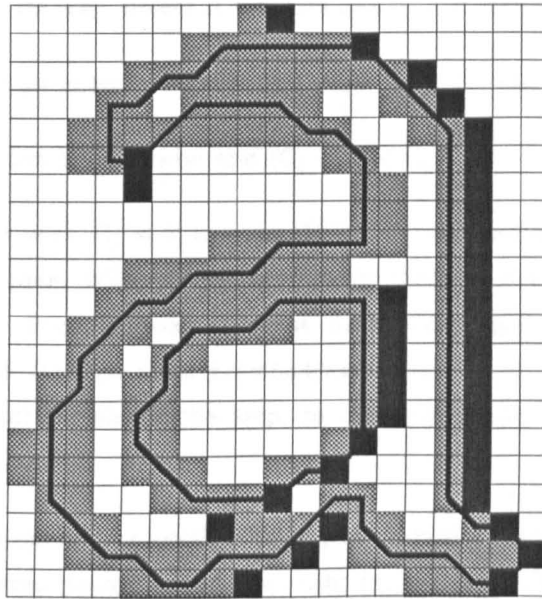


Fig. 6.3. The outline of an "a" with chequered leading edge points and solid trailing edge points. The edge path is also shown.

When the raster scanner passes over a point which is marked as a leading edge, it recognizes that any new objects encountered will be inside the closed loop and are thus children of the current object in the tree. When it passes over a point marked as a trailing edge, the raster scanner deduces that it is now inside the parent of that object in the tree. The current object in the tree is given by the label of a leading or trailing pixel.

By restricting the size of the image buffer, it can be guaranteed that no more than  $2^{16}-1$  nodes can be active at once, so the labels can be 16 bits in length. When a new buffer is read, any completed loops which are wholly above the current position of the raster scanner are moved out of the working tree into a

separate output tree. Their associated labels and tree nodes are freed and made available for new objects. The output tree uses longer pointers, giving it the capacity for any possible number of objects on a single page.

### 6.2.1 MARKING LEADING/TRAILING EDGE POINTS

Section 6.2 defined leading/trailing edge points in terms of whether the point  $(x+1,y)$  falls inside or outside the completed loop. This information must be generated before the loop is actually closed.

Let the current point be  $(oldx,oldy)$  and let the subset of the edge set at  $(oldx,oldy)$  which is selected for marking be given by  $\{(x_i,y_i), i=1,n\}$  where  $1 \leq n \leq 3$ . The chain code of  $(x_i,y_i)$  relative to  $(oldx,oldy)$  is  $C_i$ . Each edge point  $(x_i,y_i)$  is marked as leading, unless it satisfies one of the following conditions:

- (a) The line direction is anticlockwise and  $(C_i = 5 \text{ or } C_i = 6)$ .
- (b) The line direction is clockwise and  $(C_i = 2 \text{ or } C_i = 3)$ .

Condition (a) with  $C_i = 5$  is exemplified by Fig. 6.4(a). The bold arrows represent the edge path, with the thickest arrow showing the chain code 5 to the point in question. Fine arrows represent other edge points which are to be marked. Clearly, if the line is an anticlockwise traversal of the outline, then the point marked "\*" will be outside the loop, so the point at the head of the thickest arrow must be trailing.

Fig. 6.4(b) shows all the possible edge paths passing through  $(oldx,oldy)$  such that the set of marked points includes a point  $(x_i,y_i)$ , with  $C_i = 5$ , (in bold) and does not include a point at the asterisk,  $(x_i+1,y_i)$ . Clearly, if there is a point on the same line at  $(x_i+1,y_i)$ , then the mark at  $(x_i,y_i)$  is irrelevant.

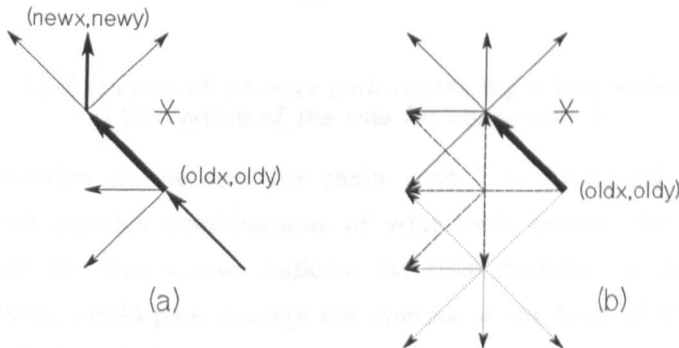


Fig. 6.4. (a) An example of the rule for  $C_i = 5$ . (b) All possible 2-step paths which include  $C_i = 5$  in the first step.

If  $C_i$  is 0, 1 or 7 then no decision can be made until the next step. Retaining the previous notation, for the original set of points, let the second step from (oldx,oldy) be to the point (newx,newy). The test depends on  $C_i$  and whether the line direction is anticlockwise or clockwise:

$C_i = 0$	Anticlockwise:	$y_i \leq \text{newy}$
	Clockwise:	$y_i > \text{newy}$
$C_i = 1$	Anticlockwise:	$x_i + y_i > \text{newx} + \text{newy}$
	Clockwise:	$x_i + y_i \leq \text{newx} + \text{newy}$
$C_i = 7$	Anticlockwise:	$y_i - x_i \leq \text{newy} - \text{newx}$
	Clockwise:	$y_i - x_i > \text{newy} - \text{newx}$

If the relevant condition above is satisfied then the point is marked trailing, otherwise it is left as leading. The condition for  $C_i = 7$  is demonstrated by Fig. 6.5(a) and (b), where bold arrows represent the edge path and the head of the thickest arrow is at the point in question. In Fig. 6.5(a), the point marked "\*" is clearly inside the loop assuming that the arrows represent an anticlockwise traversal of the edge. The situation is reversed if the direction is clockwise.

Fig. 6.5(b) shows the effect of (newx,newy) being in a different position. The asterisk is here outside an anticlockwise trace of the outline.

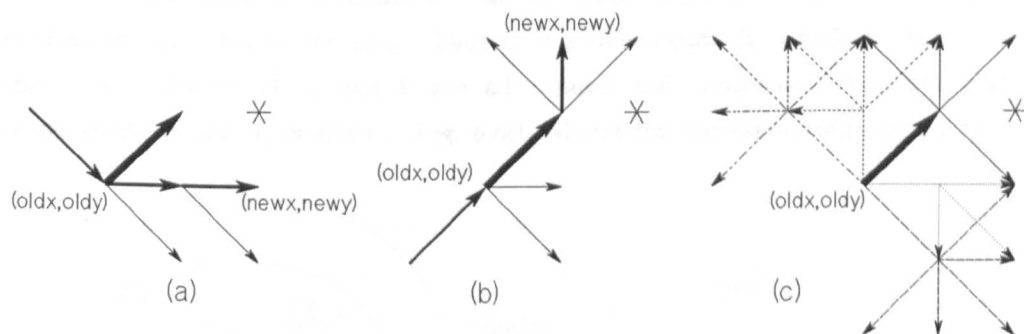


Fig. 6.5. (a), (b) Examples of an edge path containing a step with chain code 7.  
 (c) Derivation of the rule for chain code 7.

The derivation of the rule for chain code 7 is illustrated in Fig. 6.5(c), which shows all possible combinations of edge path around the chain code 7. Arrow heads on the fine arrows indicate the final position on the second step. Those paths which would pass through the asterisk or the head of the thick arrow, resulting in no decision being necessary, are not shown.

### 6.2.2. ORIENTATION OF EDGES

Due to the buffering of the image, a single object may cause several independent tree nodes to be started in one buffer. These objects will have to be coalesced when the edge lines connect further down the page and the tree will need to be modified. Fig. 6.6(a) shows a large object which is spread over three buffers and three small objects marked D, E and F. The raster scan locates edge A first, the edge is followed and is found to run out of the buffer. Edge B is then found, followed and likewise runs out of the buffer. Finally, edge C is found and that also runs out of the buffer.

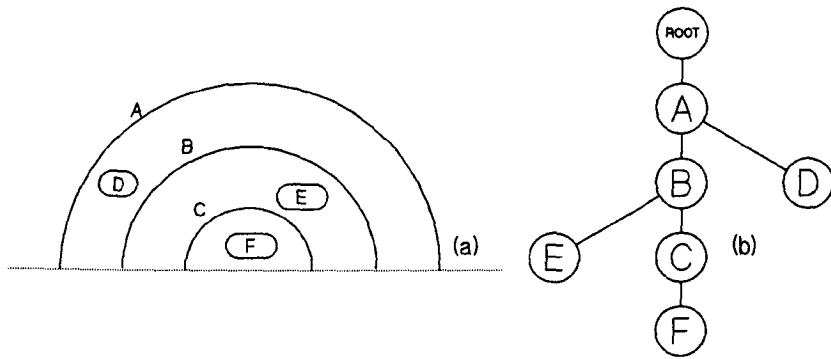


Fig. 6.6. (a) Edges found in one buffer. (b) The nesting tree for (a).

At this stage it is assumed that all these objects, when completed, will enclose the area below the arcs. Thus C is a hole inside B, which is itself a hole inside A. Objects D, E and F are all located and completed, they are placed respectively inside A, B and C. Fig. 6.6(b) shows the corresponding nesting tree.

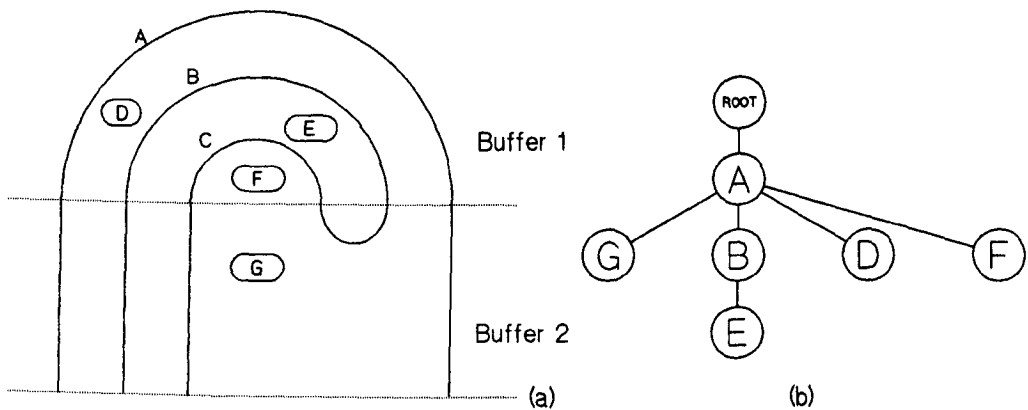


Fig. 6.7. (a) The amount of object visible in the second buffer. (b) The corresponding nesting tree.



In the second image buffer, illustrated by Fig. 6.7(a), edges B and C connect such that the area below C is in fact the area outside B. This deduction is obvious by following the edge line between B and C. Hence, at this stage it is not possible to deduce how the objects will close into a loop or loops, but it is certain that the section C is oriented inversely with respect to B. Therefore, the new object, G, being discovered after the leading edge of C, is assumed to be outside B. The tree is modified to reflect this new information, resulting in Fig. 6.7(b)

The third buffer, in Fig. 6.8(a), completes the large object. A and B coalesce such that B is inversely oriented with respect to A. This change causes the tree to be modified to the form shown in Fig. 6.8(b).

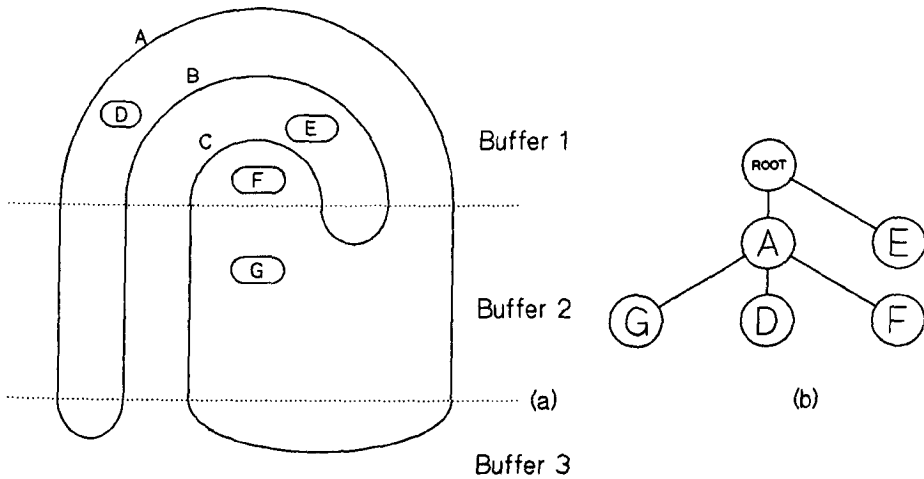


Fig. 6.8. (a) The completed object. (b) The final nesting tree.

Define the *orientation* of an object to be *true* if the area it encloses is below the start point, and to be *inverse* if the area enclosed is above the start point. When an object is first located by the raster scan, it is assumed that following the edge in chain code direction 4, i.e. left, will result in an anticlockwise traversal. This is equivalent to assuming that the point immediately below the starting point is enclosed by the edge. The orientation indicates whether this assumption is *true* or *inverted*. Thus when the large object in Fig. 6.8(a) is complete, A and C are true, while B is inverse. When the orientation of an object changes, all its children in the tree are moved to be its siblings.

As each object is found, its orientation flag is set to true. When a line coalesces with another, both orientation flags remain unchanged unless one of the following conditions is true:

- (a) The directions and orientations of the lines are both the same.
- (b) The directions and orientations of the lines are both different.

If one of the conditions is true, the orientation of one of the objects must be changed. The object to be changed is the one which is at the lowest level in the tree. The flag is changed from true to inverse or vice versa. Clearly, any other objects already connected to the changed object must also have their orientation changed.

### 6.3. THE CHOICE OF EDGE PATH

Section 3.2.3 discussed the problems in choosing between several possible edge points. This section presents a more complex solution which was derived experimentally from the failure cases on a varied set of test images.

#### 6.3.1. AVOIDING COLLISIONS

The only way to prevent collisions with existing edges is to look forward on the edge path and change course if the edge is about to collide with another edge in an undesirable fashion. Let those points in the edge set which have an edge value within a certain range of the strongest edge value, be known as the set of *useful points*. For each of the useful points, the successor edge set is considered. The result is a tree of possible edge paths similar to Fig. 6.9, where the strongest edge in each set is denoted by a bold arrow.

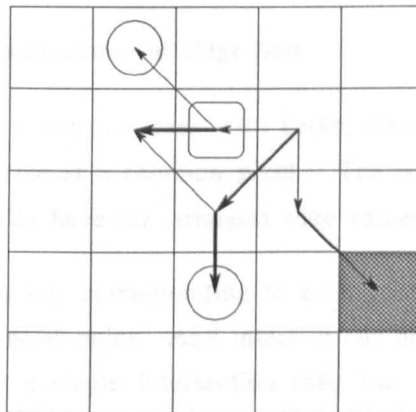


Fig. 6.9. A tree of possible edge paths.

To reduce the number of points tested, only the strongest point in each successor set is considered. For each first level useful point and its strongest successor, the two points are checked to see if they are on any used or deleted

edge. If any point tested is on another edge such that a legitimate connection is made, then the associated path is chosen immediately. Otherwise, a count is generated according to the number of points from the pair which are on another edge.

Those useful points at the first level of the tree which have more than the minimum count value are now discarded. In Fig. 6.9, the chequered pixel belongs to another edge, so the path to that point is rejected. The remaining set will be referred to as the set of *good points*. If there is only 1 such point then that is chosen immediately as the next step.

### 6.3.2. LOCATING FORK POINTS

There are now two or more good points, each with an edge set of up to five useful points. The leftmost useful point in the successor edge set to the leftmost good point and the rightmost useful point in the successor edge set to the rightmost good point are now considered. If the maximum modulus distance between these two points is more than 2 pixel units, then the present point is considered to be a *fork point*. The root point of Fig. 6.9 is a fork point as indicated by the distance between the circles.

A fork point is a pixel where the edge path appears to split into two distinct directions at the point. For non-fork points, which is the majority case, the point chosen is the one with the highest sum of present and strongest successor edge values.

#### 6.3.2.1. Fork Analysis: Intersection of Edge Sets

Fig. 6.9 illustrates a common case with forks; taking either direction at the fork, it is possible to arrive at a common pixel. The steps to the common point however, do not necessarily have the strongest edge values.

The successor edge set corresponding to each good point is compared with the set for the adjacent good point. Any intersections between these pairs of sets are counted. If there is a single intersection then this is considered a majority vote to go to that pixel. The direction chosen for the next step is one of the pair involved in the intersection such that it has fewest useful points outside the overlap region. If they are equal on that basis then the point with the strongest edge value for the second step is taken. In Fig. 6.9 the step chosen is indicated by the rounded square.

### 6.3.2.2. Fork Analysis: Backtracking

If the intersection test failed to produce a decision, a backtracking operation takes place. Fig. 6.10 illustrates the principle of backtracking. The solid arrows indicate the most extreme fork choices of Fig. 6.9. The dotted curves in Fig. 6.10(a) show one possible location of the actual edges, Fig. 6.10(b) shows another.

From the extreme fork points, the edge follower attempts to trace edges in the opposite direction. It takes the leftmost useful point of the edge set from the left fork and the rightmost from the right fork. A second step from each of these is taken, with the same left and right extremes. The results of backtracking are shown by the dashed arrows in Fig. 6.10 (a) and (b). The sum of modulus distance is then measured between each of the resulting positions and the fork point. If the distances differ by at least 2 then the direction corresponding to the point nearest the fork point is taken as the next step. The selected step is circled in Fig. 6.10(a) and (b).

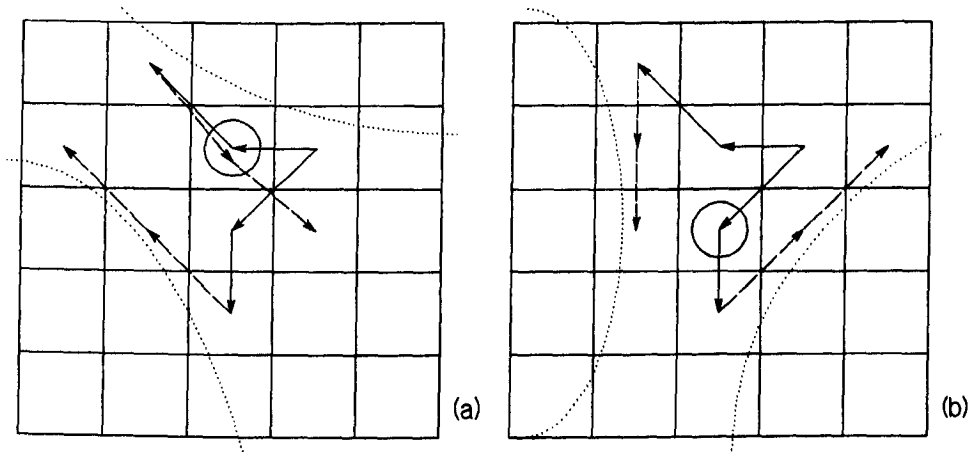


Fig. 6.10 Backtracking to resolve forks and the result with two different positions of the real edges.

If no result is obtained, then the second backtracking step is repeated using the strongest edge point, rather than the leftmost/rightmost. If there is still no decision, a reverse edge operator is used.

### 6.3.2.3. Fork Analysis: Reverse Edge Operator

At the fork point, a different edge operator is applied in the direction of the leftmost and rightmost good points. The difference is taken between pixels with chain codes  $n-2$  (modulo 8) and  $n+2$  (modulo 8). This new operator is shown

in Fig. 6.11(a) for chain code 0 and in Fig. 6.11(b) for chain code 7. It is intended to test the strength of the edge in the reverse direction.

The edge value resulting from the new operator is added to the original edge value for the two points. If this makes one much greater than the other, then the greatest is used, otherwise the new operator is applied at each of the leftmost and rightmost good points. These results are added to the previous values and the greatest is now chosen as the next point with an arbitrary choice if they are still equal.

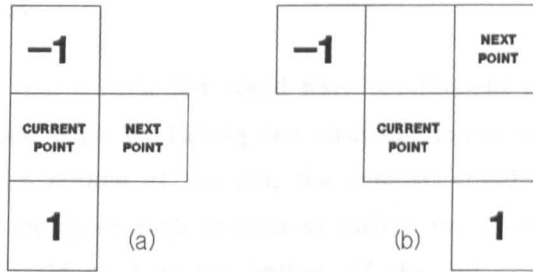


Fig. 6.11. The reverse edge operator. (a) For chain code 0. (b) For chain code 7.

### 6.3.3. RE-TRACING FAILED EDGES

The algorithm of section 6.3.2 reliably locates fork points. The described methods for choosing the path however, are by no means perfect. As an alternative to the acceptance of any closure of a loop, suggested at the end of section 3.2.4, bad collisions can be re-traced.

When a fork point is detected, its location and the path chosen are stored. If the edge subsequently collides with itself or another edge in a way which is regarded as unsatisfactory, the edge is retracted to the fork point and the alternative path taken.

This approach could be extended to a process of searching a tree of several levels of fork point, with a consequential increase in processing time. For processing text images, the increase would be negligible, since the number of incorrectly interpreted fork points is minimal. For halftoned images however, the number of fork points and collisions is extremely large, so any modification to the analysis of fork points has to take halftones into account.

## 6.4. SOME PROBLEMS CAUSED BY HALFTONED IMAGES

Text and line drawings are generally of very high contrast, being either black on white or white on black. A halftoned image however, can contain many different forms of randomly changing grey levels. The algorithm of section 6.2.1 for setting the flags to leading or trailing works correctly for text and line drawings. There are some cases however, in which it can fail. These are caused by some peculiarities of halftoned images.

### 6.4.1 FAINT EDGES

It is possible that a faint dot could have insufficient contrast at the top to be considered a good edge by failing the stringent raster scan test described in section 3.2.2. At the bottom of the dot, the contrast could be much greater. If the contrast at the top were high enough to satisfy the lower tracking threshold, the edge follower could start at the bottom of the dot and track in the usual direction. The line would go over the top of the starting point and complete a clockwise loop. The resulting object would have its leading/trailing flags totally incorrect.

An example of a correctly traced dot is given in Fig. 6.12(a). Fig. 6.12(b) shows the result of starting a dot at the bottom. Leading edge points are shown chequered and trailing edges solid. The traced edge path is superimposed in thick lines with the starting point marked "+".

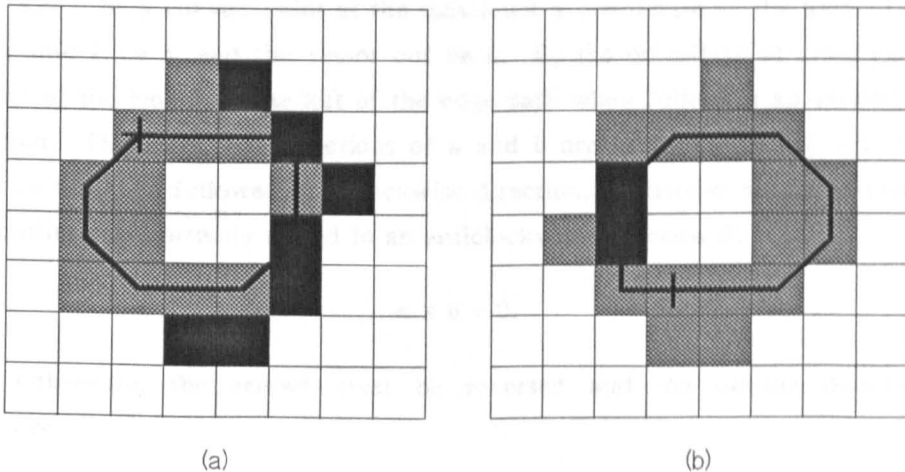


Fig. 6.12. (a) A correctly traced dot. (b) A dot started at the bottom.

As long as the leading/trailing flags are correct, the raster scanner can use the leading/trailing and orientation flags to determine whether it is entering or leaving an object. Entry to an object which is inverse is interpreted as exit from its parent and exit from an inverse object means entry to its parent. If the flags can be set incorrectly then the entire tree structure can become badly corrupted.

There is a simple test which can be applied to the completed loop to detect a clockwise traversal. Having obtained a polygonal approximation of the completed outline, the point P with the greatest y-coordinate is found. P cannot be concave since, if it were, there would be a point with a greater y coordinate, illustrated by Fig. 6.13(a). However many points there are at the maximum y-coordinate, at least one must be convex, otherwise the object would be a straight line.

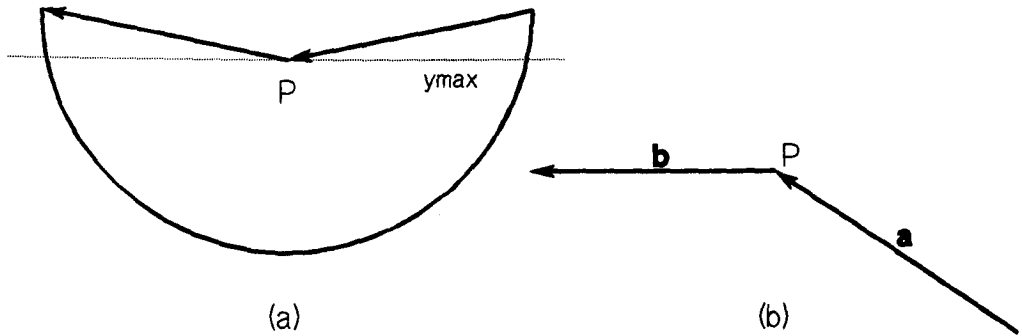


Fig. 6.13. (a) The top point must be convex. (b) An anticlockwise convex point.

Let P be a convex point at the maximum y coordinate of the loop. Let the vector into P be **a**, and the vector out be **b**. By the definition of clockwise, the interior of the loop is to the left of the edge path when following an anticlockwise direction. Therefore, the directions of **a** and **b** are as shown in Fig. 6.13(b). If the loop had been followed in a clockwise direction, the arrows would be reversed. The outline was correctly traced in an anticlockwise direction if:

$$\mathbf{a} \times \mathbf{b} > 0.$$

Otherwise, the arrows must be reversed and the outline direction is clockwise.

### 6.4.2. SELF-INTERSECTING EDGE PATHS

In Fig. 6.14 there is an edge path which crosses itself. Although the lines cross, they actually have no points in common. The edge follower cannot detect this kind of collision without searching non-edge points, leading to a large increase in processing time.

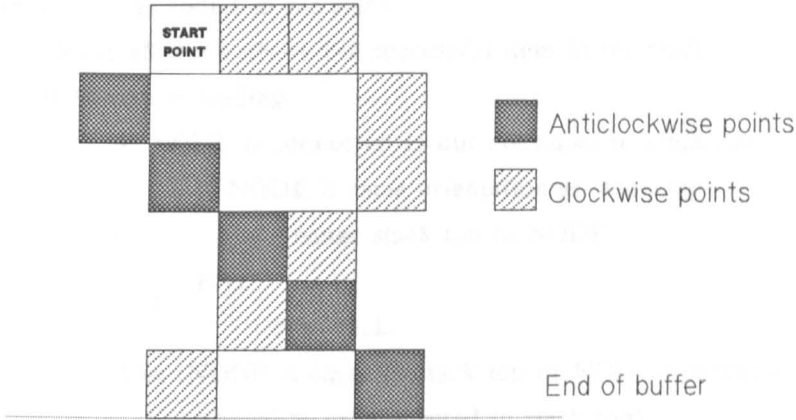


Fig. 6.14. A self-intersecting object.

The fastest solution is for the raster scanner to double check its position in the tree and delete objects which cause a discrepancy between the two tests. The second test is based on a stack. At the start of a raster line, the stack is empty. On entering each new object the label for that object is pushed onto the stack. On exit from the object, the stack is popped. If everything is correct, the new stack top will be the parent of the object just popped. If this is not the case, then the parent should be somewhere down the stack and all entries between are invalid. This simple algorithm is complicated by the existence of inversely oriented objects.

### 6.4.3. A FULL STACK-BASED SOLUTION

The following algorithm describes a full solution to the problem of detecting errors due to undetected self-intersecting edge lines. The algorithm starts with the raster scanner having encountered a PIXEL marked used i.e. leading or trailing, with a label NODE indicating to which tree node the pixel belongs.



**If NODE=stack top**

**If PIXEL is trailing**

**Pop stack.**

**Else**

**Search down the stack, checking all entries and anything they are connected to for NODE and the parent of NODE.**

**If anything was found in the stack**

**Delete all items above the successful item in the stack.**

**If PIXEL is leading**

**If NODE is connected to but not equal to stack top**

**If NODE is same orientation as stack top**

**Change stack top to NODE.**

**Else**

**Pop stack.**

**Else (NODE is equal to stack top or NODE's parent is equal to or connected to stack top)**

**If NODE's parent is equal to or connected to stack top**

**Push NODE on stack.**

**Else (PIXEL is trailing)**

**If NODE is equal to stack top**

**Or NODE is connected to & same orientation as stack top**

**Pop stack.**

**Else (Nothing found in stack)**

**If PIXEL is trailing**

**Push NODE's parent on stack.**

**Any new edges found will be placed in the tree as children of the current stack top. Any objects remaining in the stack at the end of each scan line are deleted.**

## **6.5. SUMMARY**

Construction of a full nesting tree is essential for the recognition of engineering drawings. The additional data required to construct the tree is so large that it could not be held in the memory of a single processor at a reasonable cost. It is therefore essential to operate on small sections of the image. Following the edges of objects which may be spread over several sections, causes a large increase in the complexity of the edge extraction algorithm when compared to the description in chapter 3.

In order to place new objects correctly in the tree, each edge pixel is marked leading or trailing, so that the raster scan can detect entry to and exit from objects. These marks must remain correct even when an object is split across several sections of the image. The tree is modified as parts of a single edge combine in a later section of the image.

Halftoned images pose difficult problems in maintaining the accuracy of the leading or trailing marks, with a consequential severe impact on the integrity of the nesting tree. A stack is used to confirm or contradict the evidence of the marks, enabling deletion of incorrectly traced outlines.

Chapter 3 discussed the problem of making an accurate choice of edge path. Possible collisions and fork points are detected using look-ahead. When a fork point is located, a series of tests are used to ascertain the best choice. These tests are: intersection of edge sets, backtracking and applying a reverse edge operator. As a final attempt, the opposite choice is taken at a fork point when a line has been found to collide inappropriately.

## 7. THINNING

Several algorithms for OCR by feature extraction<sup>(47-50)</sup> locate features in a thinned binary image. The principle of thinning is to reduce an image of a character or line from several pixels wide to unit width. For character recognition purposes, some of the font information is eliminated, thus providing a degree of font independence.

The functional attributes described by Shillman<sup>(34)</sup> are defined in terms of vector representations of characters, implying some form of thinning. The thinning algorithm described in this chapter, was designed with the aim of extracting these functional attributes before it was discovered that a subset, which can be extracted from outlines, is adequate for recognition. Thinning is not therefore, a necessary part of the OCR system described in chapter 2. It is however, essential for the recognition of engineering drawings and other line graphics.

### 7.1. PREVIOUS THINNING ALGORITHMS

The result of thinning is usually referred to as a *skeleton*. Montanari<sup>(51)</sup> defines a skeleton as being the result of propagating wavefronts from the inside of the outline of the figure. The intersection points of wavefronts from opposite sides of the figure form the skeleton. Although this definition is in terms of the outline of the object, it is closest to the intuitive definition assumed by the largest group of thinning algorithms, the iterative methods.

Rosenfeld and Pfaltz<sup>(52)</sup> define a skeleton to be formed by the centres of maximal discs placed within the object. The definition does not guarantee that the skeleton will be connected. Davies and Plummer<sup>(53)</sup> extend this definition to retain sufficient additional points to maintain connectivity.

#### 7.1.1. THE ITERATIVE THINNING ALGORITHMS

Many<sup>(53-66)</sup> iterative thinning algorithms have been suggested. The general principle is to iteratively remove the outside layer of pixels from an object until only a unit width skeleton remains. A 3 by 3 window operator is passed over the image in a raster scan to mark pixels for deletion, according to rules applied within the window. A second pass of the raster scan is then used to delete marked pixels. The two passes are applied repeatedly until no points are deleted.

There are several possible variations to this theme:

- (a) The second pass of the raster scan may include additional tests to prevent excessive erosion.
- (b) There may be no second pass of the raster scan; all the tests and deletion may occur in a single pass.
- (c) The window operator may be applied simultaneously on a large array of parallel processors. The operator must be modified to cope accurately with parallel deletion without excessive inter-processor communication.
- (d) A different size or shape of window operator may be used.
- (e) To reduce the processing time, the image may be split into many rectangular regions, with the terminating condition of no pixels being deleted applied individually to each region. This optimization is essential when processing a large page of text.

Since there is such a great deal of similarity between the algorithms, only one will be described here. The algorithm presented is that of Hilditch<sup>(54)</sup> as described by Naccache and Shinghal<sup>(55)</sup>.

Fig. 7.1 shows a pixel *P*, with its *8-neighbours* numbered from 0 to 7. The *4-neighbours* of *P* are the pixels numbered 0, 2, 4 and 6. Let the *crossing number* at *P* be the number of white 4-neighbours of *P* which satisfy the following condition:

If *n* is the number of the white 4-neighbour of *P*, at least one of *n+1* and *n+2* (modulo 8) must be black.

5	6	7
4	P	0
3	2	1

Fig. 7.1 The 8-neighbours of *P*.

Table 7.1 describes six tests which are applied to black pixels in the usual left to right and top to bottom raster scan order. Pixels which satisfy all of the tests are marked for deletion.

TEST	REASON FOR TEST
(1) P has at least 1 white 4-neighbour.	Ensures that P is an edge point.
(2) P has at least 2 black 8-neighbours.	Ensures that P is not an end point.
(3) At least one of the black 8-neighbours of P must not be marked.	Prevents black points at the end of thin lines from being iteratively deleted.
(4) The crossing number at P must be 1.	P must not be a break point.
(5) If point 6 is currently black and marked, setting it to white must leave the crossing number at P as 1.	Tests (5) and (6) prevent excessive erosion.
(6) If point 4 is currently black and marked, setting it to white must leave the crossing number at P as 1.	

Table 7.1. The conditions for deleting a black pixel.

Fig. 7.2(a) shows a binary image of a "Q" with the result of applying the above algorithm in Fig. 7.2(b). Deleted points are shown chequered, with skeletal points solid.

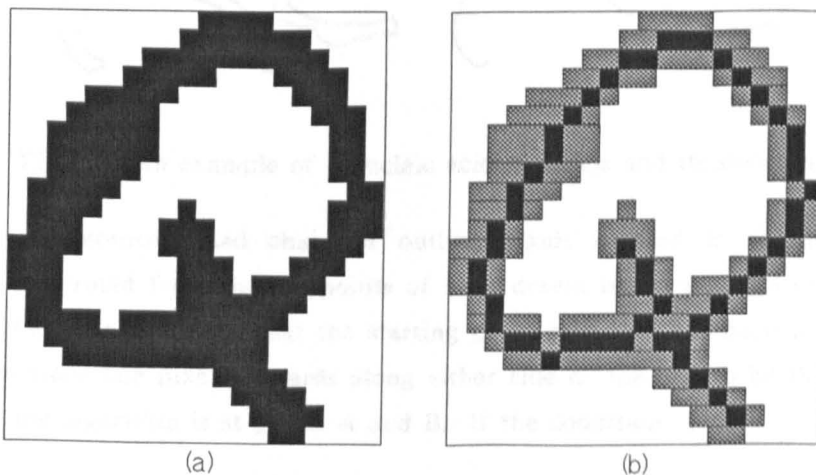


Fig. 7.2. (a) A binary image of a "Q". (b) The thinned version of (a).

Naccache and Shinghal<sup>(56)</sup> have implemented a wide selection of iterative thinning algorithms and give a useful indication of the processing time required on

a CDC 170-825. The times quoted range from 184 to 513 ms per character, with an average size of 17 by 23 pixels. The fastest of these yields a rate of almost 5.5 characters per second, which is clearly impractical for commercial character recognition.

The average size is also smaller than would be obtained from typical print at 300 pixels per inch. 25 by 30 pixels would be more realistic for a page of only 5000 characters. Because the time is a product of the area and the maximum width of any part of the character, the 184 ms would scale to approximately 520 ms, yielding only 1.9 characters per second. The only possible conclusion is that iterative thinning is too slow to be of any practical use, without using an array processor.

### 7.1.2. FASTER THINNING ALGORITHMS

Shapiro, Pisa and Sklansky<sup>(67)</sup> describe an algorithm for finding the skeleton of nucleic acid molecules. The utility of the algorithm is severely restricted by the fact that it can only be applied to objects with a central axis and protruding branches. No sub-branches or closed loops are permitted. The algorithm cannot therefore be applied to most text characters. The restriction is due to a lack of capability to locate line endings and junctions. An example of a typical outline shape and the resulting skeleton is shown in Fig. 7.3.

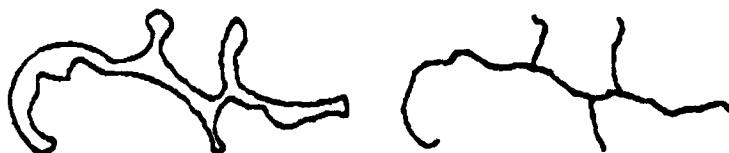


Fig. 7.3. An example of a nucleic acid molecule and its skeleton.

A non-approximated chain of outline pixels is used in thinning. The skeleton is formed from the midpoints of lines drawn between pixels on opposite sides of the object. Given that the starting point of a line has been located, the algorithm steps one pixel forwards along either side of the line to be thinned. In Fig. 7.4, the algorithm is at points A and B. If the condition

$$|AC-BD| \leq d$$

is satisfied, then the next pair of points will be D, C. Otherwise the next pair of points will be either A and C, or D and B, chosen for the minimum diagonal

length. In the above test,  $d$  is a fraction of the average inter-pixel spacing. Once line endings are located, the algorithm is linear in the number of outline points.

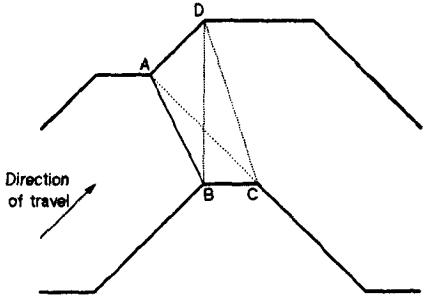


Fig. 7.4. The equal diagonal algorithm of Shapiro, Pisa and Sklansky.

Jimenez and Navalon<sup>(30)</sup> suggest a method which operates on a polygonal approximation of the outline. In Fig. 7.5, segment 2-3 has been chosen as the starting segment. The rest of the object is searched for a segment which is crossed by the perpendicular  $a$  from the midpoint of 2-3. Segment 24-23 is the nearest such segment and this is chosen as the starting segment on the opposite side of the object. The algorithm then shoots vectors in the opposite direction to  $a$  from vertices 24, 23, 22 etc. while they intersect segment 2-3. The midpoints of these lines are used as points on the skeleton, as indicated by the dotted line.

When the vector from vertex 21 is found not to intersect segment 2-3, a perpendicular is dropped from 3 and this is found to intersect segment 22-21. The process now continues with a perpendicular to segment 3-4.

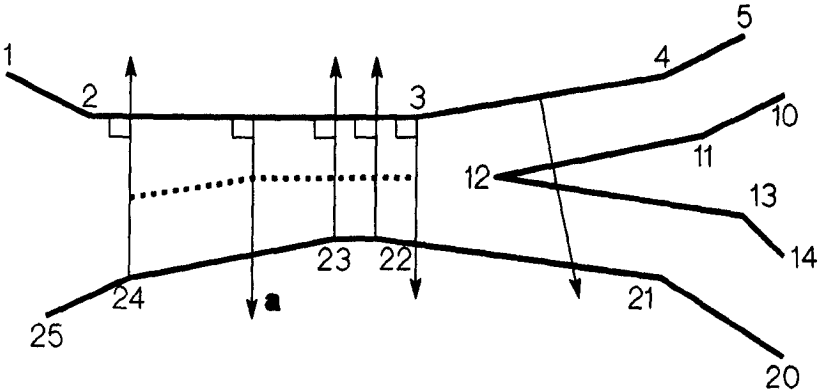


Fig. 7.5. An example of the failure to detect a junction.

Jimenez and Navalon<sup>(30)</sup> claim that their algorithm is linear in the number of source points in the outline. The test applied to the perpendicular from segment 3-4 however, must either make the algorithm squared order, or fail with the example in Fig. 7.5. The perpendicular to segment 3-4 intersects segment 22-21. There is no sudden increase in line width or any other indicator that there

is a junction. Therefore, without a linear search of all available outline segments it is difficult to guard against the situation illustrated, where segment 12-11 is in fact the nearer line. The same search must be conducted for each segment on the outline, therefore the algorithm must be squared order.

This squared order searching problem is a direct consequence of reducing the 2-dimensional bit map representation of the image to a simple 1-dimensional circular list of vectors. A simple way around the problem is to maintain a bit-map representation. The shooting of perpendiculars involves actually drawing the lines on the bit-map, and then the first line crossed actually is the nearest line. Without specialized hardware however, this approach may be even slower, depending on the width of lines and the number of vertices in the outline, since the time taken to draw a vector may exceed that to search the circular list.

## 7.2. A NEW THINNING ALGORITHM

The concept of thinning is difficult to apply to an object such as a dot. With the algorithm described in this chapter, a skeleton can be produced for any object which has either at least two line endings or a hole. A dot may have neither, in which case, the suggested skeleton is a line connecting the two most distant points on the outline.

The new thinning algorithm, described in the subsequent sections, was originally designed to operate on text outlines. It is also suitable for arbitrary line graphics, and is therefore applicable to the recognition of engineering drawings. The only problem in adapting the algorithm to line drawings, is determining which part of the nesting tree forms one object to be thinned.

Fig. 7.6 gives an example of why this is a non-trivial problem, as long as the system aims at coping with both black lines on a white background and the reverse sense on the same page. Fig. 7.6(a) shows a simple example of black lines on a white page, with the corresponding nesting tree in Fig. 7.6(b). The nodes joined by bold lines in the nesting tree indicate the different units which must be thinned to obtain the correct representation.



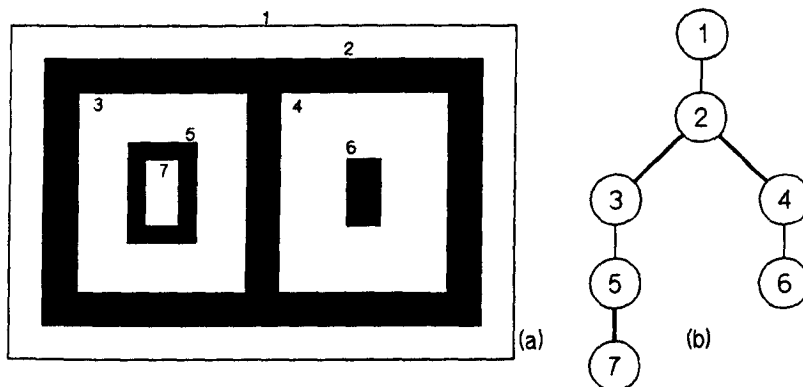


Fig. 7.6. (a) A simple page. (b) The nesting tree of (a).

The nesting tree of Fig. 7.6(b) is also produced by the page in Fig. 7.7. If the page is interpreted as white lines on a black page, then the units for thinning remain the same. If the same page were interpreted as black lines on a white page, the units for thinning would be those which are joined by the thin lines in the tree. The difficulty can be eliminated easily if the system can assume that all lines are in the opposite colour to the root of the tree. This simplification however, would restrict the recognition of grey shaded or inverse boxes within the page.

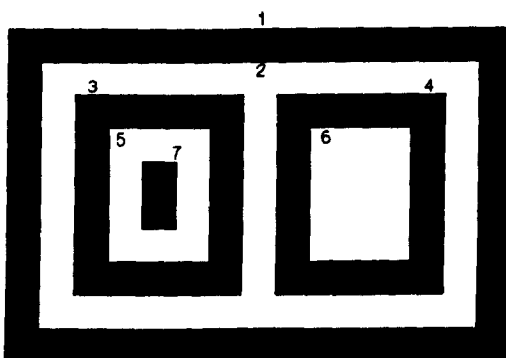


Fig. 7.7. An alternative page to produce the nesting tree of Fig. 7.6(b).

A better solution would be to measure the width of objects between adjacent levels in the nesting tree. Those edges which are close together are likely to be opposite edges of the same line.

This is a problem which does not occur when extracting only the text from a page, since no character may contain any significant mark inside a hole, nor may any character possess more than two holes. These conditions restrict the position which a character may occupy in the tree sufficiently to reliably extract text.

### 7.3. NEAREST POINTS

The remainder of this chapter make use of the notation which was defined in section 5.3. The first step of the thinning process is to find the *nearest point*,  $N_i = \text{nearest}(P_i)$ , to each point  $P_i$  on the outline, conceptually defined to be the nearest point which is on the opposite side of the object. In practice, it is the nearest in terms of distance<sup>2</sup> which also satisfies both of the following conditions:

(a) A line drawn from  $P_i$  to  $N_i$  must pass wholly within the object, holes within the object being regarded as outside it. Thus in Fig. 7.8(a),  $P_6$  cannot be the nearest point to  $P_2$ , since the line passes through part of the outside, as shown by the dotted part of the line.

(b) At least one of the following must be true:

(b1)  $P_{i+} \cdot N_{i-} > 0$  and  $P_{i+} \times P_i N_i > 0$  and  $N_{i-} \times P_i N_i > 0$ .

(b2)  $P_{i-} \cdot N_{i+} > 0$  and  $P_i N_i \times P_{i-} > 0$  and  $P_i N_i \times N_{i+} > 0$ .

Thus the sides must be nearer anti-parallel than parallel, determined by the scalar product. The cross product tests are applicable only in the case of concave points, where the angle subtended by  $P_i$  and/or  $N_i$  may be more than  $180^\circ$ . The tests guard against cases such as Fig. 7.8(b).  $P_{23}$  cannot be the nearest to  $P_{21}$ , since  $P_{21+} \cdot P_{23-} \leq 0$  and  $P_{21} P_{23} \times P_{21-} \leq 0$ . The nearest point to  $P_{21}$  is thus  $P_{17}$ .

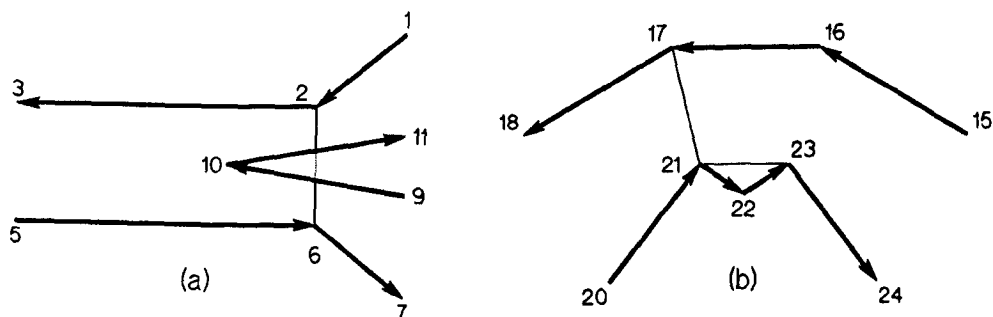


Fig. 7.8. (a) An example of nearest point test (a). (b) An example of nearest point test (b)

It is possible that there is no nearest to a particular point, because no point can be found which satisfies (a) and (b) above. The *neardistance*<sup>2</sup> of a point is defined to be the distance<sup>2</sup> to its nearest point. If there is no nearest point then the *neardistance*<sup>2</sup> is defined to be a known constant value which is larger than any possible distance<sup>2</sup> in the image. Fig. 7.9 shows some actual outlines with the nearest point relations indicated by arrows drawn from each point to its nearest point.

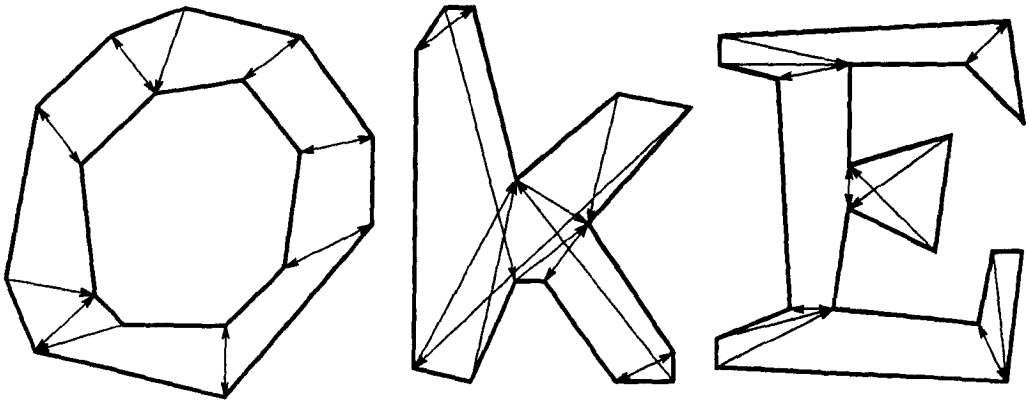


Fig. 7.9 Some actual outlines with nearest points marked by arrows.

Section 7.8 discusses how the nearest point algorithm can be optimized to prevent cubed order behaviour and to minimise the squared order component.

#### 7.4. ENDPOINTS

After the nearest point relations have been ascertained, ends of lines on the object are located. Throughout the endpoint tests, a value MAXWIDTH is used. MAXWIDTH gives a measure of the maximum width<sup>2</sup> that a line is allowed to have, and is a fixed multiple of the average neardistance<sup>2</sup> calculated in section 7.3, (excluding points with no nearest,) or a fraction of the square of the longest side of the minimum upright bounding box of the object, whichever is smaller.

##### 7.4.1. TESTING FOR ENDPOINTS

Each point  $P_i$ , on the object, undergoes the following tests to detect endpoints. Note that a *line ending* may include several *endpoints*.

- (a) If  $P_i$  is convex, and the bend at  $P_i$  is more than  $90^\circ$ , then  $P_i$  is an endpoint if  $N_i$  does not exist, or  $\text{nearest}(N_i)$  is not the same as  $P_i$ . The points marked A1 on Fig. 7.11 are endpoints with no nearest. A2 in the same figure is a point with  $\text{nearest}(N_i)$  different to  $P_i$ . Fig. 7.10(a) shows two sharp points labelled AB, on the corners of the "N" which are not endpoints because  $\text{nearest}(N_i) = P_i$ .
- (b) If  $P_i$  is convex and the bend at  $P_i$  is more than  $60^\circ$ , and the points on either side are concave, then  $P_i$  is an endpoint if it meets one of these sub-conditions:
  - (b1)  $N_{i-1} = P_{i+1}$  or  $N_{i+1} = P_{i-1}$ , i.e. there is a nearest point line joining  $P_{i-1}$  and  $P_{i+1}$ , in either direction.

(b2)  $|P_{i-1}P_{i+1}|^2 < \text{MAXWIDTH}$  and the perpendicular distance<sup>2</sup> from  $P_i$  to the line joining  $P_{i-1}$  and  $P_{i+1}$  is more than a fixed fraction of  $\text{MAXWIDTH}$  and less than the  $\text{neardistance}^2$  of  $P_i$ .

An example of condition (b1) is shown in Fig. 7.10(b) at the point marked B1. The point B2 illustrates condition (b2). The corners AB, of the "N" in Fig. 7.10(a) are also examples of points which would be rejected by test (b).

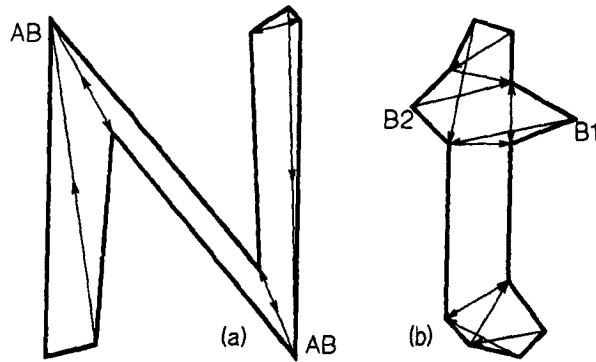


Fig. 7.10. (a) An example of sharply convex points which are not endpoints.  
 (b) Sharply convex points which are endpoints under test (b).

(c) If  $P_i$  and  $P_{i+1}$  are convex and  $|P_{i+1}|^2 < \text{MAXWIDTH}$  and the total angle of bend is more than  $60^\circ$ , then the radius of curvature is tested. If the radius of curvature is sufficiently small, then  $P_i$  and  $P_{i+1}$  are marked as an *endpoint pair*.

Fig. 7.11 repeats the outlines of Fig. 7.9 with single endpoints resulting from tests (a) and (b) marked by asterisks. Endpoint pairs produced by test (c) are shown connected by thick lines.

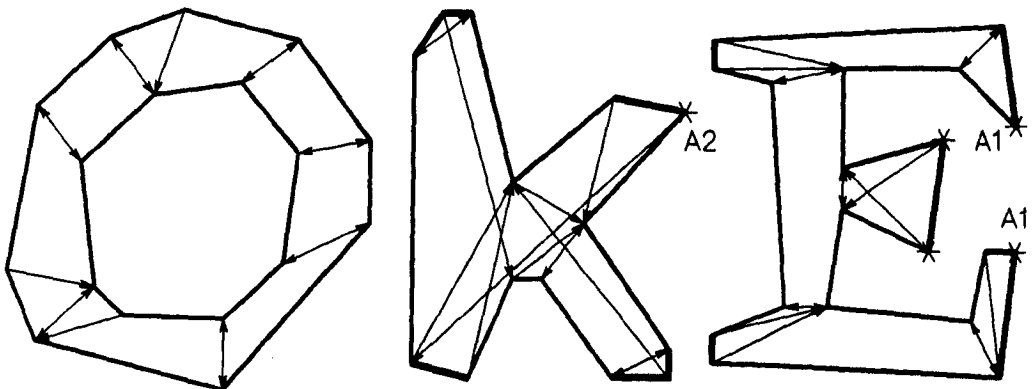


Fig. 7.11. Outlines with single endpoints marked by asterisks and endpoint pairs joined by thick lines.

## 7.4.2. THE RADIUS OF CURVATURE TEST

The radius of curvature test uses a discrete approximation to the radius of curvature function for a continuous curve. In Fig. 7.12(a) the arc length  $d$ , the angle  $A$  and the radius  $r$  of an arc of a circle are related by the equation  $r=d/A$  when  $A$  is measured in radians.

Fig. 7.12(b) shows a pair of points  $P_i$  and  $P_{i+1}$  under consideration, with their predecessor and successor points  $P_{i-1}$  and  $P_{i+2}$ . The total turn angle at  $P_i$  and  $P_{i+1}$  is shown as  $\varnothing$ . If  $\varnothing > 180^\circ$ , then  $d$  is taken to be  $|P_{i+1}|$ . Otherwise, the vectors  $P_{i-1}P_i$  and  $P_{i+1}P_{i+2}$  are scaled such that their lengths are both equal to the larger of  $|P_{i+1}|$  and the smaller of  $|P_{i-1}P_i|$  and  $|P_{i+1}P_{i+2}|$ . Let the scaled vectors be denoted by  $\mathbf{a}$  and  $\mathbf{b}$  respectively. A scaled version of Fig. 7.12(b) is shown in Fig. 7.13(a).

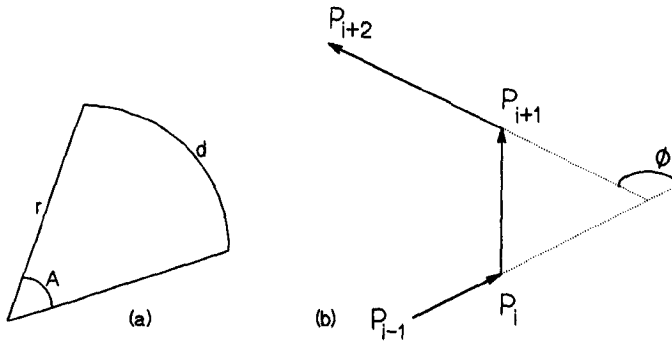


Fig. 7.12. (a) Radius of a circular arc. (b) A discrete arc.

If  $\varnothing < 180^\circ$ ,  $d$  is measured between the new positions of  $P_{i-1}$  and  $P_{i+2}$ . The reason for this complicated measurement is to prevent a false response to a pair of very short edge lines with a small angle.

To avoid floating point arithmetic and trigonometric functions, the angle  $A$  is approximated from  $\varnothing$ , using a simple property of scalar products:

$$\mathbf{a} \cdot \mathbf{b} = a b \cos \varnothing$$

where  $a$  and  $b$  are the lengths of  $\mathbf{a}$  and  $\mathbf{b}$  respectively. This relation can be used to find  $\cos \varnothing$ . Rather than calculate  $\cos^{-1}$ , a non-decreasing sinusoidal approximation of  $\varnothing$  is used as follows:

$$A = 1 - \frac{(a.b)^2}{a^2b^2} \quad \text{If } a . b \geq 0 \quad \text{and } a \times b \geq 0 \quad \text{i.e. } 0^\circ \leq \phi \leq 90^\circ$$

$$1 + \frac{(a.b)^2}{a^2b^2} \quad \text{If } a . b < 0 \quad \text{and } a \times b \geq 0 \quad \text{i.e. } 90^\circ < \phi \leq 180^\circ$$

$$3 - \frac{(a.b)^2}{a^2b^2} \quad \text{If } a . b < 0 \quad \text{and } a \times b < 0 \quad \text{i.e. } 180^\circ < \phi < 270^\circ$$

$$3 + \frac{(a.b)^2}{a^2b^2} \quad \text{If } a . b \geq 0 \quad \text{and } a \times b < 0 \quad \text{i.e. } 270^\circ \leq \phi < 360^\circ$$

The graph of this function is shown in Fig. 7.13(b). Of the various possible ways of approximating the radius of curvature, the method described above has been experimentally determined to be the most useful. The radius approximation calculated from  $d/A$  is compared with the smallest neardistance from  $P_i$  or  $P_{i+1}$ . The neardistance from  $P_i$  is not used if  $N_i$  is  $P_{i-2}$  or  $P_{i+2}$ . Similarly, the value for  $P_{i+1}$  is not used if  $N_{i+1}$  is  $P_{i-1}$  or  $P_{i+3}$ .

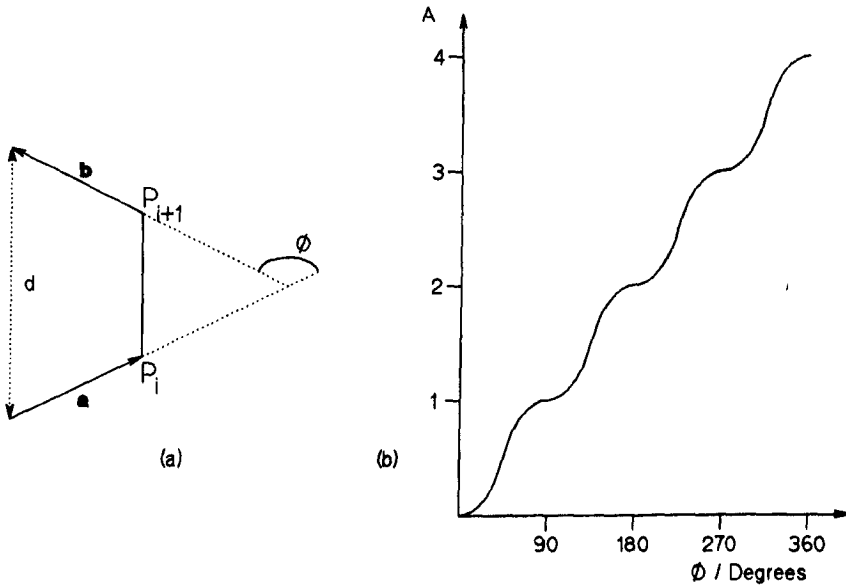


Fig. 7.13. (a) A scaled version of Fig. 7.12(b). (b) The angular approximation.

If there is a neardistance  $N$ , which can be used, then  $P_i$  and  $P_{i+1}$  make a good endpoint pair if  $d/A < 0.8N$ . If  $d/A > 1.2N$  then the points are not good endpoints. For text, the number of ends falling between these values is small. For those which do, the points can be made an endpoint pair if

$d^2 < \text{MAXWIDTH}$ . If there is no neardistance value which can be used, then the points  $P_i$  and  $P_{i+1}$  automatically become ends.

### 7.4.3. COALESCING ENDPOINTS INTO LINE ENDINGS

A line ending may consist of several endpoints or endpoint pairs. Two endpoints belong to the same line ending if they meet all of the following conditions:

- (a) There is no concave point between them of bend angle more than  $30^\circ$ .
- (b) The total distance<sup>2</sup> between the two ends must be less than  $\text{MAXWIDTH}$ .
- (c) The total bend angle over the two ends must be between  $180^\circ$  and  $360^\circ$ .

After the line endings have been found, they are checked for concave points on either side of the end region. Where there are concave points, the distance<sup>2</sup> between these is calculated. If the result is less than twice the previous distance<sup>2</sup> across the end, then the end is moved to the concave point or points. This process helps to generate straight medial axes at serifs, thus eliminating some differences between fonts. It may be an undesirable operation for engineering drawings.

This operation is illustrated by Fig. 7.14. Fig. 7.14(a) shows a line ending with a serif.  $P_i$  and  $P_j$  represent the limits of the line ending and the dotted line indicates the resulting skeleton. Clearly,  $|P_{i-1}P_{j+1}|^2 < |P_iP_j|^2$  and Fig. 7.14(b) indicates that  $P_i$  and  $P_j$  have been moved to the concave points.

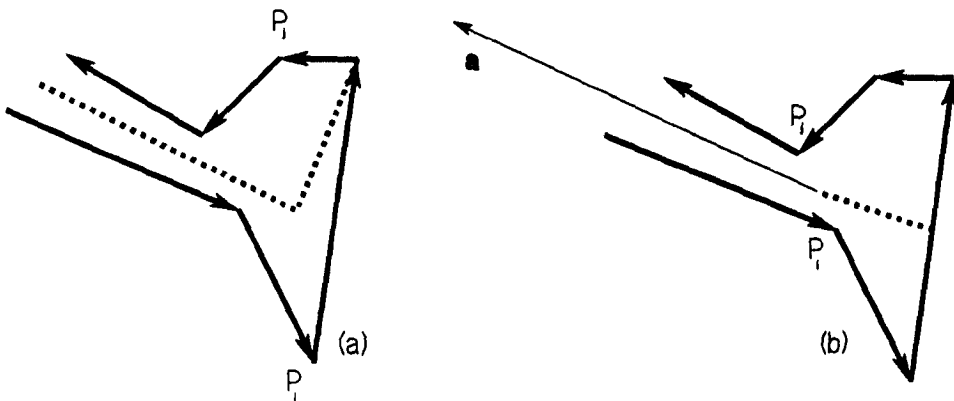


Fig. 7.14. Constructing the skeletal line at a line ending.

Where a line ending consists of more than two endpoints, part of the skeleton is constructed immediately. On the line ending in Fig. 7.14(b),  $P_i$  and  $P_j$ ,

represent the new limits of the line ending. Let  $\mathbf{a}$  be the vector  $(P_{i-})+(P_{j+})$ , or  $-(P_{i+})-(P_{j-})$ , depending on which pair is most anti-parallel. A skeletal line is constructed between the midpoint of  $P_i$  and  $P_j$  and the edge point or midpoint of an edge line between  $P_i$  and  $P_j$ , such that the skeletal line is most parallel to  $\mathbf{a}$ . The skeletal line is shown dotted in Fig. 7.14(b).

Fig. 7.15 shows the outlines of Fig. 7.11 with the resulting skeletal lines shown dotted. Nearest point arrows associated with line endings have been deleted to reduce clutter. Line endings which do not contain more than 2 edge points are shown as thick lines as in Fig. 7.11.

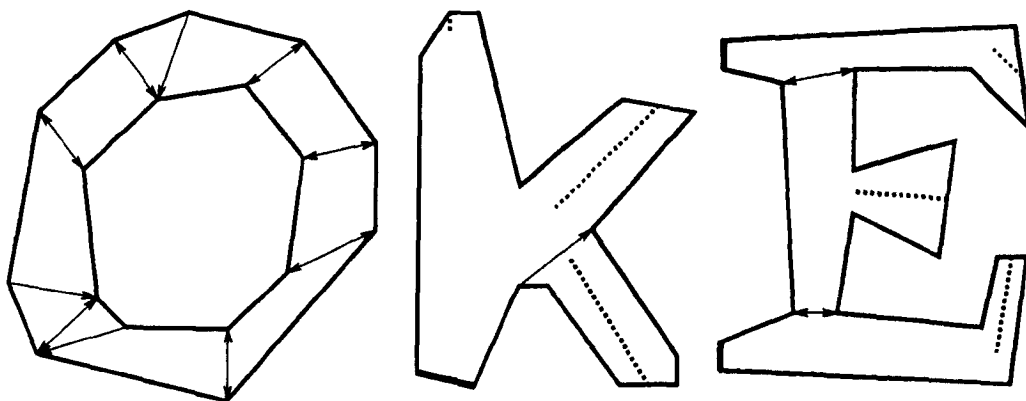


Fig. 7.15. The result after completing line endings.

When the line endings are complete, all non-endpoints are tested to see if the line to their nearest point crosses any other nearest point line. If there is any crossover, then it must be removed before processing can continue. If the two points which have crossing nearest lines are adjacent, then they are converted to endpoints. Otherwise, the points exchange their nearests. This simple procedure removes the vast majority of crossovers. If any remain after these changes, then the object cannot be thinned and is rejected.



## 7.5. CONSTRUCTING THE SKELETON

Skeletal lines are constructed starting from line endings. When each line has been traced to a junction, other lines which do not have a free end are processed. Finally, lines are connected at junctions.

### 7.5.1. TRACKING LINES FROM ENDPOINTS

Starting at each line ending, a skeletal line is constructed while the nearest points are *continuous*. The concept of continuity is illustrated in Fig. 7.16. In each of Fig. 7.16(a), (b) and (c),  $N_i = P_j$  and  $N_j = P_i$ . In Fig. 7.16(a),  $N_{i+1} = P_{j-1}$  and  $N_{j-1} = P_{i+1}$ . Fig. 7.16(b) is also continuous since  $N_{j-1} = P_i$ . Fig. 7.16(c) shows a discontinuity, since there is no nearest relation anywhere between  $P_{i+1}$  and  $P_j$  or  $P_{j-1}$  or between  $P_{j-1}$  and  $P_i$  or  $P_{i+1}$ .

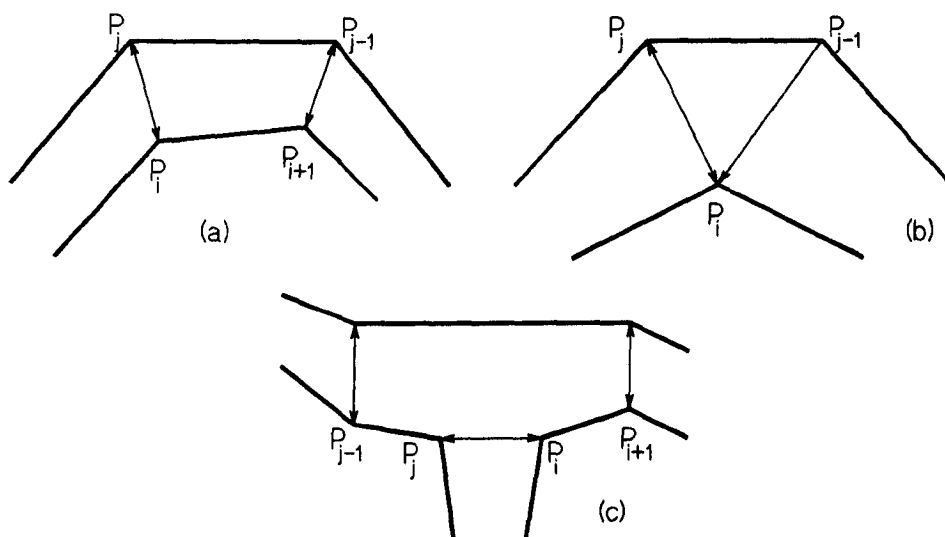


Fig. 7.16. (a), (b) Examples of continuity. (c) An example of discontinuity.

The skeleton is constructed by connecting midpoints of the lines which join points to their nearests. Generation of skeletal lines continues until a discontinuity is found. Discontinuities occur at junctions and ends of lines. The process is also terminated if one of the side points is an endpoint or it has already been used. This prevents the situation where lines could cross without being detected.

### 7.5.2. OTHER CONTINUOUS REGIONS

Once all line endings have been followed to junctions, skeletal lines are generated for the remaining regions of continuity. The "O" in Fig. 7.15 has no

line endings. Some other characters which have continuous regions that are not connected to line endings include "e", "a" and "P".

For all unused points remaining in the object, continuity of nearests is tested. Where there is continuity, the line is followed in both directions until a junction is found.

After processing continuous regions, there may still remain some points on the outline which have not been used. This will happen mainly in the case of a very short line between 2 junctions, as with the "E" in Fig. 7.17. An attempt is made to construct skeletal lines from these points. The resulting line may be as short as a single point, but the step produces an important partitioning of junctions. Partitioning junctions in this way simplifies the closure operation described in section 7.6.2.

The amount of skeleton present after this stage of processing, is shown in Fig. 7.17 as dotted lines. The next step is to isolate line junctions.

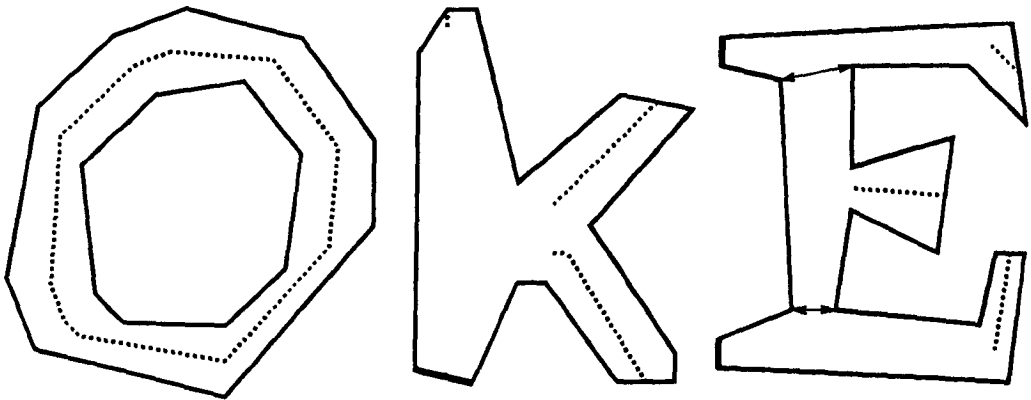


Fig. 7.17. The skeleton present after processing regions of continuity.

## 7.6. LINE JUNCTIONS

Each junction maps to a node in the output graph structure. Consequently, during construction of the graph, when each line ends at a junction, it must be attached to the same node as all the other lines ending at the same junction. This function is performed partly during creation of the skeletal lines. After all skeletal lines have been created, additional phases close junctions and partition them into manageable pieces.

### 7.6.1. CREATION OF JUNCTIONS

When each line ends in a discontinuity or it hits another junction, the last pair of points used are recorded in a structure called a *junction pair*. A junction pair contains information about one line entering a junction:

- (a) The last pair of points used on the line are the *sides* of the junction pair. The left side is the left point as seen from the line looking towards the junction
- (b) The midpoint of the sides is also stored since it gives the coordinates of the end of the line as well as a link to the rest of the line.
- (c) The junction pair also contains links to the junction pairs on the left and right. (If any.)

A completed junction (i.e. a node in the output graph) is thus a circular list of junction pairs. Fig. 7.18 shows the information stored in a junction pair. The junction pair in Fig. 7.18 has a link from the right side to the left side of the other junction pair. When these links form a circular list, the junction is *closed*.

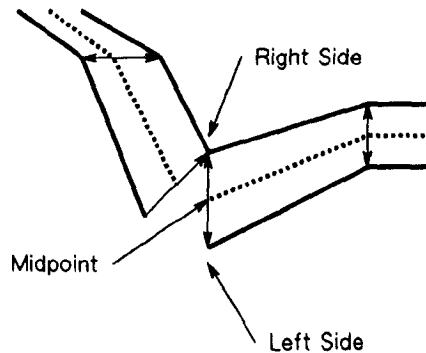


Fig. 7.18. The components of a junction pair.

When the first line to be constructed arrives at a junction, a junction pair is created for it. Whenever a new junction pair is created, its left side is checked against the right sides of all existing junction pairs. If any is found to contain exactly the same point, then it is possible, but not necessarily certain, that the junction pairs belong to the same junction. Similarly, the right side of the new junction pair is checked against the left sides of all existing ones.

In the simple case, at most one match will be found, and that will not previously have been connected to anything else. Fig. 7.19 illustrates the simplest

possible junction. In Fig. 7.19(a), there is a junction of 3 lines, one of which has been followed to the junction. In Fig. 7.19(b), the second line has ended at the junction and has a right side in common with the left side of the first line. In Fig. 7.19(c), the final line closes the loop and the junction is complete.

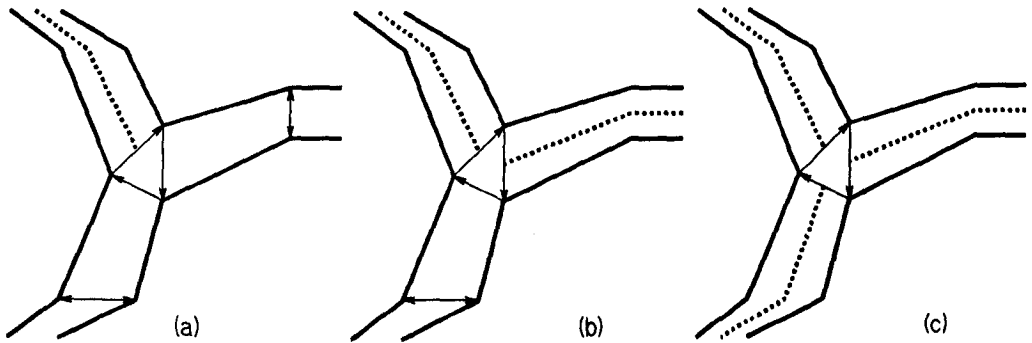


Fig. 7.19. (a) The first line ends at a junction. (b) The second line ends at the junction. (c) The junction is complete.

### 7.6.2. CLOSING JUNCTIONS

Fig. 7.19 was a slightly contrived junction. Since skeletal lines are constructed using midpoints of nearest lines, a junction pair will occur at the last nearest line of a continuous section. Fig. 7.20 shows a more realistic configuration; there is no nearest line at the junction for the third leg. Consequently, the junction pairs, (shown as thin arrows) do not form a closed loop.

In order to close the junction, it is necessary to connect all line endings that are separated by only a single edge line. Ensuring that all outline points are used, as in section 7.5.2, guarantees that there will be only one edge segment separating adjacent junction points within a junction.

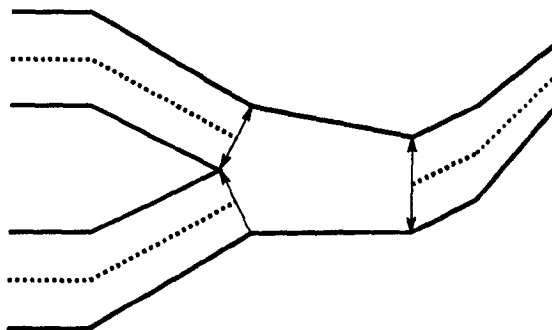


Fig. 7.20. A non-closed junction.

### 7.6.3. ADJACENT JUNCTIONS

When a new junction pair is created, it is possible for there to be more than one other junction pair sharing one of the sides. In Fig. 7.21, four junction pairs share the same edge point. Visually, it is clear which junction pairs belong to each junction.

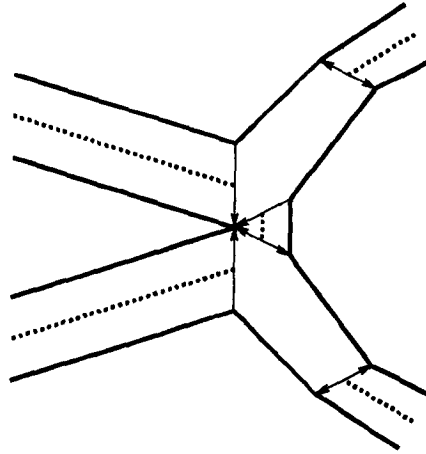


Fig. 7.21. Two adjacent junctions

Depending on the order in which the junction pairs are encountered, it is possible for confusion to arise over which pair belongs to which junction. In order to resolve any uncertainty, a new junction pair is connected such that it has least angle, and in the correct sense, with the connected pair.

Fig. 7.22 shows the example of Fig. 7.17 after this stage of processing. The midpoint of each junction pair is shown as an asterisk, with adjacent junction pairs connected by dashed lines.

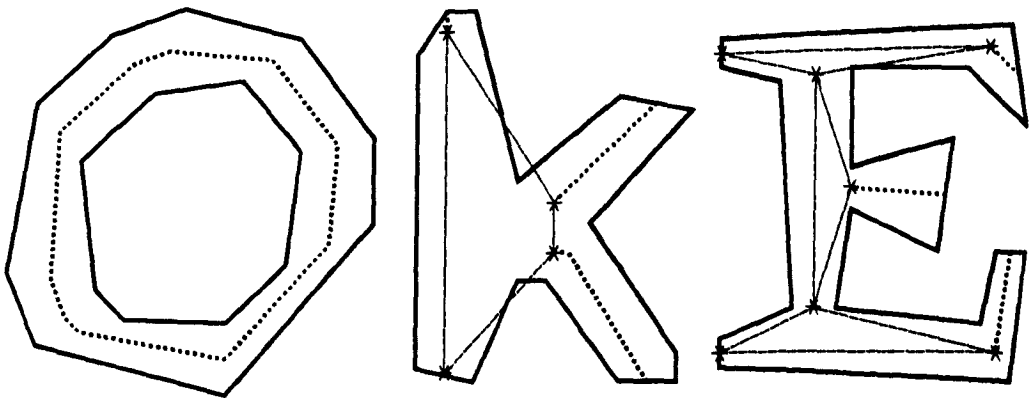


Fig. 7.22 Closed junctions are shown dashed.

## 7.7. CONNECTING JUNCTIONS

The skeletal lines entering each junction must be connected. The treatment depends on the number of lines meeting at the junction; three lines are easier to connect than four or five. Before any attempt is made to connect a junction of four or more lines, it is first partitioned, if possible, into simpler junctions.

### 7.7.1. PARTITIONING JUNCTIONS

During the thinning process, junctions of three or four lines which lie close together may be merged, i.e. the line segment between the junctions can be regarded as part of the junction. By detecting parts of junctions which resemble lines, they can be partitioned into more manageable fragments.

The first test is performed for junctions of four or more lines. An example of a junction with a suitable partition is shown in Fig. 7.23. Let  $P_i$  and  $P_j$  be two points on the outline, such that neither is shared by two junction pairs,  $P_i$  is the left side of a junction pair only and  $P_j$  is the right side of a different junction pair. If two such points exist, and they satisfy

$$|P_{i-1} \times P_{j+1}| < |P_i P_j|^2 \quad \text{and} \quad 2|P_i P_j|^2 < \max(|P_i P_{j+1}|^2, |P_j P_{i-1}|^2),$$

then the junction is split along  $P_i P_j$ . If there is more than one such pair of points then the pair is chosen such that  $|P_i P_j|^2$  is a minimum. If there is no pair of points which satisfy the condition, then there is no partition.

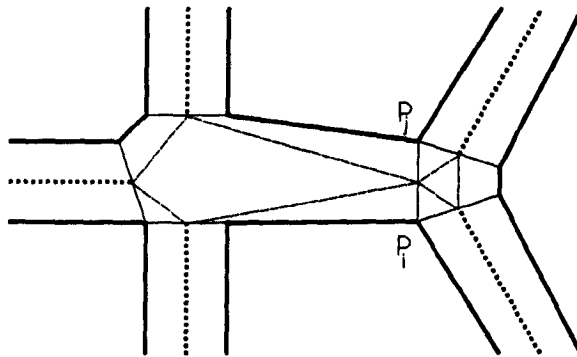


Fig. 7.23. A 5-line junction with a partition.

The test produces a pair of points at a narrow part of the object, at the end of a long line. The result of the partition is shown in Fig. 7.23 by connecting each new junction with dashed lines.

A further attempt is made at partitioning the junction if it contains exactly four lines. Fig. 7.24(a) shows a 4-line junction where the partition can be made. Since the junction has 4 lines there are only two fundamentally different ways in which it can be partitioned into two 3-line junctions. All the thin dotted lines in Fig. 7.24(a) represent possible splits which result in the same pair of lines being in each partition. The thin dashed lines represent the other possible split lines, which would result in the opposite pairs of lines being together.

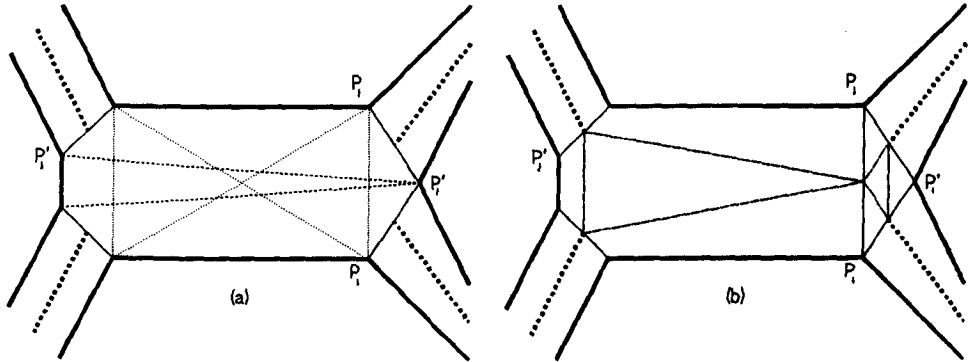


Fig. 7.24. (a) A 4-line junction with the possible partitions. (b) The resulting pair of 3-line junctions.

Let the points at either end of the shortest dotted split line in Fig. 7.24(a) be  $P_i$  and  $P_j$ .  $P_i'$  and  $P_j'$  are at opposite ends of the shortest dashed split line. If

$$3|P_i P_j|^2 < |P_i' P_j'|^2,$$

then the junction is split along  $P_i P_j$ . This is the case for Fig. 7.24(a) with the partitioned junctions shown in Fig. 7.24(b). The test is also made with the pairs reversed and if satisfied, the split is made along  $P_i' P_j'$ .

Fig. 7.25 indicates how the 4-line junction in the "k" of Fig. 7.22 is partitioned into two 3-line junctions.

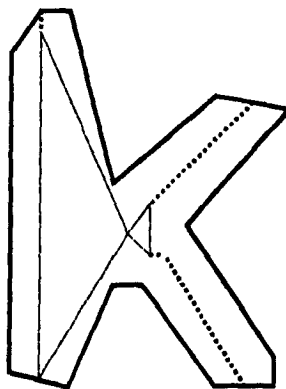


Fig. 7.25. Partitioning the 4-point junction in Fig. 7.22.

### 7.7.2. CONNECTING 4-LINE JUNCTIONS

If the diagonals which join opposite midpoints of junction pairs of a 4-line junction are within  $30^\circ$  of being perpendicular, and their point of intersection is sufficiently near the centre of one of them, then the diagonals are used to connect the junction.

Fig. 7.26 illustrates a typical 4-line junction which was not partitioned by the tests of section 7.7.1. The diagonals between the midpoints of opposite junction pairs are shown as thin dotted lines. The long diagonal intersects the short diagonal, PR, at the point Q, such that

$$4|PQ|^2 > |QR|^2 \text{ and } |PQ|^2 < 4|QR|^2.$$

Therefore, the dotted diagonals will become the skeletal lines used to connect the junction.

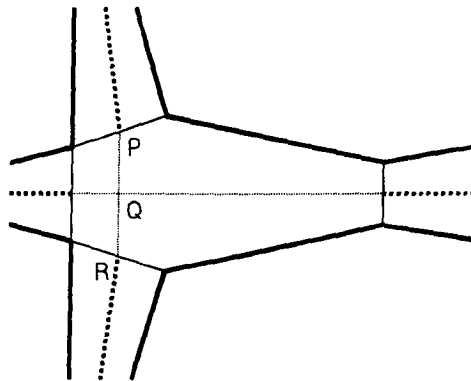


Fig. 7.26. A 4-line junction with perpendicular diagonals.

### 7.7.3. EXTRAPOLATION VECTORS

Those junctions for which the test of Section 7.7.2 is not applicable or fails, are connected using the *extrapolation vector*,  $v$ , for each junction pair in the junction. The extrapolation vector is an estimate of the line direction if it were to continue through the junction. The length of the vector is irrelevant. Let the left and right sides of the junction pair be  $P_i$  and  $P_j$  respectively. The following tests calculate the extrapolation vector for a junction pair.

- (a) If the angle between  $P_{i-}$  and  $P_{j+}$  is less than  $30^\circ$ , and  $P_{i-} \cdot P_i P_{j+1} > 0$  and  $P_{j+} \cdot P_j P_{i-1} > 0$  then  $v = P_{i-} + P_{j+}$ .



- (b) If the angle between  $P_{i+}$  and  $P_{j-}$  is less than  $30^\circ$ , and  $P_{i+} \cdot P_i P_{j-1} > 0$  and  $P_{j-} \cdot P_j P_{i+1} > 0$  and  $P_i$  is not the same point as  $P_j$  or  $P_{j+1}$ , then  $v = -(P_{i+} + P_{j-})$ . If the junction pair also satisfies (a), then the extrapolation vector is the sum of the results in (a) and (b).
- (c) If  $P_i = P_j$ , as may be the case with a single point line ending, then  $v = P_{i-} + P_{j+}$  regardless of the result of (a).
- (d) If the junction pair does not satisfy (a), (b) or (c), then the extrapolation vector is  $P_i P_j$  rotated anticlockwise through  $90^\circ$ .

Fig. 7.27 shows the extrapolation vectors as dotted arrows. The scalar product tests in (a) and (b) guard against the situation of the uppermost line in the junction, where  $P_{4-}$  and  $P_{10+}$  are almost parallel, but do not give a good indication of the line direction.

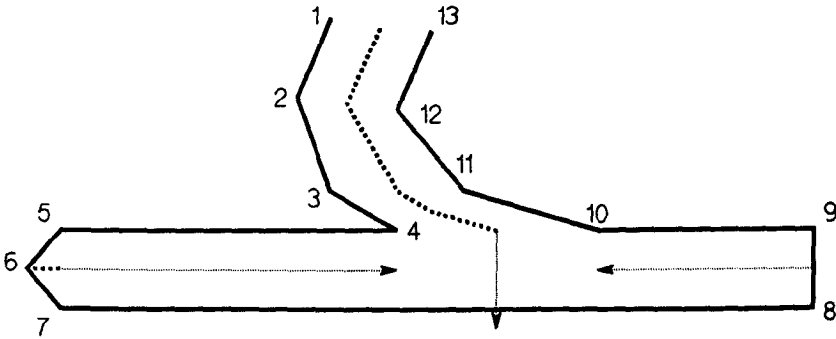


Fig. 7.27. The extrapolation vectors of a junction.

One final test is made on the extrapolation vector before it is used to connect the junction. In Fig. 7.28,  $P_i$  and  $P_j$  are a junction pair, and  $Q$  is the midpoint of  $P_i P_j$ . The extrapolation vector calculated above is  $v$ . The following simple tests detect the case shown where  $v$  may cross the edge and be unrepresentative of the line direction.

- (a) If  $P_i$  is not shared by two junction pairs, and  $v \times (QP_i + P_{i-}) < -0$  then change  $v$  to  $P_{i-}$ .
- (b) If  $P_j$  is not shared by two junction pairs, and  $v \times (QP_j + P_{j+}) > 0$  then change  $v$  to  $P_{j+}$ .

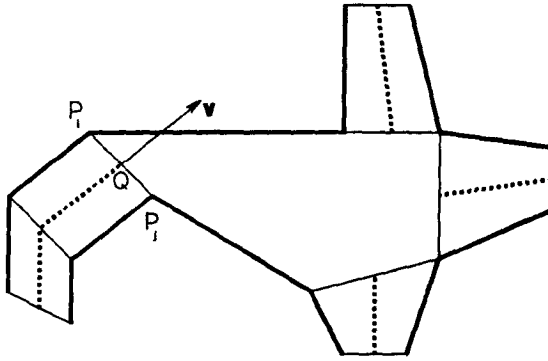


Fig. 7.28. A case of an extrapolation vector which crosses the outline.

#### 7.7.4. CONNECTING JUNCTIONS WITH EXTRAPOLATION VECTORS

The extrapolation vectors are compared pairwise in order to find the pair which are closest to anti-parallel. The pair of line endings with the angle nearest to  $180^\circ$  are joined by a line between their midpoints. All the remaining lines are extended in the direction of their extrapolation vectors until they intersect this same line segment. Those which do not cross the line anywhere along its length are defined to intersect at the end nearest to which they pass. All the lines in the junction are then connected to the mean point of intersection.

Fig. 7.29 illustrates the completion of the junctions in Fig. 7.22.

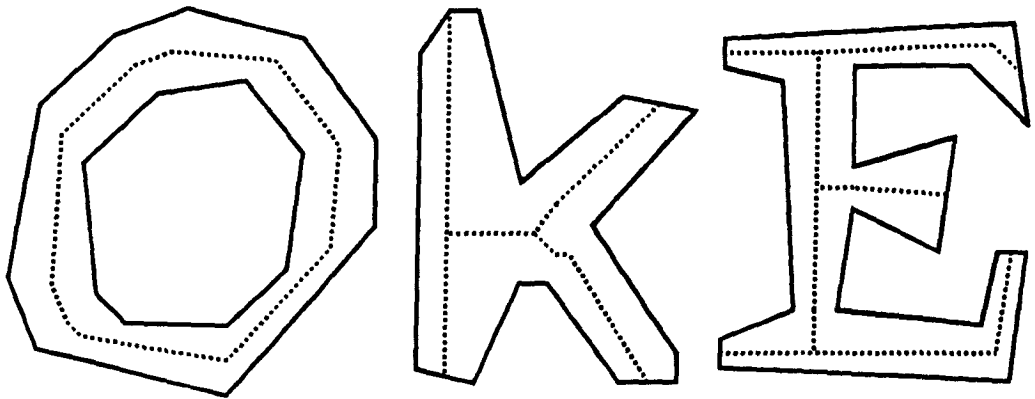


Fig. 7.29. The final result of thinning.

#### 7.8. A NOTE ON OPTIMIZATION

By the foregoing description, the thinning algorithm is heavily squared order, i.e. the processing time is a large multiple of  $n^2$ , where  $n$  is the number of points on the outline. Depending on the implementation, test (a) in section 7.3, for finding nearest points, may result in cubic behaviour. There are however,

several optimizations which can be used to eliminate any possibility of  $n^3$  order time and to reduce the multiple of  $n^2$  significantly.

### 7.8.1. OPTIMIZING THE NEAREST POINT TESTS

Given that  $N_i = P_j$ , a good first guess at  $N_{i+1}$  is  $P_{j-1}$ . Using  $P_{j-1}$  as an initial estimate of  $N_{i+1}$ , if it passes the other tests, the majority of points can be rejected simply by measuring the distance<sup>2</sup>, without performing tests (a) and (b) of section 7.3. This reduces the factor of  $n^2$  considerably.

Test (a) of section 7.3 implies that for each candidate nearest point, it is necessary to search the entire outline list to test for intersections of the line  $P_i N_i$  with an edge. A simple test which uses the same cross products as test (b) will check that the line  $P_i N_i$  lies within the object at points  $P_i$  and  $P_j$ :

- (a) If  $P_i$  is convex it is required that  $P_{i+} \times P_i N_i > 0$  and  $P_i N_i \times P_{i-} > 0$   
If  $P_i$  is concave it is required that  $P_{i+} \times P_i N_i > 0$  or  $P_i N_i \times P_{i-} > 0$
- (b) If  $N_i$  is convex it is required that  $P_i N_i \times N_{i+} > 0$  and  $N_{i-} \times P_i N_i > 0$   
If  $N_i$  is concave it is required that  $P_i N_i \times N_{i+} > 0$  or  $N_{i-} \times P_i N_i > 0$

In the vast majority of cases, if both (a) and (b) above are true, either the nearest point will be correct, or  $P_i$  has no nearest. When the final selection has been made, a single pass over all the points on the outline will verify the correctness of  $N_i$ . If  $N_i$  then fails test (a) of section 7.3, a new nearest point is generated and that is tested. Cubed order behaviour is thus eliminated.

Experimentation has shown that where the line  $P_i N_i$  does intersect one of the outlines,  $P_i$  is likely to be an endpoint. In such cases, the position of  $N_i$  is irrelevant. The test to check for intersections with outlines can therefore be eliminated completely. If an important nearest point is calculated incorrectly, then the junction closure process will fail. This situation can be detected, and the entire process restarted with the checks active.

One further optimization of the nearest point calculations is possible. The nearest point is found only for all concave points on the object. Where a nearest point is convex, the concave point is recorded as being the nearest point to the convex point, if the neardistance<sup>2</sup> is less than any other previous nearest at that point. All convex points which thereby obtain a nearest point are then excluded from the subsequent location of nearest points for the remaining convex points. This heuristic optimization does not guarantee to obtain the correct result for

convex points, but any changes are insignifiicant. In the case of curved objects, such as the "O" in Fig. 7.11, a reduction of almost 50% in the squared order factor can be achieved.

### **7.8.2. INTERSECTING NEAREST LINES**

The test at the end of section 7.4.3 for intersecting nearest lines also fails only in extreme cases. Since it requires squared order time, it can be eliminated and used only in case of a failure of the subsequent processes.

### **7.9. SUMMARY**

Thinning is an extremely useful step in the capture of engineering drawings. The iterative thinning algorithms operate on a binary image bit map and consume an impractically large amount of time. Some faster thinning algorithms have been suggested, but the problem of accurately detecting line endings and junctions has not yet been solved.

The new thinning algorithm operates on a polygonal outline approximation. It commences by locating the nearest point, on the opposite side of the object to each point on the outline. The overall principle of the new algorithm is that discontinuities in the nearest point relations provide accurate detection of line endings and junctions.

Location of the nearest points consumes time proportional to the square of the number of points in the outline, but all the remaining operations are linear. The squared order factor is minimised by several optimizations applied to nearest point generation.

The skeleton is constructed by connecting the midpoints of the lines which join each point to its nearest point, where the nearest point relation is continuous. Junctions are closed and partitioned into convenient parts. Partitioned junctions are then connected by extrapolating each line as it enters the junction.

## 8. RESULTS AND CONCLUSIONS

### 8.1. EDGE EXTRACTION AND POLYGONAL APPROXIMATION

The edge extractor described in chapters 3 and 6, provides a powerful tool for the automatic extraction of text from any kind of document. Fig. 8.1 shows an error diffused reproduction of a portion of a document containing text and pictures.

Bath and Bristol to small cottages located in attractive rural areas, including the Mendip and Cotswold Hills. Major housing schemes are currently in progress and accommodation in the area is not a problem. The supply of rented accommodation in Bristol and Bath compares favourably with other parts of the country, for many of the larger properties in these cities have been converted into small units.

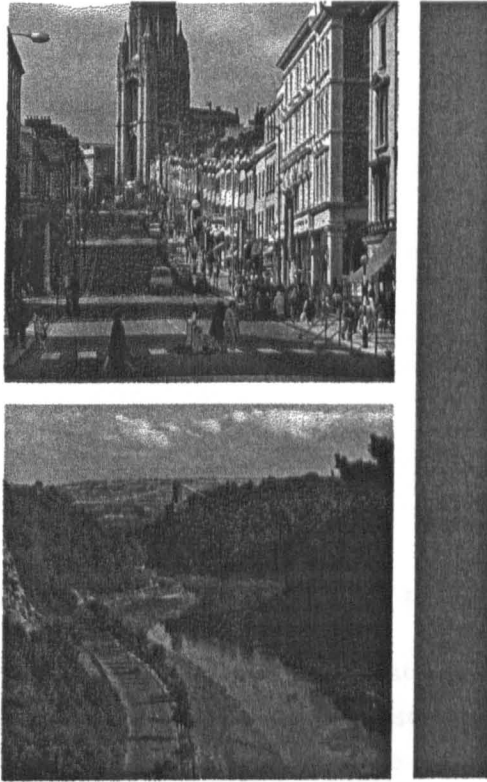


Fig. 8.1. A document containing text and pictures.

In this particular example, the contrast between the text and background is excellent. The edge extractor consequently locates the outlines of the text without a single loss. Conversely, the pictures contain very little with high contrast and the edge extractor therefore ignores them almost completely. The output from the edge extractor, after polygonal approximation, is shown in Fig. 8.2. Note that only an extremely small portion of the pictures have been retained and that the objects do not form logical lines of text. They can therefore be rejected at a later stage.

The full edge extractor of chapter 6 took 37 seconds to produce the result in Fig. 8.2 running on an HP9000/350, a 25MHz MC68020 based machine. The page

contains 338 characters, giving a speed of 9 characters per second. This is around twice as fast as the iterative thinning algorithms; a comparable pre-processing step for character recognition.

Bath and Bristol to small cottages located in attractive rural areas, including the Mendip and Cotswold Hills. Major housing schemes are currently in progress and accommodation in the area is not a problem. The supply of rented accommodation in Bristol and Bath compares favourably with other parts of the country, for many of the larger properties in these cities have been converted into small units.

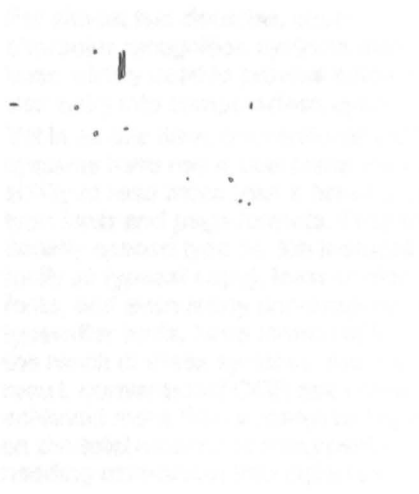


Fig. 8.2. The output of the edge extractor when applied to Fig. 8.1.

The reason why the edge extractor appears so slow is that the text occupies only 25% of the total page area. A page of the same area consisting purely of text would consume only slightly more time, yielding a speed of 25-30 characters per second. The time taken to process the picture is explained by the fact that the edge extractor will find edges which fail after some distance.

The edge extractor is also currently doing far more work than necessary to extract text. It is anticipated that an implementation of the simplified edge extractor as described in chapter 3 will yield at least 100 characters per second on pages of pure text, with a degradation in the presence of pictures, but a reduction in overall page time.

Fig. 1.7 shows a page which contains both black and white text on a grey background. Conventional OCR techniques based on thresholding may be able to locate the black text but the white text is impossible. Fig. 8.3 shows the output of the edge extractor when applied to this page.

Note that three characters and three "i" dots are missing. These losses are due to poor edge path decisions caused by the low contrast on the page. The white text however, has been extracted as easily as the black. The new simple edge path algorithm discussed in chapter 3 is likely to increase the accuracy further with its use of greater look ahead.

For documents conventional OCR can't handle, Kurzweil offers a better approach: ICR.



For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerised systems. Yet in all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

Fig. 8.3. The outlines produced from the page in Fig. 1.7.

## 8.2. FEATURE EXTRACTION

The feature extraction step requires a widely varying training set in order to learn the possible variations in the numerical feature values for each character. The present training set consists of seven different fonts printed on an HP2680 laser printer. The fonts are: Helvetica (Normal, Italic and Bold), Roman (Normal, Italic and Bold) and Courier. The training set is reproduced in Fig. 8.4.

```
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
!"#$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{}`~
```

Fig. 8.4. The training set.

Fig. 8.5 illustrates the result of applying feature extraction to the outlines of Fig. 8.2, using the above training set. Each character is the first choice from the list of characters in the recognized class, and is shown regardless of the success of the match. Each object in Fig. 8.5 was plotted to the rectangular bounding box of the original character image. The result is a somewhat uneven character position and height, caused by the graphics library choosing where in a text box a character should be drawn, and the use of capitals in inappropriate places.

Bath and Bristol to  
Small Cottages  
located in attractive  
rural areas, including  
the Mendip and  
Cotswold Hills. Major  
housing schemes are  
currently in progress  
and accommodation  
in the area is not a  
problem. The supply  
of rented  
accommodation in  
Bristol and Bath  
compares favourably  
with other parts of the  
country. For many of  
the larger properties  
in these cities have  
been converted into  
small units.

Fig. 8.5. The recognized characters from Fig. 8.2.

There are no errors in the letters on this page. Closer inspection reveals an error in punctuation, the first "i" dot in the word "cities" is shown as a single quote. This error will have no effect, since the mark above the "i" will be taken to be the dot. Several characters were rejected however, due to either a high match error, or a low match error on the second choice. Those characters which were rejected are shown in outline in Fig. 8.6. Note that on comparison with Fig. 8.5, none of the rejected characters are incorrect. This situation indicates that the number of rejects could be reduced significantly, without losing accuracy.

The time taken to produce the result in Fig. 8.5 is 4.9 seconds. This represents a speed of approximately 70 characters per second, with a mean of 17 templates tested per unknown character. There is therefore reasonable scope for



increasing the speed further by reducing the number of templates tested for each unknown character.

The reason why there are so many rejected characters in Fig. 8.6, is that there are some ambiguities in the standard templates and that the training set is insufficiently varied. Features with numerical values widely outside the acceptance range are regarded as not matching, resulting in large match errors. The main rejects in Fig. 5.6 arise as follows:

- S Ambiguity with 5.
- C Ambiguity with G.
- M Different to the training set.
- p Ambiguous with P, since they are in different classes.

A collection of scattered lowercase letters and symbols, including 's', 'c', 'M', 's', 'c', 'M', 'J', 'c', 'r', 'ss', 'c', 's', 'T', 'p', 'f', 'c', 'c', 't', 'r', 'f', 'h', 'ry.', 'p', 's', 'c'. The characters are arranged in a roughly circular pattern.

Fig. 8.6. The rejected characters from Fig. 8.5.

The ambiguity of the standard templates which result from the training set in Fig. 8.4 is extremely low for most character pairs. This indicates that the acceptance ranges could be extended considerably without increasing ambiguity. There are some ambiguities however, which pose a definite problem. The current ambiguities are shown in Table 8.1. The grade of the ambiguity gives an indication of the difference between the two templates at the closest point. Grade

1 indicates a negligible difference and grade 4 represents values just below the present ambiguity limit.

Some character pairs include special conditions for an ambiguity to be present. For instance, dot is only confused with the straight kind of comma, rather than the curly version. Each entry in the table also includes an obvious solution to the ambiguity. Clearly, context is useful for nearly all ambiguities, particularly between letters and non-letters, or between two letters. Context is therefore, only included where it is the only solution.

Where "concavity shape" is specified as the solution to an ambiguity, the assertion is that there should exist a measure of shape which is adequate, but the current measure is unsatisfactory for some cases. "Special test" indicates that some test of a specific feature is required, such as the straightness of the line at the top of a 5.

<u>PAIR</u>	<u>GRADE</u>	<u>CONDITIONS</u>	<u>SOLUTION</u>
) l	1		Symmetry, concavity shape.
] l	1		Symmetry, concavity shape.
J ]	1		Symmetry, concavity shape.
J l	1	"1" shaped l	Concavity shape.
5 S	1		Special test.
9 g	1	"9" shaped g	Positioning on line.
[ l	2		Symmetry, concavity shape.
g q	2		Concavity shape.
7 l	2	"1" shaped l	Concavity shape.
U V	2		Special test.
T l	3	"1" shaped l	Symmetry, concavity shape.
C G	3		Symmetry, special test.
/ l	3	Simple line l	Neighbouring character angle.
' .	3	Straight versions only	Special test.
f t	3		Special test.
) J	4		Symmetry, concavity shape.
B 8	4		Symmetry, concavity shape.
E F	4	Serifed F, plain E	Concavity shape.
Z l	4	Heavily serifed l	Concavity shape.
h n	4		Concavity shape.
{ f	4		Symmetry, special test.
' .	4	Small straight comma	Positioning on line.
. .	4	Small straight comma	Grammatical context.

Table 8.1. The present ambiguities, their grade and solution.

It can be seen in Table 8.1, that the number of ambiguities remaining after the introduction of a measure of symmetry and an improved measure of concavity shape will be very small. This success is indicated by the results from the feature extraction stage.



### **8.3. CONCLUSION: THE CURRENT PROBLEMS RESOLVED**

Section 1.5 listed some of the current problems in OCR. The extent to which the results presented here represent a solution to these problems will now be examined.

#### **8.3.1. RECOGNITION SPEED**

The use of an edge extractor to find outlines in the grey level image provides a considerable reduction in time over thinning - by a factor of approximately 40. Attempting to extract features from non-approximated outlines would still consume a great deal of time, as it does with the Yamamoto-Mori<sup>(7)</sup> method. The fast two-stage polygonal approximation algorithm is of considerable advantage in reducing the quantity of data passed to the feature extraction step. As a result, the time consumed in extracting the features is negligible compared to the recognition time.

The feature extraction and recognition process currently runs at around 70 characters per second. This figure is likely to increase to around 150 characters per second with the introduction of symmetry, which could be used to reduce further the number of templates tested for each unknown character.

Edge extraction would then be the slowest process, since it currently runs at around 30 characters per second on dense text. The simplified version described in chapter 3 should increase the speed to around 100-120 characters per second.

The text ordering and final classification steps will consume a small amount of time when compared with the earlier phases. It is therefore reasonable to assume that the completed system will operate at approximately 50 characters per second overall. This is comparable with state of the art pattern matching machines, which represents a considerable achievement for a system which operates on grey level images and uses feature extraction to obtain font independence.

#### **8.3.2. ACCURACY**

It is difficult to give an exact figure for the accuracy of the system at the present stage, since the feature extractor outputs generic character classes, each of which may contain several distinct characters. It should be possible to uniquely identify each character from the generic class and the physical surroundings. The

discussion in this section therefore concentrates on the accuracy of edge and feature extraction.

The accuracy of the edge extractor is at present dependent on the contrast and quality of the source image. Black and white print poses no significant problem, when the print size is 6 point or larger. Under such circumstances the edge extractor correctly locates at least 99.5% of text characters.

Some degradation in performance is inevitable when processing low contrast images, as was demonstrated in Fig. 8.3, where three important objects were lost out of around 600 characters. It is possible however, to prevent most of the losses by the application of a simple edge enhancement operation, before edge extraction.

The accuracy of the feature extractor is demonstrated by Figs. 8.5 and 8.7. Apart from a small number of insignificant punctuation errors, the text in Fig. 8.5 is recognized perfectly; yet the font is unlike any in the training set.

Fig. 8.7 shows around 30 errors in a page of around 600 characters, a 5% error rate. Comparing the errors with the rejected characters in Fig. 8.8 reveals that all except three of the errors are rejected. Apart from the merged characters, which are all rejected, all errors are due to ambiguities listed in Table 8.1, mainly 5/s and are therefore correctable. The three non-rejected errors should also be correctable with a simple context check on the two best matching classes.

It is therefore difficult to put an exact figure on the accuracy, other than somewhere between 95 and 100 percent, with improvement to over 99.5% relatively simple.

### **8.3.3. SEPARATION OF TEXT AND PICTURES**

It has been shown that applying the edge extractor to a grey level image increases the quantity of text that can be read while providing a powerful discriminator between text and non-text. This represents an important solution to the problem of reading multi-media documents, without requiring an operator to mark out the different areas on the page, as is necessary with the Kurzweil reader.<sup>(M2)</sup>

## **8.4. FURTHER WORK**

This thesis has described in detail only a small but important subset of the components of the system block diagram in Fig. 2.1. The remaining units need to be constructed before the OCR system is complete, and some of the existing components need further development.

Most of these remaining units are trivial implementation tasks rather than research topics. It is for this reason that at present these components have not yet been implemented.

### **8.4.1. HALFTONE PROCESSING**

The edge extractor presently has difficulty extracting text from halftoned backgrounds with low contrast. It would be useful, although by no means essential to be able to process images to improve the contrast of the text against the background. There are various possible approaches, such as edge enhancement, texture detection, or combined smoothing and edge enhancement.

### **8.4.2. EDGE EXTRACTION**

The simplified edge extractor described in chapter 3 must be implemented before the system will achieve a competitive speed. Further experimentation with different levels of look-ahead and edge path algorithms may also lead to a further improvement in performance.

### **8.4.3. FEATURE EXTRACTION.**

An extra feature, symmetry, must be added to improve accuracy. A different concavity shape measure would also be useful, such as the radius of curvature, or a measurement of the average angle of turn at each point in the concavity.

Other, more specific tests need to be developed for resolving certain ambiguities, such as S/5. These tests will be applied to discriminate between specific pairs of characters.

### **8.4.4. TEXT ORDERING**

The text ordering process operates at two different levels.

- (a) A typical page contains blocks of text in different fonts and sizes. It is necessary to identify individual text blocks so that ASCII codes can be generated in the correct sequence within each block. Arranging the blocks in logical reading order is difficult and can be left to the user. It is important to identify the blocks however, so that, for instance, text from different columns is not intermixed.

The feature extractor will attempt to classify any spurious objects which may be caused by a picture. The text ordering stage will determine that such objects do not form logical lines of text and can therefore be ignored.

- (b) Once individual blocks have been identified, the characters must be placed into the correct order for reading and the relevant white space characters must be inserted. Characters which are normally formed from several components, such as { !, %, i, j, :, ;, " } must be identified so that the final classification step can use the components to correctly determine the character.

The text ordering phase will also control two other important functions. Knowledge of the character cell size within a block can be used to identify broken characters and reconstruct them. Reconstructed characters can then be passed back to the feature extractor for re-classification. One of the most difficult problems of OCR by feature extraction, broken characters, can thus be overcome.

Current results indicate a high level of rejection of joined characters. This information can be used by the text ordering stage to segment such objects in an attempt to locate the individual characters. As with broken characters, results must be passed back to the feature extractor for re-classification. The result can be used to determine whether other segmentations need to be tested.

#### **8.4.5. FINAL ASCII CLASSIFICATION**

The final ASCII classification step uses the spatial information from the text ordering step to choose a specific character from a generic classification. Where the two best matching classifications are close in match rating, a dictionary look-up and other context checks will be used to resolve the ambiguity.

The simple context checks involve operations such as testing the adjacent characters to see if they are letters or digits. This will obtain the correct solution to ambiguities such as 5/S and 1/l, except in rare circumstances such as

"BRISTOL". If no clear distinction can be made, then the character must be rejected in order to avoid a substitution error.

The final classification step bears close resemblance to an expert system. Rules in a knowledge base indicate how each character is expected to lie on a page within the guidelines. The expert system uses the rules to choose the correct character from each class, given the local physical and lexical context.

#### **8.4.6. TEXT REMOVAL**

When text is recognized, the location of the rectangle surrounding each character is known. This information could be used to blank out the text from the original image. The whole of the remaining image could then be stored more easily since blank image is easy to compress by a large factor. The implementation of text removal is trivial.

Text removal has an application in the user correction of rejected characters. By showing the recognized text superimposed in a different colour on the image with the text removed, characters which have been lost by the edge extractor or rejected will be very easy to see and correct.

#### **8.4.7. FONT STORAGE/MATCHING**

Having recognized a letter "A" for instance, it would be useful to go back to the bit map and compare this with other occurrences of "A" to see if the font is the same. If not, the new A could be stored in the font table as a new font. This would enable reproduction of the text in the same font if required. The matching process would be almost identical to most current matrix matching OCR algorithms, except that it would be useful to regard characters which are the same shape but a different colour as the same font.

An interesting problem arises when a new font is read from a small document, since a small piece of text is unlikely to contain the entire character set. Any editing of the document which inserts a new character will require construction of a bit map for the new character which appears to be in the same font. Such automatic font generation would require the ability to infer line thickness, aspect ratios and embellishments from other characters in the font.



#### **8.4.8. CONSTRUCTION OF COMPREHENSIVE TEST DATA**

In order to thoroughly test the performance of the system, it will be necessary to gather together a large set of as many different fonts as can be found. These all need to be scanned, stored and re-keyed in ASCII. An easy way to obtain such a set would be to use a large set of fonts obtainable from most desktop publishing companies.

#### **8.4.9. IMAGE COMPRESSION**

Images which remain after the text has been removed could be compressed. All that is required is the selection of an appropriate algorithm for the type of image to be compressed: grey levels, containing mainly halftone pictures and no text. The halftone dots can be removed by a simultaneous application of smoothing and edge enhancement<sup>(68-69)</sup>, which would leave an image of mainly smoothly changing grey levels.

### **8.5. LONGER TERM RESEARCH**

The topics in section 8.4 are mainly implementation and small research projects. There are however, two interesting further developments, which would require considerably more research effort.

#### **8.5.1. PARALLEL PROCESSORS FOR HIGHER SPEED**

Linear speed-up should be possible using parallel processors of any type, up to a factor of 100 and beyond if necessary. The type of architecture required would be a rectangular array of processors of minimum power equal to the MC68000, not sharing memory, with 4-neighbour communication and perhaps a master processor to control the distribution of data.

Rectangular portions of the image would be shared between the processors for edge extraction, using 4-neighbour communication and some degree of image overlap to deal with objects which lie across the partitions. Each processor could then extract features independently in their own image partition, without any need for inter-processor communication.

The text ordering step would require a little more communication, but at that stage of processing, the volume of data to pass around is reduced considerably. The Inmos Transputer would be an ideal engine for this purpose, though not essential because of the low degree of inter-processor communication

required. A much faster scanner would be required in order to make use of such a large speed-up in processing the images.

### 8.5.2. CAPTURE OF ENGINEERING DRAWINGS

Another possible long term research project would be the capture of engineering drawings. CAD systems are now in common use but there remain thousands of drawings which were made manually. The conversion of these drawings back to CAD database form is still a major research topic. The main problem is one of interpretation. For instance, the system would need to determine when a series of small straight lines or dots constitutes a dotted line, and when they are independent.

The edge extractor and thinner described respectively in chapters 6 and 7 were originally developed for use in character recognition. They are however, equally useful for the capture of engineering drawings.

Fig. 8.9 illustrates the result of the thinning algorithm when applied to the outlines in Fig. 8.3. Compared with text documents, engineering drawings generally contain thinner, longer lines. Thinning is therefore likely to be more accurate than with text. There is also the advantage that compared with text documents, engineering drawings tend to be sparse. Both the edge extractor and thinner could therefore, possibly run in software on a single processor, even on an A0 drawing.

For documents  
conventional OCR can't  
handle, Kurzweil offers a  
better approach: ICR.



For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerised systems. Yet in all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

Fig. 8.9. The result of thinning the outlines in Fig. 8.3.

## 8.6. SUMMARY

Almost every state-of-the-art commercial OCR machine uses some form of matrix matching, resulting in high speed and accuracy, but a severely restrictive range of recognized fonts. Published algorithms contradict the state of the market, by concentrating on feature extraction. Feature extraction should be able to provide font independence, yet the published algorithms have previously been too slow and inaccurate for commercial use.

Both the commercial machines and published algorithms also fail to distinguish between text and non-text images. The usual solution is to indicate to the machine areas of text, so that it can avoid the rest of the page. Documents usually have to be scanned a second time to specifically store the figures.

Using an edge detection algorithm, which scans a grey level image of a page for connected closed edge loops, it is possible to extract the outlines of text in a reasonable time. The majority of non-text objects in a page contain very little of consistently high contrast, so the edge extractor fails to find many useful outlines of anything other than text. An additional capability is gained over all other OCR systems, in that text of any colour can be read from almost any background. In particular, text can be read from shaded boxes and light text can be read as well as dark. The only proviso is that the contrast must be reasonably high.

The edge extractor produces rather jagged outlines, due to the varying edge strength around each character. The chain coded outlines are approximated by polygons using a fast two-stage algorithm, in order to smooth these jagged edges. An extremely fast first stage reduces the number of outline points dramatically for straight lines and produces a good approximation for smooth curves. The second stage is applied selectively to smooth the jagged edges.

The features used in recognition are based on the psychophysical tests carried out by Shillman et. al.<sup>(34-41)</sup> Shillman's twelve functional attributes, which are defined in terms of stick figures, are replaced by five features which can be detected on outline polygons. A large proportion of the ASCII character set can be recognized using only two of the features: concavities and closure. Rather than making use of the character archetypes defined by Shillman,<sup>(34)</sup> which were only given for the upper case letters, the system can be trained on sample characters. It is intended that this process should be used only initially, and that once it has a defined standard character set, the system will not require training on any new font that it will encounter.

The current results show that after training on only seven quite similar fonts, which is not an ideal training set, the recognition algorithm provides greater than 95% accuracy on fonts different to the training set. On fonts similar to the training set, recognition accuracy exceeds 99%. The system currently only produces a generic character class for each character read. The final ASCII classification step however, is a relatively simple operation to utilize the size and positional information from neighbouring characters to determine the exact character.

A more complex edge extraction algorithm may be used to extract arbitrary text and line graphics from a page. This edge extractor is a useful first step for the interpretation of engineering drawings. The nesting tree produced by the edge extractor provides sufficient information to enable conversion of the outlines to a graph structure of the original lines on the page. Conversion is performed by a non-iterative thinning algorithm, which produces such a graph structure from the output of the edge extractor.

## REFERENCES

- 1 Wada H, Takahashi S, Iijima T, Okumura Y, Imoto K  
An electronic reading machine  
*Information Processing. Proceedings, UNESCO Conference Paris 1959,*  
227-232 (1959)
- 2 Grimsdale R.L, Sumner F.H, Tunis C.J, Kilburn T.  
A system for the automatic recognition of patterns  
*Pattern Recognition: Theory, Experiment, Computer Simulations and Dynamic Models of Perception and Discovery.* Ed. Leonard Uhr, 317-338 Wiley (1966)  
Also:*IEE Proceedings B* 106, 210-221 (1959)
- 3 Harmon Leon D.  
Automatic recognition of print and script  
*IEEE Proceedings* 60 no. 10, 1165-1176 (1972)
- 4 Ullmann J.R.  
*Pattern Recognition Techniques.* Butterworths, London. (1973)
- 5 Ullmann, J.R.  
Picture analysis in character recognition.  
*Topics in Applied Physics* 11, 295-343 (1976)
- 6 Suen C.Y, Berthod M, Mori S.  
Automatic recognition of handprinted characters - the state of the art  
*IEEE Proceedings* 68 no. 4, 469-487 (1980)
- 7 Yamamoto K, Mori S.  
Recognition of handprinted characters by an outermost point method  
*Pattern Recognition* 12 no. 4, 229-236 (1980)
- 8 Yamamoto K, Saito T.  
Pattern Reading System  
*U.S. Patent No. 4566124,* (Jan 21 1986)
- 9 Kahan S, Pavlidis T, Baird H.S.  
On the Recognition of Printed Charaters of Any Font and Size.  
*IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9 no.*  
2, 274-288 (March 1987)

- 10 Pavlidis T.  
*Algorithms for Graphics and Image Processing*. Computer Science Press,  
116-120 (1982)
- 11 Pavlidis T.  
A Vectorizer and Feature Extractor for Document Recognition  
*Computer Vision, Graphics and Image Processing* **35 no. 1**, 111-127 (1986)
- 12 Duda R.O, Hart P.E.  
Use of the Hough transform to detect lines and curves in pictures  
*Graphics and Image Processing* **15**, 11-15 (1972)
- 13 Smith R.W.  
Computer Processing of Line Images: A Survey  
*Pattern Recognition* **20 no. 1**, 7-15 (1987)
- 14 McCormick J.  
Text Scanners for the IBM PC  
*BYTE Magazine*. April 1987, 233-238
- 15 Ross, D.J.  
Extraction of Features from a Pattern  
*European Patent Application*. No. 86307868.9. (Filed 10 Oct 1986)
- 16 Morgera S.D.  
Toward a Fundamental Theory of Optimal Feature Selection Part I  
*IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6*,  
601-616 (1984)
- 17 Decell H.P.Jr, Quiereir J.A.  
An Iterative Approach to the Feature Selection Problem  
*Purdue University Conference on Machine Processing of remotely Sensed Data  
Proceedings*, 3B1-3B12 (1972)
- 18 Bidisaria H.B.  
Least Desirable Feature Elimination in a General Pattern Recognition  
Problem  
*Pattern Recognition* **20 no. 3**, 365-370 (1987)
- 19 Floyd R, Steinberg L.  
An Adaptive Algorithm for Spatial Gray Scale

- Society for Information Display. *International Symposium digest of technical papers*. 36 (1975)
- 20 Jain A.K.  
Image data compression: a review  
*IEEE Proceedings* 69 no. 3, 349-389 (1981)
  - 21 Levialdi S.  
Finding the edge  
*Digital Image Processing*. Proceedings NATO advanced study institute Bonas France, 23 June-4 July 1980, 105-148 (1980)
  - 22 Roberts L.G.  
Machine perception of 3 dimensional solids  
*Optical and Electrooptical Information Processing*, Eds. J.T. Tippert et. al. MIT press, 159-197 (1965)
  - 23 Prewitt J.M.S.  
Object Enhancement and Extraction  
*Picture Processing and Psychopictorics*, Eds. B.S. Lipkin, A. Rosenfeld. Academic Press. New York, 75-150 (1970)
  - 24 Chaudhuri B.B, Chanda B.  
The equivalence of best plane fit with Roberts, Prewitts and Sobels gradient for edge detection and a 4-neighbour gradient with useful properties  
*Signal Processing (Netherlands)* 6 no. 2, 143-151 (1984)
  - 25 Bornat R, Brady J.M.  
Using knowledge in the computer interpretation of handwritten FORTRAN coding sheets  
*International Journal of Man-Machine studies* 8, 13-27 (1976)
  - 26 Bornat R, Brady J.M.  
Finding blobs of writing in the Fortran coding sheets project  
*Proceedings, 2nd AISB Conference, Edinburgh, 1976, 47-55.*
  - 27 Brady J.M, Wielinga B.J.  
Seeing a pattern as a character  
*Proceedings, 2nd AISB Conference, Edinburgh, 1976, 63-73*

- 28 Montanari U.  
A note on the minimal length polygonal approximation to a digitized contour  
*Communications of the Association for Computing Machinery* 13, 41-47 (1970)
- 29 Pavlidis T, Horowitz S.L.  
Segmentation of plane curves  
*IEEE Transactions on Computers* C-23, 860-870 (1974)
- 30 Jimenez J, Navalon J.L.  
Some experiments in image vectorization  
*IBM Journal of Research and Development* 26 no. 6, 724-734 (1982)
- 31 Sklansky J, Gonzales V.  
Fast polygonal approximation of digitized curves  
*Pattern Recognition* 12, 327-331 (1980)
- 32 Hung S.H.Y, Kasvand T.  
Critical points on a perfectly 8- or 6-connected thin binary line  
*Pattern Recognition* 16 no. 3, 297-306 (1983)
- 33 Chaudhuri B.B, Kundu M.K.  
Digital line segment coding: A new efficient contour coding scheme  
*IEE Proceedings Part E* 131 no.4, 143-147 (1984)
- 34 Shillman R.J.  
*Character Recognition Based on Phenomenological Attributes: Theory and Methods*  
PhD. Thesis, Massachusetts Institute of Technology. (1974)
- 35 Blesser B, Shillman R, Kuklinski T, Cox C, Eden M, Ventura J.  
A theoretical approach for character recognition based on phenomenological attributes  
*International Journal of Man-Machine Studies* 6 no. 6, 701-714 (1974)  
Also: *1st International Joint Conference on Pattern Recognition*, Washington D.C, 30 Oct - 1 Nov 1973, 33-40 (1973)
- 36 Shillman R, Kuklinski T, Blesser B.  
Experimental methodologies for character recognition based on phenomenological attributes  
*2nd International Joint Conference on Pattern Recognition*, Copenhagen, Denmark, 195 (1974)



- 37 Blesser B.A, Kuklinski T.T, Shillman R.J.  
Empirical tests for feature selection based on a psychological theory of character recognition  
*Pattern Recognition* **8** no. 2, 77-85 (1976)
- 38 Shillman R.J, Kuklinski T.T, Blesser B.A.  
Psychophysical techniques for investigating the distinctive features of letters  
*International Journal of Man-Machine Studies* **8** no. 2, 195-205 (1976)
- 39 Naus Mary J, Shillman Robert J.  
Why a Y is not a V: A new look at the distinctive features of letters  
*Journal of Experimental Psychology, Human Perception and Performance* **2** no. 3, 394-400 (1976)
- 40 Suen C.Y, Shillman R.J.  
Low error rate optical character recognition of unconstrained hand printed characters based on a model of human perception  
*IEEE Transactions on Systems, Man & Cybernetics* **SMC-7** no. 6, 491-495 (1977)
- 41 Shillman R.J, Babcock R.T.  
Preliminary steps in the design of optical character recognition algorithms  
*IEEE Computer Society Conference on Pattern Recognition & Image Processing*, Jun 6-8 1977, Troy, New York, USA, 327-331 (1977)
- 42 Cox C.H.III, Cougeinoux P, Blesser B, Eden M.  
Skeletons:A link between theoretical and physical letter descriptions  
*Pattern Recognition* **15** no. 1, 11-22 (1982)
- 43 Anderberg M.R.  
*Cluster Analysis For Applications*. Academic Press, New York, USA. (1973)
- 44 Hartigan J.A.  
*Clustering Algorithms*. John Wiley, New York, USA. (1975)
- 45 Stentiford F.W.M.  
Automatic Feature Design for Optical Character Recognition Using an Evolutionary Search Procedure  
*IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-7** no. 3, 349-355 (1985)

- 46 Danielsson Per-Erik  
An improved segmentation and coding algorithm for binary and non-binary images  
*IBM Journal of Research and Development* 26 no. 6, 698-707 (1982)
- 47 Sherman H.  
A quasi-topological method for the recognition of line patterns  
*Information Processing. Proceedings, UNESCO Conference Paris 1959*,  
232-238 (1959)
- 48 Minneman M.J.  
Handwritten character recognition employing topology, cross correlation and decision theory  
*IEEE Transactions on Systems Science & Cybernetics SSC-2 no. 2*, 86-96 (1966)
- 49 Tou J.T, Gonzalez R.C.  
Recognition of handwritten characters by topological feature extraction and multilevel categorization  
*IEEE Transactions on Computers C-21*, 776-785 (1972)
- 50 Beun M.  
A flexible method for automatic reading of hand-written numerals: I. General description of the recognition method  
*Philips Technical Review* 33 no. 4, 89-101 (1973)
- 51 Montanari U.  
Continuous skeletons from digitized images  
*Journal of the Association for Computing Machinery* 16 no. 4, 534-549 (1969)
- 52 Rosenfeld A, Pfaltz J.L.  
Sequential operations in digital picture processing  
*Journal of the Association for Computing Machinery* 13, 471-494 (1966)
- 53 Davies E.R, Plummer A.P.N.  
Thinning algorithms: A critique and a new methodology  
*Pattern Recognition* 14, 53-63 (1981)
- 54 Hilditch C.J.  
Linear skeletons from square cupboards

- Machine Intelligence 4*. Eds. B Meltzer, D Michie, Edinburgh University Press, 403-420 (1969)
- 55 Naccache N.J, Shinghal R.  
An investigation into the skeletonization approach of Hilditch  
*Pattern Recognition* 17 no. 3, 279-284 (1984)
- 56 Naccache N.J, Shinghal R.  
SPTA:A proposed algorithm for thinning binary patterns  
*IEEE Transactions on Systems, Man & Cybernetics SMC-14* no. 3, 409-418 (1984)
- 57 Dinneen G.P.  
Programming pattern recognition  
*Western Joint Computer Conference 1955*, 94-100
- 58 Rutovitz D.  
Pattern recognition  
*Journal of the Royal Statistical Society Series A* 129, 504-530 (1966)
- 59 Saraga P, Woollons D.J.  
The design of operators for pattern processing  
*IEE Conference on Pattern Recognition, July 1968 CP42* 106-116 (1968)
- 60 Alcorn T.M, Hoggar C.W.  
Pre-processing of data for character recognition  
*Marconi Review* 32, 61-81 (1969)
- 61 Stefanelli R, Rosenfeld A.  
Some parallel thinning algorithms for digital pictures  
*Journal of the Association for Computing Machinery* 18 no. 2, 255-264 (1971)
- 62 Deutsch E.S.  
Thinning algorithms on rectangular, hexagonal and triangular arrays  
*Communications of the Association for Computing Machinery* 15 no. 9, 827-837 (1972)
- 63 Beun M.  
A flexible method for automatic reading of hand-written numerals: II. The thinning procedure and determination of the special points  
*Philips Technical Review* 33 no. 5, 130-137 (1973)

- 64 Arcelli I.C, Cordella L, Levialdi S.  
Parallel thinning of binary pictures  
*Electronics Letters* 11 no. 7, 148-149 (1975)
- 65 Tamura H.  
A comparison of line thinning algorithms from digital geometry viewpoint  
*4th Interational Joint Conference on Pattern Rocognition*, Kyoto Japan, 7-10  
Nov 1978, 715-719 (1978)
- 66 Suzuki S.  
Binary Picture Thinning by an Iterative Parallel Two-Subcycle Operation  
*Pattern Recognition* 20 no. 3, 297-307 (1987)
- 67 Shapiro B, Pisa J, Sklansky J.  
Skeleton Generation from x-y boundary sequences  
*Computer Graphics & Image Processing* 15 no. 2, 136-153 (1981)
- 68 Liao H.H.  
Electronic Rescreen Technique for Halftone Pictures  
*U.S. Patent no. 4259694*. (March 31 1981)
- 69 Hsieh R.C.  
Edge Extraction Technique  
*U.S. Patent no. 4638369*. (Jan 20 1987)

**REFERENCED MACHINES**

**M1 DEST Corporation**

**The Dest Workless Station**

**DEST Corporation, 1201 Cadillac Court, Milpitas, CA 95035. (408) 946-7100**

**M2 Kurzweil Computer Products**

**The Kurzweil 4000 Page Reader**

**Kurzweil Computer Products Ltd. Unit 8, Suttons Industrial Park,  
READING RG6 1AZ. (0734) 668421**

**M3 TOTEC**

**PRODATA TO-3000**

**TOTEC USA, Northridge, CA USA.**

**M4 FORMSCAN**

**The Formscan Pagereader (Palantir Compound Document Processor)**

**FormScan Ltd., Apex House, West End, Frome, Somerset. BA11 3AS. (0373)  
61446**

