RECOGNITION OF SIMPLE 3-D OBJECTS BY THE USE OF SYNTACTIC PATTERN RECOGNITION

by

GABRIEL GABRIELIDIS

A Master's Thesis submitted in
partial fulfilment of the requirements
for the award of Master of Philosophy
of the Loughborough University of Technology

June 1982

CONTENTS

ACKNOWLEDGEMENTS

I would like to thank Professor D.J. Evans for his encouragement to undertake this project.

I am also deeply indebted to my supervisor, Dr. C.J. Hinde, for his assistance at difficult points, his successful suggestions, and his guidance throughout the project.

Finally, I am grateful to all members of staff for being helpful and especially:-

Dr. C.H.C. Machin for his patience in fixing the hardware and his valuable comments,

Dr. R.L. Winder and Mr. S. Bedi for their advice in programming,

and technician, Mr. H.T. Mawson, for keeping the hardware operational.

σε ολα τα μελη της

οικογενειας μου για

την ηθικη και υλικη

τους συμπαρασταση

και ειδικα

στους γονεις μου

ελενη και θαναση

στον κωστα

και στην θεια φροσω

ABSTRACT

This project is an attempt to supply the existing hardware with adequate software, in order to develop a system capable of recognizing 3-D objects, bounded by simple 2-D planes, which in turn are bounded by straight lines.

Chapter 1 contains an introduction to pattern recognition and a survey of 3-D recognizers. Chapter 2 describes the hardware and the process of feature extraction, by the techniques of averaging, edging and isolation. Chapter 3 deals with the tracing of boundaries and contains criteria for the detection of vertices. An introduction to syntactic pattern recognition is given in Chapter 4. The 3-D and 2-D recognizers together with an introduction to PROLOG (the programming language in which they are written) are presented in Chapter 5. Finally, conclusions, results and suggestions for further improvement are given in Chapter 6.

The whole procedure is divided into three parts i) preprocessing, ii) vertex-detection, iii) recognition.

Preprocessing:- A T.V. camera pointed at the object to be recognized, takes a picture and sends it to a digitizer, which digitizes it and stores it in a frame-store. The digitized picture is then sent to a microcomputer system for further processing. The main figure of the picture is decomposed to its 2-D sides and every one of them is *averaged* and *edged.*

Vertex detection:- The boundary of every preprocessed side is traced by a follower and its vertices are detected and recorded according to a number of criteria. At the end of the tracing, a vertex-array is formed containing the vertex-coordinates for every 2-D side of the main object. This array is processed by a minicomputer, in order to verify the *real-vertices* of each 2-D shape and eliminate all those caused by noise or distortion. Finally a number of clauses are prepared, for the next phase of recognition.

Recognition:- The 3-D recognizer is based on the principles of syntactic pattern recognition. The clauses formed in the previous phase, are lists of the main components - *primitives* - which compose three basic 2-D shapes. These lists are tested against the structure of the basic shapes, and if they are described by one of them, are classified as being: a *triangle,* a *quadrilater* or *other.* The structure of a 3-D object is considered to be a combination of triangles and quadrilaterals. Thus a figure is classified into one of ten classes if it is described by its corresponding structure. Otherwise the 3-D object is classified as *other.* A 2-D recognizer similarly, performs a further classification of 2-D shapes.

The procedure was tested, both as a whole, and with respect to each of its three main parts. As a whole it works perfectly in the case of 2-D figures, and it has satisfactory results in the case of 3-D objects. Although the last two parts, tested separately, cope perfectly with every

2-D and 3-D object which they cover, there is some weakness in the isolating technique in use, to decompose the 3-D object into its actual 2-D sides. The latter requires very good lighting conditions and suitable camera settings in order to accomplish its task.

Chapter 1

INTRODUCTION

1.1 PATTERN RECOGNITION

Pattern Recognition is a branch of a broader field called Artificial Intelligence. Although some people working on it prefer not to define it, a very simple definition[1] of Pattern Recognition is:-

Pattern Recognition is the categorization (or classification) of input data into identifiable classes via the extraction of significant features or attributes of the data, from a background of irrelevant detail.

By defining the very important term of Pattern Class as, a category determined by some given common attributes, an alternative definition of Pattern Recognition could be:-

Pattern Recognition is the process of classifying sensory information into mutually exclusive categories.[2]

Recognition is regarded as a basic attribute of human beings, as well as other living organisms. A pattern is the description of an object. Both are very important for learning which is one of the most significant functions leading towards development. During the second part of the 20th century there is a tendency in humans to employ *intelligent* machines, in order to relieve their brains from doing jobs tiresome and time consuming. On the other hand, the memory storage has been quite a problem, especially considering the increasing rate of information that is generated. The computer - taking into account its tremendous development - proved a very useful solution. Its ability to store and process huge quantities of information at a very high speed, opened new dimensions to human knowledge and pushed forward all the sciences. The next step for the computer was towards recognition and learning. This is why many sicences, irrelevant to computers at first sight like:- statistics, phychology, linguistics, biology, taxonomy, switching theory, communication theory, control theory, operational research etc. have contributed to the area of Pattern Recognition.

1.2 APPROACHES TO THE PROBLEM OF PATTERN RECOGNITION - APPLICATIONS

During the past twenty years there has been a considerable growth of interest in problems of Pattern Recognition. This interest has created an increasing need for methods and techniques for use, in the design of Pattern Recognition systems. Many different approaches have been proposed, most of which deal with the decision-theoretic or discriminant method. Recently, probably because of the picture recognition or scene analysis, the syntactic or structural approach has been proposed, and some preliminary results from applying it, have shown it to be quite promising.

The syntactic approach[3] attempts to draw an analogy between the structure of patterns and the syntax of the language, by emphasising the structural description of the patterns. Mathematical linguistics constitute a very useful tool because of its syntactic nature. However the syntactic approach contains other non-linguistic methods. The description of patterns is based on class distribution or density functions in the decision-theoretic approach, syntactic rules or grammars in the syntactic one. The effectiveness of each approach depends on the particular problem and often mixed approaches need to be applied. It is difficult sometimes to distinguish sharply between syntactic and non-syntactic pattern recognizers.

Another interesting point in most Pattern Recognition systems is the processes that preceed the manipulation of the information by the computer. An image which may be derived from any one of a number of sources, acoustic, light or electronic, is broken down into a binary sequence or matrix. For example a camera produces a two-dimensional image which gives rise to a matrix. The digitized information gained from that image constitutes the pattern to be examined. Once the digitized image has been produced, it is fed to a computer, which compares the incoming patterns with sets of stored information in its memory and classifies them accordingly. An alternative technique, derives the pattern into a number of main components, called *primitives*, which are sufficient for the description of the main pattern. This time the categorization of the pattern is achieved by comparing its structure to those of pre-specified models. Both techniques involve computer *learning* which can be achieved either manually or automatically.

The latest development of RAM's (Random Access Memory) and microprocessors increased both the learning ability of the system and the speed of data processing. The technique of parallel processing dedicates a single pixel to its own microprocessor.[4]

The applications of Pattern Recognition techniques are numerous. Some of them are:-

Design or program of machines that read printed or typewritten characters. Most of the banks use an automatic pattern classification system to read the code characters of ordinary bank cheques. Recognition of hand-written characters or words used by the Post Office of some countries.

General medical diagnosis by screening electrocardiograms and electro-encephalograms. Identification of fingerprints and interpretation of photographs used by the police. Identification of faults and defects in mechanical devices and manufacturing processes. Automatic analysis and classification of chromosomes. Multispectral scanners located on aircrafts, satellites and space stations need to process and analyse the large volumes of information they receive. Some interesting applications are crop inventory, crop disease detection, forestry, geological and geographical studies, weather prediction, classification of seismic waves and scores of others. Sound recognition is also under development, moving towards recognition of spoken words and oral communication between computer and humans.

1.3 A SURVEY OF 3-D OBJECT RECOGNITION SYSTEMS

This paragraph refers to some of the work done on the recognition of 3-D objects by computer systems. Most of the algorithms and techniques given below are the first stage of a more sophisticated operation called *scene analysis*.

Robert's program:-[5] This program models objects in a scene as part of the recognition task, not as a separate process which has to be completed before recognition can begin. The basic mechanism of this system is to describe blocks in terms of unions of transformed primitive blocks (also called models). The primitive blocks he uses are a cube, a wedge and a hexagonal prism. There are two main parts to the system, entirely independent.

a) producing a line drawing from a photograph

b) producing a 3-D object list from a line drawing.

A complete line drawing, in the form of lists of lines and end-points, provides the input for the modelling/recognition program. Each end-point has pointers indicating which lines it is connected to in order of angle.

The first step is to find the polygons which make up surfaces of objects, by tracking lines and jumping to adjacent lines at junctions. Convex polygons with 3,4 or 6 sides - *approved polygons* - are looked for because these occur in the 3 primitive blocks and thus could be used as starting points in the modelling process. The second step is to match transformed primitive blocks to part or whole scene blocks. The line drawing is searched for a number of topological features which are the basis for the transformation of primitive blocks. If a transformed model completely fits a group of connected lines then the model is assumed to represent the object. If not a compound object construction procedure is used.

Yoshiaki and Saburo's program:-[6] This program is developed for the eye of a particular intelligent robot and achieves the extraction of the line drawing of 3-D objects. The procedure consists mainly of two processes. First four line drawings of the same objects illuminated from four different directions are sequentially obtained. Second, by applying 2-D logical operations to these line drawings, a complete one is extracted. This method is applicable to more than two polyhedrons in a scene which is difficult by usual methods to represent, because of too small difference of light intensities between the different planes or the effect of shadows.

Yoshiaki and Motoi's program:-[7] This is a recognition procedure with a range finder developed for the eye of the above mentioned robot. A range finder projects a light beam through a vertical slit on the polyhedrons. While the beam is moved in a field of view the picture of each instant is picked up by a TV camera. Based on the 3-D position of the slit image, the objects are recognised and their parameters are obtained. This method determines not only the boundary of the polyhedrons but also the 3-D position of their surface planes. Thus, recognition is reliable compared with one based on the line drawing of the objects. In addition, this range finder may be used to measure the 3-D position of any point for input to the usual recognition methods. The recognition procedure is free from the effects of the arrangements and shadow of objects.

Tenenbaum's program:-[8] This program deals with complex objects and is based on the idea that a robot will not have to describe a scene exhaustively but will have to find specific objects in order to carry out tasks. Vision is considered to be a problem solving process using knowledge about the robot's perceptual abilities and contextual knowledge about its world situation to recognize objects. Each model describes an object in terms of its distinguishing features. There are two classes of features used in the models. The first is obtained from the results of applying perceptual operators to instances of the object. This is performed by interactive fashion. A user outlines part of the image, using a light-pen and tells the program to use the result to update the model of the object. Colour features are recorded in this way. The second class of features describe the object in terms of simple shape characteristics, such as the ratio of height to base area. The system uses sensory inputs: brightness, colour and range. When the system is told to find an object in a scene it proceeds in two stages:-

a) acquisition stage, where the scene image is sampled in a search for characteristic features of the object.

b) validation stage, where each of the possible instances found at the acquisition stage is checked in more detail.

Popplestone and Ambler's program:-[9] This program is designed to form body models automatically using range data. The input is provided by the striper, a triangulation ranging device which enables the 3-D position of all points visible to both camera and a light projector to be calculated. The object to be modelled is placed on a turntable and images are stored from views at several angles of rotation. The construction of the models takes place in four stages:-

a) inspection, which collects stripe data and information about the position and operation of the various parts of the striper system and then reduces the amount of stripe data by segmenting each stripe into a number of straight line segments.

b) plane finding, where the information supplied by the segmenter is used to indicate smoothly joined planes.

c) cylinder finding, which tries to fit cylinders to groups of smoothly joined planes.

d) body model building, where a body model is formed by the union of intersections of two basic primitives. These two primitives are half-spaces and length cylinders. The planes found by the plane-finder are represented by transformed half-spaces; cylindrical faces are represented by transformed infinite cylinders. It is assumed that bodies will have only planar and cylindrical faces.

Shneier's program:-[10] This system also uses the striper to provide the input data. After finding plane and cylindrical faces it uses completely different representation for objects than the previous one. The models for individual objects do not exist as separate entities but are all part of a global data-base which is a semantic net. The nodes of the net represent faces of objects, and the links represent relations between faces. In addition surfaces which are of the same type and dimensions are represented by one node. The modelling operation is not totally automatic, the user supplies a name for the object and a list of relations to be used when constructing the model. The program creates the model, integrating it with the existing data-base. The recognition process proceeds in two parts:-

a) find interpretations for the fragments of a scene

b) find the best interpretation of the scene as a whole.

A range map is processed to find some of the surfaces present in the scene. The surfaces found are matched to nodes in the net and the results of these matches are used to construct a scene graph which contains all the possible interpretations of each surface. In a second stage a special constraint analysis algorithm is used to find the best overall interpretation of the scene.

Nevatia and Binford's program:-(11) This system also uses as input, only range data, obtained by using laser-based triangulation ranging. The basic principle is that bodies are segmented into simpler parts. Models are based on descriptions of these parts and on the connectivity relations between them. The domain of objects modelled included toy dolls and horses, and hand tools. Interestingly the same processing is used to generate the symbolic description of an object for both modelling and recognition. The main primitives used to describe objects are generalised cones. After a first stage of segmentation, the program generates a symbolic description of the object consisting of:-

a) connectivity relations, which are in the form of a semantic net with nodes representing parts and links the relations between them.

b) part descriptions, containing a summary of the size and gross shape of the part which is used for quick matching and more detailed description of the linear cone which is fitted to the part.

c) junction description, consisting of a list of parts connected at the joint in cyclic order and a note of the widest piece at the joint.

d) global properties which are the number of pieces, joints and descriptions of *distinguished pieces* (parts with special properties).

The recognition stage consists of the description codes of distinguished pieces in the scene, which are compared with those of stored models. Those compared successfully are taken on to the matching stage which is performed by growing the graph starting from distinguished pieces.

SUMMARY - CONCLUSIONS

- Pattern Recognition is the process of classifying sensory information into mutually exclusive categories.
- Many sciences have contributed to the area of Pattern Recognition.
- There are two main approaches to the problems of Pattern Recognitions, the decision-theoretic or discriminant method, and the syntactic or structural one.
- The applications of Pattern Recognition are numerous and spread over a large area. Some of them affect every-day life.
- A survey of 3-D object recognition systems reveals that they follow two basic stages:-
  - a) modelling
  - b) recognition.

REFERENCES

1. J.T. Tou, R.C. Gonzalez: *"Pattern Recognition Principles"*, Addison-Wesley Publ.Comp. Inc., Reading, Massachusetts, 1974, Chapter 1.
2. Nils J. Nilsson: *"Adaptive Pattern Recognition: A Survey"*, 1966 Bionic Symposium, Bryton, Ohio (May '66), pp.103-115.
3. K.S. Fu: *"Syntactic Methods in P.P."*, Academic Press, 1974, Preface.
4. Paul Reed: *"Pattern Recognition"*, New Electronics, September 16, 1980, pp.49, 54-58.

Chapter 2

PREPROCESSING METHODS AND FEATURE EXTRACTION

2.1 INTRODUCTION

Preprocessing is defined as a method of representing a scene by a set of numbers.[1] A visual scene might be divided into a raster of cells and then the light intensity of each cell, converted to an electrical signal. However an accurate representation of the complete scene, could well lead to a large set of numbers. Furthermore, the probability density functions needed to represent the scatter of a pattern set produced by such preprocessing, are likely to be very complex.

However, only the significant *features*, are extracted from the sensory information, provided by the mass of complex data resulting from such simple preprocessing schemes. These features would have either binary values indicating their absence (∅) or presence (1), or any numerical value indicating the *intensity* of the feature. The collection of feature values for a scheme is used as the numerical representation, X, for the scene. Feature extraction methods are a major element of the recognition process. Unfortunately there seems to be no general theory to guide the search of relevant features in any given recognition problem. The design of feature extractors is mainly empirical, following different ad-hoc rules, found to be useful in special situations.

The primary input to a visual processor is a rectangular grid of cells, each one of which takes a value from Ø to 63 ~~15~~. These values represent 16 gray levels from white to black respectively. Before any extraction of features takes place from the grid, some preliminary operations need to be done on the grid itself. Their main tasks include, speck removal, gap-filling, thickening, thining, edging and isolation. The first part of the following paragraphs gives a description of the used hardware. In the second part a detailed analysis of the averaging, edging and isolating operations, is attempted.

2.2 THE HARDWARE

The hardware consists of the following main devices: a T.V. camera, a T.V. monitor, a frame store digitizer and the appropriate interface, a microcomputer system, a lineprinter and a V.D.U.

2.2.1 The Camera

This is a SANYO video T.V. camera with a 12.5-75 mm zoom lense of aperture range 1.8-22f.

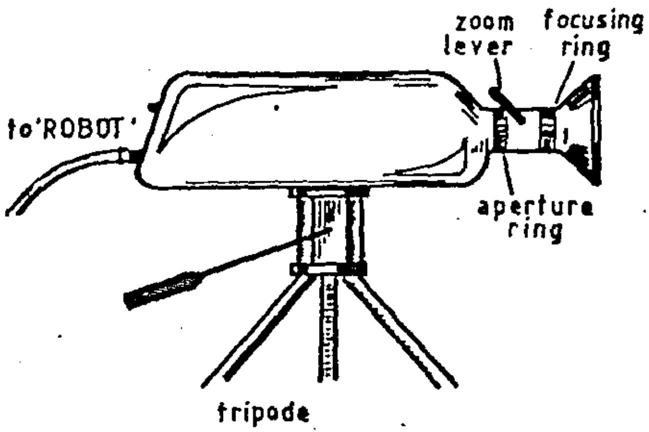


fig. 2.1

It is situated on a tripod and is pointed at the object that is to be recognized. The picture that is taken, is displayed on a T.V. monitor. If the digitizer is interposed between the camera and the monitor, the picture on the screen is a digitized representation of the actual picture.

2.2.2 The Digitizer ('Robot')

This is a device that digitizes the actual picture that is taken by the camera, and stores it in a 16K×4 bit memory frame store. There are 3 switches and 3 knobs on the front and two sockets on the back of the 'Robot'.

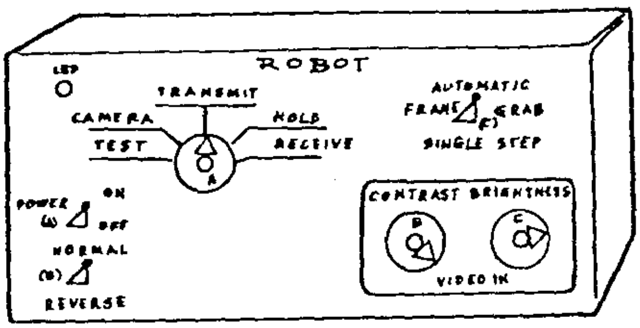


fig. 2.2

Switch A is an 'ON-OFF' switch. Switch B gives the actual (or digitized) picture in its normal form, when switched to 'NORMAL' and the negative picture, when switched to 'REVERSE'. Switch C is a 'FRAME GRAB' switch, which means that it causes the device to split the actual picture into 218×128 cells, and stores it in the 16K memory, each time it is pressed down. Knob A can be set to one of five positions. At position 'TEST' a test pattern of four stripes of different gray levels (those corresponding to hex numbers Ø,7,8 and F). That helps adjusting the contrast and brightness by using knobs B and C respectively. At position 'CAMERA' the digitized picture of the object is displayed on the screen. At position 'TRANSMIT' the *frozen* digitized picture that is stored in the memory is displayed on the monitor. This setting has been designed to send the frame to a telephone line. At the next position 'HOLD' the digitized picture displayed on the screen is held. Finally position 'RECEIVE' has been designed to allow the device to receive a frame from a telephone line. The transmit and receive operations are not used at the moment, because the special circuitry for them does not exist. Hence the effect of the first on the screen, is the same as at 'HOLD' and that of the second, reception of random garbage that fills the picture. A block diagram of the 'Robot' is shown in Fig. 2.3a.

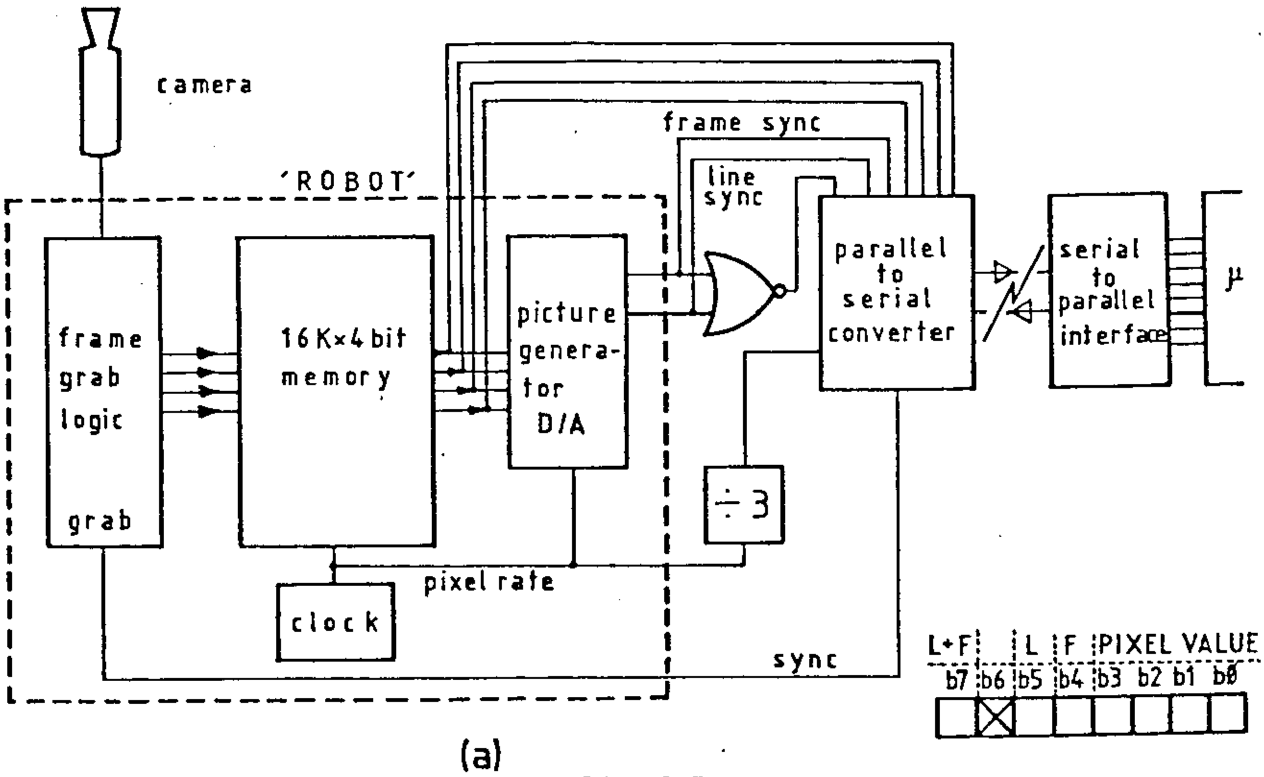


fig. 2.3

The picture taken by the camera, is passed to the 'Robot' (inside the dotted block). The analog signals which are received from the camera are given to a *Frame Grab Logic*. This converts the analog signals to digital ones and stores them in a 16K×4 bit memory frame, whenever the *Frame Grab* switch is pressed down. The memory is organized in an 128×128 byte matrix and each byte contains an integer in the range Ø-15 (or ØØ-ØF in hex). Each byte is called a *pixel* and the number that it contains *pixel value*. Every pixel value corresponds to one of 16 gray levels, which are used to represent the light intensity of that particular pixel. The whole of 128×128 pixels constitute the digitized picture. From the memory the picture is passed through a *Picture Generator*, which gives a visual representation on the monitor, by converting the digital signals back to analog ones.

The link between the 'Robot' and the microcomputer system is done by a *Parallel to Serial Converter*, that sends the bytes serially through a *Serial to Parallel Interface* to the micro. The format of each byte that arrives to the micro is shown in Fig. 2.3b and it is the following:-

a) the four low order bits constitute the pixel value, which is an integer number from Ø to 15.

b) the four high order bits are:-

bit four: the *Frame Sync*. It indicates a new frame when set (1).

bit five: the *Line Sync*. It indicates a new line when set (1).

bit six: is not used.

bit seven: the logic OR of bits four and five, (b_7=b_4 OR b_5)

The pixel rate in which the Robot transmits the pixels to the micro is too fast and the picture can not be collected by program. So, a rate divider, that divides the pixel rate by 3, has been interposed between the clock and the parallel to serial converter. Thus every third pixel is transmitted and since 3 is comprised with 16,383 (=16K) the sequence is:

$$
\begin{array}{ccccccccc}
P_{\emptyset} & & P_3 & & P_6 & & P_9 & & \cdots \\
& P_2 & & P_5 & & P_8 & & P_{11} & \cdots \\
P_1 & & P_4 & & P_7 & & P_{1\emptyset} & & \cdots
\end{array}
$$

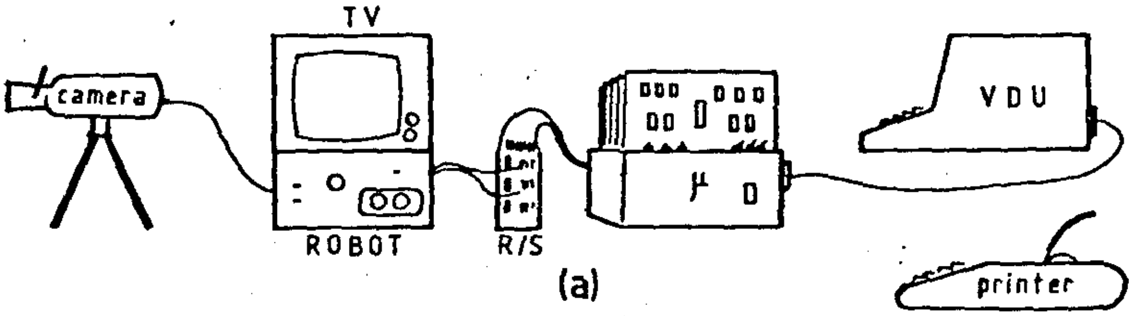
This covers the whole picture in exactly three passes. Every time a bit seven is checked to indicate a new frame or line.

2.2.3 The Microcomputer System the V.D.U. and the T.V. Monitor

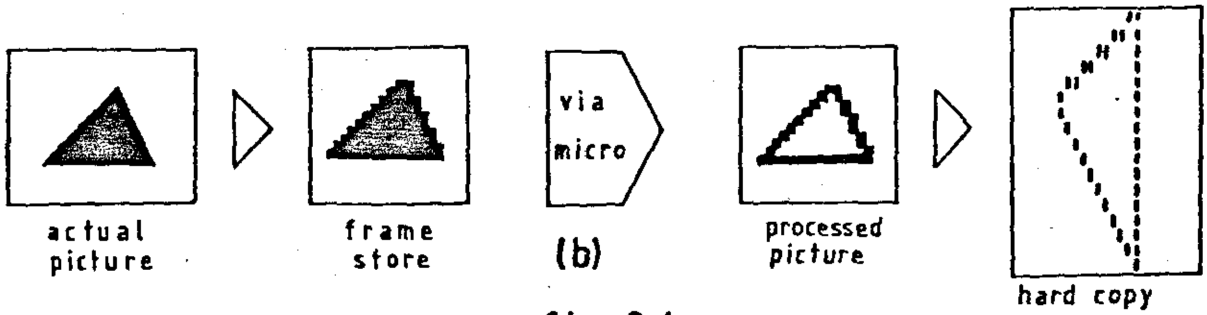
This consists of a Z-80 microprocessor board with direct assembler and 2K of memory on it, two 16K static memory boards, one 16K EPROM memory board and one 8K dynamic memory board. There are also available, two 4K non-volotile memory boards which were used during the development of the programs and an EPROM programmer which was used to *blow* the programs into the 1K EPROM chips. All the boards are *slotted in* on a QUARNDON motherboard which offers power supply, reset and restart buttons, single step switches, and memory location and data LED's in HEX.

A special program[3] (see Appendix 2) transfers the digitized picture to the two 16K memory boards. One of them is saved and the other is processed by the program, that is located in the 8K dynamic memory board. The program is loaded there from the 16K EPROM memory where it resides. This is done to allow the user to change the variables according to the circumstances of the problem. The language that is used is the *Z-80 Assembly Language*. The assembler offers a ½K memory for the label table but it does not allow forward references. Finally a number of function keys provide the user with elementary operations such as: accessing of memory locations and registers, and change of their contents, transfer and display of data blocks, and program execution. The arithmetic system used is the HEXadecimal. Fig. 2.4a shows the connection of the used equipment. The

The input unit is an ordinary Newbury V.D.U. which operates at 48ØØ c/s Baud rate. The T.V. monitor is a simple black and white CRT with brightness and contrast control knobs.



(a)



(b)

fig. 2.4

2.2.4 The Printer

This is a TPEND printer that operates at 3ØØ c/s Baud rate. Its moving head is a 5×7 dot matrix and it prints a maximum of 64 characters per single line. The printer is used to obtain a hard copy of the digitized picture. The idea is to give a visual representation by printing a character (or a number of overprinted characters) for every pixel relevant to its pixel value. The maximum of necessary overprints is 3 and thus every line is the result of 3 overprintings. For example the pixel value ØF which corresponds to gray level black is depicted by overwriting the characters N,Z and $. The effect of this is shown in Fig. 2.5.

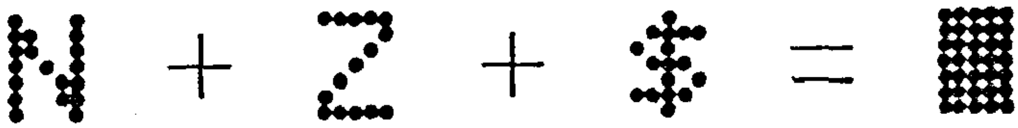


fig. 2.5

The problem of printing 128 characters, with only 64 possible in a single line, is solved by printing the picture sideways and in two halves; botton half and top half. The procedure is illustrated in Fig. 2.6a.

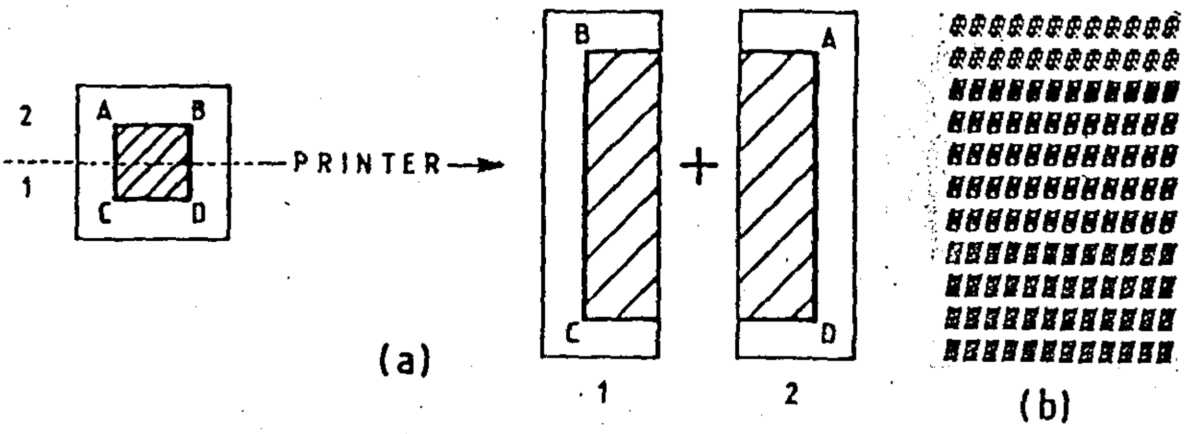


fig. 2.6

Of course the result is to have a picture different (prolonged) from the one displayed on the monitor due to the fact that the characters are rectangular and the distance between adjacent characters of the same row are different from those of the same column (Fig. 2.6b). A detailed description of the combinations of characters used is given in Appendix 1.

2.3 AVERAGING

As it has been mentioned before, the primary input to the visual processor, is a rectangular grid of cells, each one of which has a pixel value between Ø and 15. The objective of preprocessing is to reduce this grid of numbers, to a manageable set of features, relevant to the classification of the scene.

The original picture could be divided into 3 main sets of pixels. The background pixels, which are all the pixels that do not represent any part of the main object. The latter are selected so that they all have the same pixel value (preferably all black -pix.val=Ø- or all black -pix.val.15-) and they are of high contrast with the main object. The main object pixels, which are all the pixels that represent the studied object and they can take any pixel value in the range Ø-15. And the noise, which are pixels caused by noise and can be found in any of the two previous sets. In this set can also be included pixels that have been caused some kind of distortion in the picture. The elements of the first two sets are necessary for the recognition, while the elements of noise are not. The first of the preprocessing operations, has the task to remove the undesired noise pixels and if possible correct the pixels caused by distortion, leaving otherwise the background and main object unchanged.

Averaging[4] is a simple operation, which can achieve gap filling, speck removal, thickening and thining. It does that, by modifying each cell of the grid representation accordingly, with respect to its neighbourhood.

2.3.1 How It Works

Every cell is made the centre of a number of surrounding cells, which are called *window*. The size and shape of the selected window, varies according to the requirements and the main purpose of the operation. Basically the window is a square or rectangle with sides formed by an odd number of cells. The cell in the middle of the window is the one that gets modified. Before getting into the complex case of 16 pixel values, the function of averaging is examined on a picture with only two pixel values i.e. black 1 and white Ø.

The number of pixels with value 1 within the window is summed up and the result is compared to a prespecified threshold. If the sum exceeds the threshold the middle cell is turned to black; otherwise it is turned to white, i.e. it is set to Ø.

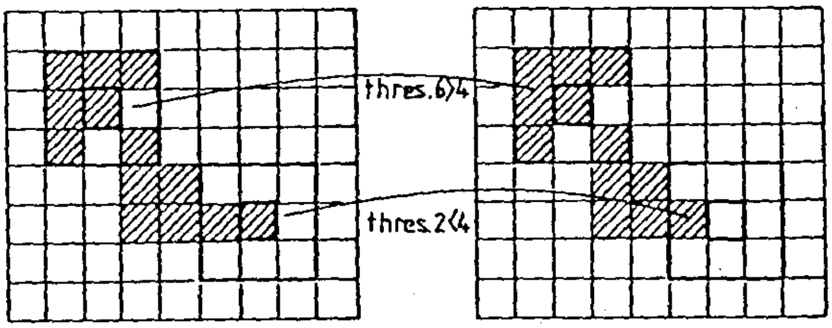


fig. 2.7

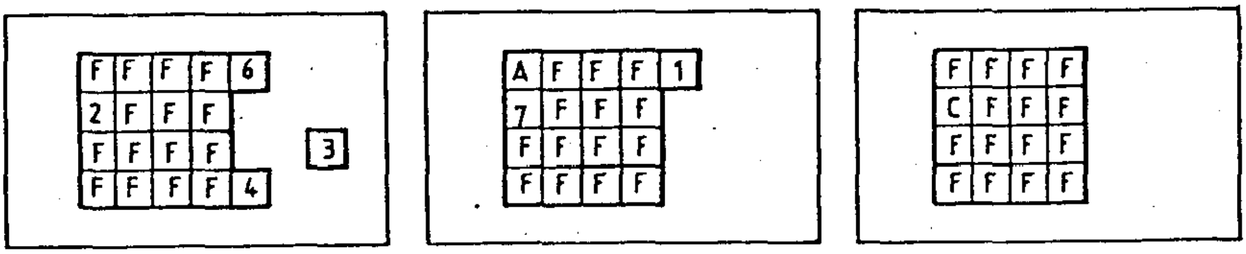
Fig. 2.7 gives an example of the effect of averaging using a 3×3 window with threshold 4.

The window is shifted one cell across every time, so that it is centred on the next cell and the operation is repeated until the end of the row is reached. Then the window is moved one row below and the same routine is followed until the whole grid is covered. Assuming a black background, white specks surrounded by black neighbours are likely to be noise and they are removed by being turned to black. This operation is known as speck removal. The opposite operation is black spots on the main object are turned to white and thus removed. This is called gap filling. For low thresholds, white lines on a black background are thinned and black lines on a white background are thickened. For high thresholds the effect is exactly the opposite. The size of the window controls the extent of the filling and removing operations. Small windows produce little change in the scene, while larger ones destroy detail. By using different threshold settings and different window sizes optimum effects are obtained.

2.3.2 Averaging in Pictures with More Than 2 Gray Levels

When the gray levels are more than two the technique has to be modified so that it copes with the larger range of pixel values.

The sum of the window cells is similarly compared to a threshold θ. If the sum is greater than θ, then a *quantity* or *intensification factor* is added to the pixel value of the cell in the centre of the window. Likewise the same number is subtracted from it, if the sum is found less than the threshold θ. Small intensification factors cope with speckles of low pixel value near the main object or random noise. Larger intensification factors are more effective when high contrasted edges are sought. Sometimes consecutive averagings with constant threshold can clear up pictures with great amount of noise and extract the main figure from a very grainy background (see application in Appendix 2).



actual picture

threshold: 3F
intensification factor: 5

: 3F
: A

fig. 2.8

The example in Fig. 2.8 assumes 16 gray levels Ø-F hex and a threshold of 3F. In case b) with intensification factor 5 speckles 4 and 8 have been removed and the gap of 2 has been filled. Speckle 6 has been virtually removed, considering that the next operation deals with pixel values greater than 7. In case c) the results are more obvious at the expense of weakening the top-left corner.

It must be noted that if the value of the modified pixel becomes greater than F or less than Ø, then it is set to F and Ø respectively.

The above technique has very good results, when it is applied on pictures, where the main object consists of pixels with only one gray level. Once a suitable threshold has been selected, it remains constant throughout the procedure. The problems arise when 3-D objects are represented on a 2-D screen. Normally there are 2 or more areas with different gray levels. This means, that if the threshold was kept the same for all of these areas, important edges might be destroyed. Fig. 2.9 shows an example of constant threshold averaging with intensification factor 5 and threshold 3F. The effect is that the front side goes from 5 to Ø, the top side goes from C to F and the right side remains unchanged. The effect is that edges xw and wz are intensified while edge wr completely disappears.

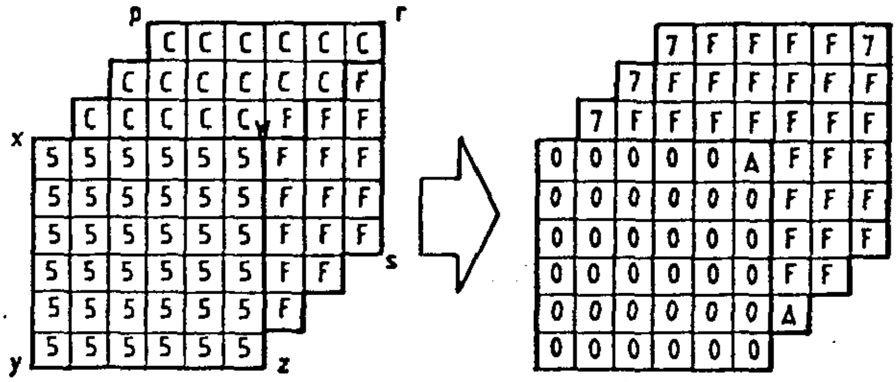


fig. 2.9

This indicates the need of changing the threshold according to the surrounding area. To get round this a new technique is adopted which is called *voting technique* or of *variable threshold.*

According to this method the mean of the surrounding cells in the window is taken as the value for the threshold and this is compared with the value of pixel at the centre of the window. An intensification factor is added to or subtracted from the pixel value of the centre pixel if it is found less than or greater than the threshold respectively. Otherwise its value remains as it is. The advantages of this method over the previous one are:-

a) it removes the noise near the boundary of the object as before, but it does not change the main pattern of the different areas.

b) it tends to set the pixel value of the centre cell to the gray value of its environment, rather than change it if that environment differs from a standard value. In Fig. 2.10 the voting technique is demonstrated on the example of Fig. 2.9.

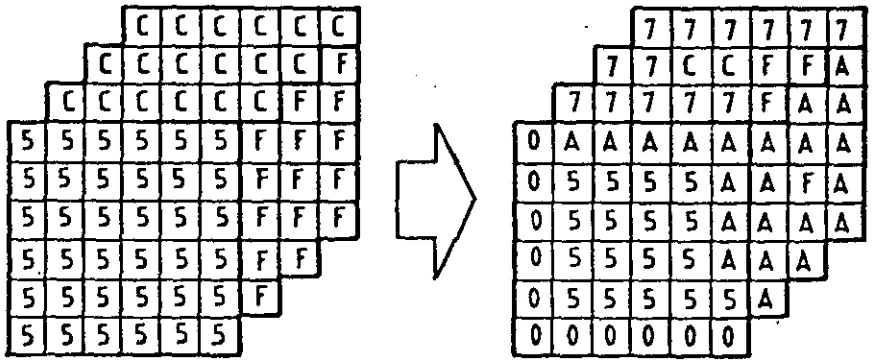


fig. 2.10

The main drawback of this method is that although it achieves its purpose in the middle of large areas, it alters strongly the pixel values of cells near areas with high contrast (i.e. edges). This gives rise to either thicker or double edges, which will certainly *confuse* the edging operator in the next stage.

As a result it is clear that averaging on a picture with a broader spectrum of gray levels is not as effective as it is in the two level system. This is because sometimes the neighbourhood of the centre pixel is misleading.

2.3.3 Masking the Main Object - Back to 2 Gray Levels

So far the averaging techniques that have been described could be called *intensification techniques*, because they average the scene by increasing or decreasing pixel values according to a constant or variable threshold. In this paragraph a new technique, which tries to use the two gray level operator is suggested.

As it has been mentioned before the representation of a 3-D object on a 2-D screen is a combination of two or more areas with different gray levels. The fact that the levels are more than two causes the problems that were discussed in the previous methods. Thus the idea is to *isolate* each area every time and treat the rest of the picture as if it were background. Before proceeding to any operation the following assumptions are made:-

a) a shape is considered as a 3-D shape if it consists of two or more areas with pixels, the values of which differ by a certain amount.

b) a number of pixels are considered to belong to the same side if their values lie between certain limits.

c) a number of pixels less than a certain fraction of the whole number of pixels constituting the main object, are considered to be noise.

d) pixels with the highest (lowest) pixel value, are considered as background pixels.

Thus before applying the averaging operator consecutive scannings search for pixels with the same value (other than that of the background). These are grouped together and their sum is stored in different locations for each of the gray levels. Then the total of the pixels of the main object is formed, by adding together all the partial sums. The pixel values, the partial sums of which are greater than a certain fraction of the main sum, are put in a special array. The others are simply turned to background pixels. The consecutive elements of this array are tested. If they differ by more than a certain limit then they are kept, otherwise they are replaced by the value below. Finally every element of the array indicates the pixel value of cells that belong to the same side. This area is turned to white and the rest is made black (background by convention). The averaging operator can be applied now with all the advantages of the 2 gray level scene. The same procedure is followed until the whole of the array has been used. Thus every time the main object is masked, so that concentration is kept on one of its sides each time. Of course the contrast between the two sets, main object and background is the highest possible and this will make the edges easier to inspect. The main concern is on the selection of the limits, so that pixels that actually belong to different sides are not put in the same group and vice versa. Another point that needs to be considered carefully is the selection of the main sum fraction with which the partial sum are compared. This must be such that only the pixels, that are not many enough to be forming a side, are eliminated as noise.

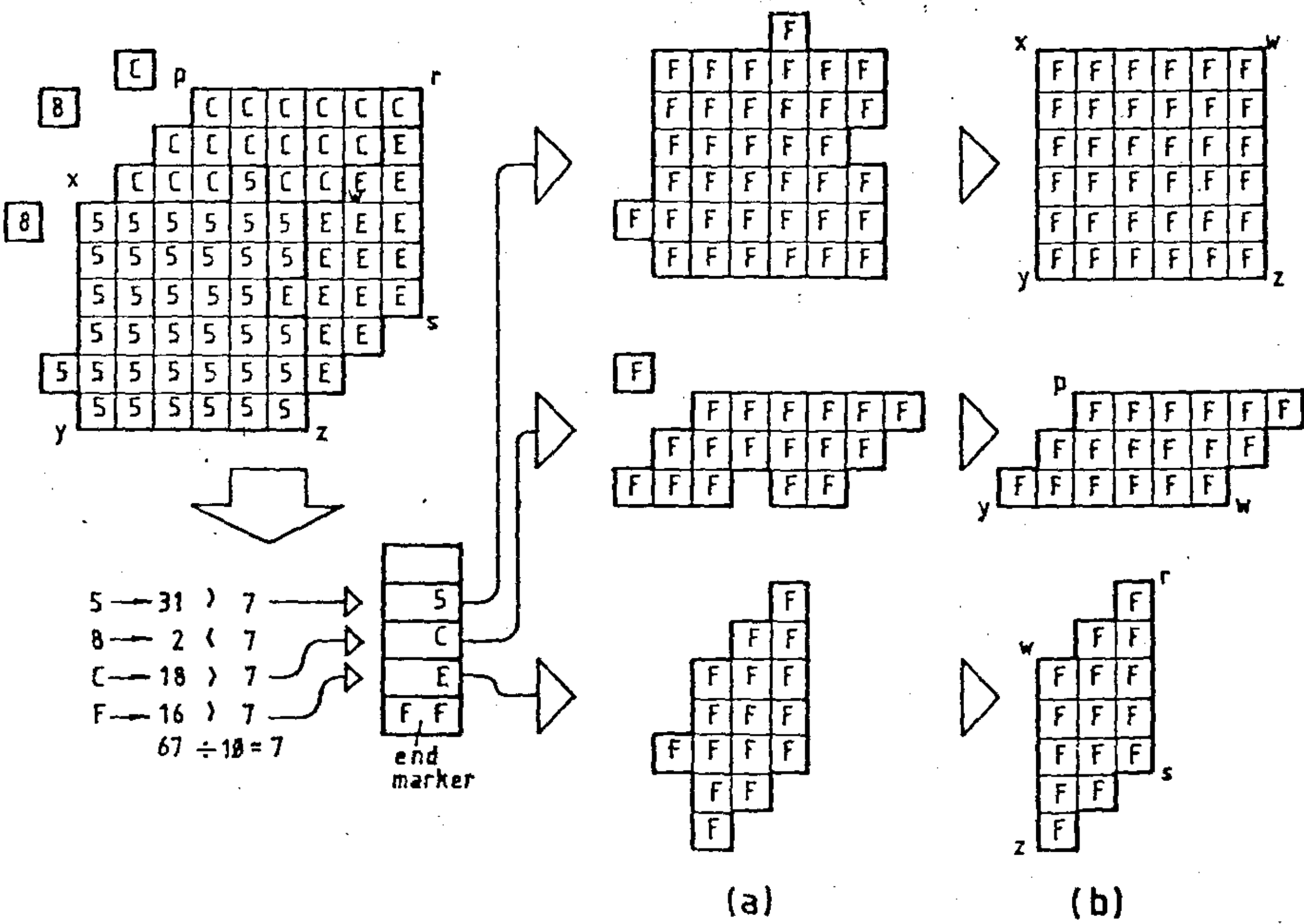


fig. 2.11

Fig. 2.11 demonstrates the method. In a) the result of the masking is shown and in b) the result of the averaging.

2.3.4 Averaging in Practice

The picture here is a 128×128 grid of bytes with 16 gray level resolution. Black is chosen to be the background (pix.val.ØF hex) in order to eliminate the shadow from the main object. The window is a 3×3 square and the threshold is taken equal to 5*. Because of the size of the window, the actual picture becomes 126×126 because the first and last row and the first and last columns are not used as centres of the window (Fig. 2.12a).

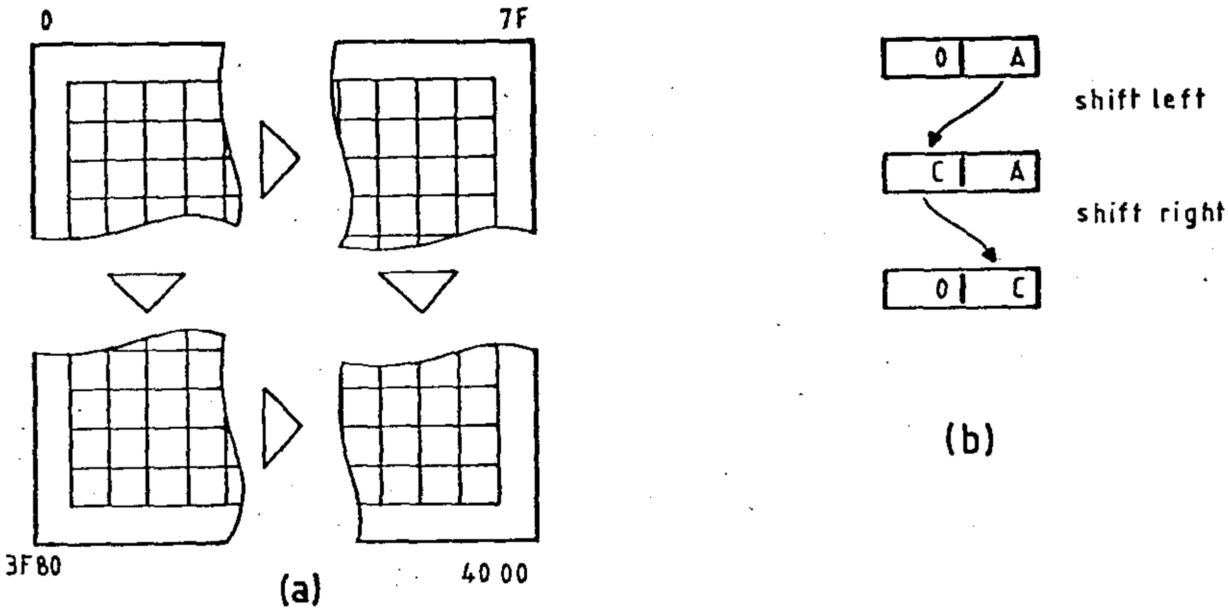
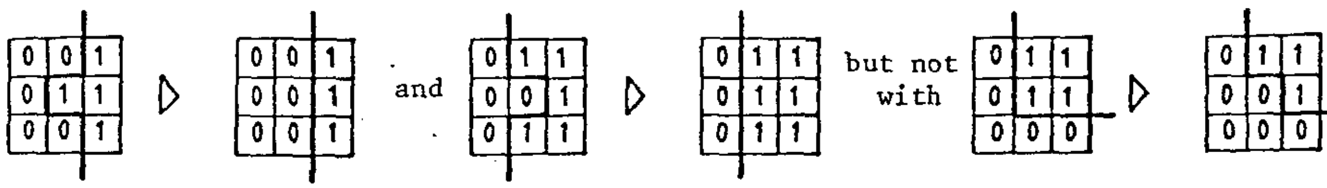


fig. 2.12

The first four bits of every byte contain information that is no longer useful, once the picture has been collected. So a considerable saving of storage (16K) is achieved by shifting the modified pixel value four places to the left. After the end of the averaging the first four bits are shifted

---

* this copes with



FLOW CHART FOR AVERAGING

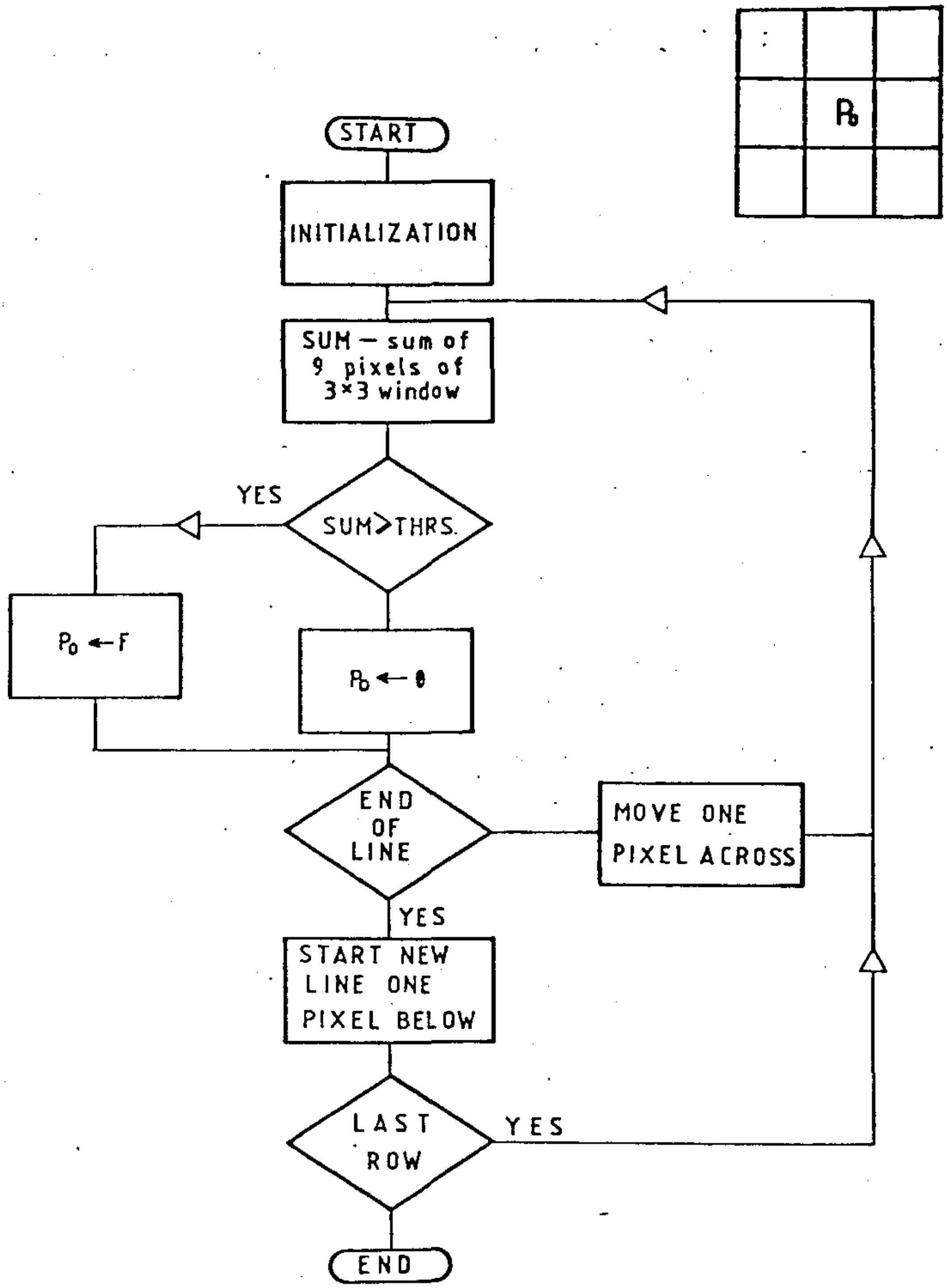


fig. 2.13

four places to the right. This means that the original picture is destroyed after every averaging. Thus a copy of the original picture is always kept in the second of the two 16K memory boards.

2.4 EDGING

The previous operations of averaging and intensification remove the noise and present a picture the main characteristic of which is areas of cells with the same pixel value. As it has been mentioned before, assuming uniform illumination, each one of these areas represents a side of the object. Every one of these areas is separated from its adjacent areas or the background, by a single pixel line, which is the contour of the side. Considering every side as a separate 2-D shape, every 3-D shape consists of a number of 2-D shapes connected together. The contour of these sides is the feature that distinguishes them from others with different contour and groups them together with those of similar contour. Edging is the operation that extracts the contour of a shape. It preserves centres of asymmetry such as edges, corners, junctions and ends of lines. The edging operator sharpens differences and it could be seen as a two-dimensional derivative since changes about the centre element are counted.

Like in averaging, edging is examined separately on pictures with two and more than two gray levels.

2.4.1 How It Works

The edging operator[6] is a square window, centered on each cell. After a number of tests have been performed, the window is centred on the next cell across until the end of the row is reached. Then the window moves to the next row below and the operation continues until the whole of the grid is covered. Considering a 3×3 window on a two gray level picture (Ø,1), the operation is as follows:-

The 9 cells of the window are numbered as in Fig. 2.14a. Next the following (Fig. 2.14b) eight tests are performed.

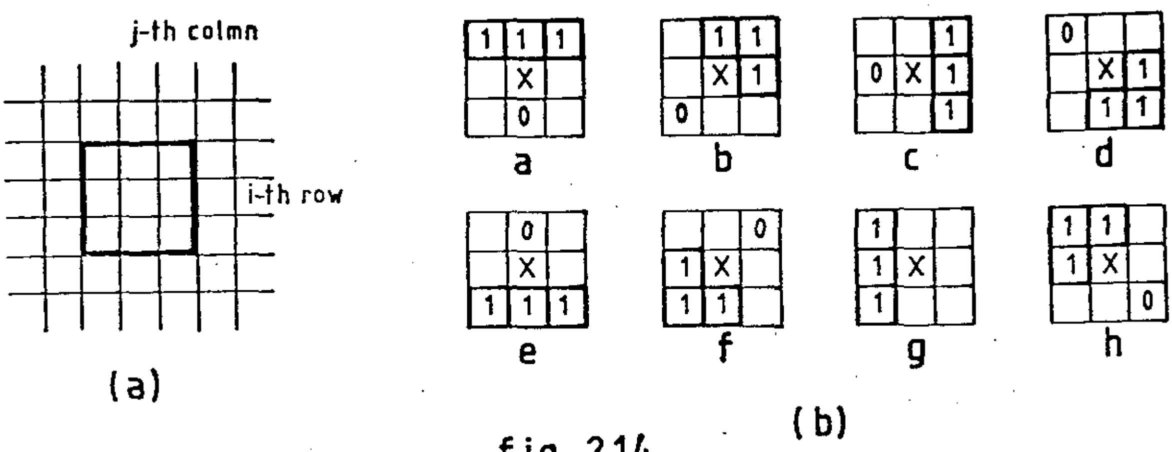


fig. 2.14

A one is scored if any of the combinations in Fig. 2.14b or the complentary ones (where 1→Ø and Ø→1) occurs and the number of occurrences is summed and stored in a *tally*. If the tally is greater than a threshold θ, then the pixel in the middle is set to be black (pix.val.=1), in the transformed scene. Otherwise it is turned to white (pix.val.=Ø). The threshold θ could be determined by first counting the total number of black cells in the window and then setting the threshold to some fraction of this number.

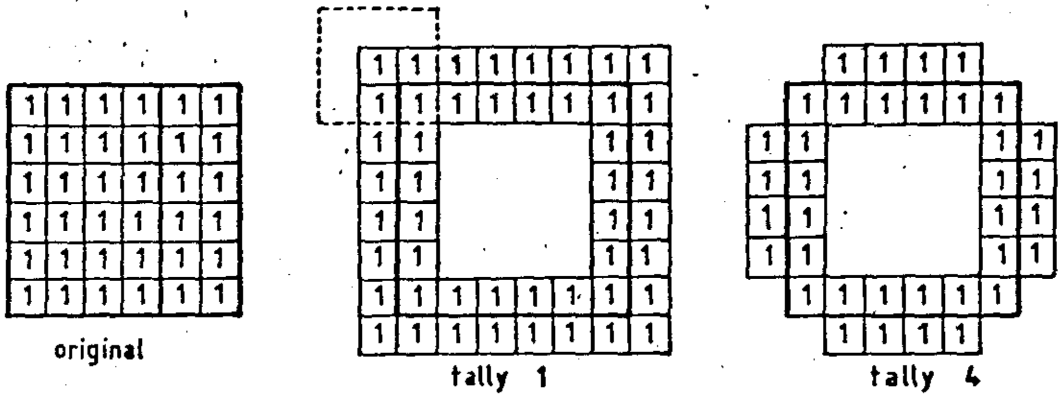


fig. 2.15

Fig. 2.15 demonstrates the effect of edging on a square, using different thresholds. The result is that a double edge is obtained. The inside one is the contour of the square and the outside one is caused by the background. This double edge will cause problems to the boundary follower (external loops) and as such needs to become single. This is achieved by centering the window on black cells only. Thus the white cells of the background that caused the outer edge, will no longer be used by the operator and the result will be a single edge. Fig. 2.16 shows the result of centering the window only on black cells.

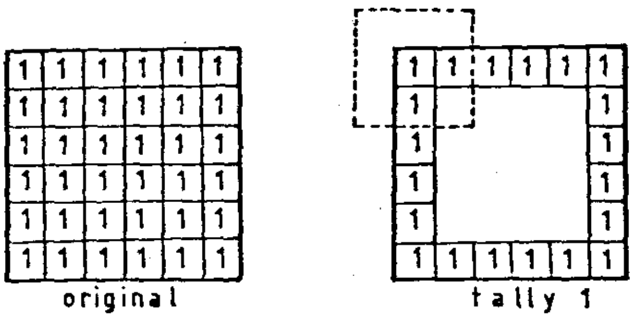


fig. 2.16

2.4.2 <u>Edging in Pictures with More Than 2 Gray Levels</u>

In the case of more than two gray levels, the algorithm is modified, in order to cope with the higher resolution picture. First of all, pixels with values smaller than a certain limit are considered to be noise. This is reasonable, because pixels with relatively low value, will have been reduced to such by the previous averaging-intensification operation. This means that they were mainly noise and not elements of the main object. By introducing a threshold I, these pixels will be virtually eliminated from the scene if their values are below this threshold. In fact the operator will be centred only on pixels with value greater than threshold I.

After this first thresholding the algorithm performs the sequence of tests. This time however, a one is scored if the absolute difference between the value of the middle pixel and the value of each one of the three opposite cells is greater than a threshold II. When the above condition is satisfied for a certain number of times, the middle pixel is set to black (pix.val.=F, in case of 16 gray levels). Otherwise it is turned to background. When the whole picture has been covered the procedure ends. The result is a two level picture with white background and the boundary of the shape as a single pixel line of black cells.

The above have good results when a single 2-D shape is represented in the picture. In the case of 3-D objects the thresholds have to be altered otherwise some of the sides will be eliminated as noise. The solution to this is the masking operation which has been discussed in Section 2.3.3. The edging operator follows the averaging operator every time a new side is processed. Masking brings the problem back to two gray levels and everything described in Section 2.4.1 applies in this case too.

2.4.3 Edging in Practice

Like in averaging the picture becomes effectively 126×126 large, because of the size of the window. The same shifting left and right technique, to store the modified pixel value is used again.

Finally, a point that needs special consideration is that of the selection of the tally. The sum will be an integer between ∅ and 8, i.e. $\emptyset \leq$ tally ≤ 8. The problem is to find the lowest tally, below which the middle pixel will not be marked as a boundary cell. From various examples it is deduced that the higher the tally, the more the gaps in the final boundary. Fig. 2.17 demonstrates some of these results.

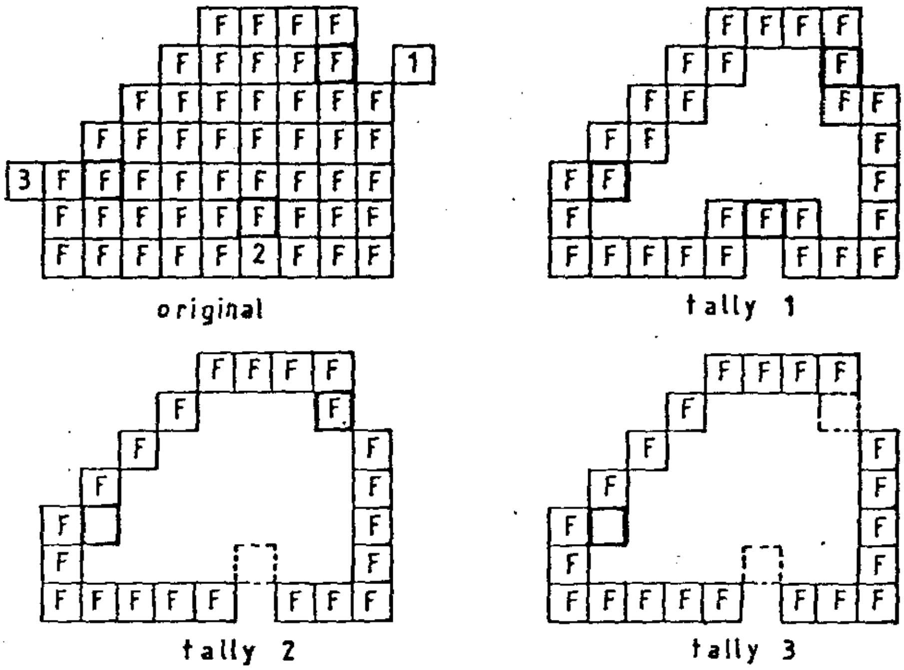


fig. 2.17

By concentrating on the three cells 1,2 and 3 it is seen that for tally=1, the contour is followed perfectly, but fails to give a single diagonal line, which is highly undesirable. For tally=2 a gap appears in the cell at position 2. For tally=3 another gap appears at position 3. In general since the operator is centred on black cells, if the number of white squares inside the 3×3 window is less than the used tally, there is discontinuity in the boundary of final picture.

Tally=2 is the best selection because the gap in the original shape will be filled by the averaging operator (see § 2.3.3) and consequently there will not be such a case.

FLOW CHART FOR EDGING

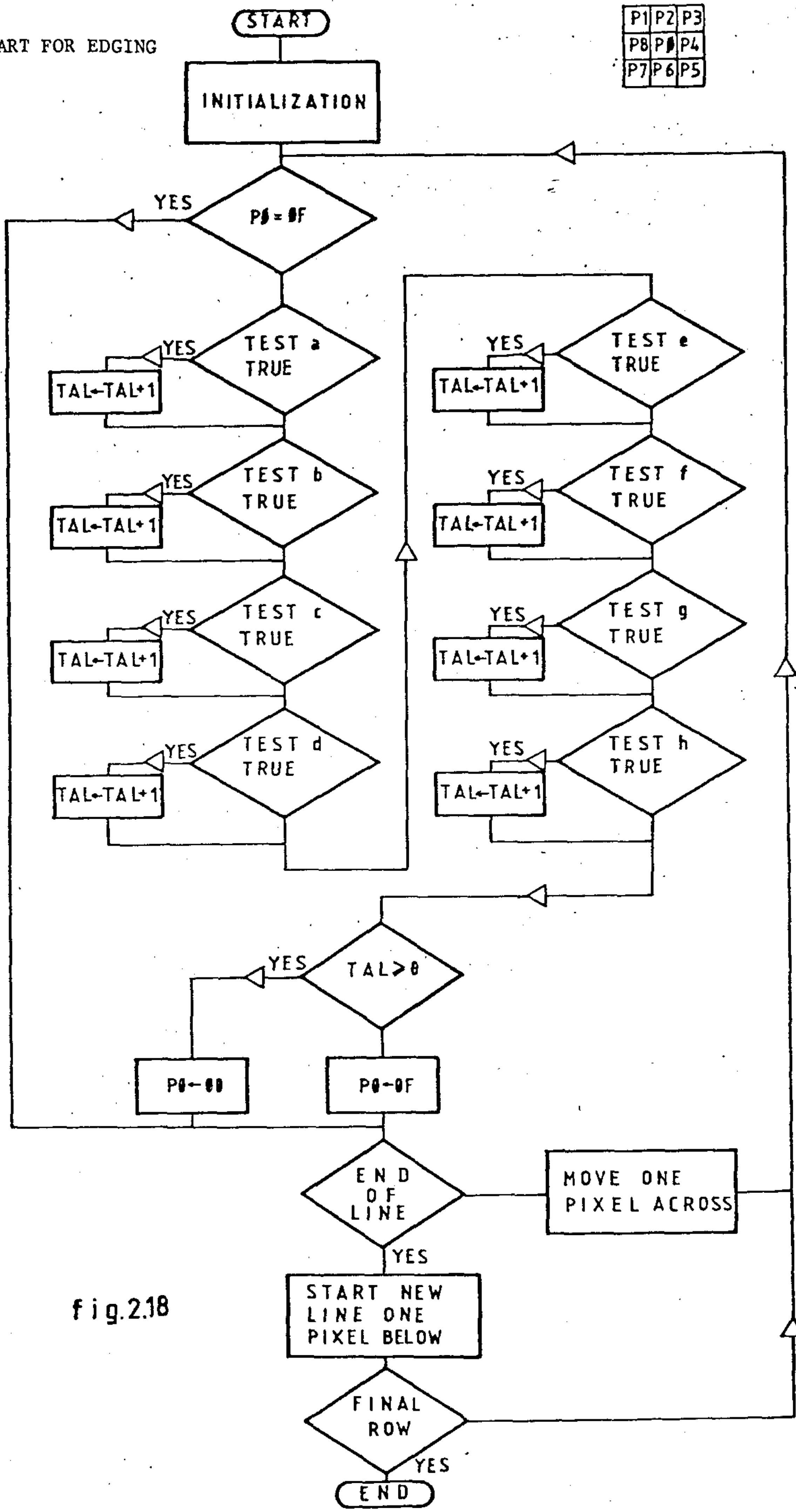


fig.2.18

2.5 ISOLATION[7]

In Section 2.3.2 a kind of isolation operator was mentioned. Below a more effective isolation operator is discussed.

This operator selects the connected figure in a scene and erases all other parts of the scene not connected to it. In this operation an assumption is made that the figure is made up of black (pix.val=1) and the background is made up of white (pix.val=Ø) cells. It begins by selecting an arbitrary black cell. A window is centred on this cell and all black cells (including the original one) inside the window are marked retained. The window is then centred on each *retained* cell in turn and new black cells are marked *retained* by the same criterion. This operation continues until no new *retained* cells are left to become window centres. Then each *retained* cell in the grid that has its corresponding cell in the transformed grid is set equal to 1. All other cells are set equal to Ø. Thus if the window is 3×3, only the black cells connected (diagonally or adjacently) to the original cell are preserved. All other block (figure) cells are converted to white (background) cells. Larger windows allow the process to hop *across* small gaps in an otherwise connected figure.

The above described operator can be modified to cope with more than two gray level pictures. In this case pixels will be marked *retained* if they lie within an upper and a lower threshold.

SUMMARY - CONCLUSIONS

- Preprocessing is a method of representing a scene by a set of numbers.
- A visual scene is divided into a grid of cells, the light intensity of which is converted to an electrical signal and represented by a four bit number.
- A T.V. camera takes a picture of an object that is digitized and stored in a 16K (128×128 pixel range) framestore. The picture is presented on a T.V. monitor and through a special routine a hard copy of it is obtained by a printer. A microcomputer system administers the previous routine as well as the following preliminary operations.
- Averaging-intensification: the middle cell of a 3×3 window is modified according to the relation of the sum of the 9 cells to a given threshold. The window operator scans the whole picture.
- It achieves gap filling, speck removal, thickening and thinning.
- Edging: the middle cell of a 3×3 window is turned to ∅ (white) or 1 (black), according to the difference of pixel values in a number of combinations between opposite lying cells.
- It sharpens differences and presents the contour of the figure.
- Isolation: it selects one connected figure in a scene.
- Careful selection of the various thresholds in averaging and edging provides better results.
- The preliminary operations *clean up* the scene leaving the dimensions of it unchanged.

REFERENCES

1,4,6,7. N.J. Nilsson, *"Adaptive Pattern Recognition: A Survey"*, 1966, Symposium, Bryton, Ohio (May 1966), pp.103-115.

2. C.H.C. Machin, *Personal Communication - Robot block diagram.*

3,5. Suggestions by C.H.C. Machin, Department of Computer Studies, Loughborough University of Technology.

Chapter 3

EDGE FOLLOWER AND VERTEX DETECTOR

3.1 INTRODUCTION

One of the main problems on the digitized picture is the representation of straight lines. The latter are important, because they are the main components of the shapes to be recognized. Once the 2-D shape has been preprocessed, it is left with only its contour, which consists of straight lines connected together. The objective of the next process, is to follow this contour and identify the points at which the direction changes. In other words, locate the vertices of the shape. The coordinates of the vertices are stored in special memory locations and are given to the next process which will check which of these vertices are more likely to be true ones. The others, which could be called *pseudo-vertices,* will be dropped. So the result of the procedure is a two-dimensional array of vertices - representing X and Y coordinates respectively - which are the set of vertices that determine the 2-D shape, to be recognized. This *true-vertex determination* is written in C and its last task is to form the necessary PROLOG unit clauses, which will be used as data for the PROLOG recognition program.

The technique of this first part i.e. the vertex detection, is based on a series of special tests, which check the change of the pattern representing straight lines with different slopes. The second part, i.e. the verification of the true vertices is based on the check of the distances of the *suspected* or *pseudo* vertices from the equation of the straight lines.

3.2 BOUNDARY FOLLOWER

It is true from Analytical Geometry that the equation of the straight line is:-

$$y = ax + b \tag{1}$$

with x taking values in $\mathbb{R}$, a is the slope of the straight line defined as $a=\frac{dy}{dx}$, and b is value of coordinate y for $x=\emptyset$, i.e. y_0. The equation of a line segment $\overline{A\ B}$ is given by (1), with x defined on the domain $(A,B)\in\mathbb{R}$ and taking values in $\mathbb{R}$.. Hence, every root of the equation (1) on (A,B) is given by the function

$$y = f(x) = ax + b \tag{2}$$

which is continuous.

In the digitized picture however, a different representation of straight lines takes place, because there is a definite number of points - every pixel can be represented by its centre - and at fixed positions. Hence, although the end points of a straight line segment can be determined, they can not be joined together - generally - with a number of points that belong to the set of roots of the equation defined by them. This is illustrated in Fig. 3.1 a and b.

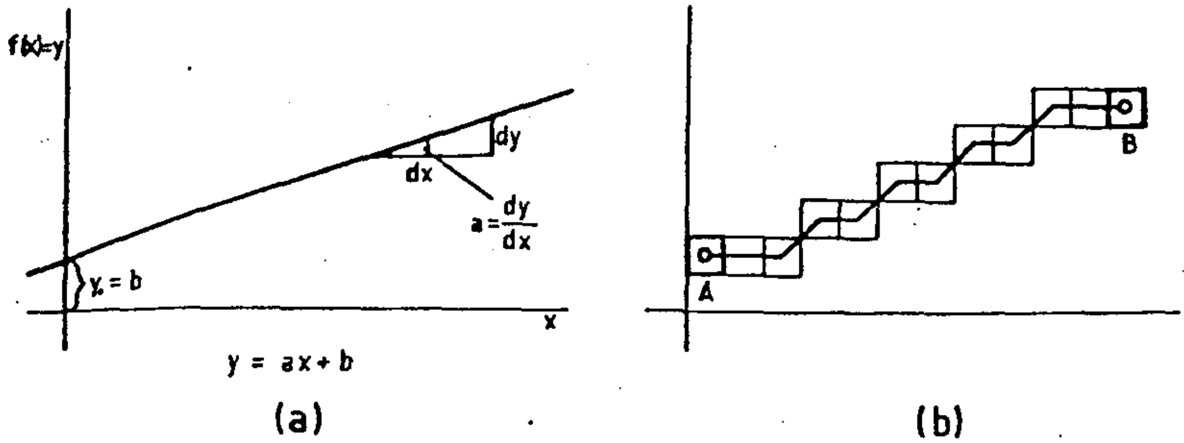


fig. 3.1

It is easy to see that the only cases when the digital representation coincides with the one given by the equation, defined by the two points, is horizontal, vertical and 45^{o} lines (Fig. 32a,b,c).

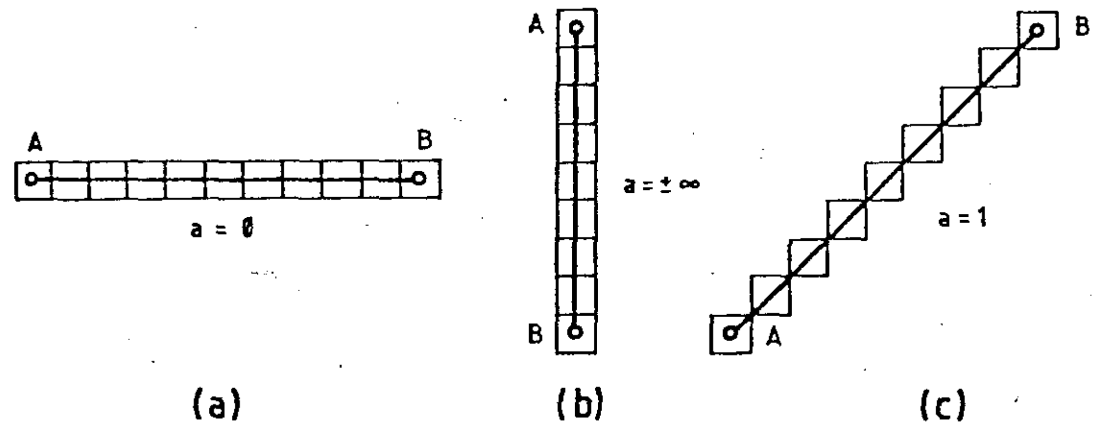


fig. 3.2

So, before starting discussions on the technique of the boundary follower, it would be wise, if a good study of different types of straight line representation in a digital picture was made.

3.2.1 <u>Straight Line Representation in Digital Form</u>

From the above it is obvious that the best approximation of straight lines is by joining together horizontal or vertical segments which belong to adjacent rows or columns respectively. Supposing that each pixel is represented by a point located at its centre; then by joining together this point with its 8 neighbouring centres eight directions are formed. These are the possible ways in which one can move in order to find the next adjacent point of a straight line (Fig. 3.3a).

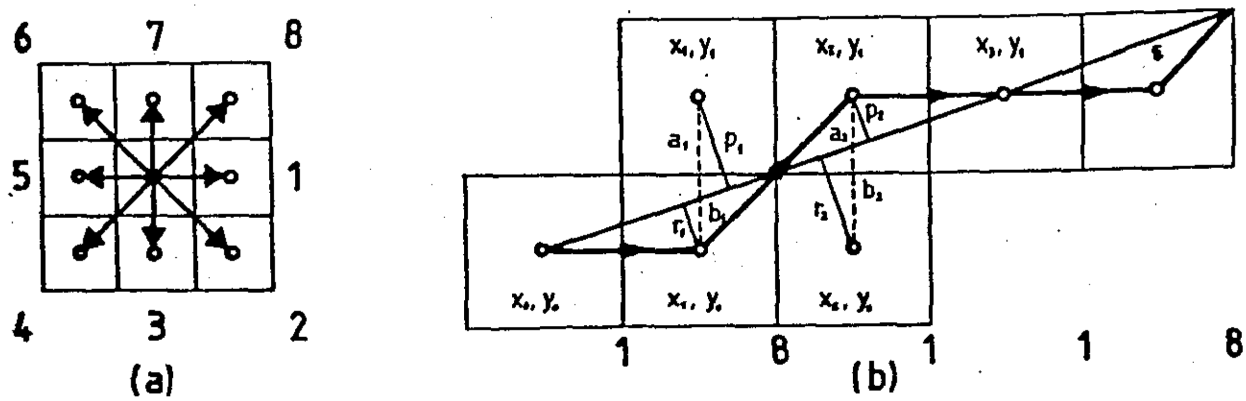


fig.3.3

The procedure that generates the right direction is given by the rule of the least distance from the actual straight line.[1] For example, let (x_0,y_0) be the coordinates of a point that belongs to the straight line ε. The two points with the least distances from ε are (x_1,y_1) and (x_1,y_0). From Fig. 3.3b it is true that, $r_1<p_1$ or (because of the similar triangles) $b_1<a_1$, and hence the point (x_1,y_0) will be preferred to (x_1,y_1). At this point it is noticeable that, it would be convenient if a name was given to each of the 8 directions (Fig. 3.3a) so one could refer to that by it. The easiest way is to start by calling a certain direction *number 1* (in the above the one across right was chosen) and continue clockwise, by increasing the numbers by 1. So in the above example the first move was a '1'. Then, because of $a_2<b_2$, (x_2,y_1) is the next point, and this is obtained by using an '8' from the point (x_1,y_0). Next a '1' comes up again, followed by another '1' and so on.

From the above it can be seen that the straight line ε is approximated by a pattern of *unit lines* of standard direction of the form ...18118118... It will be proved now, that this pattern remains unchanged with respect to the same straight line (no distortion is assumed, case which will be examined later on).

The example examined above is considered again, assuming that the pattern does not remain the same:-

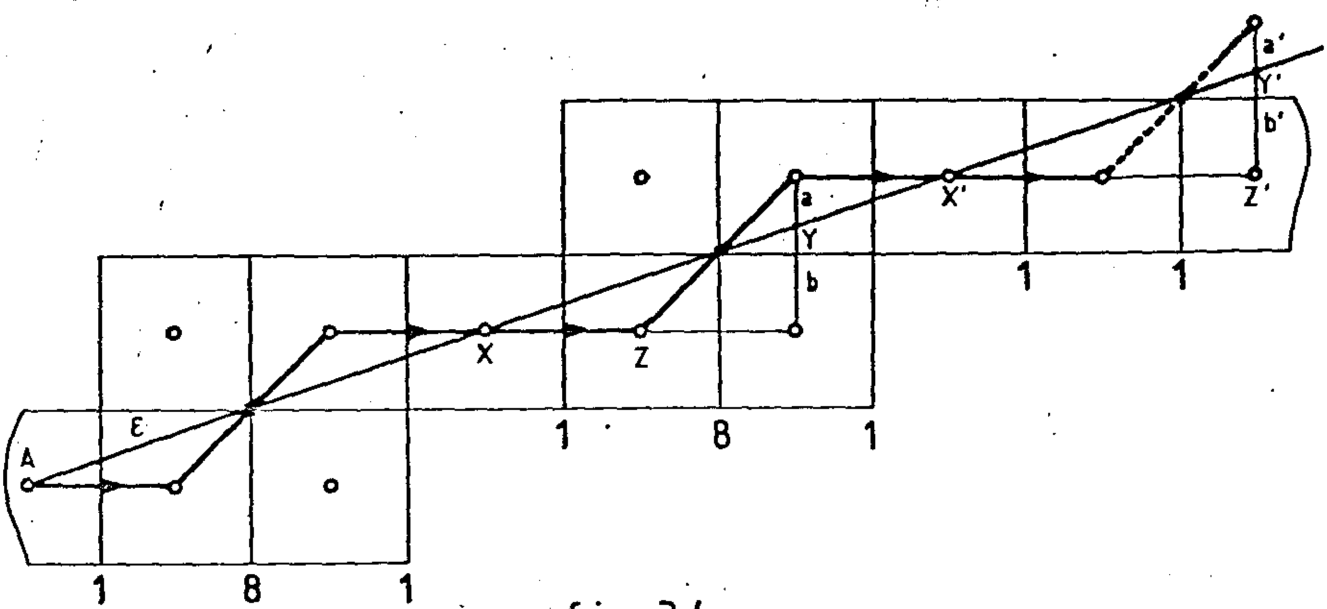


fig. 3.4

It is true that the slope s of the straight line is ct. An assumption is made that the pattern starts 18118118... and at the point Z' changes to 8111 (Fig. 3.4). This means that the distance a'>c' and thus a step '1' is taken instead of an '8'. From the equal triangles $X \overset{\triangle}{Z} Y$ and $X' \overset{\triangle}{Z'} Y'$ (they have $X \hat{Z} Y = X' \hat{Z'} Y' = 1L$, $\overline{XZ}=\overline{X'Z'}$, and $Z \hat{X} Y=Z'\hat{X'}Y'=S$) it is deduced that $\overline{Y'Z'}=\overline{YZ}$ and b'=b. Similarly a'=a. But it is a'>b' and so it should be a>b, which contradicts the original assumption that the pattern at that point is ...118111.... . Hence it was proved that the pattern, which represents a straight line remains unchanged through the line. Here it must be pointed out that at the two ends of a straight segment it looks as if the pattern is slightly different. This is only due to the fact, that the pattern is *picked up* at some point and there is no interest in the pattern beyond these points, e.g. Fig. 3.5 shows this.

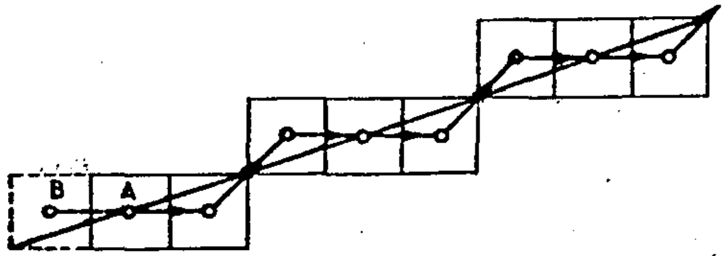


fig. 3.5

The pattern here is ...118118118... but because the main interest is in the part from A onwards, the first bit BA is dropped and it looks as if the pattern is 1811811811... . This will cause some problems later.

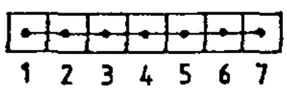
3.2.2 Different Kinds of Patterns - The Link

It was mentioned in previous paragraphs that the representation of a straight line on a digital picture is a sequence of horizontal and vertical line segments, joined together. Here an attempt is made to analyse the main features of this representation, in order to use them as key points for the location of the vertices.

First, the *length* of a straight line segment or *unit* is defined* as:- the number of pixels in it minus one.

Length = no. of pixels -1

e.g.

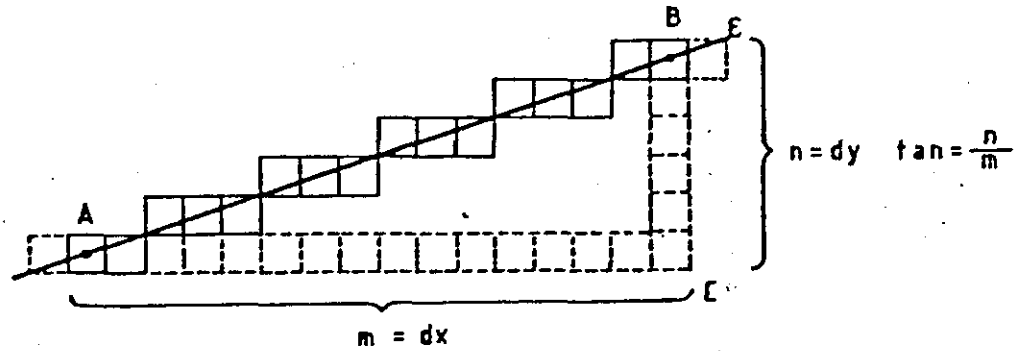


has length 6 (7-1)

---

*If a cell belongs to two different units counts for both in the calculation of the length.

There has also been shown that the pattern remains unchanged with respect to the same line. Next it is examined how the length of the straight segments, which comprise the line, is related to the slope of the segment.



f i g. 3.6

The slope s of the straight line ε is given by the quotient:- $s=\frac{dx}{dy}$. In the above example, in order to find the slope of each line segment the right angled triangle ABC is formed (Fig. 3.6) and as dx and dy are used the lengths ℓn and ℓm of its two sides BC and AB respectively. By taking the $\max(\ell m,\ell n)$ and the $\min(\ell m,\ell n)$ and forming $p=\frac{\max(\ell m,\ell n)}{\min(\ell m,\ell n)}$, then $[p]$ is the number of pixels in every straight segments - *unit* - of the straight line AB. If p is an integer then the length of the units is constant $\ell u=p-1$ otherwise it varies from $[p]$ to $[p]+1$. So the main pattern is a recurrence of units with length ℓu and $\ell u+1$. The units are connected with joints of length 1, which are called *links*. The links remain the same throughout the whole line, and this is the feature that distinguishes between parts - units - of the same line and parts of another line (change of direction i.e. location of a vertex). So far the only units considered were horizontal ones connected with diagonal links. Let the first quadrant be taken to examine how the direction of the links and the units change, by moving from $\emptyset^{o}$ to $9\emptyset^{o}$ (Fig. 3.7).

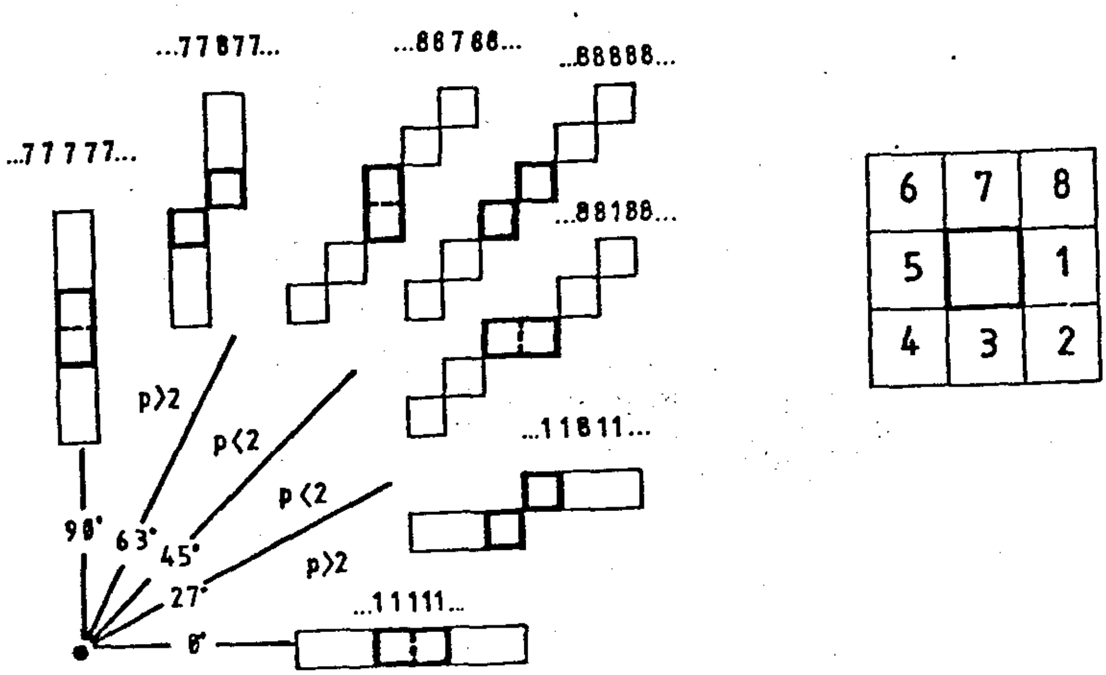


fig. 3.7

At $\emptyset^{o}$ the link degenerates so the pattern is a single unit of ones (or fives) - the directions left to right and bottom to top are considered - Moving towards 45^{o} p approaches 2 and so the length of the units decreases until it becomes $\emptyset$ at 27^{o} ($p=1.94 \Rightarrow [p]=1$ and $\ell u=[p]-1=\emptyset$). The units from 27^{o} to 45^{o} are of length $\emptyset$ and 1 and the length of the link starts increasing becoming maximum at 45^{o} where the length of the units is $\emptyset$ only. From 45^{o} to 63^{o} p is less than 2 and so there are some units of length one with the only difference that they are vertical this time. Finally from 63^{o} to $9\emptyset^{o}$ one finds increasing vertical units connected with diagonal links. At $9\emptyset^{o}$ the pattern is again one single unit without any links. Thus it is clear that the three directions horizontal ($\emptyset^{o}$), diagonal (45^{o}) and vertical ($9\emptyset^{o}$) are the only ones, in which all the points of their digital representation lie on the same line. That means that the two representations digital and real coincide. Inside the sector 27° to 63° there are units of length one only, so one could consider the sequence of links as units connected with horizontal or vertical links of length one. The three directions which determine more or less the direction of the units are called *dominant*. So the link can be defined as:- a unit of length one, that connects the main units which represent a straight line in a digitized frame.

3.2.3 Edge Following Operator

After the first three preprocessing operations, the only pattern in the frame store, will be that of the contour of the shape. On the screen this looks like a - rather - continuous sequence of white pixels in a black background. All that needs to be done now is to follow this contour and at the same time detect and locate the vertices of the shape.

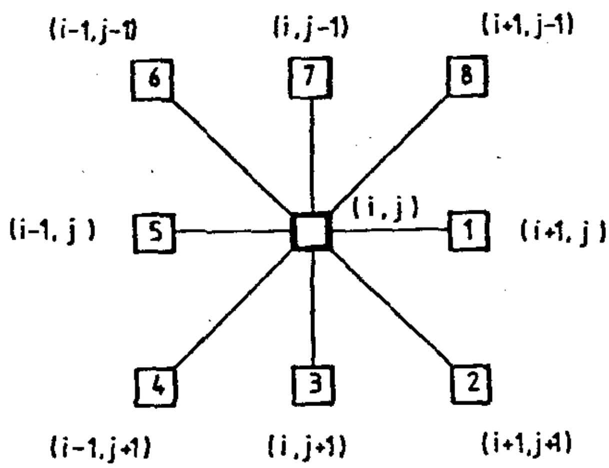


fig. 3.8

There are two main steps involved in the boundary following algorithm[2] described below.

a) The search strategy.

The picture is scanned left-to-right, top-to-bottom until a white cell is found. Because of the averaging operation there will not be any single white cells and, the ones met will belong to the boundary of our shape. This is considered as the starting boundary point, and the position of this cell (i_s,j_s) is saved in the first element of a boundary array.

b) The follow strategy.

The neighbours of a boundary cell are numbered as shown in the Fig. 3.8. From the starting point (i_s,j_s) found by the search strategy, the boundary of the shape is followed in such a manner, that the region within the boundary is kept always to the right of the path being followed.

The neighbours of the currently considered boundary cell (i,j) are now checked for the next boundary-white-cell. The checking begins with neighbour $n-2$, with $1 \leq n \leq 8$. Where n is the number that indicates which of the neighbours of the preceding boundary cell, the currently considered cell is. Every new boundary cell is marked with a number indicating its direction. Thus all the white cells of the boundary are turned to n, $1 \leq n \leq 8$, apart from the ones being considered as vertices, which are marked differently. When a neighbouring boundary cell is found, it will become the next currently considered boundary cell.

This following strategy is repeated until the starting point (i_s,j_s) is again encountered. At this time it is likely that the complete boundary has been traced. In order to search for a second boundary in the picture, all the non-white cells are turned to black-background pixel value - and a new scanning is tried according to the search strategy. When the whole frame has been covered and there are not any white cells left, the following operation finishes. An example of the route of the follower and the result of the marked cells is given in Fig. 3.9.

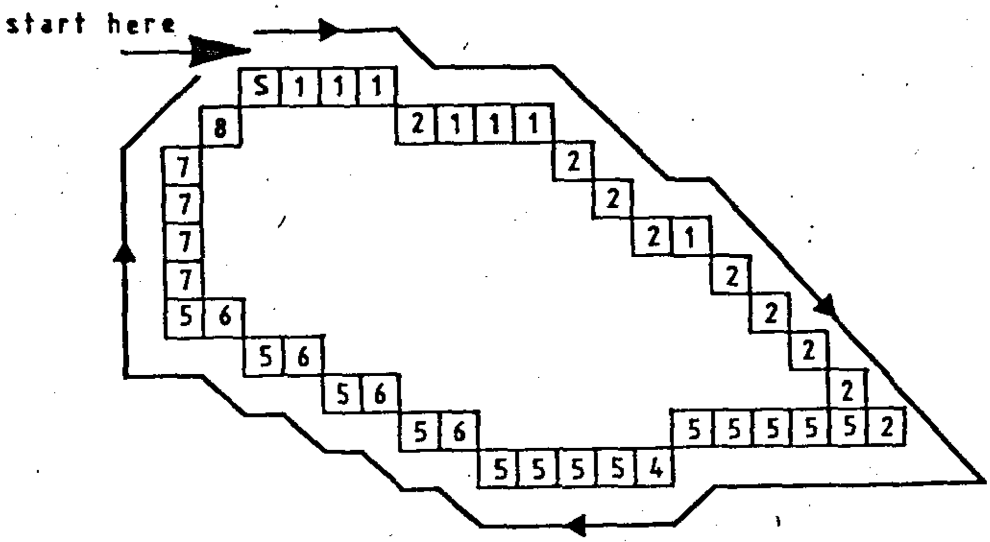


fig. 3.9

The *idea of starting the following strategy* not at n but at n-2 is to cope with cases like the one below, where the n-follower would have missed part B of the boundary out (Fig. 3.10). The n-follower after the last *six* looks for a new *six* (same direction) and then for a *seven* which is in fact the starting cell. On the other hand the n-2-follower looks for a *four* and *it finds it* including part B in the boundary of the shape. The only case in which the follower does not quite *follow* the *boundary* is that shown in Fig. 3.11.

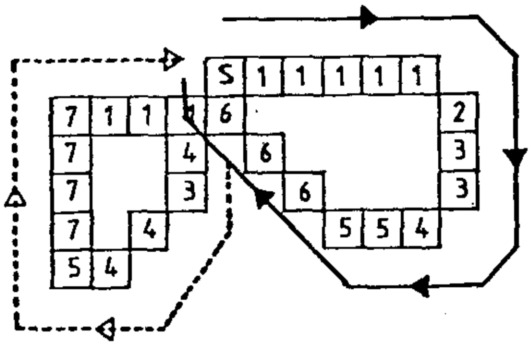


fig. 3.10

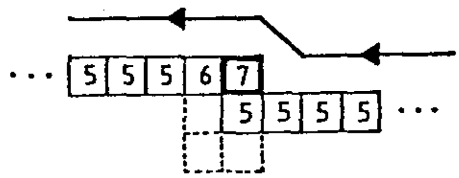


fig. 3.11

The leftmost '5' in the bottom row is not followed by any '3' (=5-2) neither any '4' nor '5', but by a '6' and a '7'. In this case '7' will remain white but such a case is prevented by the previous operator of edging so actually there will be no such cell as the one at position '7'.

So far it has been assumed that the boundary to be followed is continuous. This means that there are no gaps- black cells - in the path. But what happens if one comes across such a gap, which occurred due to noise or inefficiency of the two previous operators? The problem can be solved by improving the follower, so that it copes with small gaps, within a circle of radius $2\sqrt{2}$ of the last boundary cell.

Supposing that (i,j) (Fig. 3.12), is the current boundary cell and after a complete round, none of the eight neighbouring cells belongs to the boundary. Now say that the previous direction was '7'. The operator is centred on the (i,j-1) cell and it is applied once more. By doing this, one checks if any of the A_1 cells is *white* - (i-1,j-1) and (i+1,j-1) have already been checked in the previous round - and if one is encountered

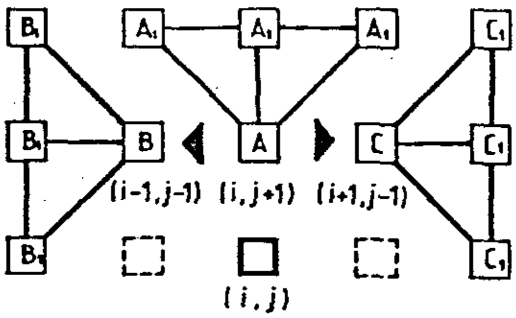


fig.3.12

continues from there. If not, cell B is the new centre and B_1 cells are checked. If it fails again a new attempt is tried on C by checking the C_1's. If this last search does not retrieve the next boundary cell, the follower fails and the procedure stops there. A message 'FAIL' indicates that the system can not go any further, and the method can not recognize the shape.

After a successful application of the boundary follower the processor halts and a message 'SEND' indicates that the array of data can be handed over to the VAX mini-computer for further calculations. If the re-application of the search strategy finds single white cells the procedure goes on until a whole string of them is encountered. When no more shapes are left an 'END' message appears on the screen.

3.3 VERTEX DETECTION AND LOCATION

The definition of a 2-D shape vertex, which consists of straignt lines only is:-

the intersection of two sides. The coordinates of the vertex are given by the common solution of the equations of these two sides.

The objective of the vertex detector is to spot the vertices as the boundary follower continues its operation, by using certain criteria. From the definition, a vertex is the intersection of two straight lines, which implies that these two lines have different slopes. So the main criterion that indicates the presence of a vertex is the change of direction. The method that is used here is:-

a) detect all the changes of direction and store the coordinates of the points, where the change happens, in a vertex-array.

b) form the equations of every two adjacent points and check if the rest of them lie within certain limits.

c) delete the vertices that are within the above limits and store the ones left in a new array, in order to use it for the next stage.

3.3.1 Change of Direction Criteria

As mentioned before, the digital representation of a straight line, is a sequence of equal length units, connected with links of length one.

So, it can be said that there are three main numbers which determine straight lines of the same direction. The number that shows the direction of the unit, the number that shows the direction of the link and the length of the unit. One can refer to them as N_U, N_L and L_U respectively. Whenever a change in any of these three is detected, a change of direction occurs and a vertex should be indicated.

The above is the main and general criterion for the indication of vertices. But before proceeding to the final expression of more specific criteria two other factors that might *confuse* the vertex detector must be taken into account. These are random noise and picture distortion, caused by the digitizer due to overheating of some chips, or caused by the camera. These make the straight lines look slightly curved. As a result of that, there are changes in the pattern, which in normal circumstances would suggest change of direction and existence of vertices. In order to overcome this ambiguous point, two kinds of vertices are introduced. The ones that occur due to definite change of direction, which are called *Certain-Vertices* or *Real-Vertices*, and the ones that occur due to change of direction, but not as clear as in the previous case, which are called *Suspected-Vertices* *Pseudo-Vertices*. The coordinates of the vertices of the first kind, are used to form the equations of the straight lines against which the vertices of the second kind will be checked. If their distance from the straight line is within a certain tolerance, the pseudo-vertices are dropped from the final vertex-array. Otherwise they become real-vertices themselves and new equations are formed to include them, according to the new rearrangement of the set of real-vertices.

The main criterion that indicates a real-vertex is that the current N_U and the next N_U differ by more than one direction numbers, i.e. they are not adjacent.

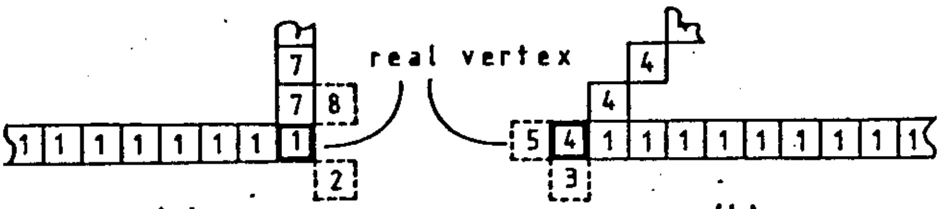


fig.3.13

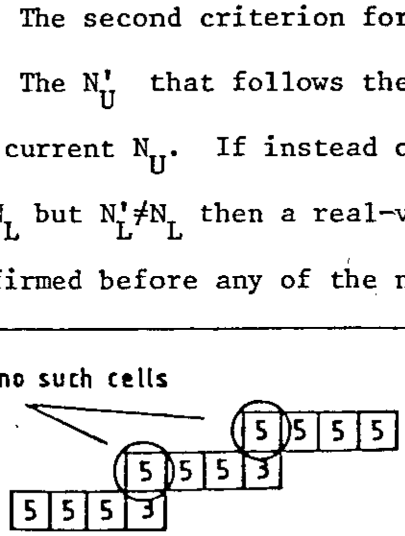
In the first example (Fig. 3.13a) N_U=1, so if the next cell is different to one and it is part of the same straight line there has got to be a link* i.e. one of the '8' or '2', which is not - seven here -. The result is that the last cell is marked as a real-vertex. Similar is the case demonstrated in Fig. 3.13b. The vertex detector looks always one cell ahead and at the same time saves the current N_U and N_L. It also saves the address of the last cell in case where a vertex is to be indicated. The coordinates of all the vertices which have been encountered are stored in a vertex-array. A second array points at the real-vertices and thus the next process uses only the vertices indicated by the pointers to form the equations of the sides of the shape.

The second criterion for real-vertices is:-

The N'_U that follows the link is of direction different from that of the current N_U. If instead of the expected link a new unit occurs, with $N'_U=N_L$ but $N'_L \neq N_L$ then a real-vertex is indicated. A real-vertex is not confirmed before any of the new N'_U or N'_L is met. In the example in Fig. 3.14

---

*



such a case as the one in the figure does not exist because of the way the edging operator works; see also edging §2.4.2.

N_U=1 and N_L=2. Then instead of the normal N'_U=1, a N'_U=2 occurs and this indicates a new unit. The first '3' could be just the new link N'_L but the occurrance of the second one confirms the existence of a real-vertex. The same could have happened if instead of a new unit the expected link occurred, followed by the new unit. Of course it is assumed that $N_U \neq N'_U$.

The third criterion for real vertices is:-

Instead of the expected link a new unit occurs with N'_U=N_L. In the example in Fig. 3.15 after the last '1' of the last unit a link '2' would be expected, but the occurrence of the new unit with N'_U=8 ($\neq N_L$=1) marks the existence of a real vertex at cell (i,j). The last two criteria are slight variations of the main one and there is small difference between them and the following pseudo-vertex criteria.

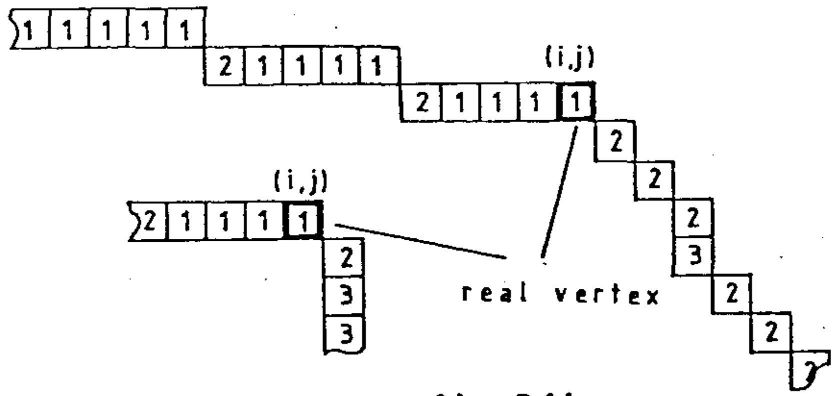


fig. 3.14

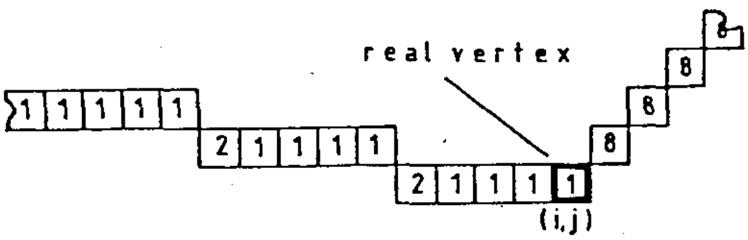


fig. 3.15

As it was mentioned before this type of vertices take their name from the fact that they may have been caused by distortion of the straight lines and not by actual change of direction. The first criterion for detection of pseudo-vertices is based on the feature that units of the same straight line have lengths that differ by 1 at least. Thus if the length of the new unit is different from the expected one, a change of direction is detected and a pseudo-vertex is indicated. This is true with the assumption that $N_U=N'_U$ and $N_L=N'_L$. In a genuine case one would have two sets of units. In each of them the lengths of every two elements differ by one, while the difference between elements of the two sets is greater than one. In reality though, units of the same straight line can have lengths with difference more than one. This means that the criterion was to be modified to cope with the latter case. The modification is:-

when the link is met the length of the unit is compared to the average of the lengths of the previous units (of the same straight line of course). If the absolute difference is greater than the 1/4 of this average, the last cell of the previous unit is marked as a pseudo-vertex; otherwise the follower continues its path. In the example in Fig. 3.16, cell (i,j) is the pseudo-vertex because the length of the new unit is 7, the average 2.5 and $|7-2.5|>2.5/4$. Like the second criterion for real-vertices the vertex detector is always one unit ahead.

The second criterion for the pseudo-vertices is:-

the direction N'_U of the next unit is the same as the current link N_L and the direction N''_U of the unit after the next is the same as the current, i.e. $N_U=N''_U$, and $N_L=N'_U$. This criterion copes with the case, where the link of the pattern changes slightly, or in other words a unit has taken the place of the link. Of course if N''_U were different then N_U, cell (i,j) would be turned into a real-vertex. In the example in Fig. 3.17 cell (i,j) is deemed to be a pseudo-vertex since the link '2' has been extended to a new unit of '2's. The same applies to cell (i+3,j+3) because after $N'_U=2$ a new $N''_U=1$ occurs, instead of an expected link $N'_L=1$ or 3.

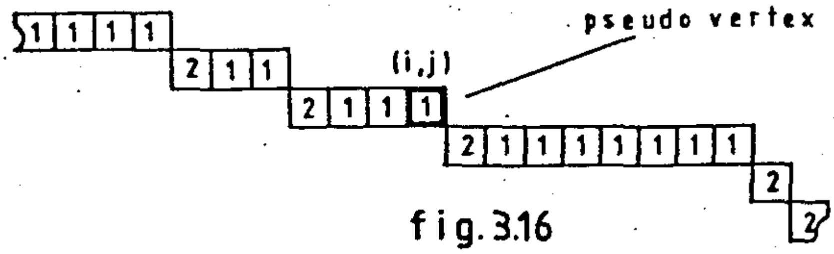


fig.3.16

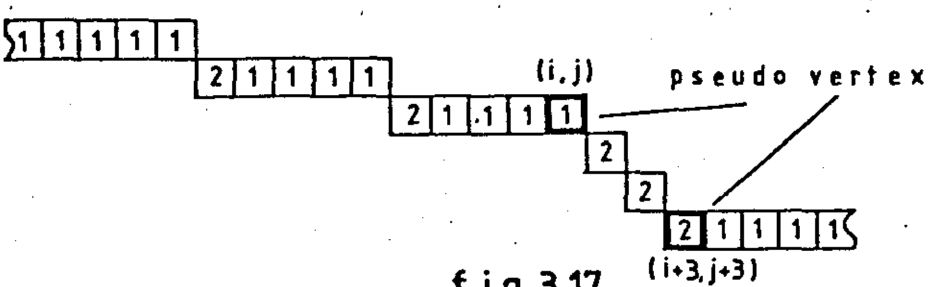


fig.3.17

The third criterion for the pseudo-vertices is:-

the direction N'_U of the new unit equals the link N_L of the current one, and the link N'_L of the new pattern equals the direction N_U of the current unit. This is slightly different from the previous one in that the cell (i,j) is the start of a straight line with a completely new direction from the previous line. That could be sufficient to mark (i,j) as a real-vertex, but care is taken to make it cope with cases, where the unit length is only one and the slopes are very close indeed. In Fig. 3.18 the two cases are demonstrated clearly. It can be noticed that had the new link

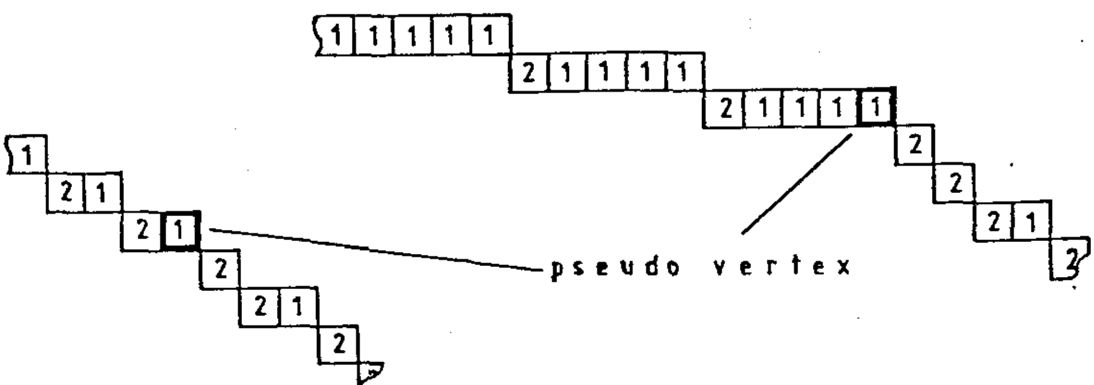


fig.3.18

N'_L been different from the current unit direction N_U, the cell (i,j) would have been a real-vertex according to the second criterion for the real-vertices.

The fourth criterion for pseudo-vertices is:-

the direction N'_U of the new unit is the same as the current one N_U, but the link N'_L is different from the current one N_L. The reason for not indicating a real-vertex here is, cases like the one in Fig. 3.19b. Here because of some unexpected distortion the difference in slope is not very clear and it is wiser to mark cell (i,j) as pseudo-vertex rather than as a real one.

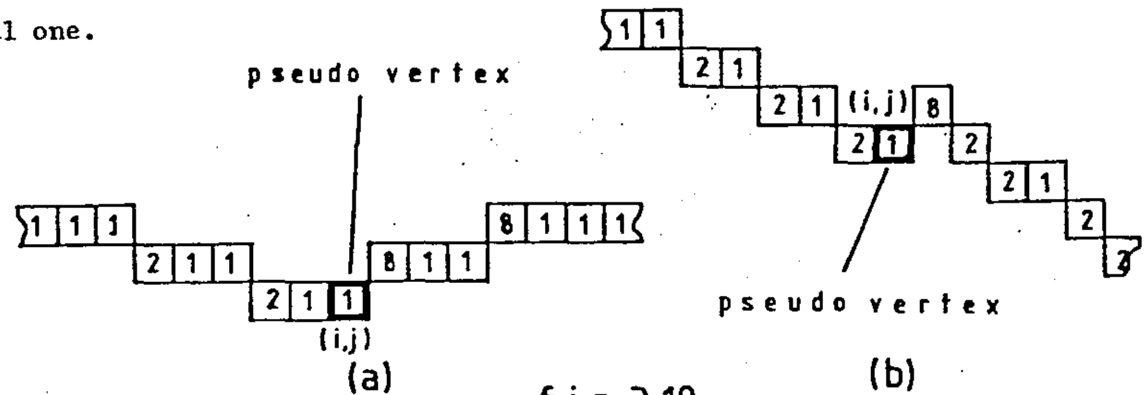


fig. 3.19

Finally by definition, the first and the last boundary cells met by the follower are marked as real-vertices. It will be seen later how a *virtual* first vertex can be eliminated from the vertex-array.

3.3.2 General Discussion on the Vertex Detector

The vertex detector is the main subroutine of the boundary follower, and consists of several shorter subroutines which perform the operation. Every time the follower finds a new cell that belongs to the boundary, the vertex detector is called to do the following jobs:-

a) Mark every new boundary cell with its direction number N. This is done in order to be able to detect the change of direction by comparing the new direction number to the old one. Three cells of the same N constitute a unit and N becomes the direction number of the unit N_U.

b) Save the current N_U in a special location, in order to compare it with the N'_U of any new unit. As new cells of the same N are met, a sum of them is formed. This represents the length L_U of the unit which is saved too, for later comparisons. The first non-N_U cell which differs* by one from N_U is deemed to be the link direction or link N_L and it is saved.

c) Mark the current vertex and save it. Vertices unlike other common cells, which are marked with a one digit number from 1 to 8, are marked with two digit numbers. The very first vertex is marked by number 11 and then every real-vertex by adding 1Ø to the previous real-vertex. The pseudo-vertices are marked by adding 1 to the previous vertex number. The position of the previous pseudo-vertex is saved too, in case that it needs to be turned into a real-vertex due to consecutive changes of the N_U by more than one.

d) Store the coordinates of the vertices and the pointers to the real ones into the vertex-array and pointer-array respectively. (Fig. 3.20).

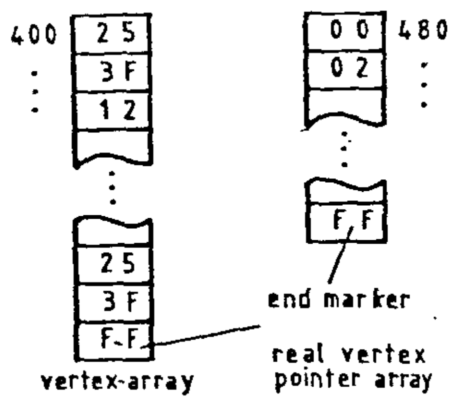


fig.3.2Ø

The top-left pixel is taken as the Ø,Ø point and every cell is determined by two numbers from Ø to 7F (hex), that are the X and Y coordinates of the cell. At the end of every array an end marker is placed to indicate that there are no more elements left. FF hex (or -1 decimal) is the end marker used here. The coordinates of the first vertex are taken again as the last couple of elements in the vertex-array before the end marker. The memory locations 4ØØ-47F

*Here is meant positional difference and not arithmetic difference e.g. '8' differs by one from '7' and '1', '2' differs by two from '4' and '8' etc.

are kept for the real-vertex pointer-array. The whole program is written in Z-8Ø assembly language. A flow chart of the boundary follower and vertex detector is given in Figs. 3.21-3.23.

3.4 REAL-VERTEX VERIFICATION

The two arrays obtained by the previous operations are fed to a VAX mini-computer for further processing. The result of this process is the formation of unit clauses, which will be the data for the recognizer. The main objective of this phase is to eliminate all the excessive vertices, in order to be left with the genuine ones only. The whole procedure can be divided into three main parts:-

a) the elimination of pseudo-vertices

b) verification of real vertices

c) formation of unit clauses.

3.4.1 The Elimination of Pseudo-Vertices

It was said before that the pseudo-vertices are caused by either change of direction or by distortion of the pattern. Those belonging to the first set need to be turned to real ones, while the vertices of the second set need to be eliminated from the final database.

The two arrays of data which are given by the vertex detection phase are handed over to the new program of real-vertex verification and are stored in two new one-dimensional arrays. These are V[i] for the vertices and N[i] for the pointers. The dimensions of these two arrays vary according to the desirable maximum number of vertices, that can be allowed in the procedure. At the end of each array -1 is placed as an end marker. After the loading procedure, the elements of the pointer-array are checked,

SEARCHING ROUTINE FLOWCHART

VERTEX DETECTOR FLOW CHART

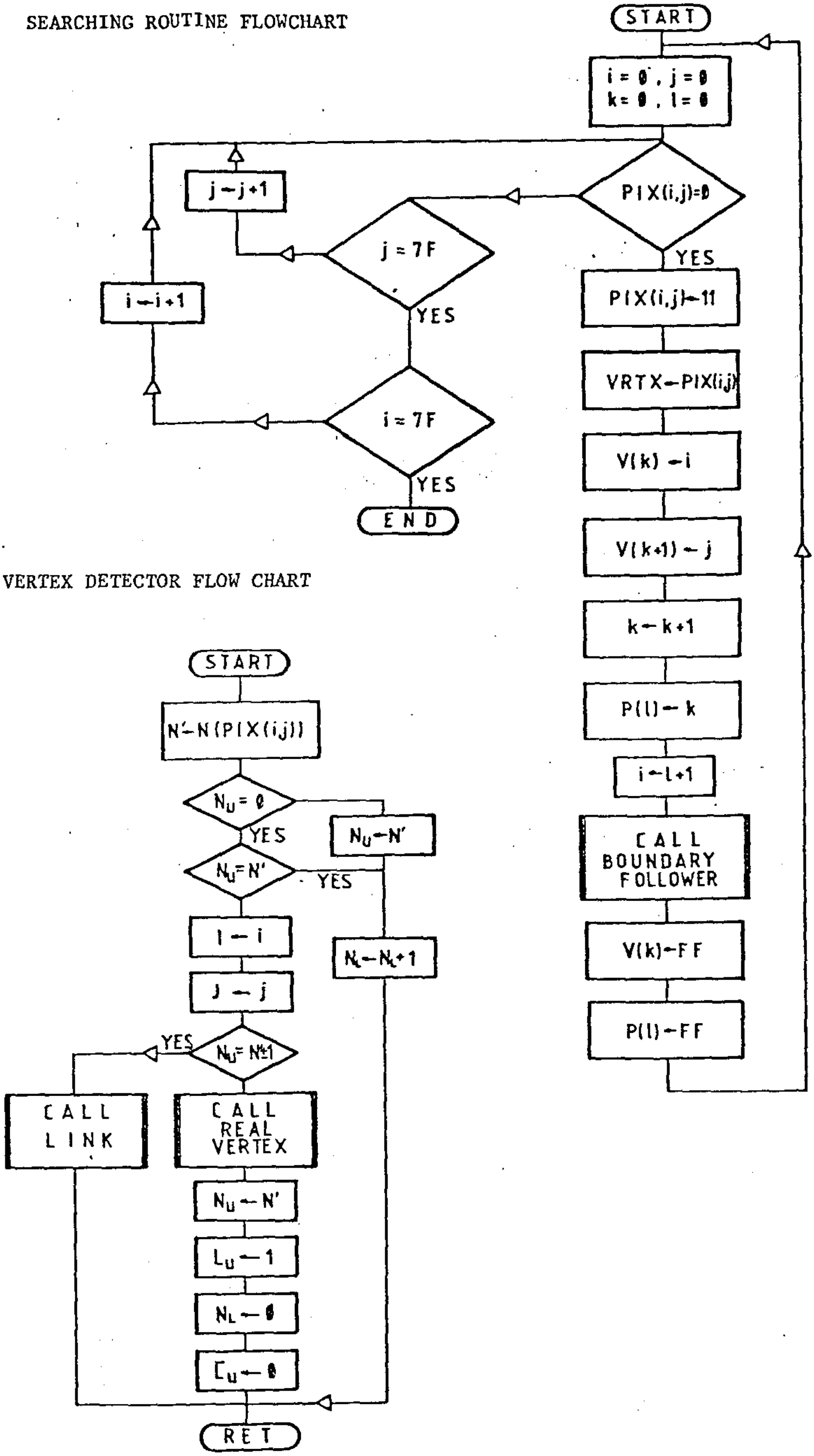
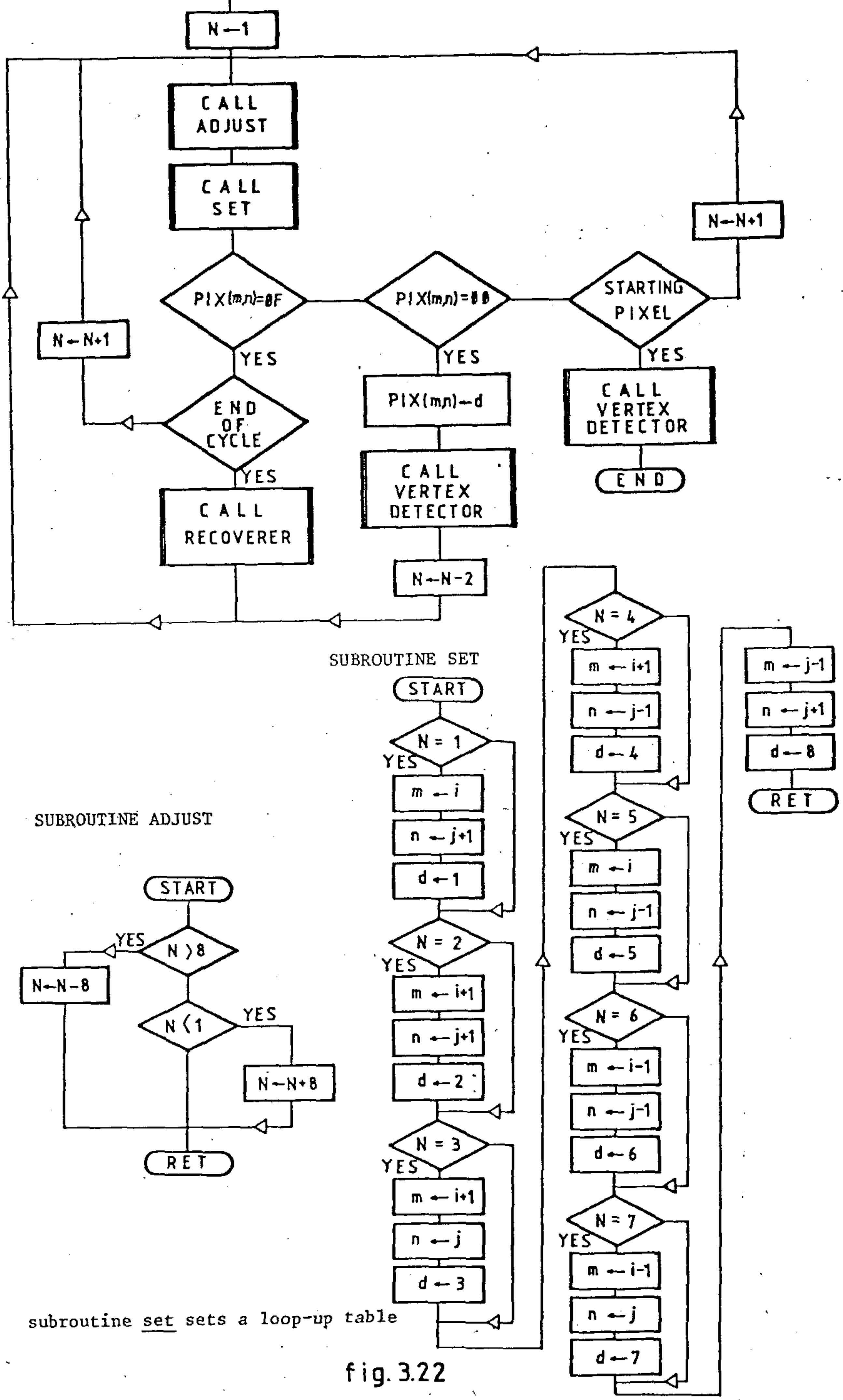


fig. 3.21



subroutine set sets a loop-up table

fig. 3.22

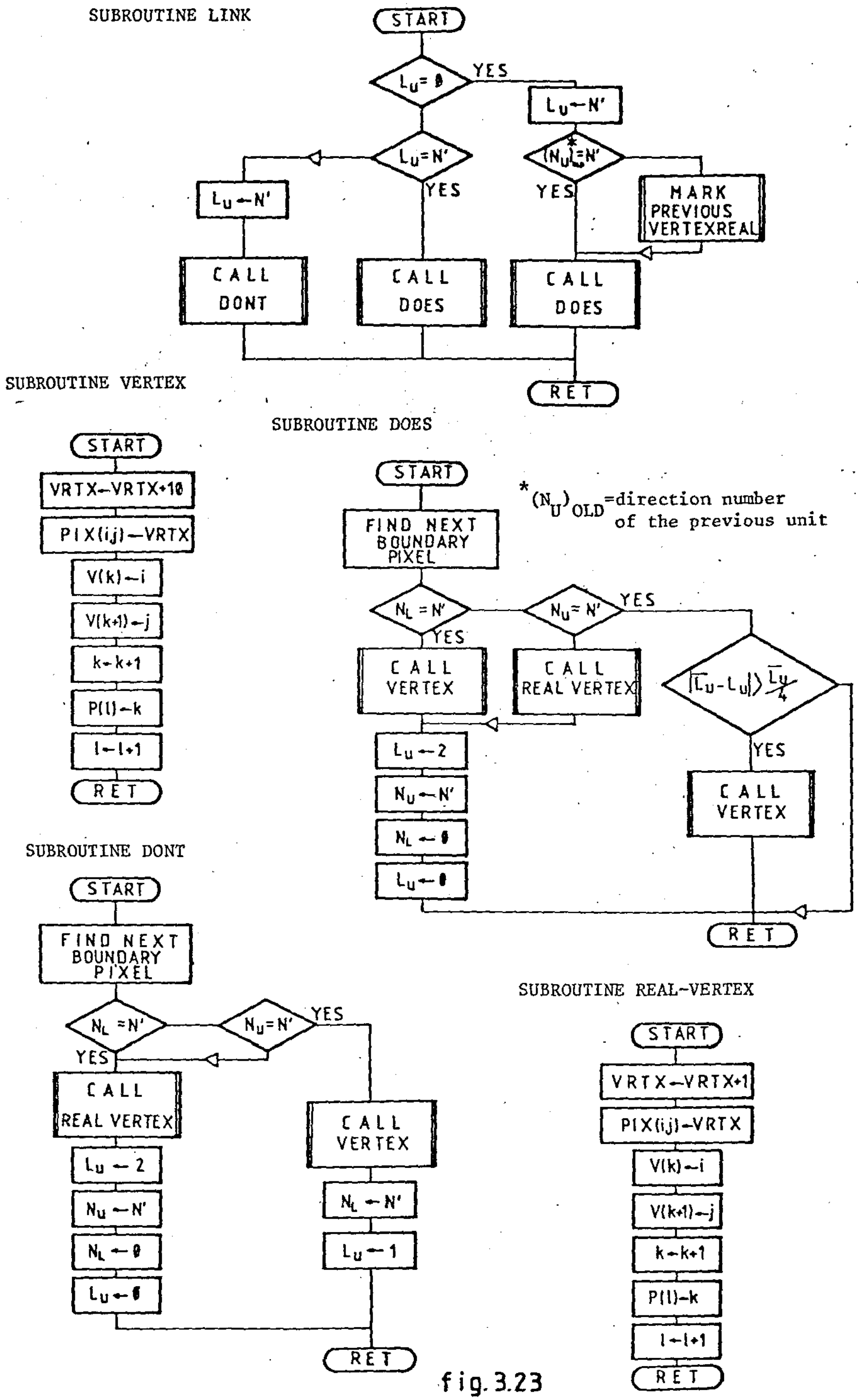

SUBROUTINE LINK
START
Lu= 0
YES
Lu←N'
Lu=N'
YES
(Nu)*=N'
YES
MARK PREVIOUS VERTEXREAL
Lu←N'
CALL DONT
CALL DOES
CALL DOES
RET
SUBROUTINE VERTEX
START
VRTX←VRTX+10
PIX(i,j)←VRTX
V(k)←i
V(k+1)←j
k←k+1
P(l)←k
l←l+1
RET
SUBROUTINE DOES
START
*(NU)OLD=direction number of the previous unit
FIND NEXT BOUNDARY PIXEL
NL = N'
YES
Nu= N'
YES
CALL VERTEX
CALL REAL VERTEX
|L̄u-Lu|> L̄u/4
YES
CALL VERTEX
Lu←2
Nu←N'
NL←0
Lu←0
RET
SUBROUTINE DONT
START
FIND NEXT BOUNDARY PIXEL
NL =N'
Nu=N'
YES
YES
CALL REAL VERTEX
CALL VERTEX
Lu←2
Nu←N'
NL←0
Lu←0
NL←N'
Lu←1
RET
SUBROUTINE REAL-VERTEX
START
VRTX←VRTX+1
PIX(i,j)←VRTX
V(k)←i
V(k+1)←j
k←k+1
P(l)←k
l←l+1
RET

fig. 3.23

in order to examine if the end marker exists. If not the method *fails* and a message is returned to indicate that the 2-D shape has more vertices than the array elements. The shape is classified as *other* automatically by default, but if an answer is wanted by the recognizer, the dimension of N[i] has to be increased sufficiently.

Next the coordinates of the first two real-vertices, pointed to by the first two elements of the pointer-array are used, to form the equation of the first side of the shape. The equations for these two vertices are:

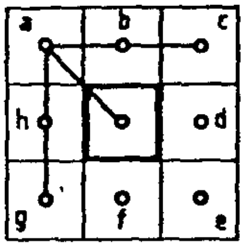
$$y_1 = \tan.x_1 + b \quad (1), \qquad y_2 = \tan.x_2 + b \quad (2)$$

and thus
$$\left.\begin{array}{l} \tan = (y_2-y_1)(x_2-x_1) \quad (3) \text{ and} \\ b = (x_2y_1-x_1y_2)/(x_2-x_1) \quad (4) \end{array}\right\} \text{ with } x_1 \neq x_2$$

By calculating the slope and the constant factor of the equation for the straight line determined by the two real-vertices, every vertex between these two is checked to find out if it belongs to the same straight line or not. Here again because of the digitized picture, a tolerance has to be introduced, taking into account the idiomorphic structure of the quantized straight line. Since a vertex can be, practically, approximated by any of the nine pixels of a 3×3 square (Fig. 3.24a) and the largest distances from the middle is that of the diagonal, the tolerance chosen is $\sqrt{2}$. Pixels with distance greater than this from the straight, are eliminated. The distance of the point A from the straight line (Fig. 3.24b) is given by:

$$\frac{1}{x^2} + \frac{1}{y^2} = \frac{1}{d^2}$$

and a vertex is eliminated if $d>\sqrt{2}$ or if $\max(x,y)>2$. This is the condition which is used here. x and y are the values obtained by solving the equations $y=\tan.x_n+b$, $x=\frac{y_n}{\tan}-b$, with x_n and y_n being the coordinates of the n^{th} pseudo-vertex between real-vertices 1 and 2 and tan and b, given from (3) and (4) respectively.



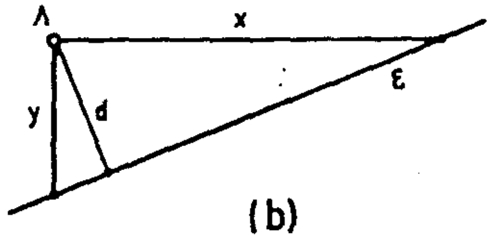


fig. 3.24

If the distance of some of the pseudo-vertices is greater than the given tolerance, it means that some of them are real-vertices and a new test should be tried according to the emerged new conditions. Supposing that there are n pseudo-vertices to be checked against the equation of the real-vertices V_i and V_{i+1} and m of them are found with distances greater than d_{lim} from the straight $\overline{V_i,V_{i+1}}$. If V_{max} is the pseudo-vertex with the maximum distance then, this is taken as a new real vertex and the original straight line $\overline{V_i,V_{i+1}}$ is changed to $\overline{V_i,V_{max}}$ and $\overline{V_{max},V_{i+1}}$. The rest of the n-1 pseudo-vertices are checked against the equation of the real-vertices between which are included in the array. If their distances are within the allowed limit, they are ignored - effectively eliminated - otherwise the same procedure is followed until all of the vertices are checked. The new real vertex is marked as such by inserting an extra pointer to it in the pointer-array and moving the pointers below by one position downwards (Fig. 3.26). The coordinates of every real-vertex met are stored in a new two-dimensional array A[i][j].

In the example of Fig. 3.25, v_2 and v_1 have distances greater than the limit with $d_2>d_1$, while V_3 is within tolerance. Two new straight lines are formed $\overline{V_1\ v_2}$ and $\overline{v_2V_2}$ and the new tests show none of the remaining v_1,

v_3 is a new real vertex. Thus v_2 goes into the final vertex-array. When all the real-vertices have been used and all the pseudo-vertices have been checked, the procedure ends.

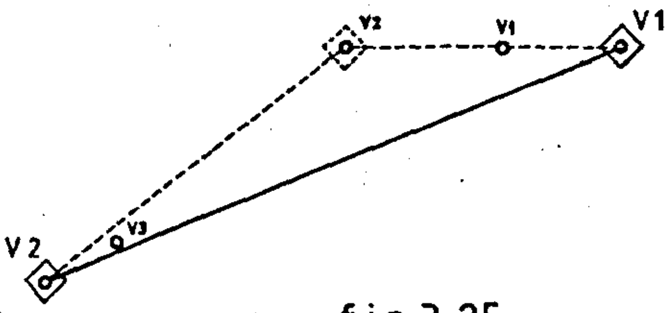


fig.3.25

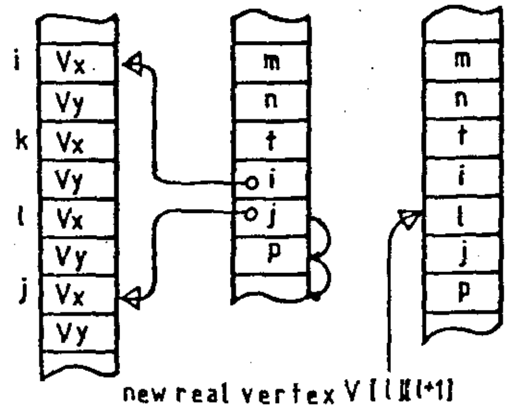


fig.3.26

3.4.2 Real-Vertex Verification

Before the array of real-vertices, produced by the previous procedure has reached its final form, two points must be examined. The existence of nearby vertices and the very first vertex.

It has been mentioned earlier that points within the same 3×3 square can be considered as one. This implies that vertices with distance less than $2\sqrt{2}$ - max distance across the diagonal of the 3×3 square - from each other should be considered as the same vertex. It is common that the vertex of some - mainly acute-angles are approximated by patterns like the ones in the Fig. 3.27. This happens because the averaging operator rounds the sharp parts by removing the odd pixels. This of course causes the existence of more than one real-vertex in a very small area due to abrupt changes of direction. A solution to this problem is, to check the distances of the adjacent vertices against a certain maximum distance, within which the two vertices are considered to be one. This is obtained by moving the elements of the array one position upwards and eliminating - by overwriting - the first of the two neighbouring vertices. The above technique will take care of the first and last pixels which will be taken as one vertex - only one cell apart - although both are marked as real-vertices.

The very first boundary pixel is always a real-vertex except for one case. The proof is the following:-

Since the follower searches from left to right, top to bottom, the very first boundary pixel will be the end of an ascending line and the beginning of a descending - or an horizontal - one. Hence it is a real vertex. But it fails when the ascending line is very close to the horizontal.

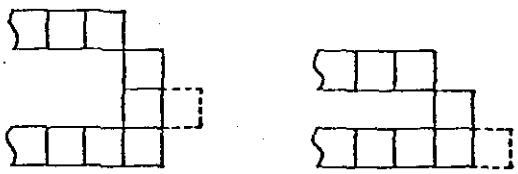


fig.3.27

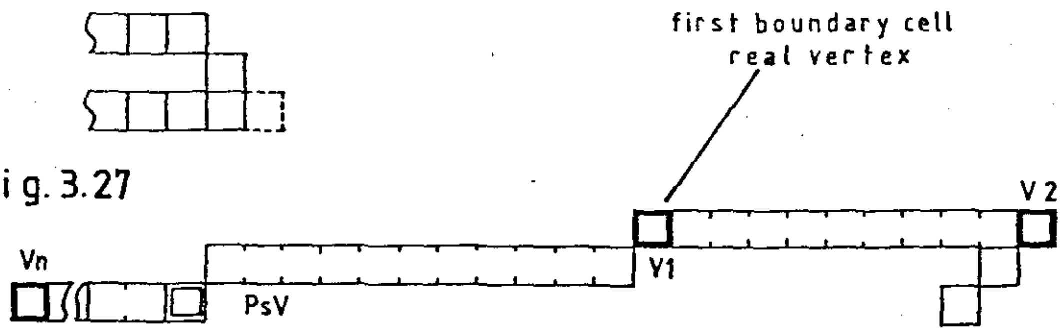


fig.3.28

To cope with this after the test of adjacent vertices has been completed, the equation of the straight line between the second and penultimate vertex is formed and the first one is checked against it. Since it belongs to that straight line it will be eliminated without causing any further problems.

The existence of a pseudo-vertex in the above case will not cause any problems if there are any pseudo-vertices between the first and the last real-vertices. If there is such a vertex, it will be below the first - otherwise it would have been the first -. The test for this PsV (Fig.3.28) will be between V_n and V_1 instead of V_n and V_2, but since V_1 and V_2 belong to the same straight line, it will not make any difference to the final result.

3.4.3 Formation of Unit Clauses

This part prepares the data which will be consulted by the recognizer, to make the final classification of the shape. The unit clauses created in this phase are:- a) conn(c_1,c_1 c_2,c_2) b) line(c_1c_2,n), c) slope(c_1 c_2,m), d) sqrline(c_1 c_2,ℓ), and e) angle(c_1,k).

c_1,c_2,c_3 are characters from the set ch={a,b,c,...,x,y,z}, and ℓ,m,n are integer numbers.

a) conn(X,Y,Z):- The conn unit clause is the most important one because it indicates the connectivity between the vertices and hence the structure of the shape. The first and last variables refer to the vertices being connected, while the one in the middle refers to the line that joins them. For example 'conn(a,ab,b)' means that vertex a is connected to b by the side ab. The direction of the connectivity is important, so conn(a,ab,b)$\neq$ conn(b,ba,a).

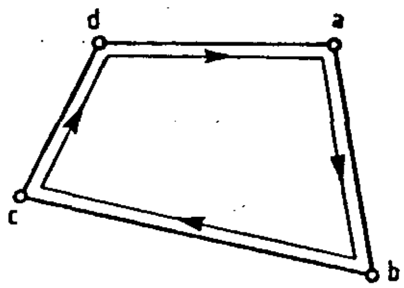
$$\left.\begin{array}{l} A_{\emptyset\emptyset} \rightarrow ChA_{\emptyset} \\ A_{1\emptyset} \rightarrow ChA_1 \end{array}\right\} \quad \text{conn}(ChA_{\emptyset}, ChA_{\emptyset}ChA_1, ChA_1)$$

$$A_{2\emptyset} \rightarrow ChA_2 \;\} \quad \text{conn}(ChA_1, ChA_1ChA_2, ChA_2)$$

$$\vdots$$

$$A_{n\emptyset} \rightarrow ChA_n \;\} \quad \text{conn}(ChA_{n-1}, ChA_{n-1}ChA_{\emptyset}, ChA_{\emptyset}) \qquad (A_{n\emptyset} \equiv A_{\emptyset\emptyset})$$

A character array is associated with every vertex, by using the same subscript. Successive vertices mean connected points and subsequently the forming of *conn* unit clauses. The respective elements of the character array ChA fill the places of the variables as shown above. When a vertex equal to the first is met, the end of the procedure is marked. Of course the fact that the first and last vertices are the same, is deliberate to supply the connectivity between last and first vertex (which would not have been successive otherwise).



conn(a,ab,a).
conn(b,bc,c).
conn(c,cd,d).
conn(d,da,a).

fig. 3.29

The example in Fig. 3.29 illustrates the order and direction of the connectivity that gives rise to the respective conn's.

b) <u>Line(X,n) and sqrline(X,m)</u>:- The first one is used to compare the sides of the shape when equality is demanded. X is the side concerned and n an integer number representing the length of the side. The second unit clause is similar to *line*, but m is n^2 and it is used when the theorem of Pythagoras is needed. n and m are calculated as the distances of consecutive vertices as shown below.

$$D_1 = A[i][\emptyset]-A[i+1][\emptyset] \qquad (1)$$

$$D_2 = A[i][i]-A[i+1][1] \qquad (2)$$

$$S[i] = \sqrt{D_1^2+D_2^2} + \emptyset.5 \qquad \begin{array}{l} n = s[i] \\ m = (s[i])^2 \end{array}$$

A[i][Ø], A[i+1][Ø]: the ordinates of vertices i and i+1

A[i][1], A[i+1][1]: the abscissae of vertices i and i+1 s[i] is an integer array and $\sqrt{D_1^2+D_2^2}$ a real number, which means that s[i] will be assigned the integer part of the square root. To round the number correctly, ∅.5 is added to the square root. m is obtained by squaring n. Both m and n have to be integer because in PROLOG (the language in which the recognizer is written) only integers can be used. The X is filled in by the same method as variable Y in 'conn(X,Y,Z)'.

c) <u>Slope (X,θ)</u>:- Because of the use of a non-conventional system of co-ordinates the slope $S\ell$ of straight line ε (Fig. 3.29) is given by:-

$$S\ell = \tan\theta = \frac{dx}{dy}$$

The slope is calculated by using D_1 and D_2 from (1), (2) of the previous calculations of the line (X,n):-

$$T[i] = \frac{D_2}{D_1} \quad \text{with } D_1 \neq \emptyset \quad \text{and hence}$$

$$\theta[i] = \frac{18\emptyset}{3.1416} \arctan T[i]+\emptyset.5 \text{ when } D_1 \neq \emptyset$$

$$\theta[i] = 9\emptyset^{o} \quad \text{when } D_1 \neq \emptyset$$

This is the angle θ in degrees that ε forms with the $\vec{OX}$ axis (clockwise is taken as positive). θ is also an integer and x is supplied as described above. The 'slope(X,k)' unit clauses are used to indicate parallel lines.

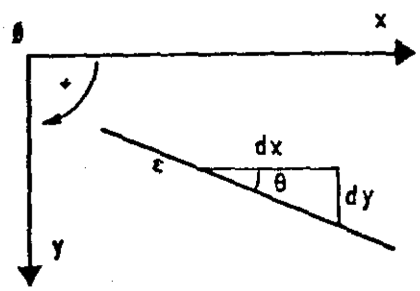


fig. 3.29

d) angle(Z,ℓ):- This unit clause is used to indicate a non-convex quadrilateral. Since the equality of angles of a shape is implied by the equality of subtended sides, the *angle* unit clauses are of no other use. Y is the character that determines the angle and ℓ is an integer which gives the amplitude of the angle in degrees. The amplitude a of an angle that is left on one side of a line parallel to the OX axis and passing through its vertex V (Fig. 3.30a) is given by:

$$\alpha = |\phi - \theta| \qquad (1)$$

with ϕ and θ the *slopes* of the two lines of the angle as defined in the previous section. In case of angle β, the amplitude of β' is calculated by (1), which however is equal to β. The formula is different when the parallel to OX intersects the angle (Fig. 3.30b):-

$$\alpha = 180 - |\phi - \theta| \quad (2)$$ (angles measured in degrees).

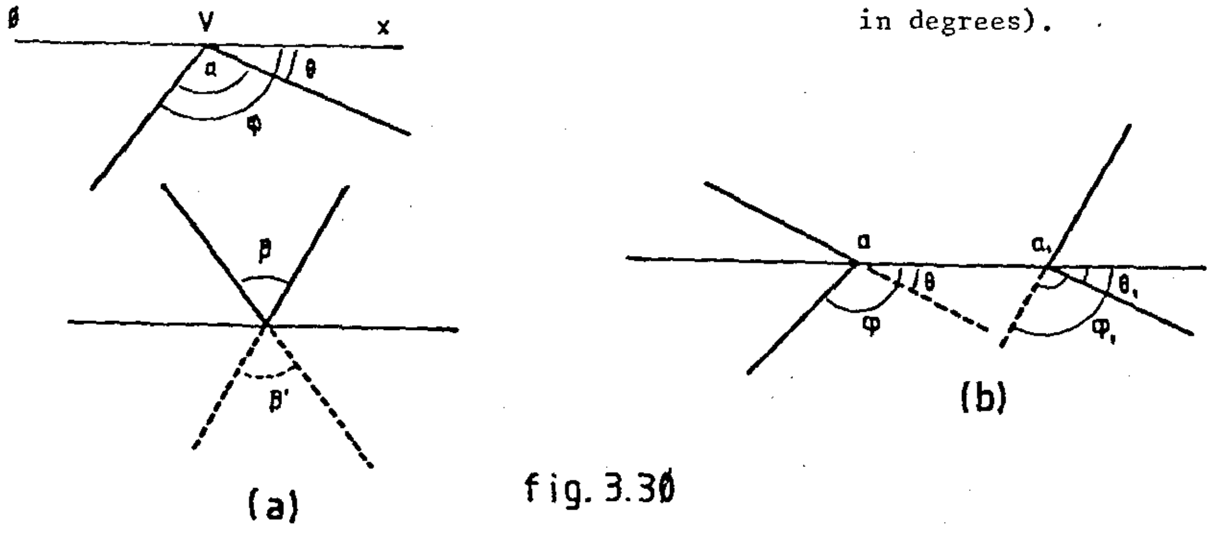


fig. 3.30

In a convex quadrilateral there are two angles that belong to the first category at least. These can be found by comparing the abscissae of their vertices. They are the ones with the minimum and maximum distance from OX respectively. Thus by finding the vertices with the smallest and biggest abscissae, the use of the adequate formula is chosen. In the case when there are two maxima and two minima the first and third angles are calculated by formula (1). In both of the shapes in the example of Fig. 3.31a, the angles a and c are calculated by formula (1) and b and d by (2).

In a convex quadilateral (abcd) there is the following relation between the abscissae of the vertices.

If $V_y(1)$ = min and $V_y(2)$ = max, then $V_y(3)>V_y(4)$ *always*. This does not happen in the case of non-convex quadrilaterals (a' b' c' d') and thus is the condition for using formula (1) only for all four angles. Both (1) and (2) calculate angles smaller than 18$\emptyset^o$, and so fail in the case of a non-convex quadrilateral, because they calculate the exterior of the real angle (e.g. c' and c"). However this is easy to be checked by using the known relation a'+b'+c'+d'=36$\emptyset^o$ (3). If the sum is less than 36$\emptyset^o$, a non-convex-quadrilateral is indicated. In the case of a"b"c"d", which is a non-convex quadrilateral with $V_y(3)>V_y(4)$ the formula for convex quadrilaterals is used, but the fact that (3) is not satisfied reveals it is non-convex.

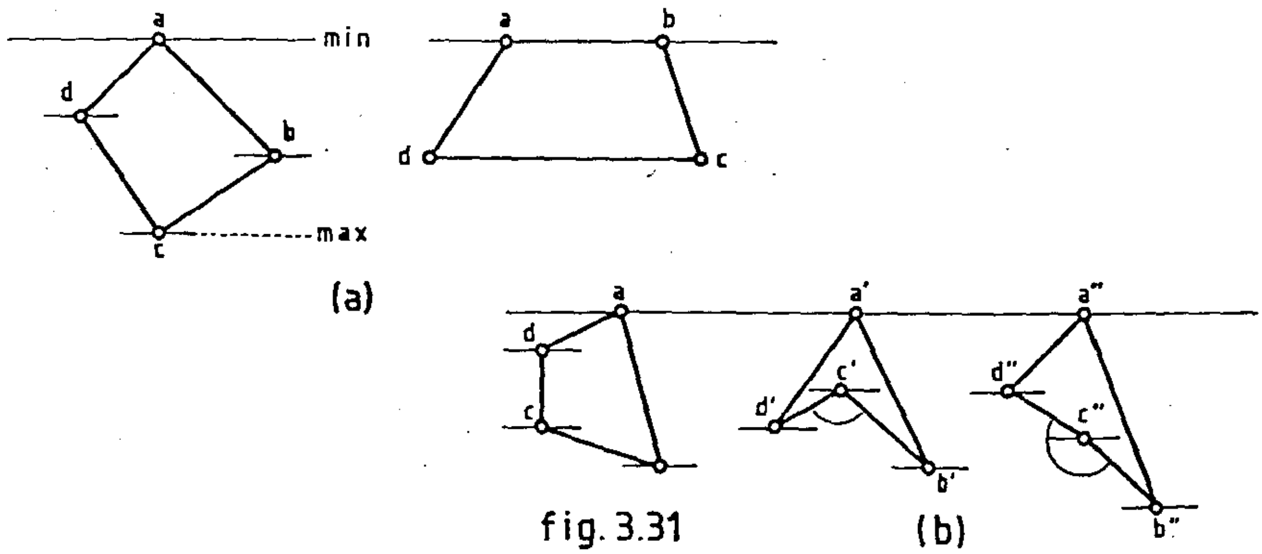


fig. 3.31

SUMMARY - CONCLUSIONS

- The representation of straight lines on a grid is given by sequences of horizontal or vertical lines of pixels connected with diagonal joints called links.
- The pattern that represents a straight line on a digital grid remains unchanged throughout the line. Likewise the limit remains

the same with respect to the same line.

- The edge follower is a 3×3 square window centred on cells belonging to the boundary of a shape. The eight cells round the middle cell are checked for the next boundary cell using an n-2 sequence. A recovery operator takes care of small gaps in the boundary.
- The vertices of a shape are points at which a change of direction is observed. The change of direction is implied by a change in the pattern that represents the two sides of the particular vertex. Such genuine vertices are called *real-vertices*.
- Vertices caused by distortion of the picture due to noise, the camera, the screen or other reasons are called *pseudo-vertices*. These may become real in certain cases.
- A number of criteria checks the type of the vertices and forms two arrays with their coordinates. The real-vertices are used to form the equations of the sides of the shape. If any of the pseudo-vertices are farther than a limit distance from these sides they become real-vertices themselves.
- When all the real-vertices have been verified they form a final array with their coordinates, which is responsible for the formation of the data used by the next stage of recognition.
- The data used by the recognizer carry information about the structure of the shape - *conn* - or its special properties such as length of its sides, amplitude of its angles etc.
- If the number of vertices identified by the vertex detector is too large the method fails. In this case the shape is classified as *other* by default.

REFERENCES

1. M.C.V. Pitteway:- *"Algorithm for drawing ellipses or hyperbolae with a digital plotter"*, The Computer Journal, Vol.1Ø, No.3, Nov. 1967.

2. S.A. Dudani:- *"Region Extraction Using Boundary Following"*, Hughes Research Laboratories, Malibu, C.A., 'Pattern Recognition and A.I.', edited by C.H. Chen, pp.217-22Ø, 1976.

Chapter 4

SYNTACTIC PATTERN RECOGNITION

4.1 INTRODUCTION

This chapter examines a relatively new and promising approach to pattern recognition, based on the utilization of concepts from formal language theory. This approach is frequently referred to as *Syntactic Pattern Recognition*, although terms such as linguistic pattern recognition, grammatical pattern recognition and structural pattern recognition are often found in the literature.

The basic difference between syntactic pattern recognition and the other approaches is that the former explicitly uses the structure of patterns in the recognition process. Analytical approaches, on the other hand, deal with patterns on a strictly quantitiative basis, thus ignoring inter-relationships between the components of a pattern. From the viewpoint of pattern description or modelling, class distribution or density functions are used to describe patterns in each class in the decision-theoretic approach but syntactic rules or grammars are employed to describe patterns in the syntactic approach. The effectiveness of these approaches appears to be dependent upon the particular problem at hand. Often a mixed approach needs to be applied. As a matter of fact, it is sometimes difficult to distinguish sharply between syntactic and non-syntactic recognizers. Of course the existence of a recognizable *structure* is essential for the success of the syntactic approach. For this reason, syntactic pattern recognition research has been largely confined so far to pictorial patterns which are characterized by recognizable shapes, such as characters, chromosomes and particle collision photographs

The interest in syntactic pattern recognition may be traced to the early 1960's, although research in this area did not gain momentum until later in that decade. Even today, however, many of the major problems associated with the design of a syntactic pattern recognition system have been only partially solved.

In the following paragraphs a general model of a syntactic pattern recognition system is discussed, and some basic concepts of the formal language theory are given. The two main kinds of recognizer are analysed and some pattern recognition languages are mentioned.

4.2 SYNTACTIC APPROACH TO PATTERN RECOGNITION

The many different mathematical techniques used to solve pattern recognition problems may be grouped in two general approaches:- the *decision-theoretic* (or *discriminant*) approach and the *syntactic* (or *structural*) approach. In the decision-theoretic approach, a set of characteristic measurements, called features, are extracted from the patterns. The recognition of each pattern (assignment to a pattern class) is usually made by partitioning the feature space. In some pattern recognition problems, the syntactic or structural approach has been proposed. This approach draws an analogy between the (hierarchical, or treelike) structure of patterns and the syntax of languages. Patterns are specified as being built up out of subpatterns in various ways of composition, just as phrases and sentences are built up by concatenating words, and words are built up by concatenating characters. Evidently, for this approach to be advantageous, the simplest subpatterns selected, called *pattern primitives*, should be much easier to recognize than the patterns themselves. The *language* that provides the structural description of patterns in terms of a set of primitives and their composition operations is sometimes called the *pattern recognition language*. The rules governing the composition of primitives into patterns are usually specified by the so-called *grammar* of the pattern description language. After each primitive within the pattern is identified, the recognition process is accomplished by performing a *syntax analysis* or *parsing* of the *sentence* describing the given pattern to determine whether or not it is syntactically (or grammatically) correct with respect to the specified grammar. In the meantime, the syntax analysis also produces a structural description of the sentence representing the given pattern (usually in the form of a tree structure).

The syntactic approach to pattern recognition provides a capability for describing a large set of complex patterns, by using small sets of simple pattern primitives and grammatical rules. One of the most essential aspects of this capability is the use of the recursive nature of a grammar. A grammar (rewriting) rule can be applied any number of times, so it is possible to express in a very compact way some basic structural characteristics of an infinite set of sentences. Of course, the practical utility of such an approach depends on our ability to recognize the simple pattern primitives and their relationships, represented by the composition operations.

the structural information which describes each pattern is important, and the recognition process includes not only the capability of assigning the pattern to a particular class (to classify it), but also the capacity to describe aspects of the pattern that make it ineligible for assignment to another class. A typical example of this class recognition problem is picture recognition or, more generally speaking, scene analysis. In this class of recognition problem, the patterns under consideration are usually quite complex and the number of features required is often very large. Thus the idea of describing a complex pattern in terms of a (hierarchical) composition of simpler patterns is adopted. Also, when the patterns are very complex and the number of possible descriptions is very large, it is impractical to regard each description as defining a class. Consequently, the requirement of recognition can only be satisfied by a description for each pattern rather than by the simple task of classification. Fig. 4.1b shows the structural description of scene 4.1a.

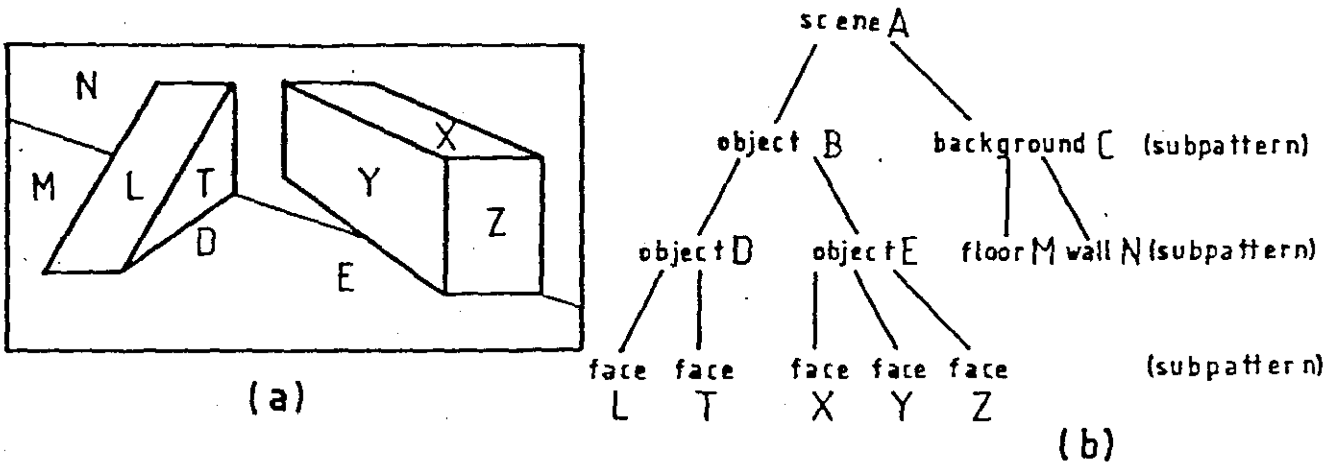


fig. 4.1

In order to represent the hierarchical (treelike) structure information of each pattern, that is, a pattern described in terms of simpler subpatterns

The various relations or composition operations defined among subpatterns can usually be expressed in terms of logical and/or mathematical operations. For example, if *concatenation* is chosen as the only relation (composition operation) used in describing patterns, then for the pattern primitives shown in Fig. 4.2a, the rectangle in Fig. 4.2b would be represented by the string aaaabbccccdd. More explicitly, if '+' is used for the *head-to-tail concatenation* operation, the rectangle in Fig. 4.2b would be represented by a+a+a+a+b+b+c+c+c+c+d+d, and its corresponding treelike structure would be that shown in Fig. 4.3.

An alternative representation of the structural information of a pattern is a *relational graph*.[3] Fig. 4.4b shows a relational graph

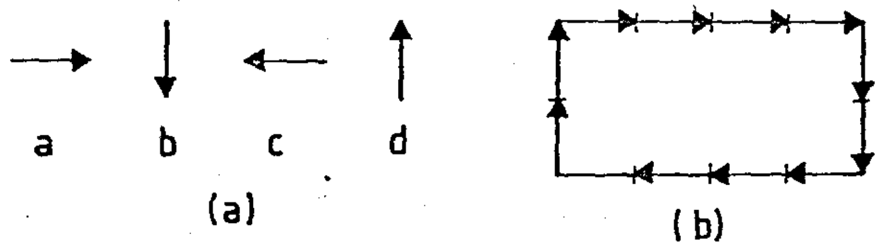


fig.4.2

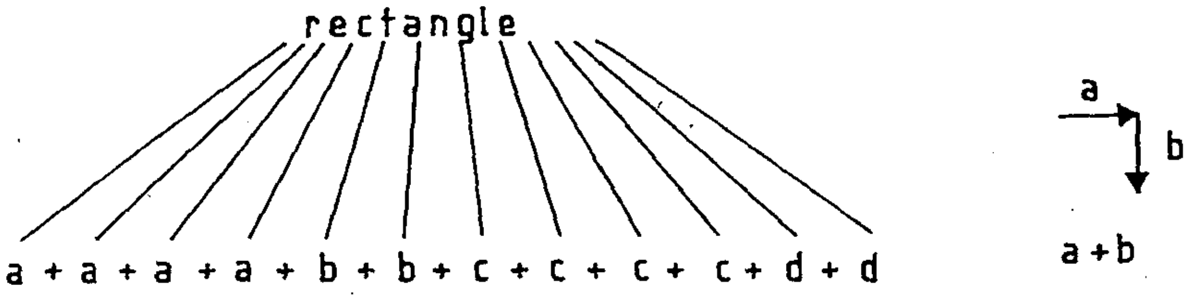


fig. 4.3

of picture F in Fig. 4.4a. Since there is a one-to-one corresponding relation between a linear graph and a matrix, a relational graph can also be expressed as a *relational matrix*. In using the relational graph for pattern description, the class of allowed relations can be broadened to include any relation that can be conveniently determined from the pattern. It is worth noticing that:

a) *the concatenation is the only natural operation for one-dimensional languages*

b) a graph, in general, contains closed loops, whereas a tree does not. With this generalization, richer descriptions can be expressed than with tree structures.

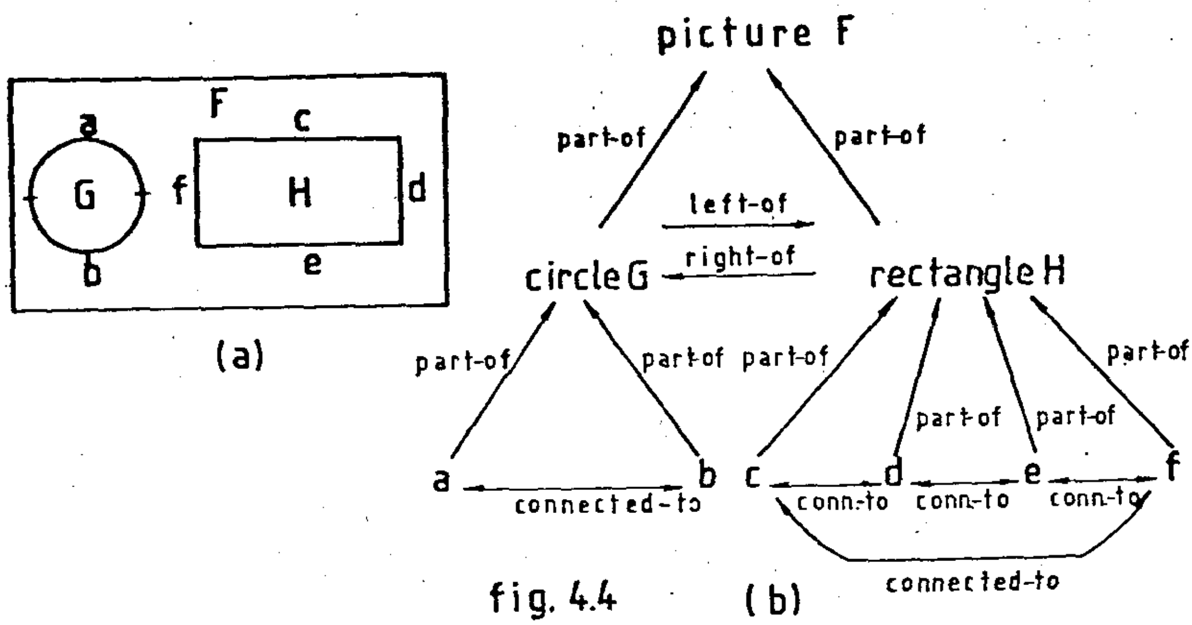


fig. 4.4

However, the use of tree structures provides a direct channel for adapting the techniques of formal language theory to the problem of compactly representing and analysing patterns containing a significant structural content. Because of the adaptation of techniques from formal language theory, the syntactic approach is also sometimes called the *linguistic approach*. Nevertheless, it is probably more appropriate to consider that the techniques of formal language theory are only tools for the syntactic approach rather than the approach itself.

4.3 SYNTACTIC PATTERN RECOGNITION SYSTEM

A syntactic pattern recognition system can be considered as consisting of three major parts, namely, preprocessing, pattern description or representation, and syntax analysis. Usually, the term *syntactic pattern recognition* refers primarily to the last two parts. A simple block diagram[1a] of the system is shown in Fig. 4.5.

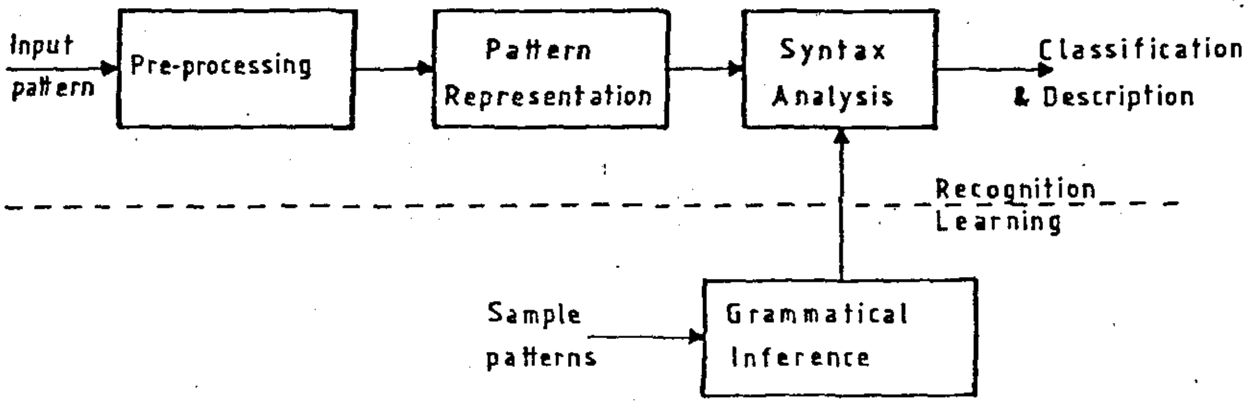


fig.4.5

The functions of preprocessing include pattern encoding and approximation, and filtering, restoration and enhancement. An input pattern is first coded or approximated by some convenient form for further processing. For example, a black-and-white picture can be coded in terms of a grid (or a matrix) of Ø's or 1's, or a waveform can be approximated by its time samples. In order to make the processing in the later stages of the system more efficient, some sort of *data compression* is often applied at some stage. Then, techniques of filtering, restoration and/or enhancement are used to clean the noise, to restore the degradation and/or improve the quality of the coded (or approximated) patterns. The output of the preprocessor gives patterns of reasonably *good quality*. Each pattern is then represented by a language-like structure (e.g. a string).

The pattern representation process consists of pattern segmentation, and primitive (feature) extraction. In order to represent a pattern in terms of its subpatterns, the pattern must be segmented, and the primitives in it identified (or extracted). This means that, each preprocessed pattern is segmented into subpatterns and pattern primitives based on prespecified syntactic or composition operations. Then, in turn, each pattern is represented is identified with a given set of pattern primitives. At this point, each pattern is represented by a set of primitives with specified syntactic operations. For example, in terms of the concatenation operation each pattern is represented by a string of (concatenated) primitives. The decision whether or not the representation (pattern) is syntactically correct (i.e. belonges to the class of patterns described by the given syntax or grammar) will be made by the *syntax analyser* or *parser*. When performing the syntax analysis or parsing, the analyser can usually produce a complete syntactic description, in terms of a parse or parsing tree, of the pattern, provided that the latter is syntactically correct. Otherwise, the pattern is either rejected or analysed on the basis of other given grammars, which presumably describe other possible classes of patterns under consideration.

The simplest form of recognition is probably *template matching*. The string of primitives representing an input pattern is matched against strings of primitives representing each prototype or reference pattern. Based on a selected *matching* or *similarity* criterion, the input pattern is classified in the same class as the prototype pattern that is the *best* match to the input. The hierarchical structural information is essentially ignored. A complete parsing of the string representing an input pattern, on the other hand, explores the complete hierarchical structural description of the pattern. In between there are a number of intermediate approaches.

For example, a series of tests can be designed to test the occurrence or nonoccurrence of certain subpatterns (or primitives) or certain combinations of subpatterns or primitives. The result of the tests (e.g. through a table look-up, a decision tree, or a logical operation) is used for a classification decision. The selection of an appropriate approach for recognition usually depends on the problem requirements. If a complete pattern description is required for recognition, parsing is necessary. Otherwise, a complete parsing could be avoided by using other simpler approaches to improve the efficiency of the recognition process.

In order to have a grammar describing the structural information about the case class of patterns under study, a grammatical inference machine is required, that can infer a grammar from a given set of training patterns in language-like representation.[1c] The function of this machine is analogous to the *learning* process in a decision-theoretic pattern recognition system. The structural description of the class of patterns under study is learnt from the actual sample patterns from the class. The learnt description, in the form of a grammar, is then used for pattern description and syntax analysis. A more general form of learning might include the capability of learning the best set of primitives and the corresponding structural description for the class of patterns concerned.

4.4 CONCEPTS FROM FORMAL LANGUAGE THEORY[1b,2]

This section contains some essential ideas from formal language theory, as related to problems in syntactic pattern recognition.

An *alphabet* is any finite set of symbols.

A *sentence* over an alphabet is any string of finite length composed of symbols from the alphabet. For example, given the alphabet {∅,1}, the following are valid sentences: $\{\emptyset,1,\emptyset\emptyset,\emptyset 1,1\emptyset,\ldots\}$. The term *string* and *word* are also commonly used to denote a sentence.

The sentence with no symbols is called the *empty sentence*. The empty sentence will be denoted by s_0. For any alphabet V, V* will be used to denote the set of all sentences composed of symbols from V, including the empty sentence. The symbol V^+ will denote the set of sentences V^*-s_0. For example, given the alphabet $V=\{a,b\}$, $V^*=\{s_0,a,b,aa,ab,bb,\ldots\}$ and $V^+=\{a,b,aa,ab,bb,\ldots\}$.

A *language* is any set (not necessarily finite) of sentences over an alphabet.

A *grammar* is defined as a fourtuple:

$$G = (V_N,V_T,P,S) \ . \tag{1}$$

where: V_N is a set of *nonterminals* (variables);

V_T is a set of *terminals* (constants);

P is a set of *productions* or rewriting rules;

S is the *start* or *root* symbol.

It is assumed that S belongs to the set V_N and that V_N and V_T are disjoint sets. The alphabet V is the union of sets V_N and V_T.

The language generated by G, denoted by L(G), is the set of strings which satisfy two conditions:

a) each string is composed only of terminals (i.e. each string is a *terminal sentence*.

b) each string can be derived from S by suitable applications of productions from the set P.

The following notation will be used. Nonterminals will be denoted by capital letters:- S,A,B,C,... . Lower-case letters at the beginning of the alphabet will be used for terminals:- a,b,c,... . Strings of terminals will be denoted by lower-case letters towards the end of the alphabet:-

v,w,x,... . Strings of mixed terminals and nonterminals will be represented by lower-case Greek letters:- $\alpha,\beta,\gamma,\delta,\ldots$.

The set P of productions consists of expressions of the form $\alpha\rightarrow\beta$ where α is a string in V^+ and β is a string in V^*. The symbol $\rightarrow$ indicates replacement of the string α by the string β. The symbol $\underset{G}{\Rightarrow}$ will be used to indicate operations of the form $\gamma\alpha\delta\underset{G}{\Rightarrow}\gamma\beta\delta$ in grammar G, that is, $\underset{G}{\Rightarrow}$ indicates the replacement of α by β by means of the production $\alpha\rightarrow\beta$, γ and δ being left unchanged. It is customary to drop the G and simply use the symbol $\Rightarrow$ when it is clear which grammar is being considered. For example considering the grammar $G=(V_N,V_T,P,S)$, where $V_N=\{S\}$, $V_T=\{a,b\}$, and $P=\{S\rightarrow aSb,\ S\rightarrow ab\}$, if the first production is applied m-1 times.

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow a^3Sb^3 \Rightarrow \ldots\ a^{m-1}Sb^{m-1}$$

is obtained. Applying now the second production results in the string

$$a^{m-1}Sb^{m-1} \Rightarrow a^mb^m$$

the language generated by this grammar consists of an infinite number of strings or sentences and can be expressed as $L(G)=\{a^mb^m|m\geq 1\}$.

4.4.1 <u>Types of Grammars</u>(1b,2)

The grammars considered in this section as specific examples of equation (1) of the previous paragraph. They are all of the general form $G=(V_N,V_T,P,S)$ differing only in the type of productions allowed in each.

An *unrestricted grammar* has productions of the form $\alpha\rightarrow\beta$, where α is a string in V^+ and β is a string in V^*.

A *context-sensitive grammar* has productions of the form $\alpha_1A\alpha_2\rightarrow\alpha_1\beta\alpha_2$, where α_1 and α_2 are in V^*, β is in V^+, and A is in V_N. This grammar allows replacement of the nonterminal A by the string of β only when A appears in the context $\alpha_1A\alpha_2$ of strings α_1 and α_2.

A *context-free grammar* has productions of the form $A \rightarrow \beta$, where A is in V_N and β is in V^+. The name context free arises from the fact that the variable A may be replaced by a string β regardless of the context in which A appears.

A *regular* (or *finite-state*) *grammar* has productions of the form $A \rightarrow aB$ or $A \rightarrow a$, where A and B are variables in V_N and a is a terminal in V_T. Alternative valid productions are $A \rightarrow Ba$ and $A \rightarrow a$. However, once one of the two types has been chosen, the other set must be excluded.

These grammars are sometimes called *type Ø, 1, 2 and 3 grammars*, respectively. They are also often referred to as *phrase structure grammars*.

It is interesting to notice that all regular grammars are context-free all context-free grammars are context-sensitive and all context-sensitive grammars are unrestricted. As expected, unrestricted grammars are considerably more powerful than the other three types. However, their generality presents some serious difficulties in the theoretical and practical applications of the parameters. This is also true of context-sensitive grammars. There is also a major difference between type Ø, 1, 2 and 3 grammars. A type 3 grammar is a finite machine and can be recognised by a *finite automaton*, while the others cannot. They are recognized though by other automata which can also recognize type 3 grammars. This is very important for grammatical inference.

4.5 FORMULATION OF THE SYNTACTIC PATTERN RECOGNITION PROBLEM[2]

Using the concepts of the previous sections, the problem of pattern recognition described in paragraph 3 can be regarded as follows. Suppose that two pattern classes ω_1 and ω_2, are considered. Let the patterns of these classes be composed of features from some finite set. These features will be called *terminals* and denote the set of terminals by V_T. The term *primitives* is also often used in syntactic pattern recognition terminology to denote terminals. Each pattern may be considered as a string or sentence, since it is composed of terminals from the set V_T. Assume that there exists a grammar G with the property that the language it generates consists of sentences (patterns) which belong exclusively to one of the pattern classes, say ω_1. This grammar can clearly be used for pattern classification since a given pattern of unknown origin can be classified as belonging to ω_1 if it is a sentence of L(G). Otherwise the pattern is assigned to ω_2. For example, the context-free grammar $G=(V_N,V_T,P,S)$ with $V_N=\{S\}$, $V_T=\{a,b\}$, and production set $P=\{S\rightarrow aaSb,\ S\rightarrow aab\}$, is capable of generating only sentences which contain twice as many a's and b's. Considering a hypothetical two-class pattern recognition problem in which the patterns of class ω_1 are strings of forms aab,aaaabb, etc., while the patterns of ω_2 contain equal numbers of a's and b's (i.e. ab,aabb, etc.), it is clear that classification of a given pattern string can be generated by the grammar G discussed above. If it can, the pattern belongs to ω_1. If it can not, it is automatically assigned to ω_2. The procedure used to determine whether or not a string represents a sentence which is grammatically correct with respect to a given language is called *parsing*.

The above classification scheme assigns a pattern into class ω_2 strictly by default. However, it is possible that the pattern does not belong to ω_2 either. It may represent a noisy or distorted string which is best rejected. In order to provide a rejection capability it is necessary to determine two grammars, G_1 and G_2 which generate languages $L(G_1)$ and $L(G_2)$. A pattern is assigned to the class over whose language it represents a grammatically correct sentence. If the pattern is found to belong to both classes it may be arbitrarily assigned to either class. If it is not a sentence of either $L(G_1)$ or $L(G_2)$, the pattern is rejected. Thus in the

M-class case M grammars and their associated languages $L(G_i)$, $i=1,2,\ldots,M$. An unknown pattern is classified into class ω_i if and only if it is a sentence of $L(G_i)$. If the pattern belongs to more than one language, or if it does not belong to any of them, it may be arbitrarily assigned to one of the ambiguous classes or rejected, respectively. A block diagram[1d] of the syntactic recognizer is shown in Fig. 4.6.

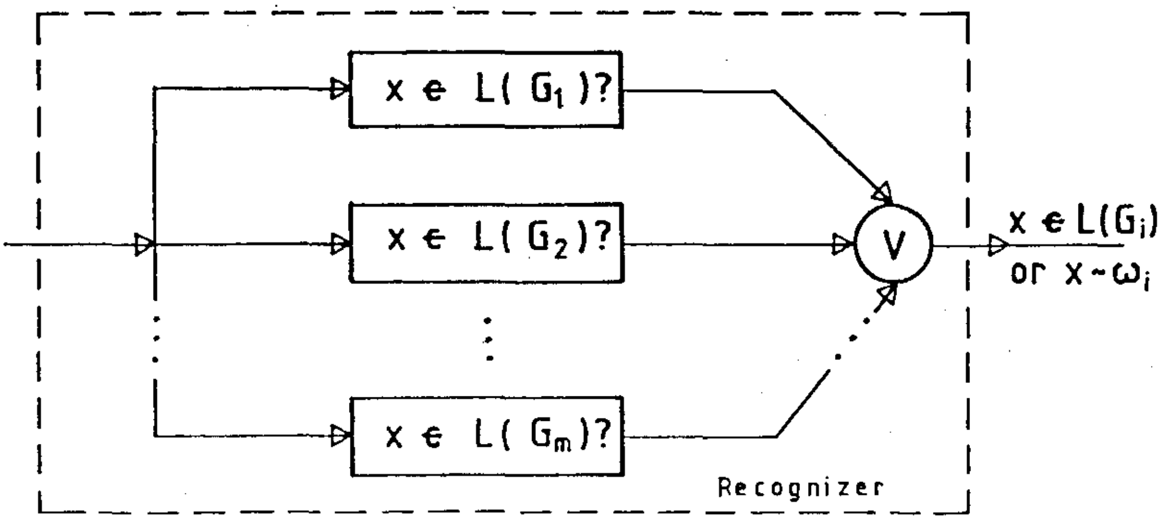


fig. 4.6

4.6 SYNTAX-DIRECTED RECOGNITION[2]

It has been indicated that formal grammars can be used for pattern recognition by determining whether a given pattern represents a terminal sentence which can be generated by any of the grammars under consideration for a specific problem. The procedure that determines whether or not a given pattern represents a valid sentence, in formal language theory, is known as parsing. Basically, two main types of parsing techniques will be considered: *top-down* and *bottom-up*. By referring to a tree structure the following analogies can be drawn. The top or *root* of the (inverted) tree is the start symbol S. The terminal sentences (patterns) represent the bottom or leaves of the tree. The top-down[4] technique starts with the root symbol S and, through repeated applications of the productions of the grammar, attempts to arrive at the given terminal sentence. The bottom-up approach, on the other hand, starts with the given sentence and attempts to arrive at the symbol S by applying the productions in reverse. In either case, if the parse fails, the given pattern represents an incorrect sentence and is therefore rejected.

It is evident that the parsing schemes described above are inherently inefficient since they involve essentially an exhaustive search in the applications of the productions of the grammar. However, it is seldom necessary to carry a sentence of productions all the way through, since partial results can be checked against the desired goal in order to determine whether a given sequence of productions has the potential to produce a successful parse.

The parsing process can be further improved by employing the rules of *syntax* of a grammar. Syntax is defined as the concatenation of objects. A *rule of syntax* states some permissible (or prohibited) relations between objects. For example the concatenation www never occurs in the English language. In this terminology, a grammar is nothing more than a set of rules of syntax which define the permissible or desired relations between objects. A *syntax-directed* parser, therefore, employs the syntax of the grammar in the parsing process.

4.6.1 Recognition of Graph-Like Patterns[2]

The problem with the previous approach is that, scanning for the primitives or substructures of interest in a two-dimensional situation could be a formidable task for a machine. Today, the most successful attempts in this area have involved patterns which can be reduced to graph-like structures.

An interesting application of linguisitic concepts to pattern recognition is the Picture Description Language (PDL).[5,6] A primitive in PDL is any n-dimensional structure with two distinguished points, a *tail* and a *head*, as shown in Fig. 4.7a for two-dimensional structures. It is worth noticing that a fairly general structure can be abstracted as a directed line segment since there are only two points of definition.

A primitive can be linked to other primitives *only* at its tail and/or head. On the basis of this permissible form of concatenation, the structures of PDL are directed graphs, and can be handled by string grammars. The principal rules for the concatenation of abstracted primitives are shown in Fig. 4.7b. It is important to point out that *blank* primitives may be used to generate seemingly *disjoint* structures while preserving the rules of connectivity. Also, it is often useful to consider a *null point* primitive having identical head and tail.

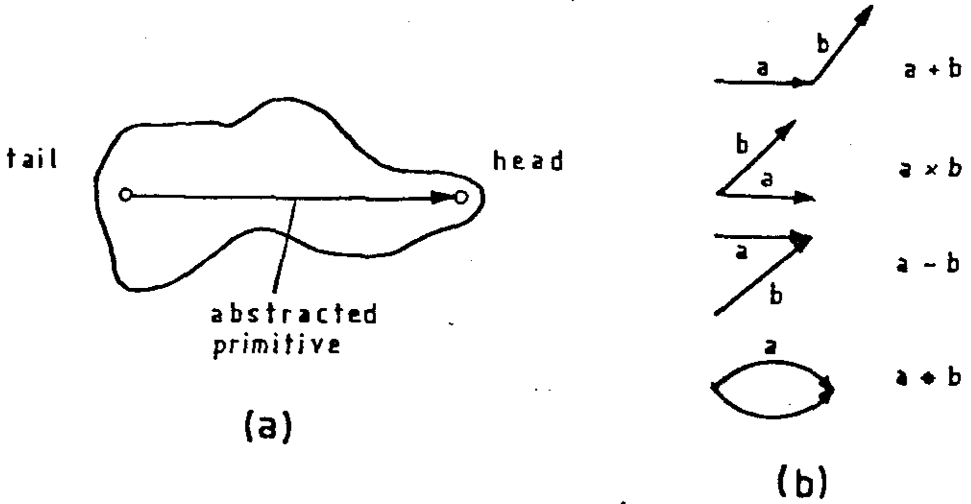


fig. 4.7

When n-ary relations are involved, a graph-representable description can be obtained by transforming all relations into binary ones. A unary relation $r(x)$ can be changed to a binary one $r'(X_1\lambda)$ where λ denotes the 'null' primitive. The relation $r(X_1,\ldots,X_n)$ $(n>2)$ can be transformed into a composition of binary relations, such as:

$$r_1(X_1,r_2(X_2,\ldots,r_{n-1}(X_{n-1},X_n))$$

or into a conjunction of binary relations

$$r_1(X_{11},X_{12})\wedge r_2(X_{21},X_{22})\wedge\ldots\wedge r_k(X_{k1},X_{k2})$$

or into a combination of these. For example, the ternary relation TRIANGLE (a,b,c) would be transformed into either one of the following equivalent binary relations:-

$$CAT(a,b)\wedge CAT(b,c)\wedge CAT(c,a) \quad \text{or} \quad \Delta(b,CAT(a,c))$$

where $CAT(x,y)$ means that head(X) is concatenated to tail(Y), that is, $CAT(X,Y)=X+Y$ and $\Delta(X,Y)$ means that the line X is connected to form a triangle with the object Y consisting of two concatenated objects.

Another grammar, which generates languages with terminals having an arbitrary number of attaching points for connecting to other primitives or sub-patterns is the *PLEX* grammar.[8] The primitives of the plex grammar are called *N-attaching point entities* (NAPEs). Each production of the plex grammar is in a context-free form in which connectivity of primitives or subpatterns is described by using explicit lists of labelled concatenation points (called joint lists). Sentences generated by a plex grammar can be transformed to directed graphs, by assigning labelled nodes to both primitives and concatenation points or by transforming primitives to nodes and concatenations to labelled branches.

An extension of the concept of string grammars to grammars for labelled graphs are the *WEB* grammars.[9] Labelled node-oriented graphs are explicitly used in the productions. Each production describes the rewriting of graph

α into another graph β and also contains an *embedding* rule E which specifies the connection of α to its surrounding graph (host web) when α is rewritten.

A further generalization of string grammars to graph grammars can be done by including non-terminal symbols which are not simple branches or nodes.[10] An mth-order non-terminal structure is defined as an entity that is connected to the rest of the graph by m nodes. In particular, a second-order structure is called a branch structure and a first-order structure a node structure. Then an mth-order context-free grammar G_g is a quadruple $G_g=(V_N,V_T,P,S)$ where V_N is a set of mth-order non-terminal structures:- nodes, branches, triangles,..., polygons with m vertices; V_T is a set of terminals:- nodes and branches; P is a finite set of productions of the form $A\to\alpha$, where A is a non-terminal structure and α a graph containing possibly both terminals and non-terminals (α is connected to the rest of the graph through exactly the same nodes as A); S is a set of initial graphs. The expression A*B denotes that the two graphs A and B are connected by a pair of nodes, and N(A+B+C) denotes that the graphs A,B and C are connected through a common node N. Finally ANB denotes a non-terminal subgraph consisting of a branch structure A with nodes X and Y connected to the node structure N through Y and a branch structure B with nodes Y and Z connected to N through Y. The subgraph is connected to the rest of the graph through the nodes X and Z. Fig. 4.8a,b and c illustrate the above.

Another interesting application of syntactic pattern recognition deals with automatic classification of chromosomes.[7] A context-free grammar classifies a chromosome as being either *submedian* or *telocentric*. The primitives used in this application are shown in Fig. 4.9a; typical submedian and telocentric chromosomes are shown in Fig. 4.9b.

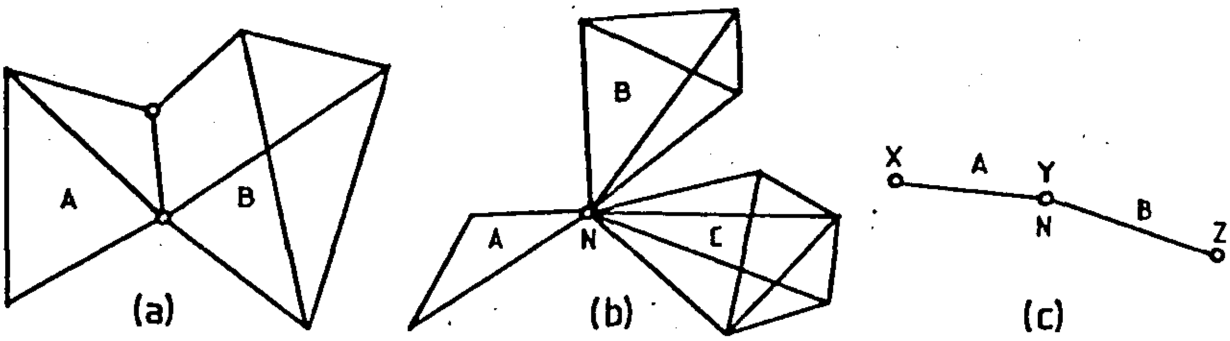


fig. 4.8

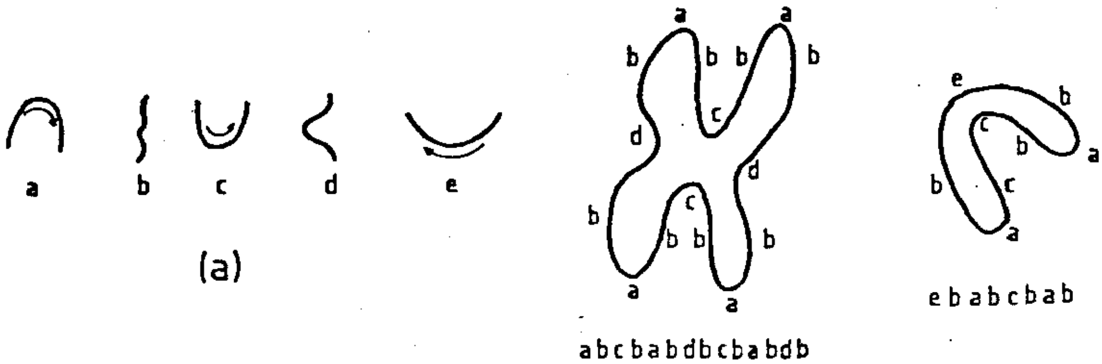


fig. 4.9 (b)

In terms of these figures, operator '.' is interpreted as describing simple connectivity of parts as a chromosome boundary is tracked in the clockwise direction.

4.6.2 Recognition of Tree Structures[2]

In order to handle tree structures it is necessary to modify slightly the concept of a grammar. A *tree grammar* is defined as a quintuple

$$G=(V_N,V_T,P,R,S) \qquad (2)$$

where V_N and V_T are, as before, sets of non-terminals and terminals, respectively; S is the start symbol which can, in general, be a tree; P is a set of productions of the form $\Omega\to\psi$, where Ω and ψ are trees; and

R is a *ranking function* which denotes the number of direct descendants of a node whose label is a terminal in the grammar. An example of a tree grammar is illustrated in Fig. 4.10.

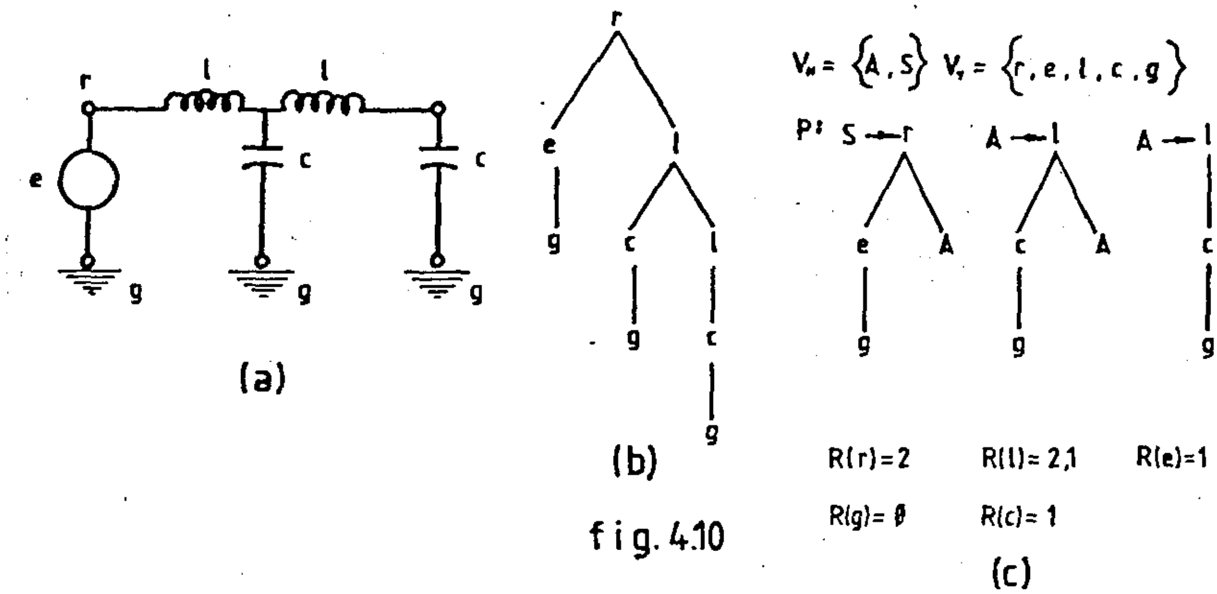


fig. 4.10

4.7 LEARNING AND GRAMMATICAL INFERENCE[1e]

The use of formal linguistics in modelling natural and programming languages and in describing physical patterns and data structures has recently received increasing attention. Grammars or syntax rules are employed to describe the syntax of languages or the structural relations of patterns. In addition to the structural description, a grammar can also be used to characterize a syntactic source which generates all the sentences (finite or infinite) in a language, or the patterns belonging to a particular class. In order to model a language or to describe a class of patterns or data structures under study more realistically, it is hoped that the grammar used can be directly inferred from a set of sample sentences or a set of sample patterns. This problem of learning a grammar based on a set of sample sentences is called *grammatical inference*.

The problem of grammatical inference is concerned mainly with the procedures that can be used to infer the syntactic rules of an unknown grammar G, based on a finite set of sentences or strings, S_t from L(G), the language generated by G, and possibly also on a finite set of strings from the complement of L(G). The inferred grammar is a set of rules for describing the given finite set of strings from L(G) and predicting other strings which in some sense are of the same nature as the given set. A basic block diagram of a grammatical inference machine is shown in Fig.4.11.

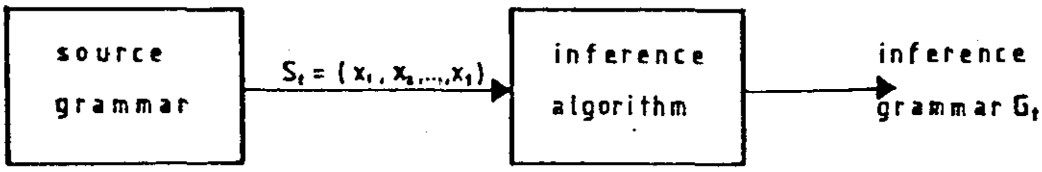


fig.4.11

The inferred grammar for S_t is considered to be *good* if it yields a satisfactory result. In recent years some measures of *goodness* have been defined in terms of complexity of the inferred grammar, and they have been applied to grammatical inference problems.

SUMMARY - CONCLUSIONS

- The syntactic approach to pattern recognition provides a capability for describing a large set of complex problems, by using small sets of simple pattern primitives and grammatical rules.
- A syntactic pattern recognition system consists of three major parts: preprocessing, pattern description or representation and syntax analysis.

- Preprocessing functions include pattern encoding, approximation filtering, restoration and enhancement. Pattern representation consists of pattern segmentation and feature extraction. Syntax analysis or parsing makes the decision whether or not the pattern belongs to the class described by the given syntax and grammar.
- A grammar is defined as:- $G=(V_N,V_T,P,S)$ (V_N=nonterminals, V_T=terminals, P=productions, S=start symbol). The commonest grammars are the:- unrestricted, contex-sensitive, contex-free and regular.
- An unknown pattern is classified into class ω_i if and only if it is a sentence of language $L(G_i)$, $i=1,2,3,...,M$. Otherwise it is rejected.
- The two main parsing techniques are top-down and bottom-up. Top-down starts with the start symbol S and attempts to reach the terminals with repeated applications of the productions P. Bottom-up is the reverse procedure.
- A syntax directed parser employs the syntax of the grammar in the parsing process.
- Picture Description Language is implemented as an application of graph-like pattern recognition.
- In order to modify tree structures the *ranking function*, R is introduced to the fourtuple of grammar.
- Grammatical inference deals with the problem of learning a grammar, based on a set of sample sentences.

REFERENCES

1. K.S. Fu, *"Syntactic Methods in Pattern Recognition"*, Academic Press, 1974

    a. Chapter 1

    b. Chapter 2

c. Chapter 3

d. Chapter 4

e. Chapter 7

2. J.T. Tou, R.C. Gonzalez, *"Pattern Recognition Principles"*, Addison-Wesley, Publ.Comp. Inc., Reading, Massachusetts, 1974, Chapter 8.
3. A. Bundy, *"A.I. An Introductory Course"*, Edinburgh University Press, 1978, Chapter 4.
4. A.V. Aho, J.D. Ullman, *"Principles of Compiler Design"*, Addison-Wesley Publ. Comp. 1978, Chapter 5.
5. A.C. Shaw, *"The Formal Description and Parsing of Pictures"*, Rep. SLAC-84, Stanford Linear Accelerator Centre, Stanford Univ., Stanford, California, 1968.
6. A.C. Shaw, *"A Formal Picture Description Scheme as a Basis for Picture Processing Systems"*, Information and Control 14, 9-52 (1969).
7. R.S. Ledley, L.S. Rotolo, T.J. Golab, J.D. Jacobsen, M.D. Ginsburg and J.B. Wilson, FIDAC: *"Film Input to Digital Automatic Computer and Associated Syntax Directed Pattern Recognition Programming System"*, in Optical and Electro-Optical Information Processing (J.T. Tippett et al, eds.), Chapter 33, pp.591-614, M.I.T. Press, Cambridge, Massachusettes, 1965.
8. J. Feder, *"Plex Languages"*, Information Sci. 3, 225-241 (1971).
9. J.L. Pfaltz and A. Rosenfeld, *"WEB grammars"*, Proc.Int.Joint Conf.A.I. 1st, Washington, D.C., May 1969, pp.609-619.
10. T. Pavlidis, *"Graph Theoretic Analysis of Pictures"*, in 'Graphic Languages (F. Nake and A. Rosenfeld, eds.), North-Holland Publ. Amsterdam, 1972.

Chapter 5

THE RECOGNIZER

5.1 INTRODUCTION

After the pattern has been preprocessed and its main features have been extracted, it is segmented into subpatterns called *pattern primitives*, which are easier to recognize than the pattern itself. Thus the structure of the pattern is described by a *pattern recognition language* that is based on the set of primitives and their composition operations. The rules governing the composition of the primitives into patterns is usually specified by the *grammar* of the pattern description language. After each primitive within the pattern is identified, the pattern is ready to be recognized. The recognition process is accomplished by performing a *syntax analysis* or *parsing* of the *sentence* describing the given pattern to determine whether or not it is syntactically (or grammatically) correct with respect to the specified grammar. Finally the pattern is classified, i.e. it is assigned to a particular class determined by the above mentioned grammar.

The recognizer discussed in this chapter is capable of recognizing three-dimensional objects that consist of simple straight-line two-dimensional shapes. It functions in two stages. In the first stage all the 2-D shapes, which are eventually the visible sides of the 3-D object to be recognized, are classified by a 2-D recognizer. Then, in the second stage, a 3-D recognizer is called to classify the 3-D object composed by the 2-D sides previously recognized. Both recognizers belong to the top-down type, i.e. they start with the start symbol (root of the tree) and attempt to arrive at the given terminal sentence, by repeatedly applying the productions of a certain grammar. The grammar used is a *string grammar* which means that the primitives or terminals are strings (of some special form because of the use of PROLOG), with concatenation as the main relation between them. Finally both recognizers are written in PROLOG.

The following sections attempt to analyse the function of the recognizer. First an introduction to PROLOG is given for better understanding of the special primitives used. Then the recognizer is examined in two parts, the 2-D recognizer and the 3-D recognizer.

5.2 AN INTRODUCTION TO PROLOG[1]

PROLOG is a simple and powerful programming language for non-numeric applications. It was originally devised around 1972 for the purpose of implementing a natural language question-answering system at the University of Marseille. A PROLOG compiler/interpreter for the DEC-system-10 has been produced at the University of Edinburgh.

The basic idea of PROLOG is that a collection of logic statements of a restricted form (clauses) can be regarded simply as a program, and that the execution of such a program is nothing other than a suitably controlled logical deduction from the clauses forming the program. A PROLOG program can be regarded as a collection of statements of fact - the *declarative view*. The program can also be understood as a number of procedure definitions - the *procedural view*. The different clauses in a procedure represent alternative *cases* of the procedure. The appropriate clause (or clauses) is selected by a *pattern matching* operation *(unification,* explained later on) according to the form of the procedure call. Pattern matching is the *sole* data manipulation operation. Data items in PROLOG are called *terms* and may be thought of as complex record structures written in a textual, machine independent form, not involving the notion of reference or pointer.

Procedures:- A PROLOG program consists of a sequence of statements called *clauses*. Here is a simple example, consisting of six clauses:

```
descendant(X,Y):- offspring(X,Y).
descendant(X,Y):- offspring(X,Y) , descendant(Y,Z).
offspring(abraham,ishmael). offspring(abraham,isaac).
offspring(isaac,esau).      offspring(isaac,jacob).
```

Clauses can be understood in two ways. Firstly, they can be interpreted as statements of fact. For instance the first clause says that, whatever may be the values of the *variables* X and Y, 'Y is a descendant of X if Y is one of the offspring of X'. And the last clause says that 'jacob is one of the offspring of isaac'. Note that the variables in different clauses are considered distinct, even if they have the same name. The second way to understand clauses is as pieces of program. Each clause corresponds to a *case* of a *procedure*. Looked at in this way, the first clause can be read as 'To find a Y that is a descendant of X, find a Y that is one of the offspring of X', and the last clause as 'When seeking an offspring of isaac, return the solution jacob'.

The six clauses of the example serve to define two procedures, named 'descendant' and 'offspring'. Each clause consists of a *head* or *procedure entry point*, followed by a (possibly empty) *body*. A clause with an empty body is called a unit clause. A PROLOG program works as follows:-

To run the program, one provides an initial goal such as:-

descendant(abraham,X).

The result of executing this goal will be to enumerate descendants of abraham and return them, one by one, as values of the variable X. In order to execute such a goal, the PROLOG system *matches* it against the head of some clause and then executes the goals (if any) in the body of that clause, in left-to-right order. In seeking a match, PROLOG tries the clauses of procedure concerned, in the order they appear in the program text. The matching process, known technically as *unification*, succeeds if the goal and the clause head can be made identical by *filling in* suitable values for the variables. For example the goal 'offspring(X,ishmael)' matches the first clause for 'offspring' if X is given the value 'abraham'. The variable X is then said to be *instantiated* to *abraham*. When one solution to a goal has been finished with, or when no match can be found for a goal, the PROLOG system *backtracks*. That is, it goes back to the most recently executed goals, and looks for an alternative match. If backtracking generates more than one solution to a goal, the corresponding procedure is said to be *non-determinate*.

In the example above suppose that the initial goal 'descendant(abraham,X)' is executed. Through matching the goal against the first clause for 'descendant', PROLOG starts off by looking for the immediate offspring of abraham, and returns successively X='ishmael' and X='isaac'. Then back-tracking causes the second clause for 'descendant' to be used. This results in the 'descendant' procedure being called recursively for each of the abraham's offspring, giving further descendants, esau and jacob.

Structures:- PROLOG data objects are called *terms*. Variables and unstructured constants are terms called *atoms*. PROLOG also provides for structured data objects called *complex terms*. An example is the binary tree data type. The following procedure checks whether a particular item is present in an ordinary binary tree

in(X,tree(T1,X,T2)).
in(X,tree(T1,Y,T2)):- before(X,Y),in(X,T1).
in(X,tree(T1,Y,T2)):- before(Y,X),in(X,T2).

Here 'tree' is a *function* of 3 *arguments*. It can be thought of as a record type with 3 fields. The arguments stand for the left subtree, the item at the root node, and the right subtree. The first clause says that X is present in the ordered binary tree <T1,X,T2>, for any values of X,T1 and T2. The last clause says that X is present in the ordered binary tree <T1,Y,T2> if Y is before X and X is present in T2, for any values of X,Y,T1 and T2.

Another, very commonly used data type is the *list*. For example, the PROLOG procedure for concatenating lists is:-

concatenate([],L,L).
concatenate([X|L1],L2,[X|L3]):- concatenate(L1,L2,L3).

From a practical point of view, PROLOG enables the programmers to write clearer, more concise programs, with less effort, and with less likelihood of error. The language could perhaps be summed up as *pointer manipulation made easy.*

PROLOG has been put to practical use in a number of areas outside pure research. Examples include a package for doing algebraic *symbol crunching,* an architectural design aid to assist in planning the layout of a building, a system to help predict the properties of organic compounds, and the implementation of a compiler (DEC-1Ø PROLOG). Applications within

Artificial Intelligence research include programs for plan generation, equation solving, natural language analysis, and solving mechanics problems. All the above are large and complex programs which would probably never have got written at all with the available manpower, were it not for the relative ease of writing them in PROLOG.

5.3 THE 2-D RECOGNIZER

Although this is basically part of the recognizer of 3-D objects, it could also be viewed as a separate program capable of recognizing 2-D shapes. The 2-D recognizer has as its input sets of primitives representing 2-D shapes and classifies them into one of the following three classes:-

a) *triangle*

b) *quadrilateral*

c) *other*

Once the shape has been assigned to one of the first two classes a further classification is obtained by using relations between the features of the studied shape. For example a triangle with two equal sides is an *isosceles-triangle* or a quadrilateral with two pairs of parallel sides is a *parallelogram* etc. Each of these *primary* or *secondary* classes is considered as a separate goal and is subject to a different grammar. In Fig. 5.1 the function of the 2-D recognizer is shown, in the form of a tree structure.

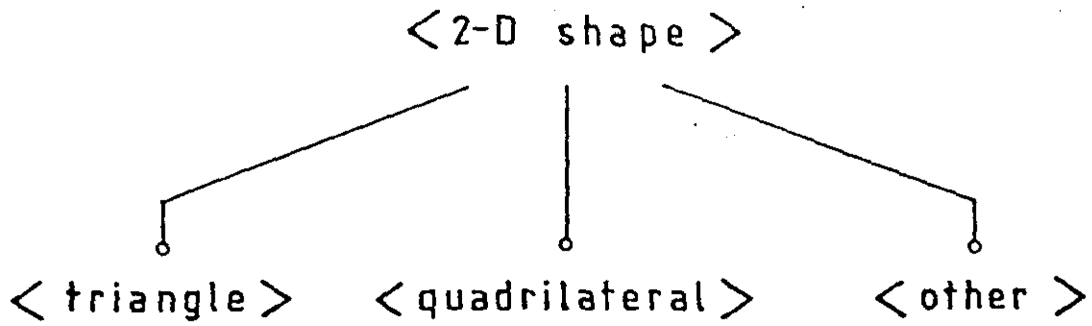


fig. 5.1

5.3.1 Triangle

A triangle is defined by three straight line segments connecting together three points which are not in the same straight line. The three straight line segments are called the *sides* of the triangle and the points are called the *vertices* of the triangle. From the definition of the triangle it is obvious that sides are its main components. These can be represented by predicates of the form:- *conn(A,AB,B)*. Where A and B are the two vertices connected by the side AB and *conn* stands for *connect*, which is the relation between A and B. Thus a triangle is considered as the structure that consists of the conjunction of three *conn* predicates, expressing the relation between its three vertices and three sides. For example triangle (a,b,c) is given by the following clause:-

triangle(A,AB,B,BC,C,CA):-conn(A,AB,B),conn(B,BC,C),conn(C,CA,A). *

by substituting variables A,B,C with a,b, and c respectively.

Predicate *conn* is considered as directed and thus *conn(a,ab,b)*≠ *conn(b,ba,a)*. By convention the clockwise rotation order is used, because that is the direction in which the boundary follower works. The purpose of keeping a certain direction is to cope with ambiguous situations where two 2-D shapes have common sides (more in the 3-D recognizer).

Every triangle is considered with respect to one of its vertices as a point of reference. Thus if for example the data is:- *conn(a,ab,b).*, *conn(b,bc,c).*, *conn(c,ca,a).*, the question *'shape(b,X)'*. will be answered by *'shape(b,triangle)'.*, which means that 'there is only one shape connected to vertex b, and this is a triangle'. If other alternatives are sought, the answer will be that 'there are not any'. However, if the question is

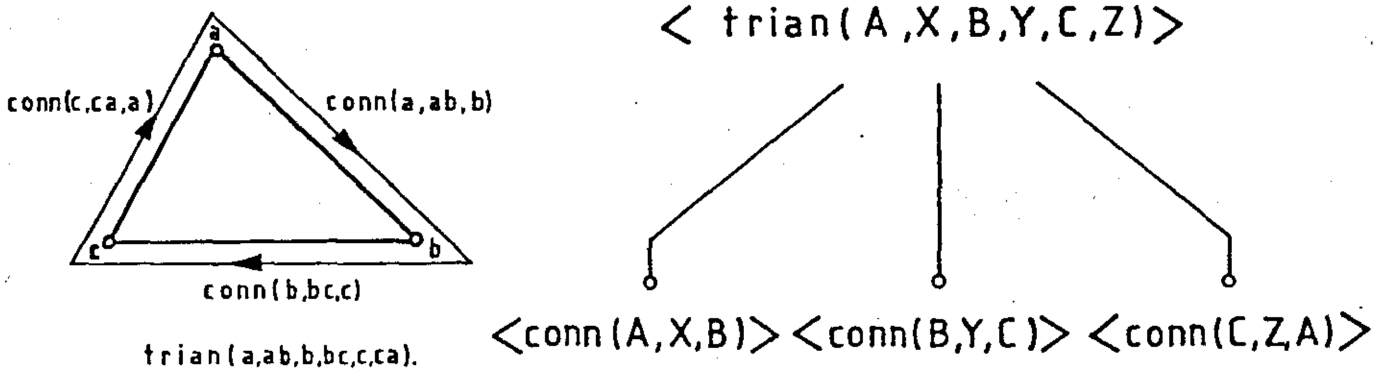
---

* ',' is the symbol for logic AND and ';' the symbol for logic OR in PROLOG(2)

put more generally *'shape(Y,X)'*, that is 'name all the triangles connected to any of the vertices' the answer will be 'shape(a,triangle)', with 'shape(b,triangle)' and 'shape(c,triangle)' as the alternatives of the first answer. The actual triangle is one and the two alternatives represent the same triangle with respect to a different vertex each time. This of course may prove confusing when the data represent more than one triangle (or shape in general) connected with common vertices or even sides (3-D case). In the last case it is undesirable that a triangle once identified as such, is taken under consideration again within the same frame. This is achieved by using the predicate *assert(a triangle(A,X,B,Y,C,Z))*,(2) which adds the unit clause *a triangle(A,X,B,Y,C,Z)* at the end of the database. By modifying the definition of the triangle as follows the two previously mentioned problems are solved.

```
trian(A,X,B,Y,C,Z)):- conn(A,X,B),conn(B,Y,C),conn(C,Z,A),
                      not(atrian(A,X,B,Y,C,Z)),
                      not(atrian(B,Y,C,Z,A,X)),
                      not(atrian(C,Z,A,X,B,Y)),
                      assert(atrian(A,X,B,Y,C,Z)).
```

Referring to the same example mentioned above, the *assert* predicate will add to the data the unit clause *atrian(a,ab,b,bc,c,ca)*, which will effectively *remove trian(a,ab,b,bc,c,ca)* from the data (since *not(atrian(a,ab,b,bc,c,ca))* will become false) and hence the same triangle will not be taken into account when alternatives are asked for. The extra two *not(atrian(...))* predicates will prevent any recycling, and so to the original question *'shape(Y,X)'* will be given only one answer, *'shape(a,triangle)'*. (it is '*a*' because *conn(a,ab,b)* happens to be the first unit clause in the data). After the recognition of the same frame has been completed a special clause called *init 2D* (see Section 5.4) will retract all *atrian's* from the data to allow the next frame to be recognized correctly. Fig. 5.2b shows the tree-like structure of the definition for the triangle, and a illustrates the example of 5.3.1.



(a) fig. 5.2 (b)

5.3.2 Quadrilateral

A quadrilateral is defined by four co-planar points every three of which are not in the same line, connected together by four straight line segments. In other words a quadrilateral consists of four vertices and four sides. By using arguments similar to the ones in the case of the triangle the definition of the quadrilateral could well be, a sequence of four consecutive *conn's* with the appropriate *not(aquadril(...))'s* and an *assert(aquadril(...))* at the end. This is correct, apart from the fact that, since the 2-D recognizer is deemed as a part of the 3-D one, two more things must be taken into account. Fig. 5.3 explains these two cases.

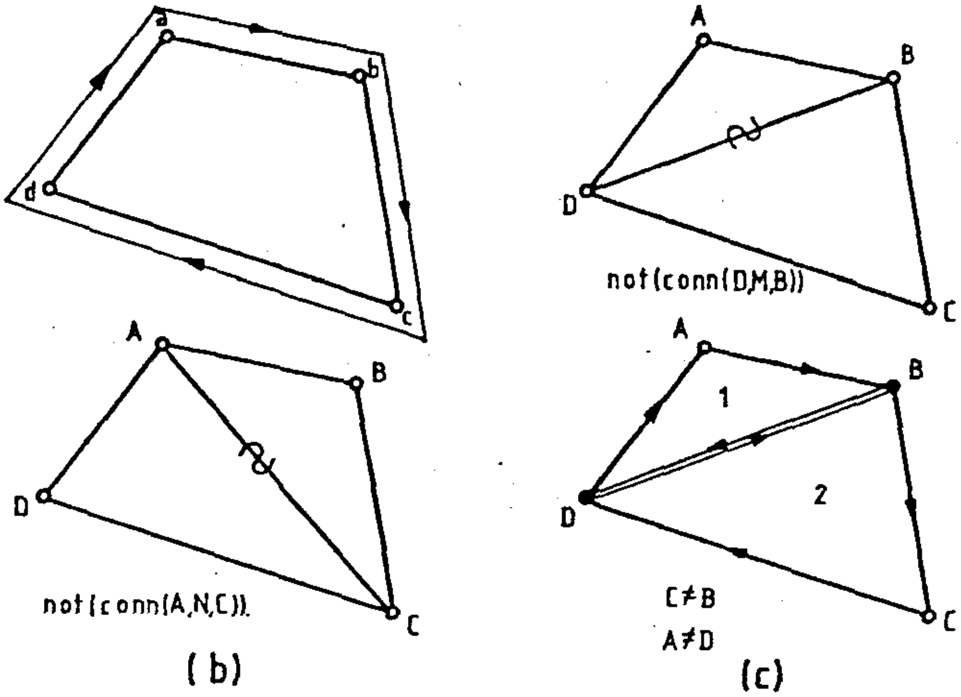


fig. 5.3

Since a quadrilateral is considered as a 2-D shape, there should not be any straight line segment representing either of its diagonals. This implies that such clauses as *conn(A,N,C)* or *conn(D,M,B)* or the symmetric ones *conn(C,L,A)* or *conn(B,K,C)*, should not exist in the database. Thus if any of the above cases exists then the shape will be interpreted as two triangles connected by a common side rather than a quadrilateral. Supposing now that the following clauses exist in the database:-

1) conn(a,ab,b). *2) conn(b,bc,c).* *3) conn(c,ca,a).*
4) conn(b,bd,d). *5) conn(d,dc,c).* *6) conn(c,cb,b).*

Considering the suggested definition of the four *conn's*, there should not be any quadrilateral because of the *conn's* no.2 and 6. But the following combination may give rise to a *false* quadrilateral:-

(1) ... conn(A,W,B),conn(B,X,C),conn(C,Y,D),conn(D,Z,A) ...
conn(a,ab,b).,conn(b,ba,a)., conn(a,ab,b).,conn(b,ba,a).

which is a quadrilateral according to the definition (1), since there is neither *conn(a,aa,a)* nor *conn(b,bb,b)*. Thus care must be taken that A≠C and B≠D.*

After this discussion the definition of the quadrilateral becomes:-

```
quadril(A,W,B,X,C,Y,D,Z):- conn(A,W,B),conn(B,X,C),conn(C,Y,D),
                                                    conn(D,Z,A),
                           not(conn(A,K,C)),not(conn(C,L,A)),(A\==C),
                           not(conn(B,M,D)),not(conn(D,N,B)),(B\==D),
                           not(aquadril(A,W,B,X,C,Y,D,Z)),
                           not(aquadril(B,X,C,Y,D,Z,A,W)),
                           not(aquadril(C,Y,D,Z,A,W,B,X)),
                           not(aquadril(D,Z,A,W,B,X,C,Y)),
                           assert(aquadril(A,W,B,X,C,Y,D,Z)).
```

The tree-like structure of the definition for the quadrilateral is illustrated in Fig. 5.4.

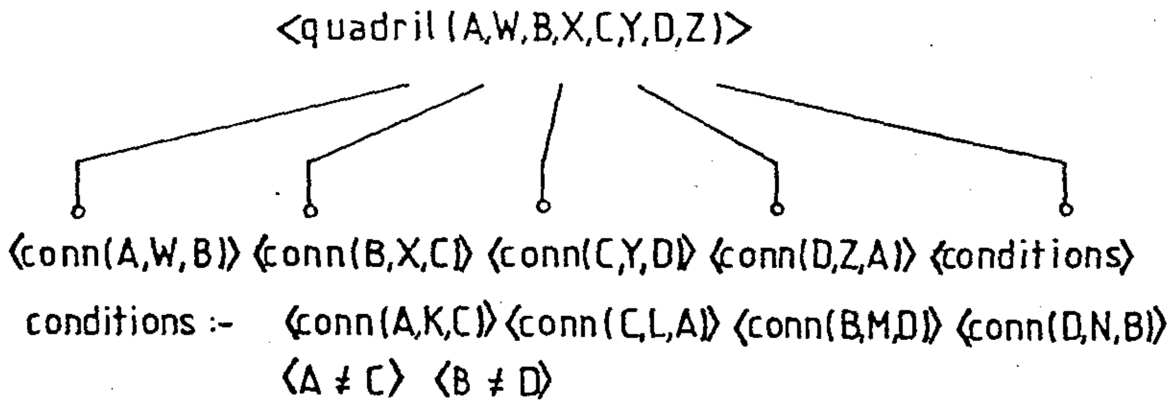


fig. 5.4

As in the case of triangle, the operation *init 2D* will clear all the *aquadril*

---

*A≠C can be expressed in PROLOG by 'A=\=C' which means that integer expressions A and C are not equal, and 'A\==C' which means that A and C are not identical. Here the second expression is used to denote that vertex A and vertex C are not identical or in other words they do not coincide. Obviously the second expression is stronger than the first one.

clauses, preparing the 2-D recognizer for the next application.

5.3.3 Secondary Classes for the Quadrilateral

So far the only unit clauses used were the *conn's*, denoting the basic structure of the main figure. These are enough to make a *first* classification of the 2-D shape to one of the *primary* classes, that is triangle or quadrilateral. If a further classification is required, then the other unit clauses related to the 2-D shape supplied by one of the previous procedures (Section 3.4.3), must be used. These unit clauses are:- *line(X,K)*, *sqrline(X,L)*, *slope(X,M)* and *angle(X,N)*. The comparison of unit clauses of the same type for a quadrilateral can reveal equal sides and angles, parallel sides, and right angles. Before examining each secondary class in detail the definition of some of the operations between the unit clause is given.

equalline(X,Y):- line(X,M),line(Y,N),equal1(M,N).

This says that two lines are equal if the integer numbers M and N that represent their length satisfy the clause *equal1(M,N)*. The latter is true if the absolute difference of the two numbers is smaller than two. Similarly *equal2(M1,N1)* is true if the absolute difference between the two numbers (which in this case are the squares of the previous ones M and N) is smaller than four.

paral(X,Y):- slope(X,M),slope(Y,N),equalslope(M,N).

Two sides are parallel if their slopes (defined by 3.4.3c), satisfy the clause, *equalslope(M,N)*. This is true if the absolute difference between M and N (in degrees) is less than three degrees.

rect(X,Y):- slope(X,M),slope(Y,N),right(M,N).

Finally two lines are perpendicular if *right(M,N)* is true, i.e. if the absolute difference of the two slopes differs in absolute value from ninety degrees less than three degrees.

The secondary classes have in common the fact that they are all quadrilaterals in the first place and then, depending on the relations between their sides and angles are one of: a) *parallelogram*, b) *rectangle*, c) *square*, and d) *rhombus*.

a) parallelogram:- A parallelogram is a quadrilateral with its opposite sides parallel to each other in pairs. Thus the definition of a parallelogram is:-

paralgrm(A,W,B,X,C,Y,D,Z):- quadril(A,W,B,X,C,Y,D,Z),paral(W,Y),paral(X,Z).

A parallelogram and its tree-like structure is given in Fig. 5.5.

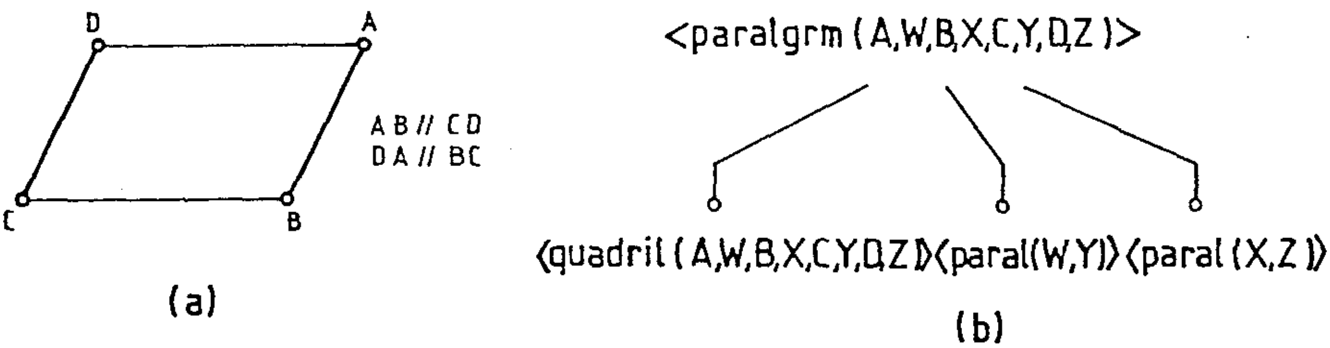


fig. 5.5

b) rectangle:- A rectangle is a parallelogram with two of its sides perpendicular, i.e. forming a right angle. The definition of a rectangle is given by:-

rectan(A,W,B,X,C,Y,D,Z):- paralgrm(A,W,B,X,C,Y,D,Z),rect(W,X).

Fig. 5.6 illustrates a rectangle and its tree-like structure.

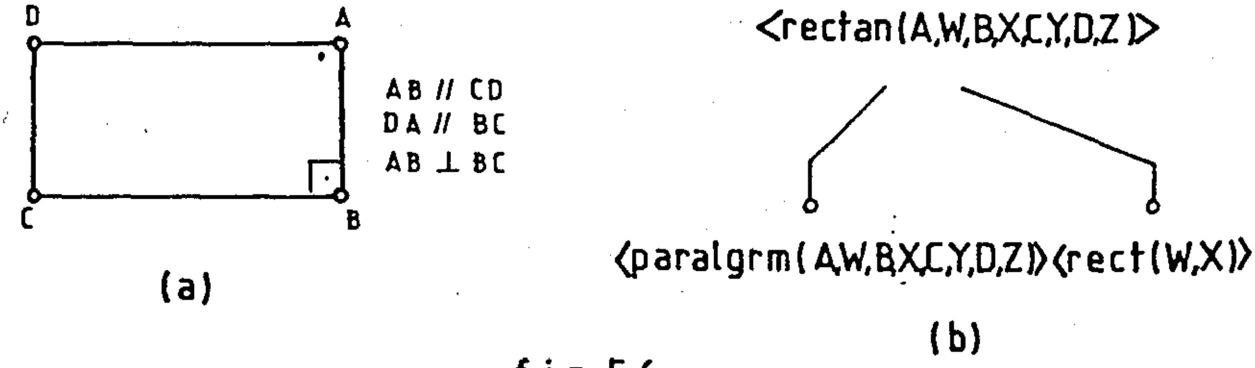


fig. 5.6

c) square:- A square is a rectangle with two adjacent sides equal. Its definition is:-

square(A,W,B,X,C,Y,D,Z):- rectan(A,W,B,X,C,Y,D,Z),equalline(W,X).

A square and its tree-like structure is shown in Fig. 5.7.

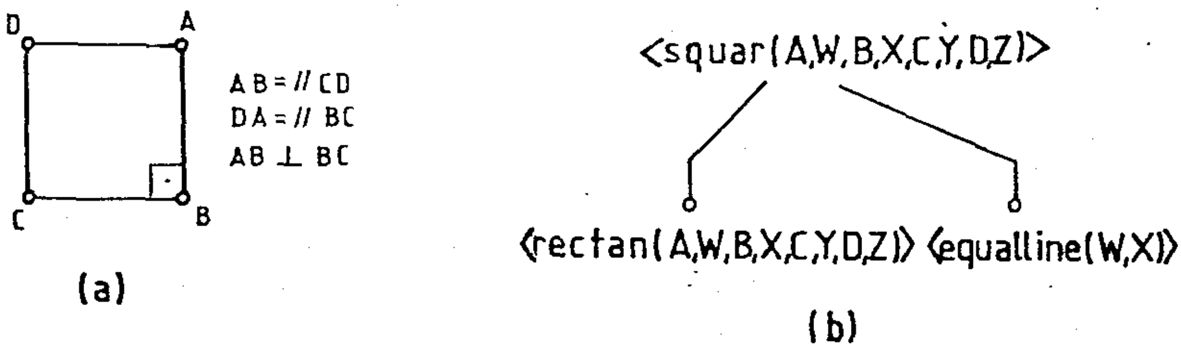
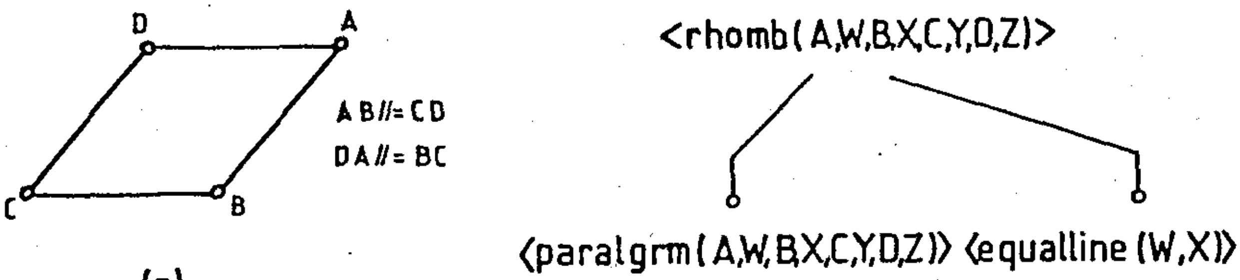


fig. 5.7

d) rhombus:- A rhombus is a parallelogram with two adjacent sides equal. The definition of the rhombus is:-

rhomb(A,W,B,X,C,Y,D,Z):- paralgrm(A,W,B,X,C,Y,D,Z),equalline(W,X).

Fig. 5.8 illustrates the rhombus and its tree-like structure.



(a) (b)

fig.5.8

This was the 2-D recognizer examined as part of the 3-D recognizer. At the end of this chapter there will be a section examining the 2-D recognizer as an autonomous system, capable of recognizing single (unconnected) 2-D shapes.

5.4 THE 3-D RECOGNIZER

This is the second part of the recognizer, and its task is to recognize simple objects composed of sides which the 2-D recognizer can identify. Before attempting to describe how the 3-D recognizer works, it would be helpful if the representation of 3-D objects on a 2-D picture was examined first.

It was said in the previous sections that a 2-D shape is a combination of a number of sides and vertices (three and four respectively) connected together in a particular way. Since the shapes are considered to be planar, a 2-D representation of them would be sufficient to describe exactly their structure. In other words a 2-D shape is by definition whatever its structure tells that it is and nothing else.

On the other hand a 3-D shape is a composition of 2-D shapes (in this case triangles and quadrilaterals) joined together by common sides - edges -.

Thus if a one-to-one correspondence was tried between a 2-D and a 3-D shape this would be:

$$\begin{array}{ccccc} \textit{points(vertices)} & + & \textit{straight line segments(sides)} & \rightarrow & \textit{2-D shape} \\ \updownarrow & & \updownarrow & & \updownarrow \\ \textit{straight line segments(edges)} & + & \textit{2-D straight line shapes(sides)} & \rightarrow & \textit{3-D shape} \end{array}$$

It is obvious that the best space for a perfect representation of 3-D shapes would be the three dimensional space. But even in this case the fact that some 2-D sides prevent the observer from having a direct view of all the other sides, makes it difficult for him to categorically identify the object. Actually he may need to either move or move the object itself in order to give a definite answer. The difficulties increase when an answer is attempted from a 2-D representation of a 3-D shape. Actually the 2-D representation of a 3-D object looks like a combination of 2-D shapes connected together by common sides. The difference is that these 2-D shapes are not what they appear to be, but they represent the visible sides of the 3-D object as these are seen from a particular angle. Taking into account the factor of perspective, things become more complicated. For example a 3-D shape composed by a triangle ABC and a trapezium ADEB could well represent a prism or a square-pyramid (Fig. 5.9). Depending on the angle of view, perspective

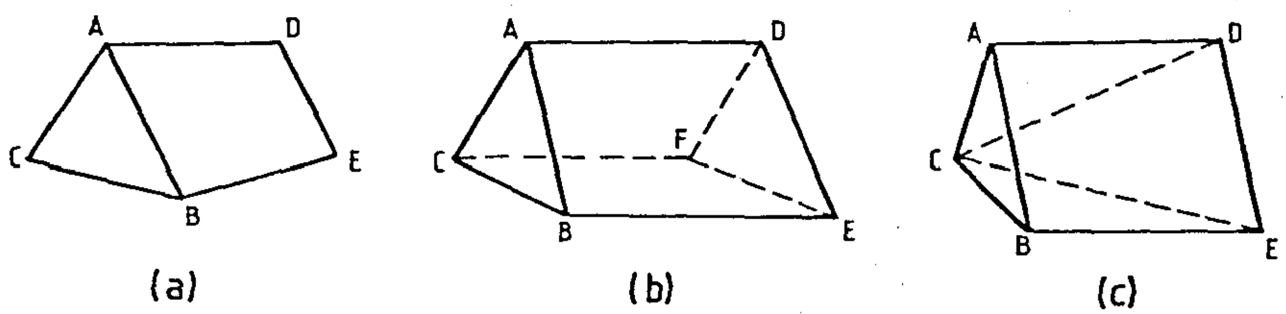


fig. 5.9

could make the actual parallelogram ADEB (Fig. 5.9b) look like a trapezium (Fig. 5.9a). On the other hand since there is no clue about the invisible side of the particular shape two possible solutions could be the ones shown by Fig. 5.9b and c. Considering all these possibilities, the 3-D recognizer tries to classify the given representation making a number of assumptions each time. These assumptions will be seen in detail when every class is examined.

Another interesting point is that the question given to the recognizer is:- '*shape(a,X)*', that is the given object is examined with respect to its vertex (3D-vertex) '*a*'. Since the way the vertices of the 2-D shapes are marked depends on the order that they are met in the picture, many possibilities can occur for each 3-D shape. The idea is that the first triangle met is marked as ABC then the second one is marked DEF. Now if BC and EF are the common sides of the two triangles then EF is substituted by BC and the whole 3-D shape becomes ABCD (Fig. 5.1∅). The two triangles

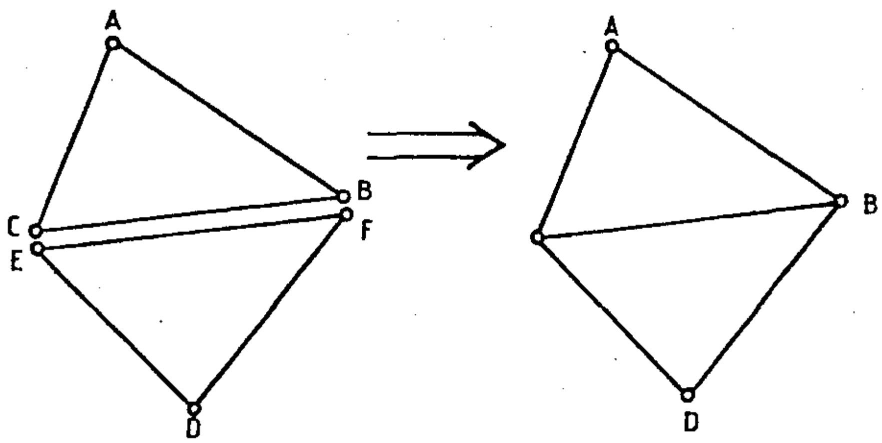


fig.5.1∅

are now ABC and BDC. The condition that checks if two points are near enough to represent the same vertex is their distance to be less than three (units of length). If this happens the letter that represents the vertex of the second shape is replaced by the one representing the vertex of the first one. In the example above F becomes B and E becomes C.

The classes that will be discussed in the following sections can be grouped into three larger groups. The first group contains 3-D shapes consisting of triangles only, the second group examines 3-D shapes consisting of both triangles and quadrilaterals and the third one looks at 3-D shapes consisting of quadrilaterals only. A main assumption for all three groups is that all the existant 2-D shapes comprising the 3-D object have at least one common vertex.

Finally every time a new alternative is tried the clause *init 2D* clears all the *atrain's* and *aquadril's*. This is defined:-

init 2D:- retractall(atrian(A,B,C,D,E,F)),rectractall(aquadril(G,H,I,J, K,L,M,N)).

5.4.1 Definitions

a) tetrahedron:- is the 3-D shape that consists of four 2-D sides each one being a triangle (Fig. 5.11a).

b) square-pyramid:- is the 3-D shape that consists of five 2-D sides four of them being triangles and the fifth - called basis - being a square (quadrilateral in general). (Fig. 5.11b).

c) truncated-triangular-pyramid:- is the 3-D shape that consists of two triangular bases and three quadrilaterals (Fig. 5.11c). It is basically a tetrahedron with a small tetrahedron missing from its top.

d) triangular-prism:- is the 3-D shape that consists of two triangular bases and three parallelograms (Fig. 5.11d).

e) truncated-square-pyramid:- is the 3-D shape that consists of two quadrilaterals as bases and four more quadrilaterals (Fig. 5.11e). It is a square pyramid with its top being cut off.

f) square-prism:- is the 3-D shape that consists of two quadrilaterals as bases and four parallelograms for the rest of its sides (Fig. 5.11f).

g) parallelepiped:- is the 3-D shape that consists of eight sides all of them being parallelograms (Fig. 5.11g).

h) rectangular-parallelepiped:- is a parallelepiped with all six sides being rectangles (Fig. 5.11h).

i) rhomboid:- is parallelepiped with its six sides being rhombuses Fig. 5.11i).

j) cube:- is the rectangular-parallelepiped with all of its six sides being squares (Fig. 5.11j).

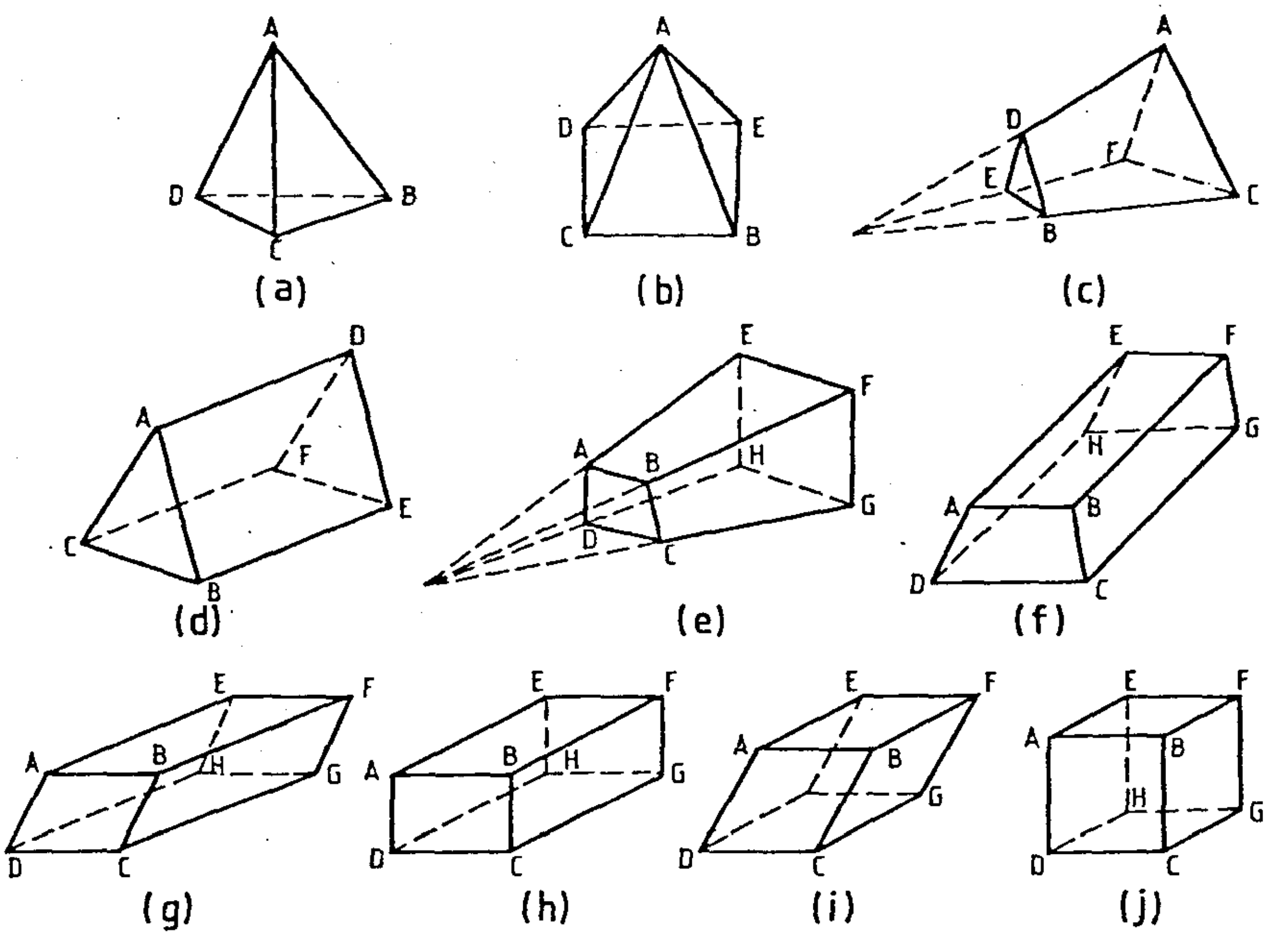


fig. 5.11

5.4.1 Group A (triangle-triangle)

The first member of this group consists of two triangles joined by a common side. This combination is most likely to represent a tetrahedron, but it could possibly be a square-pyramid, or something else. There are three different cases in this type of connection which give rise to the following definition:-

shape3D(A,tetrahedron):- tetra(A,B,C,D).

$$tetra(A,B,C,D)\text{:- } init2D, trian(A,B,C)^{*}\left\{\begin{array}{l} ,trian(C,B,D), \\ ,trian(A,C,D), \\ ,trian(A,D,B), \end{array}\right\}!,\ not\ shape, not\ (other1(A,B,C,D)).$$

$$\left.\begin{array}{l} shape3D(A,square\text{-}pyramid) \\ shape3D(A,other) \end{array}\right\}\text{ :- } tetra(A,B,C,D).$$

The three cases are illustrated in Fig. 5.12a.

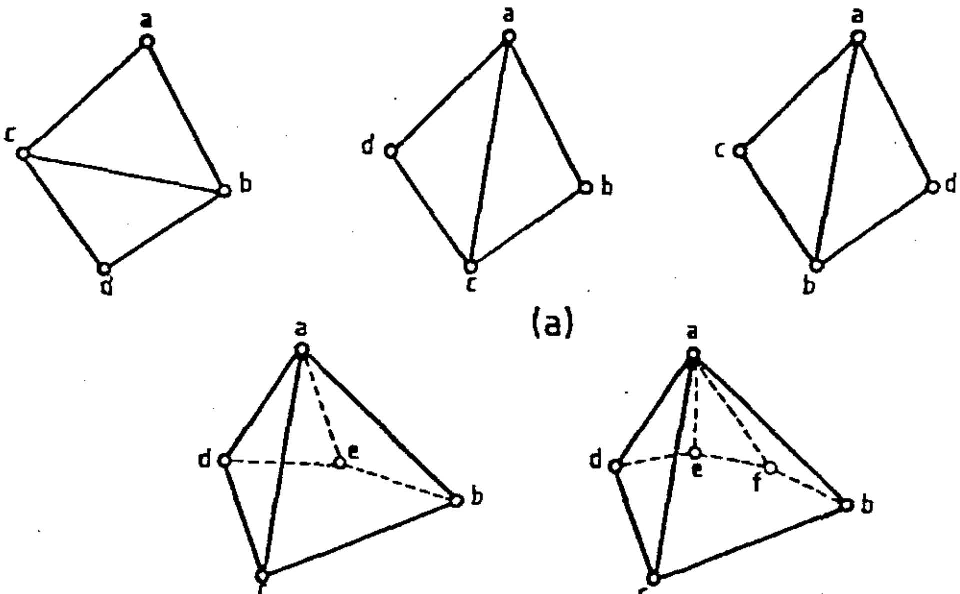


fig. 5.12

Fig. 5.12b shows the cases of *square-pyramid* and *other* respectively.

Clause notshape is defined as:-

notshape:- not(trian(A,B,C)),not(quadril(D,E,F,G)).

and makes sure that no other triangle or quadrilateral is connected to the two original triangles.

---

*Here, a shorter form for the predicate 'trian' is used instead of trian (A,X,B,Y,C,Z) for simplicity reasons. A similar simpler form is used for 'quadril' too.

Clause *other1* ensures that the two shapes are not connected to anything other than each other, and its definition is:-

other1(A,B,C,D):- conn(I,J,K),(K\==A),(K\==B),K\==C),(K\==D).

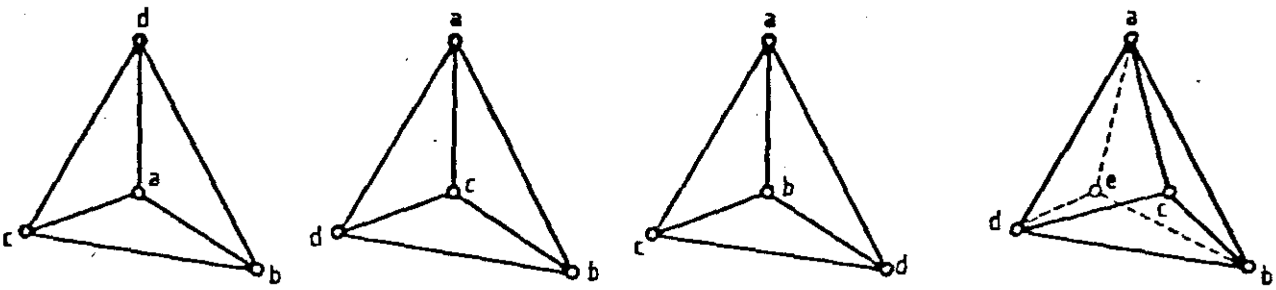
The use of '!' (cut operator)(1) is to prevent other alternatives to be sought if the goal before it fails.

The next three cases define again a *tetrahedron* or an *other*.

$$tetral(A,B,C,D) :- init2D, trian(A,B,C), \left\{ \begin{array}{l} trian(A,D,B), trian(A,C,D) \\ trian(A,C,D), trian(A,B,D), trian(C,B,D) \\ trian(A,D,B), trian(A,D,C), trian(B,D,C) \end{array} \right\},$$

notshape,not(other1(A,B,C,D)).

$$\left. \begin{array}{l} shape3D(A,tetrahedron) \\ shape3D(A,other) \end{array} \right\} :- tetral(A,B,C,D).$$



(a) (b)

fig.5.13

Fig. 5.13a illustrates the three cases and 5.13b the *other*. In the last two cases of tetral the extra triangle stands for the base of the tetrahedron. For example the *conn's* in the last case will be:-

1)	*conn(a,b)*	*4)*	*conn(b,d)*	**7)*	*conn(a,d)*
2)	*conn(b,c)*	**5)*	*conn(d,c)*	*8)*	*conn(d,b)*
**3)*	*conn(c,a)*	*6)*	*conn(c,b)*	*9)*	*conn(b,a)*

1,2 and 3 constitute *trian(a,b,c)*, 4,5 and 6 constitute *trian(b,d,c)* 7,8 and 9 constitute *trian(a,d,b)* and 7,5 and 3 constitute an extra *trian(a,d,c)*, which would not give a *tetral* because of the notshape clause.

The last member of this group is the combination of three triangles one next to the other, with the one in the middle having one side in common with the outside ones. This can be a square-pyramid or something else according to the following definition:-

```
                                          ⎧trian(A,C,D),trian(A,E,E),(E\==B)⎫
                                          ⎪trian(A,D,B),trian(A,C,E),(D\==E)⎪
                                          ⎪trian(A,D,B),trian(A,E,D),(C\==E)⎪
                                          ⎪trian(C,B,D),trian(C,D,E),(A\==E)⎪
pyram(A,B,C,D,E):- init2D,trian(A,B,C),   ⎨trian(A,D,B),trian(C,B,E),(E\==D)⎬;!,
                                          ⎪trian(C,B,E),trian(A,C,E),(E\==D)⎪
                                          ⎪trian(C,B,D),trian(D,C,E),(B\==E)⎪
                                          ⎪trian(A,D,B),trian(D,E,B),(C\==E)⎪
                                          ⎩trian(C,B,D),trian(B,F,D),(A\==E)⎭

                        notshape,other2(A,B,C,D,E).

shape3D(A,square-pyramid)⎫
                         ⎬:- pyram(A,B,C,D,E).
shape3D(A,other)         ⎭
other2(A,B,C,D,E):- conn(I,J,K),(A\==K),(B\==K),(C\==K),(D\==K),(E\==K).
```

The nine cases are illustrated in Fig. 5.14a and the case for *other* in Fig.14b.

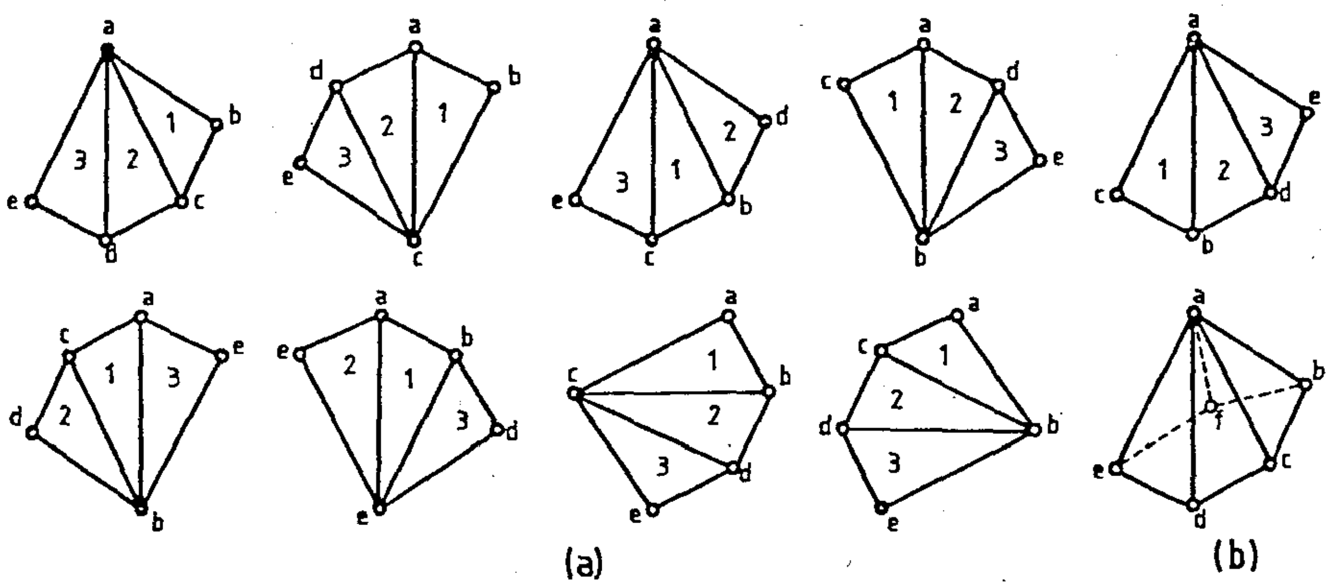


fig. 5.14

5.4.2 Group B (triangle-quadrilateral)

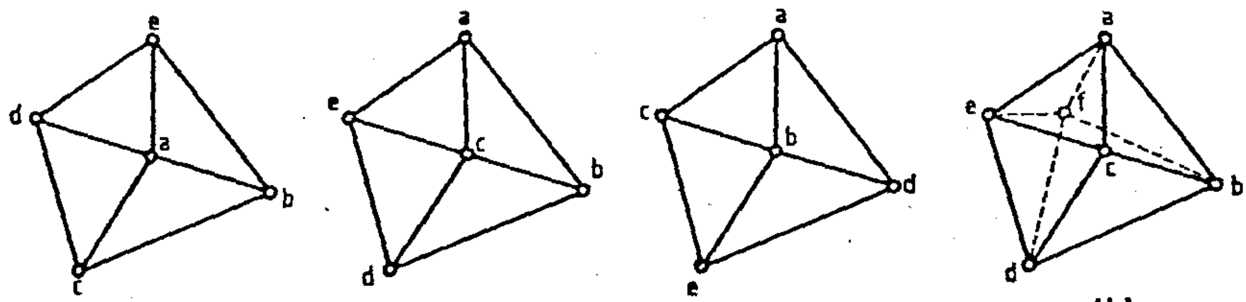
The first member of this group consists of *square-pyramids* according to the definition:-

$$\mathit{pyram1}(A,B,C,D,E)\text{:- } \mathit{init2D}, \mathit{trian}(A,B,C), \begin{cases} \mathit{trian}(A,C,D), \mathit{trian}(A,D,E), \mathit{trian}(A,E,B), \\ \mathit{trian}(C,B,D), \mathit{trian}(C,D,E), \mathit{trian}(E,A,C), \\ \mathit{trian}(B,A,D), \mathit{trian}(B,D,E), \mathit{trian}(B,E,C), \end{cases}$$

$$\left.\begin{matrix} \mathit{quadril}(B,C,D,E) \\ \mathit{quadril}(E,A,B,D) \\ \mathit{quadril}(A,D,E,C) \end{matrix}\right\}, \mathit{notshape}, \mathit{not}(\mathit{other2}(A,B,C,D,E)).$$

$$\left.\begin{matrix} \mathit{shape3D}(A,\mathit{square\text{-}pyramid}) \\ \mathit{shape3D}(A,\mathit{other}) \end{matrix}\right\}\text{:- } \mathit{pyram1}(A,B,C,D,E)$$

The *quadril* stands for the base of the square-pyramid. Fig. 5.15a shows the three cases and 5.15b the *other*.



(a) (b)

fig. 5.15

The second member contains the combination of a triangle connected with a quadrilateral by a common side. This can be a truncated-triangular-prism or a square pyramid or something else defined as:-

$$\mathit{truntripyr}(A,B,C,D,E)\text{:- } \mathit{init2D}, \left.\begin{cases} \mathit{trian}(A,B,C), !, \begin{cases} \mathit{quadril}(A,D,E,B) \\ \mathit{quadril}(A,C,E,D) \\ \mathit{quadril}(B,D,E,C) \end{cases} \\ \mathit{quadril}(A,B,C,D), !, \begin{cases} \mathit{trian}(D,C,E) \\ \mathit{trian}(B,E,C) \end{cases} \end{cases}\right\}, !, \mathit{notshape},$$

$$\mathit{not}(\mathit{other2}(A,B,C,D,E)).$$

shape3D(A,truncated-triangular-pyramid)
shape3D(A,square-pyramid) } *:-truntripyr(A,B,C,D,E).*
shape3D(A,other)

Fig. 5.16a shows the 5 cases of the *truntripyr* and Fig. 5.16b the two alternatives *square-pyramid* and *other*.

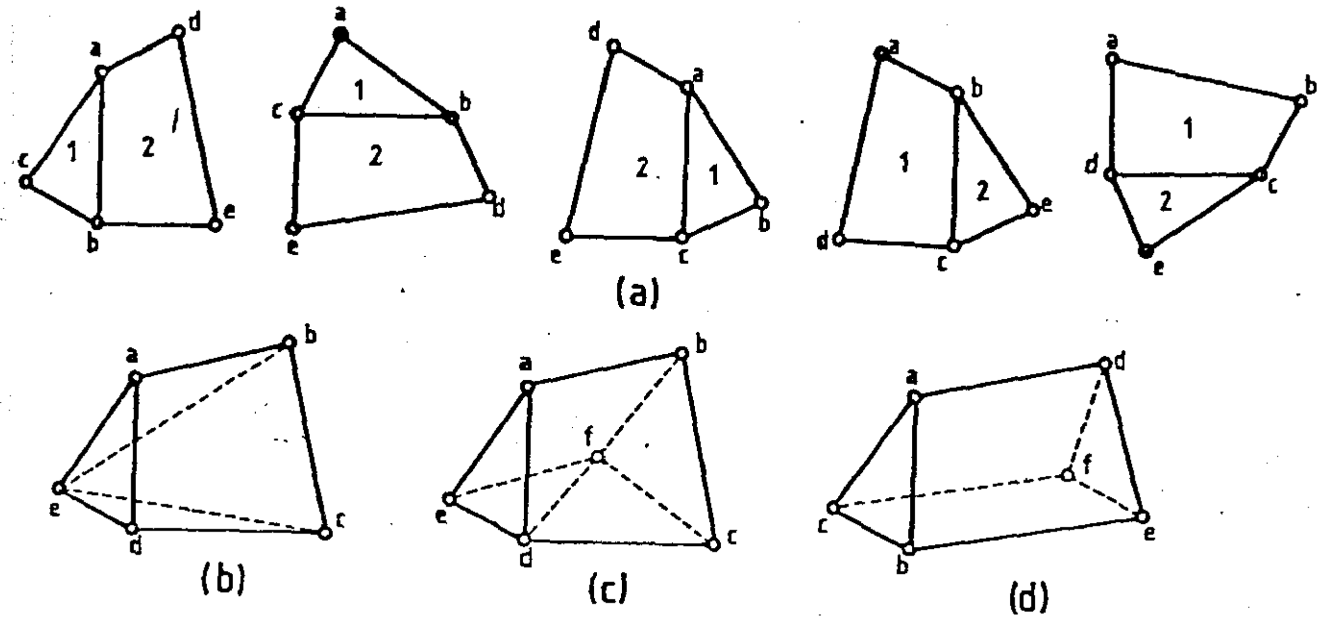


fig. 5.16

If instead of *quadril,* *paralgrm* is used in the above definition similarly the 3-D shape *triangular-prism,* is formed,

shape3D(A,triangular-prism):- triprism(A,B,C,D,E).

as shown in Fig. 5.16c.

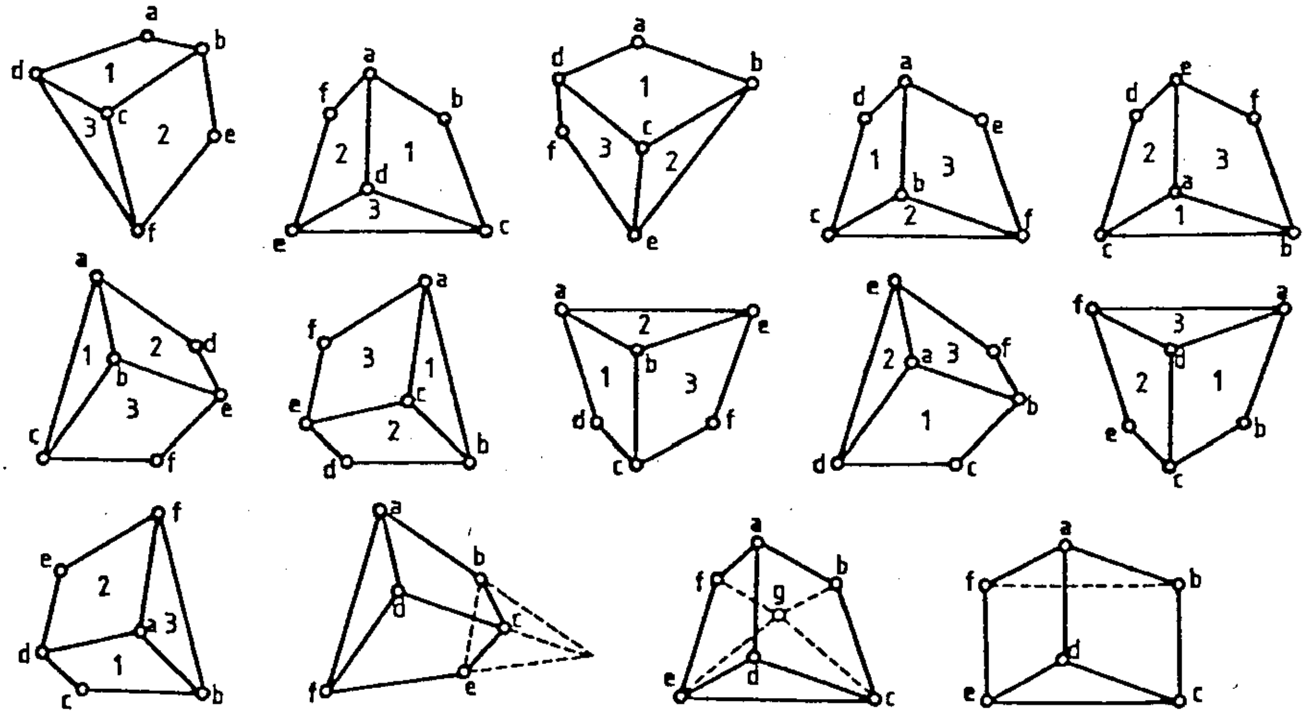
The second member of this group contains combinations consisting of two quadrilaterals and a triangle. The result is interpreted as a truncated-triangular-pyramid according to the definition:-

```
truntripyr1(A,B,C,D,E,F):- init2D,
  { quadril(A,B,C,D),!,
      { quadril(C,B,E,F),trian(D,C,F),
        quadril(A,D,E,F),trian(D,C,E),
        quadril(D,C,E,F),trian(C,B,E),
        quadril(B,A,E,F),trian(C,B,F),
        quadril(C,B,E,F),trian(B,A,E),
        quadril(A,D,E,F),trian(B,A,F),
        quadril(D,C,E,F),trian(A,D,F),
        quadril(B,A,E,F),trian(A,D,E),
    trian(A,B,C),!,
      { quadril(A,C,D,E),quadril(B,A,E,F),
        quadril(B,A,D,E),quadril(C,B,E,F),
        quadril(C,B,D,E),quadril(A,C,E,F), } ,!

     notshape, not(other3(A,B,C,D,E,F)).
other3(A,B,C,D,E,F):- conn(I,J,K),(A\==K),(B\==K),(C\==K),(D\==K),(E\==K),(F\==K)

shape3D(A,truncated-triangular-pyramid) }
shape3D(A,other)                        } :- truntripyr1(A,B,C,D,E,F).
```

The 11 cases of *truntripyr1* are shown in Fig. 5.17a and *other* in 5.17b.



(a) (b) (c)

fig. 5.17

If in the place of *quadril*, *paralgrm* is used, another class is defined by:-

shape3D(A,triangular-prism):- triprism1(A,B,C,C,D,F).

shown in Fig. 5.17c.

For the classes *triprism* and *triprism1* definition for *other* is not necessary since these are special cases of *truntripyr* and *truntripyr1* respectively, which include a definition for *other*.

5.4.3 Group C (quadrilateral-quadrilateral)

The first member of this group contains combinations of two quadrilaterals connected by a common side. The result is class *truncated-square-pyramid* to be formed, which is defined by:-

$$\mathit{trunsqrpyr}(A,B,C,D,E,F)\text{:-}\ \mathit{init2D},\mathit{quadril}(A,B,C,D),\left\{\begin{matrix}\mathit{quadril}(C,B,E,F)\\ \mathit{quadril}(B,A,E,F)\\ \mathit{quadril}(A,D,E,F)\\ \mathit{quadril}(D,C,E,F)\end{matrix}\right\},!,$$
$$\mathit{notshape},\mathit{not}(\mathit{other3}(A,B,C,D,E,F)).$$

$$\left.\begin{matrix}\mathit{shape3D}(A,\mathit{truncated\text{-}square\text{-}pyramid})\\ \mathit{shape3D}(A,\mathit{other})\end{matrix}\right\}\text{:-}\ \mathit{truntripyr}(A,B,C,D,E).$$

Fig. 5.18a shows the four cases and 5.18b the *other*.

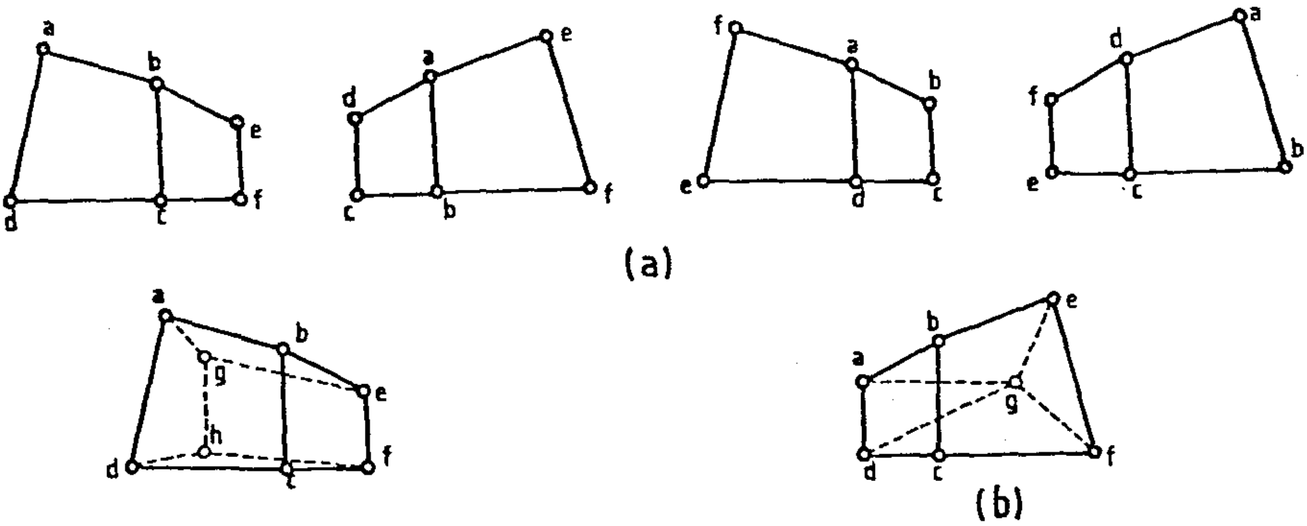


fig. 5.18

By substituting the *quadril's* with *paralgrm's*, *rectan's*, *rhomb's* and *squar's* the following classes arise respectively:-

$$\left.\begin{array}{l} shape3D(A,square\text{-}prism) \\ shape3D(A,triangular\text{-}prism) \\ shape3D(A,parallelepiped) \end{array}\right\} :-\ sqrprism(A,B,C,D,E,F).$$

shape3D(A,rectangular-parallelepiped):- rectparalgrm(A,B,C,D,E,F).

shape3D(A,rhomboid):- rhomboid(A,B,C,D,E,F).

shape3D(A,cube):- cube(A,B,C,D,E,F).

These six new classes are illustrated in Fig. 5.19.

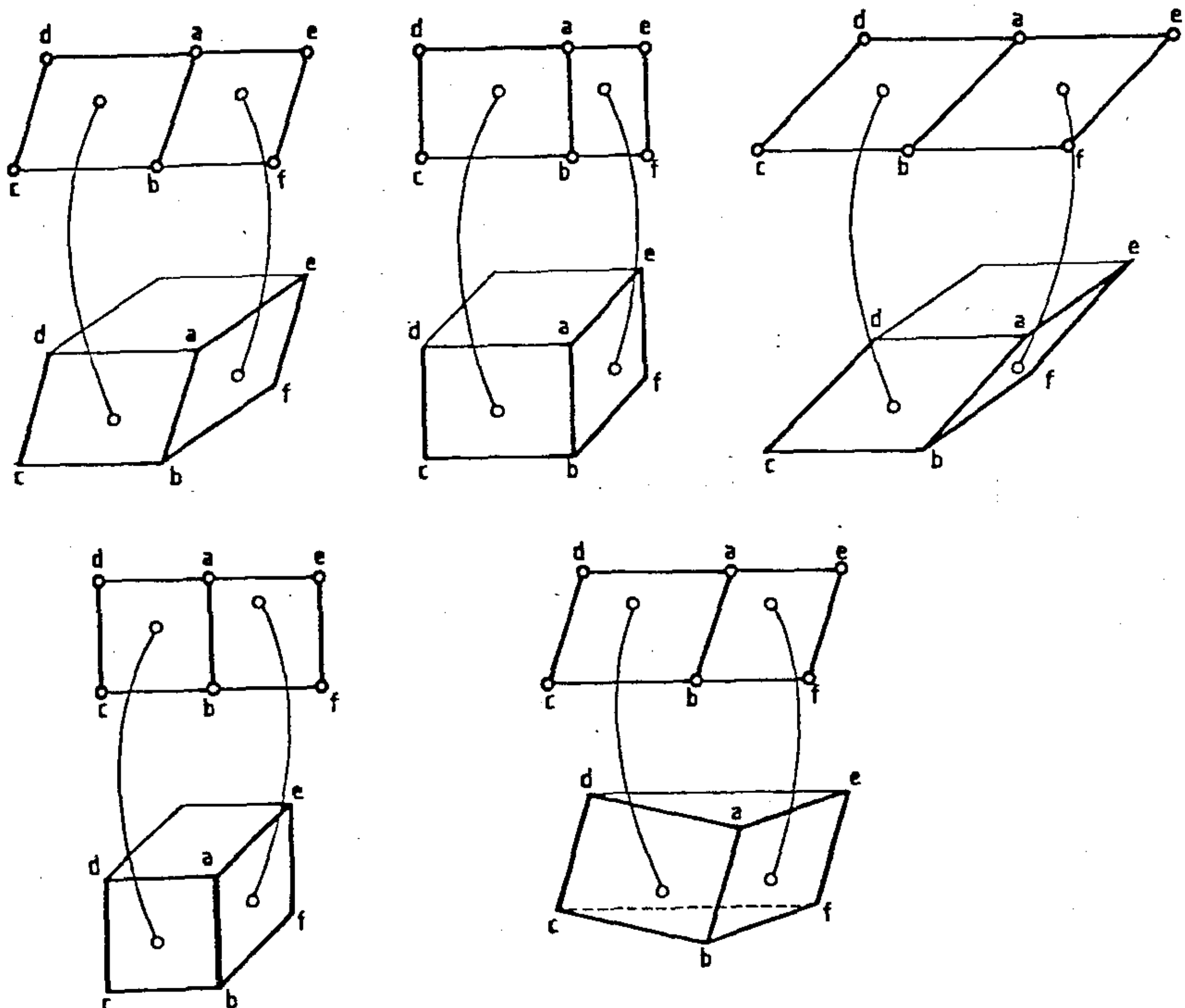


fig. 5.19

Finally the last member of this group is three quadrilaterals connected according to the definition:-

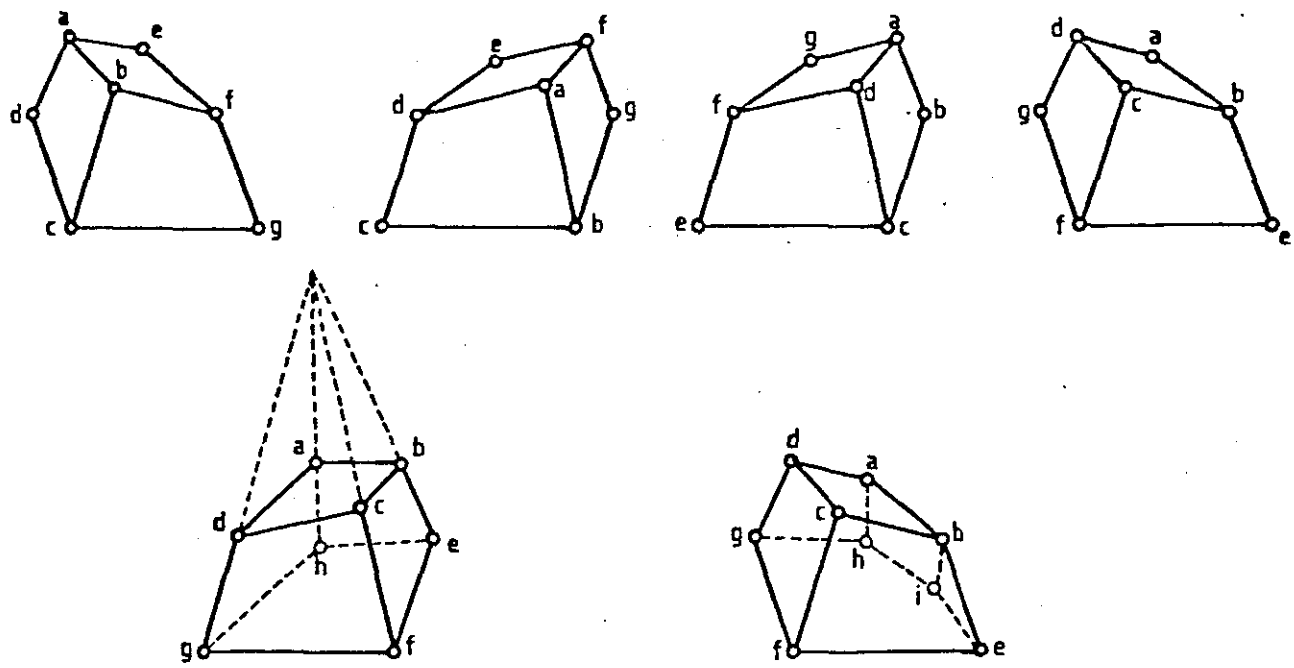
$$\mathit{trunsqrpyr}(A,B,C,D,E,F,G)\text{:- } \mathit{init2D}, \mathit{quadril}(A,B,C,D), \left\{\begin{array}{r} \mathit{quadril}(B,A,E,F), \mathit{quadril}\\ (C,B,F,G)\\ \mathit{quadril}(A,D,E,F), \mathit{quadril}\\ (B,A,F,G)\\ \mathit{quadril}(D,C,E,F), \mathit{quadril}\\ (A,D,F,G)\\ \mathit{quadril}(C,B,E,F), \mathit{quadril}\\ (D,C,F,G) \end{array}\right\},$$

$$\mathit{notshape}, \mathit{not}(\mathit{other4}(A,B,C,C,E,F,G)).$$

$$\mathit{other4}(A,B,C,D,E,F,G)\text{:- } \mathit{conn}(I,J,K), (A\backslash==K), (B\backslash==K), (C\backslash==K), (D\backslash==K), (E\backslash==K), (F\backslash==K), (G\backslash==K).$$

$$\left.\begin{array}{l} \mathit{shape3D}(A, \mathit{truncated\text{-}square\text{-}pyramid})\\ \mathit{shape3D}(A, \mathit{other}) \end{array}\right\}\text{:- } \mathit{trunsqrpyr1}(A,B,C,D,E,F,G).$$

Fig. 5.20a shows the four cases of *trunsqrpyr1* and 5.20b of the *other*.



(a) fig. 5.20 (b)

If instead of *quadril*, *paralm* and *rhomb* is used respectively in the above definition, four more classes are formed:-

```
shape3D(A,parallelepiped):- parallelepiped1(A,B,C,D,E,F,G).
shape3D(A,rhomboid)                    }
shape3D(A,rectangular-parallelepiped)  }:- rhomboid1(A,B,C,D,E,F,G).
shape3D(A,cube)                        }
```

Fig. 5.21a illustrates these classes.

A combination of one *quadril* and two *paralgrm's* defines a *square-prism*.

A combination of one *rectan* and two *paralgrm's* defines a *rectangular-parallepiped.*

And a combination of one *squar* and two *rhomb's* defines a *cube.* In all of these definitions it is important in what order the different types of quadrilaterals appear and thus there are 12 cases instead of 4 of the original definition.

Fig. 5.21b illustrates a representative of each case.

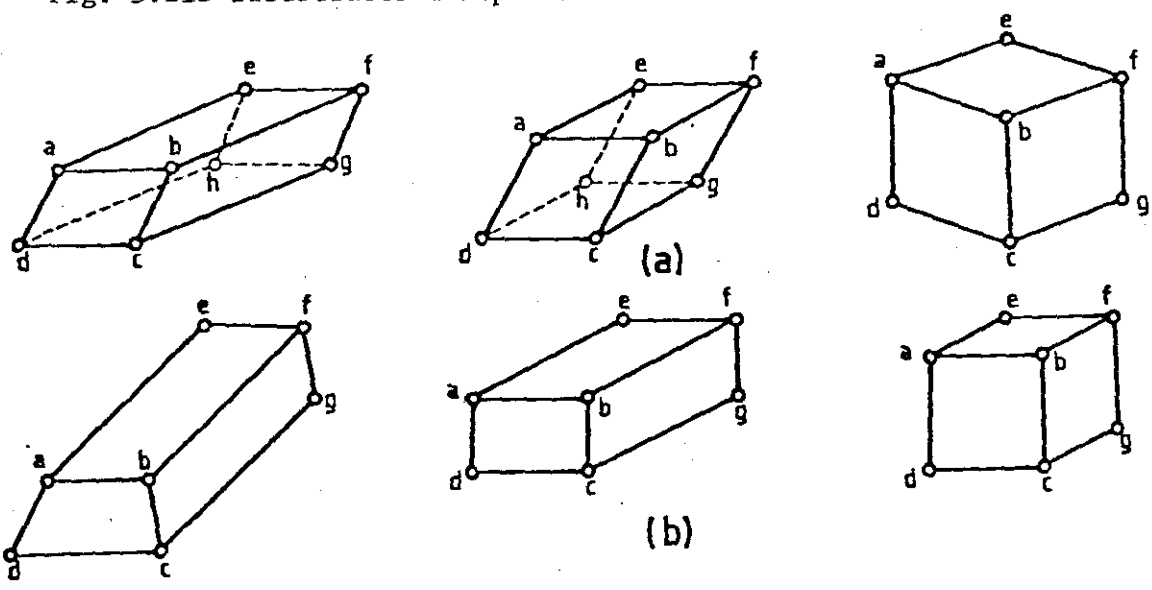


fig. 5.21

As in the previous group B, every one of the combinations just mentioned will be interpreted as *other* since they are subcases of the general forms *trunsqrpyr* and *trunsqrpyr1* which include the interpretation *other* in their definitions.

Finally any combination that does not belong to the classes examined by the three groups, is classified as *other*.

5.5 THE 2-D RECOGNIZER AS AN AUTONOMOUS SYSTEM

As an autonomous system the 2-D recognizer can recognize single or unconnected 2-D shapes and classify them into one of the following classes:- *triangle*, *quadrilateral*, and *other*. Its main difference from the 2-D recognizer as part of the 3-D recognizer is, that the former can produce some further classifications in the case of triangle and a few extra ones in the case of a quadrilateral. Another main difference is the use of *init2D* in the definitions of the secondary classes. This is necessary because, for example a triangle after its identification is effectively removed from the data by use of the *assert* predicate, and thus a second use of it is impossible. This means that a definition such as:- shape(A,isosceles):- trian(A,X,B,Y,C,Z), equal pair(X,Y,Z), would fail. This is because a trian(a,ab,b,bc,c,ca) has already been met, and since atrian(a,ab,b,bc,c,ca) has been asserted to the data, the first goal of the body would fail.

According to this 2-D recognizer, a triangle is *isosceles* if it has a pair of equal sides:-

shape(A,isosceles):- init2D,trian(A,X,B,Y,C,Z),equal pair(X,Y,Z).

or it is *equilateral* if it has all its three sides equal:-

shape(A,equilateral):- init2D,trian(A,X,B,Y,C,Z),equalline(X,Y),equalline (X,Z).

or it has a *right angle* if the theorem of Pythagoras is applied among the length of its sides:-

shape(A,right angled):- init2D,trian(A,X,B,Y,C,Z),right(X,Y,Z).

or finally it has an *obtuse angle* if the theorem of the obtuse angle is applied among its sides:-

shape(A,obtuse angled):- init2D,trian(A,X,B,Y,C,Z),obtuse(X,Y,Z).

On the other hand a quadrilateral is *non-convex* if the sum of its angles is less than 360^{o} (see also 3.4.3d), *shape(A,non-convex):- init2D,quadril(A,B,C,D),non-convex(A,B,C,D).*

or it is a *trapezium* if it has *one* pair of opposite sides parallel:

shape(A,trapezium):- init2D,quadril(A,W,B,X,C,Y,D,Z),((paral(W,Y),not ((paral(X,Z)));(paral(X,Z),not(paral(W,Y)))).

or finally it is an *isosceles trapezium* if it is a trapezium with equal non-parallel sides.

shape(A,trapezium):- trapez(A,W,B,X,C,Y,D,Z),((equaline(X,Z),not equalline(W,Y));(equalline(W,Y),not (equalline(X,Z))).

As *other* is classified any shape that is neither a triangle nor a quadrilateral:-

shape(A,other):- init2D,not(trian(A,B,C)),not(quadril(D,E,F,G)).

5.6 A COMPARISON OF THE 2-D AND THE 3-D RECOGNIZER

The 2-D recognizer examines combinations of 1-D sides to define its classes while the 3-D one examines combinations of 2-D sides.

The *conn* predicates in the 2-D recognizer correspond to a single 2-D shape, while in the 3-D one they correspond to more than one 2-D shape.

A shape recognized by the 2-D recognizer is classified as *other* if it does not belong to any of the other classes. In the 3-D one, every 3-D shape can be classified as *other* because there are invisible sides.

The question 'shape(A,X)' refers to any of the vertices of the shape in the 2-D recognizer, while in the 3-D one 'a' is preferred to 'A'. This is because there is more than one way of connection between the 2-D shapes.

In the 3-D recognizer 4 types of *other* are used to prevent 2-D shapes, non-recognizable by the 2-D recognizer, to be taken as *noshape*. Every one of them corresponds to a different number of *visible* vertices of the 3-D object.

The 3-D recognizer allows multiple classifications, because of the ambiguity in the interpretation of some 3-D shapes.

The complete recognizer written in PROLOG is presented in Appendix 4.

SUMMARY - CONCLUSIONS

- The recognizer can be split into two interrelated parts, the 2-D recognizer and the 3-D recognizer.
- The 2-D recognizer examines the structure of 2-D shapes, consisting of 1-D side combinations and classifies them into *one* of the classes:- *triangle, quadrilateral, other.*
- The 3-D recognizer examines the structure of 3-D shapes, consisting of 2-D side combinations and classifies them in *some* of the classes:- *tetrahedron, square-pyramid, truncated-triangular-pyramid, truncated-square-pyramid.*
- By using relations among the main components of the 2-D shapes further secondary classifications are made:-
  triangle *(isosceles, equilateral, right angled, obtuse angled)*
  quadrilateral *(non-convex, trapezium, isosceles-trapezium, parallelogram, rectangle, rhombus, square).*
- These give rise to further classifications of the 3-D shapes:- *triangular-prism, square-prism, parallelepiped, rectangular-parallelepiped, rhomboid, cube.*
- The 2-D recognizer is mainly part of the 3-D one but it can be used on its own with some minor modifications.

REFERENCES

1. David Warren, *"Expert Systems in the Microelectronic Age"*, ed. D. Michie, Edinburgh University Press, 1980.
2. W.F. Clocksin and C.S. Mellish, *"The UNIX PROLOG System"*, Software Report 5, University of Edinburgh, 198Ø, Section 4.

Chapter 6

CONCLUSION - DISCUSSION

6.1 INTRODUCTION

In the previous chapters the successive stages leading towards the recognition of 3-D shapes have been examined separately. This chapter attempts to describe how the whole process works and to draw some conclusions from it. The process is divided into four parts:- the function of the hardware, the preprocessing programs, the real-vertex verification programs and the recognition programs. The effectiveness of each one of them is examined and a number of suggestions for further development is made. Finally, the last paragraph contains figures and photographs which illustrate some results of the project.

6.2 THE PROCESS AS A WHOLE

The process starts by pointing the camera at the object which is to be recognized. The object is normally white (or of some light colour) and is placed against a black background. Once in focus, the brightness and contrast knobs are adjusted until a satisfactory picture appears on the T.V. monitor. If the object is very small, the zoom facility of the camera is used to magnify it appropriately. A spot light, illuminates the object from one side or above, in such a way that its shadow disappears and the different faces (sides) of it, have different gray level representations on the screen. This is very important, since a successful recognition is very much dependent on the condition that different sides are represented by pixels of different gray values (normally two levels apart, although this depends on the parameters of the isolation procedure). With all the above adjustments carefully made, a frame is grabbed by pressing the 'FRAME GRAB' switch of the ROBOT. From this point the programs are ready to run. By giving the appropriate commands ('MAIN$G' for the micro and 'a.out' for the VAX) the two programs start and do the following:-

The micro transfers the digitized picture into locations 4ØØØ-7FFF and starts the preprocessing operations. First it saves a copy of the original picture in locations AØØØ-DFFF in order to use it later. The first transfer lasts about 3Ø seconds. Then the picture is scanned and all the pixels of different pixel values, are summed up and stored in locations 3ØØ-31D (every sum is stored in two bytes). The pixels of the background are not taken into account. The above found sums are added together to form the number of pixels which constitute the main object, and this value is stored in location 32Ø. The next process selects the gray levels with partial sum over 6% of the entire number of the main object pixels, which are saved in locations 16Ø onwards. However there is a condition, that a gray level is saved, only if it differs by two at least from the previous saved gray level. At the end of this procedure an end marker FF is placed at the end of the array of *significant* gray levels. For every one of these values until FF is reached the following sequence of operations is performed. The original frame is masked so that the current significant gray level value is turned to white (∅) and all the other pixels to black (∅F). The obtained picture (which is an isolated face of the object) is then averaged and edged and the edge is followed. If the following is successful, two arrays are formed, one for the coordinates of the detected vertices (4∅∅-47F), and one of the pointers to those of the previous addresses that contain real vertices (48∅-4FF). If the procedure of edge-following fails and the recoverer can not pick up the edge again, then the procedure is ended with a message *FAIL* on the screen. In this case the object can not be recognized and is by default classified as *other*. After a successful edge-following a message *SEND* appears on the screens and it signals that the two arrays for one side of the object are ready to be handed to the VAX minicomputer for the verification phase. At this point it should be mentioned that all the just traced cells of the boundary (all different from ∅) become black, except for those being white. This copes with the case of two disjoint sides with the same pixel values. After this the follower is called again, and this is continued until no more white cells exist on the picture (single pixels are not taken into account). Next the original picture is loaded back to 4∅∅∅-7FFF and it is masked according to the new significant gray level. The same sequence as before is then followed until FF is met. This is signalled with an *END* message on the screen.

On the other side, the program in VAX is waiting for data, i.e. the values of the arrays which determine the vertices of each side of the object. When that data has been received two new arrays are created V[51] for the vertices and N[16] for the pointers. The end of these two arrays is marked by the end marker -1. If there is no end marker at the end of the pointer array N[16] the procedure fails and a message: *'2-D shape with more than 15* *vertices, algorithm fails'* is given on the screen. This suggests automatically that the 3-D object is classified as *other*. On the other hand if V[51] has got no end marker, a message: *'too many vertices, augment dimension of V[]'* indicates that V[51] is not enough to contain all the coordinates and needs enlargement before the procedure is resumed. Finally, every time a new set of arrays arrive, a counter is kept and when more than 5 are received, a message:- *'solid with more than 5 sides, algorithm fails'* is printed on the screen. This means again that the object is classified as *other*. For each one set of arrays a corresponding set of clauses is formed and are written in a file named *data* which is created in the first round. It is obvious that before every application of the whole process this file needs to be deleted, so it does not contain any old clauses. When the first value of N[51] is the end marker the end of the procedure is signalled by a message:- *'end of procedure'*.

The final phase, is the phase of recognition. This is done by *entering* PROLOG first. Then the two recognizers *2D* and *3D*, and file *data* are consulted. Finally a goal *'shape 3D(a,X)'* is given. If the object is described by any of existing structures then a message:- *'**(top)PROVED: shape 3D(a,cube)'* will be a possible answer. By typing ';' each time at the end of every answer, all the alternatives will be obtained. When there is no alternative an answer *'no'* signals the end of the procedure. If the first answer is *other* then the 2-D recognizer can be tried by typing:- *'shape(A,X)'*. If the shape is a recognizable 2-D one, the correct answer will appear on the screen, otherwise a message:- *'**(top)PROVED: shape(a, other)'* will be the end of the recognition phase. Of course alternatives can be sought in this case too by the use of ';'. Fig. 6.1 presents a flowchart of the whole process.

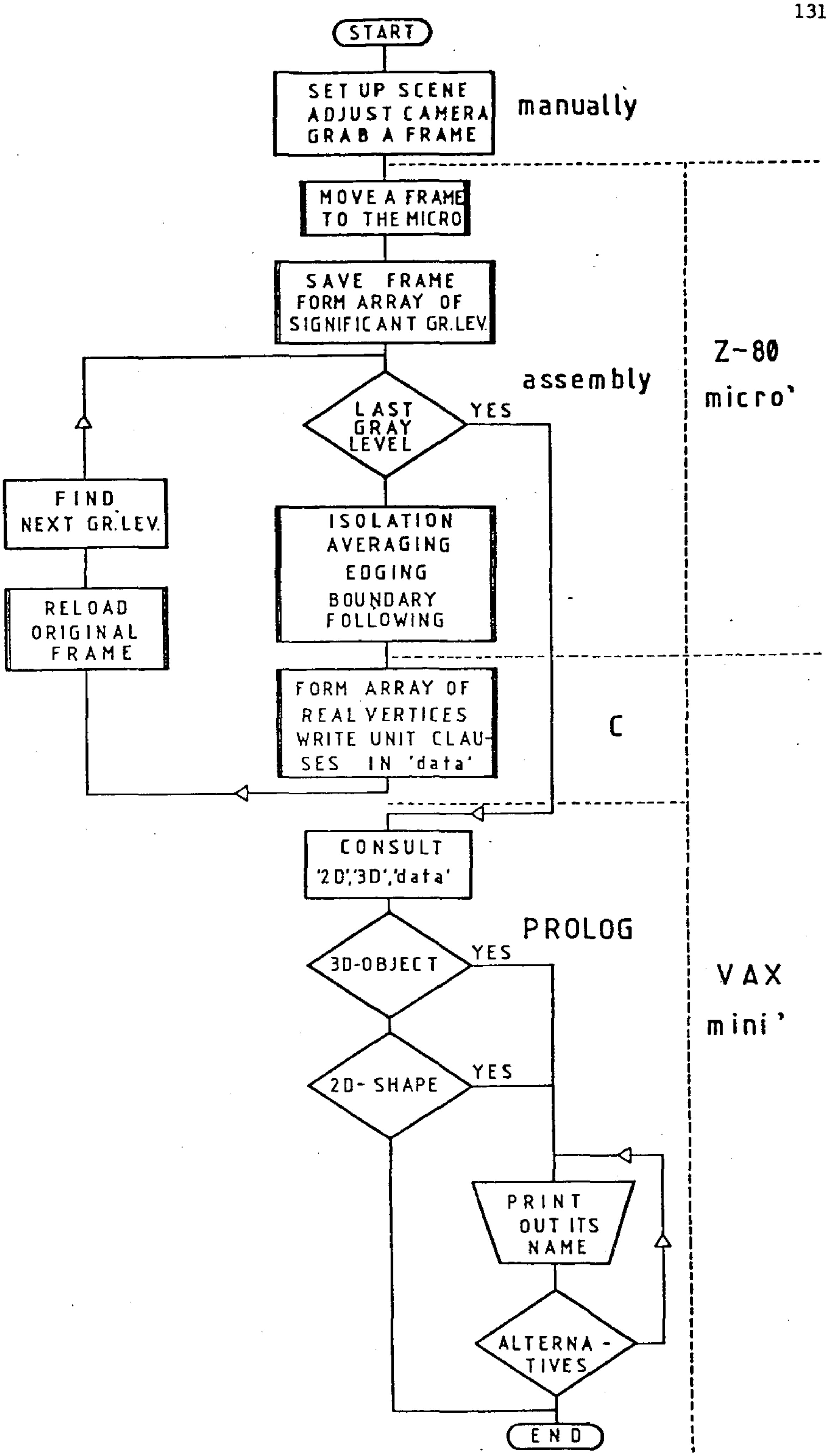

START
SET UP SCENE
ADJUST CAMERA
GRAB A FRAME
manually
MOVE A FRAME
TO THE MICRO
SAVE FRAME
FORM ARRAY OF
SIGNIFICANT GR.LEV.
Z-80
micro'
assembly
LAST
GRAY
LEVEL
YES
FIND
NEXT GR.LEV.
RELOAD
ORIGINAL
FRAME
ISOLATION
AVERAGING
EDGING
BOUNDARY
FOLLOWING
FORM ARRAY OF
REAL VERTICES
WRITE UNIT CLAU-
SES IN 'data'
C
CONSULT
'2D','3D','data'
PROLOG
3D-OBJECT
YES
VAX
mini'
2D-SHAPE
YES
PRINT
OUT ITS
NAME
ALTERNA-
TIVES
END

fig. 6.1

6.3 DISCUSSION

First of all, it should be pointed out that the objective of this project is the development of a system that recognizes simple 3-D objects, by combining efficiently the developed software with the existing hardware. The latter consists mainly of a number of elementary, unsophisticated and relatively inexpensive equipment, which often need the support of the software to perform their tasks. In the following discussion the function of the main phases of the system is examined with respect to their further development and improvement, keeping the cost as low as possible.

The hardware, as it stands, is quite inflexible because there is no communication between the processor and the camera or the lighting. This means that once set, no further adjustments can be made to the scene to obtain different views of the object or more detailed shots of some ambiguous parts of it. Basically, a great deal of human factor is involved in the setting up of the scene, and there is no automation in the various adjustments. What it is suggested in this case is a *mechanical arm* with *pan* and *tilt* facilities which is directed from the processor itself. This can move the camera so that shots from different angles are taken and they are either combined or the most descriptive is kept. When ambiguous areas exist, where the detection of an edge is not very easy, the mechanism should be able to move the camera closer to these areas, so that a clearer shot be taken. An automatic zoom control could do that. The information of the different shots will be processed so that a more perspective image of the object be obtained. This could be the first part of a modelling process that gives a very descriptive model of the object with even the hidden parts of it. A more sophisticated technique could perhaps involve two cameras taking a *stereoscopic* view of the object. The quantization of the picture into 128×128 pixels, could be increased to 256×256 or 512×512 for better quality of picture. The resolution (16 gray levels) is sufficient. The ROBOT should be modified so that the processed image is transmitted to the T.V. monitor for a faster and more precise representation of it. The EPROM of the printer can be re-programmed so that a special set of 16 characters represent the 16 gray levels of resolution. That saves time by avoiding multiple overprintings and at the same time gives a better visual impression to the human eye.

The main advantage of using the microcomputer system is that the programs can be written into EPROMs and occupy very little space. For example, a microcomputer system adequately programmed could be part of the visual system of a robot. On the other hand there are drawbacks, such as limitations in memory and slower processing. It takes 3Ø seconds for the micro to transfer a frame from the ROBOT to its own memory. The longest of the preprocessing operations is the edging; it takes about 3½ minutes to be completed! Finally the programming is done in Assembly, which is a low level language. The latter has the only advantage that it copes easily and efficiently with bit manipulation, but in general it has limited capabilities. Arithmetic operations are performed in the *Hexadecimal* numerical system (not very familiar) and multiplication and division have to be written by the programmer as separate routines. There are no array facilities and no trigonometric and other common functions. Finally the code is not very clear for everyone to follow. A weakness of the preprocessing phase is that its effectiveness depends very much on a good original picture. The different sides of the object have to be very distinct from each other and this means perfect illumination and sufficient contrast between them. This weakness can be overcome by introducing a *spatial differentiation* i.e. an operation in which the 2-D difference of light intensity at each pixel and the direction of the gradient are calculated by a special 3×3 array.

This involves some trigonometric functions and thus a more powerful language is required.

The program in C is a transit phase between the use of the micro and the final phase of recognition. Basically it deals with some operations which would take a very long time for the micro to perform. Thus they speed up the process and prepare the unit clauses which will be the *data* for PROLOG. A further development of this phase could be to supply the program with a routine capable of drawing lines. This would give the process the advantage of drawing the figures, that would act as more precise models extracted from the original picture. The latter combined with the mechanical arm facility mentioned earlier, could cope with the invisible sides of the object and their representation on the drawn figure.

Both the 2-D and 3-D recognizer are written in PROLOG. A major advantage of this is that the structure of PROLOG makes easy further modifications so that more specific classifications can be obtained. On the other hand the programs can be made capable of recognizing more complicated objects. This is achieved by simply adding the necessary clauses that cope with the new figures at the end of the program. Minor alterations have to be made though, in order to make the two programs compatible. The 2-D recognizer copes with all the possible straight line shapes with maximum number of vertices four apart from the shape in Fig. 6.2a (which is taken as two triangles with a common vertex). The 3-D recognizer, deals with most of the objects that are combinations of recognizable 2-D shapes. Every object consists of four triangles or three quadrilaterals at most. This leaves out the cases of Fig. 6.2b and c. The effect of perspective view and the invisible sides make the task of the 3-D recognizer more difficult. Thus the last alternative of every case is *other*.

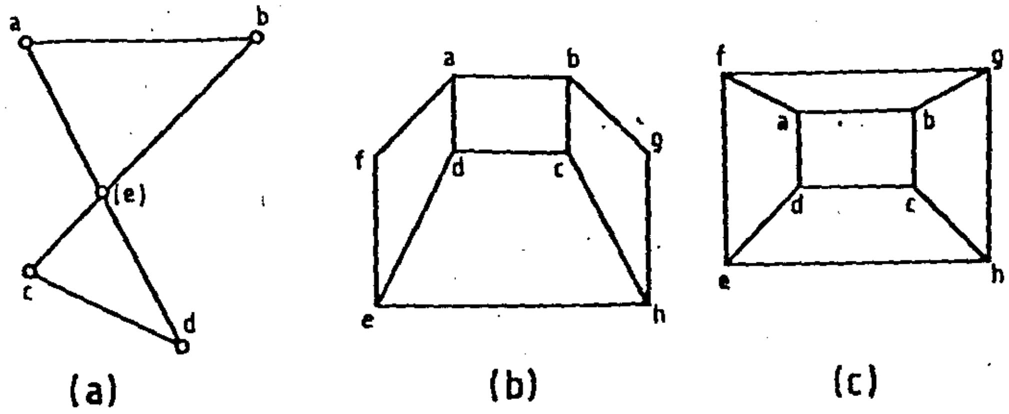


fig. 6.2

Of course that could be avoided if the mechanical arm was used to move the camera round the object in order to *see* all its sides.

A very interesting point arises from the comparison of the present grammar with the PLEX grammars. The latter use as primitives entities (line segments) with N-attaching points, i.e. they treat the 3-D shapes as a set of interconnected line segments, according to their rules. The present grammar gets round this by treating the 3-D shapes as combinations of interconnected 2-D shapes. In other words it interprets a concatenation of 2-D shapes as a 3-D object. The advantage of this is that the only primitives it uses are the *conn's*, which are the same for both 2-D and 3-D, while a plex grammar would use .napes with 2 and 3 attaching points respectively.

6.4 RESULTS

This section contains some results from a number of applications on 2-D and 3-D shapes.

a) *a square:-* The actual frame (the gray levels are in reverse order) as is shown in Fig.3. Some noise can be observed above the top-left corner and near the left-hand side of the shape, in the form of two white stripes parallel to it. Finally there is some distortion near the bottom-right corner. Fig. 4 shows the shape after the averaging-intensification operator

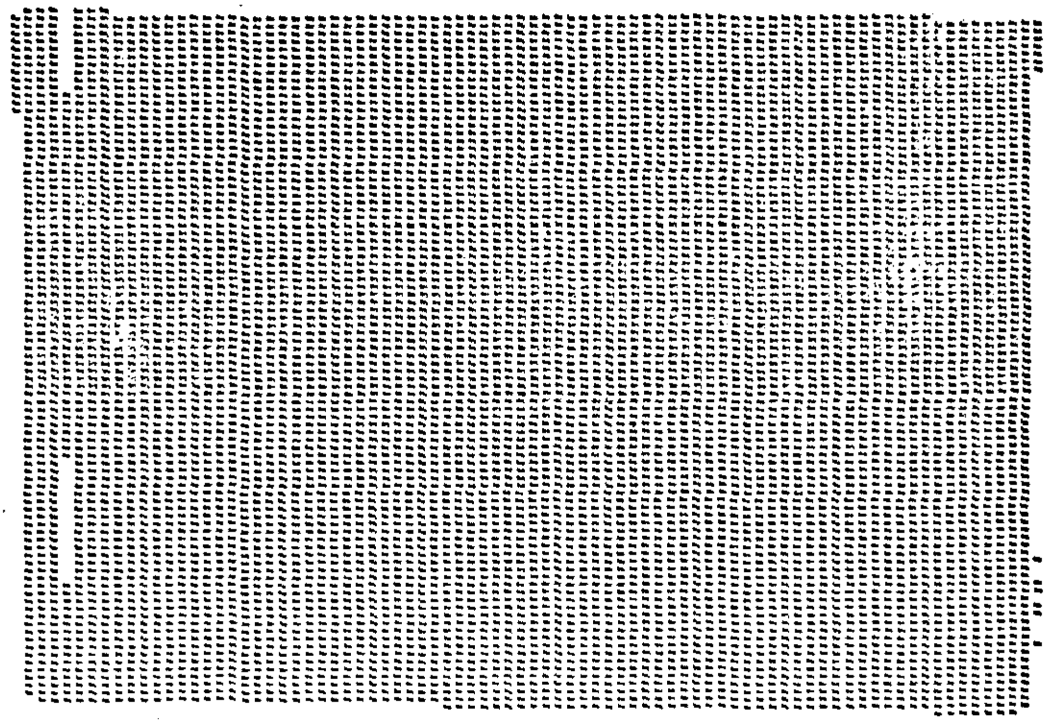


FIGURE 3

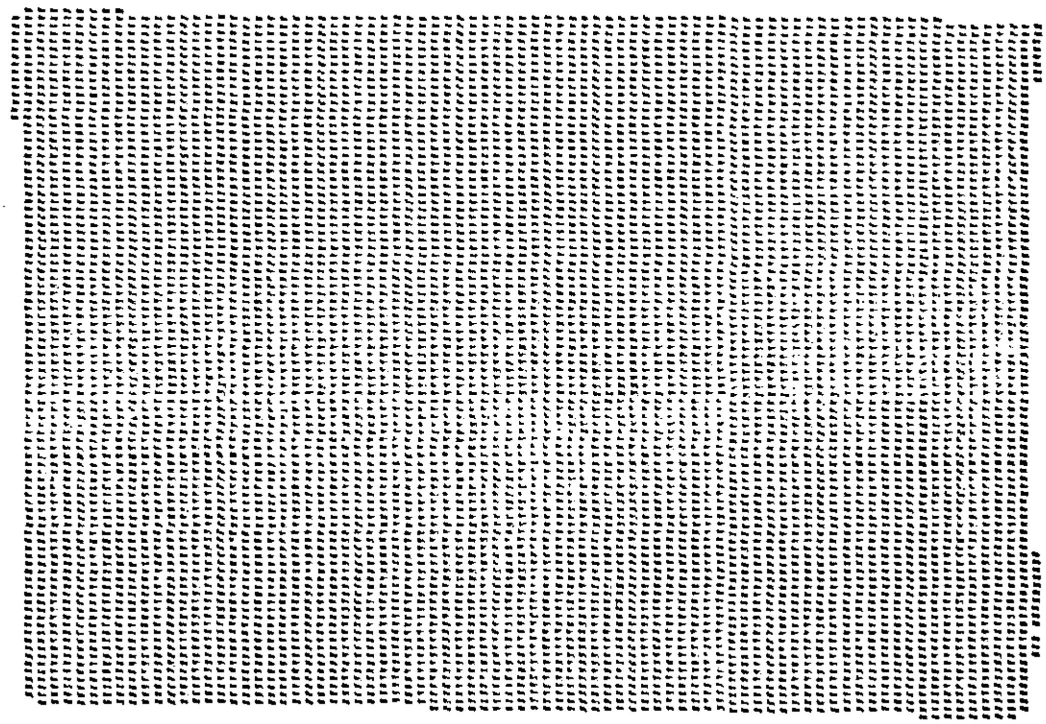


FIGURE 4

has been applied on the original picture. The results are obvious. The noise has been eliminated, the white gaps have been filled and the rough part of the right-hand side has been smoothened-up. The edge operator leaves the boundary of the shape as shown in Fig. 5.

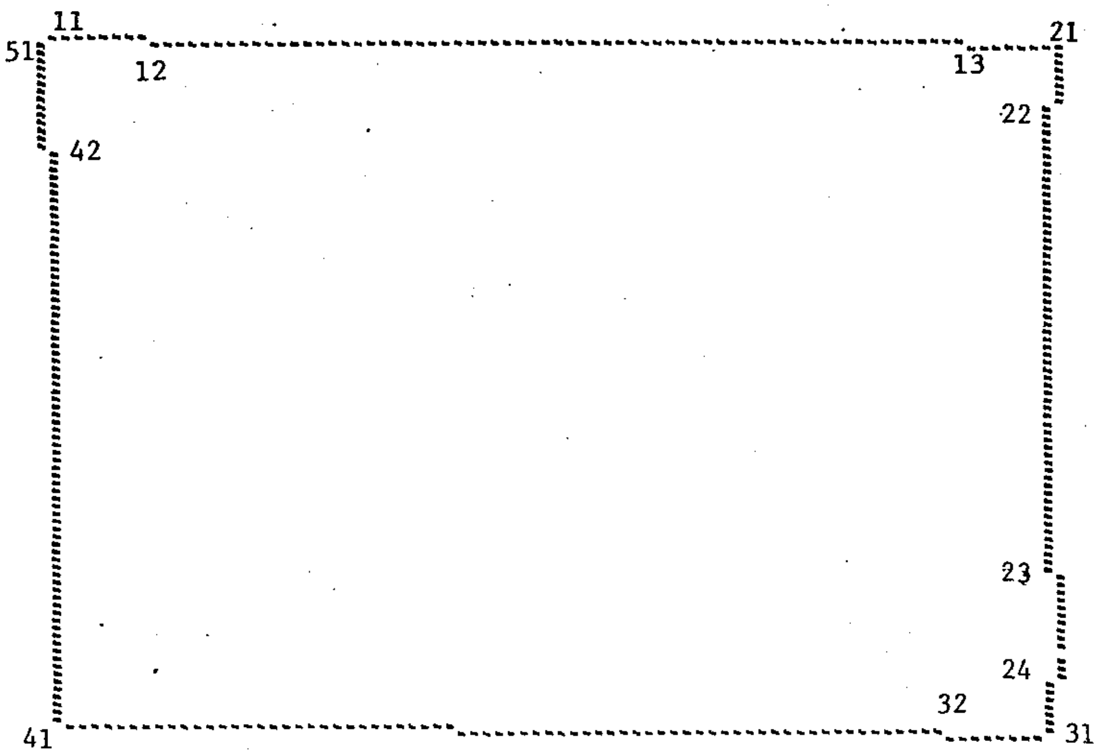


FIGURE 5

The explanation of the little gap near the botton of the right-hand side is given in 2.4.3. The follower marks the 11,21,31,41,51 as real-vertices and 12,13,22,23,24,32,42 as pseudo-vertices. It also copes with the gap of the boundary between 23 and 24. All the pseudo-vertices are eliminated because they are within the prespecified tolerance and 51 is also eliminated for being too close to 11. In more detail the vertex-detector marks 11 as the first rear-vertex and continues until it comes across 13. At this point 12 was not marked as anything because it was just the link of two units. By applying the first criterion for pseudo-vertices both 12 and 13 are marked as such. 21 is marked as a real-vertex according to the first criterion for real-vertices. 22,32,42 are marked as pseudo-vertices for the same reason as 12 and 13, and, 23 and 24 according to the fourth criterion (§3.3.1). Finally, 51 is the last boundary cell and as such is a real-vertex. These four real-vertices give rise to the following list of unit clauses and answers from the recognizer:-

```
conn(a,ab,b).
line(ab,89).
slope(ab,91).
sqrline(ab,7921).

conn(b,bc,c).
line(bc,78).
slope(bc,1).
sqrline(bc,6Ø84).

conn(c,cd,d).
line(cd,89).
slope(cd,89).
sqrline(cd,7921).

conn(d,da,a).
line(da,81).
slope(da,1).
sqrline(da,6561).

angle(a,9Ø).
angle(b,9Ø).
angle(c,92).
angle(d,88).

shape 3D(a,X).
no.
shape(A,X).
** (top) PROVED : shape(a,quadrilateral) ;
** (top) PROVED : shape(a,parallelogram) ;
** (top) PROVED : shape(a,rectangle) ;
no.
```

The actual shape is a square but because of some distortion in the digitized picture adjacent sides ab and bc are not *equal* (within the tolerance of *equal1)* and thus the final classification is: *rectangle*.

b) *a triangle:-* The picture of the shape is given in Fig.6. The procedure is similar to the previous case and the unit clauses with the answers of the recognizer are given below:-

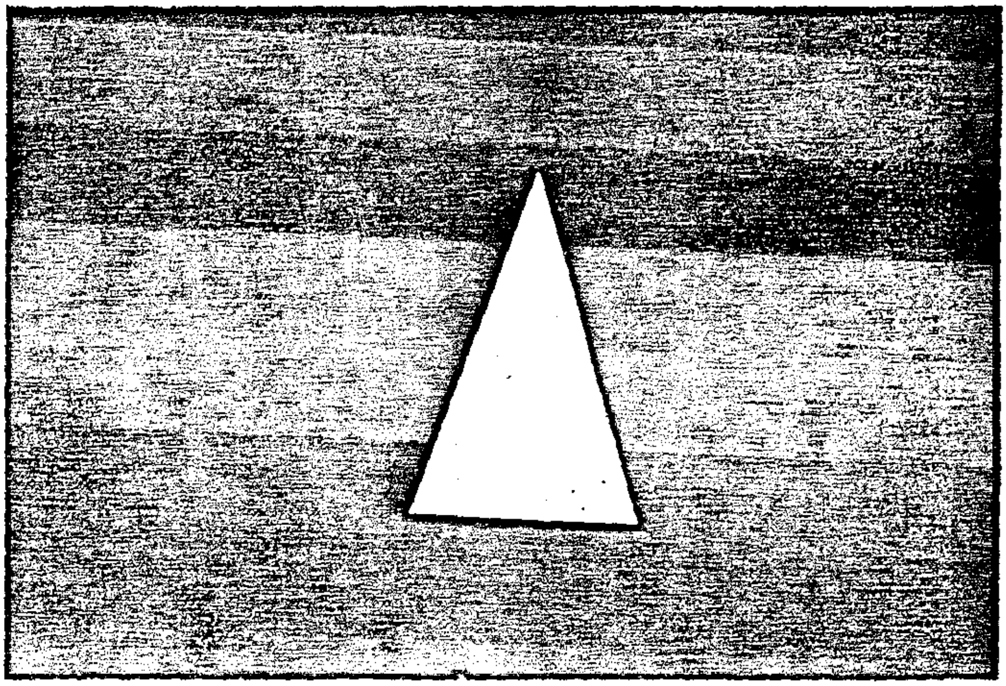


FIGURE 6

```
conn(a,ab,b).
line(ab,1Ø6).
slope(ab,76).
sqrline(ab,11236).

conn(b,bc,c).
line(bc,62).
slope(bc,6).
sqrline(bc,3844).

conn(c,ca,a).
line(ca,1Ø2).
slope(ca,11Ø).
sqrline(ca,104Ø4).

angle(a,34).
angle(b,7Ø).
angle(c,76).

shape 3D(a,X).
no.
shape(A,X).
** (top) PROVED : shape(a,triangle) ;
no.
```

The lengths of ab and ca are not quite equal and thus it is not an isosceles-triangle.

c) *a non-convex-quadrilateral:*- Fig.7 shows the actual picture of the shape and the answers of the recognizer are given below:-

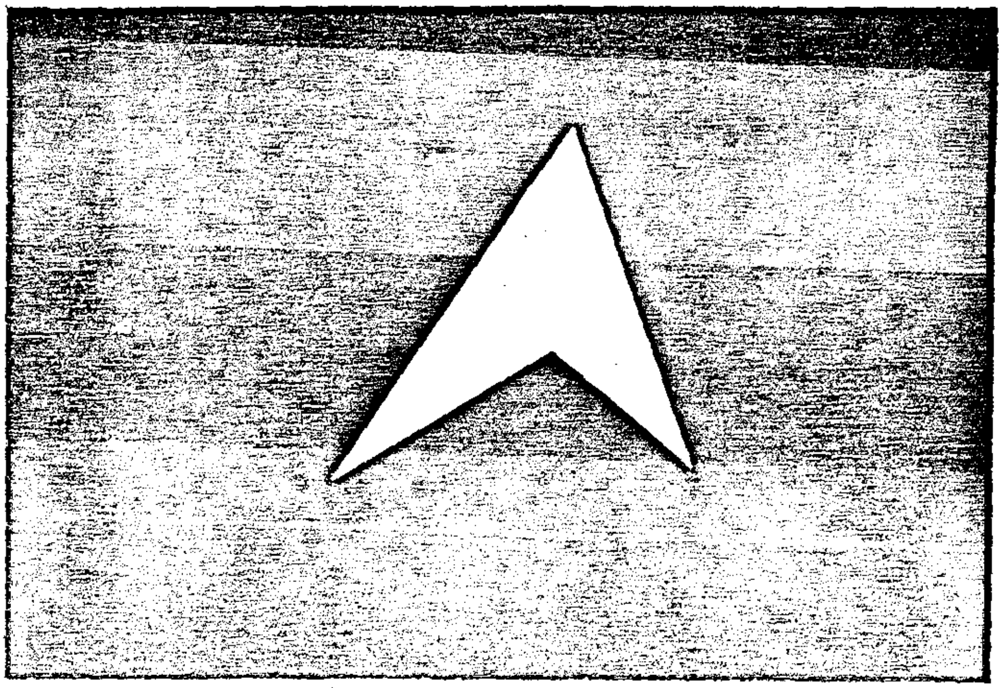


FIGURE 7

```
conn(a,ab,b).
line(ab,1Ø3).
slope(ab,71).
sqrline(ab,1Ø6Ø9).

conn(b,bc,c).
line(bc,49).
slope(bc,43).
sqrline(bc,24Ø1).

conn(c,cd,d).
line(cd,67).
slope(cd,148).
sqrline(cd,4489).

conn(d,da,a).
line(da,116).
slope(da,12Ø).
sqrline(da,13456).

angle(a,49).
angle(b,28).
angle(c,1Ø5).
angle(d,28).

shape 3D(a,X).
no.
shape(A,X).
** (top) PROVED : shape(a,quadrilateral) ;
** (top) PROVED : shape(a,non-convex-quadrilateral) ;
no.
```

Here the fact that the sum of its four angles is less than 360 classifies it as *non-convex*.

d) *a tetrahedron:-* The picture of the object is shown in Fig. 8. First the left-hand side is preserved and the other is masked to background (Fig. 9a). Then the right-hand side is processed (Fig. 9b). Finally the unit clauses obtained and the answers of the recognizer are given below:-

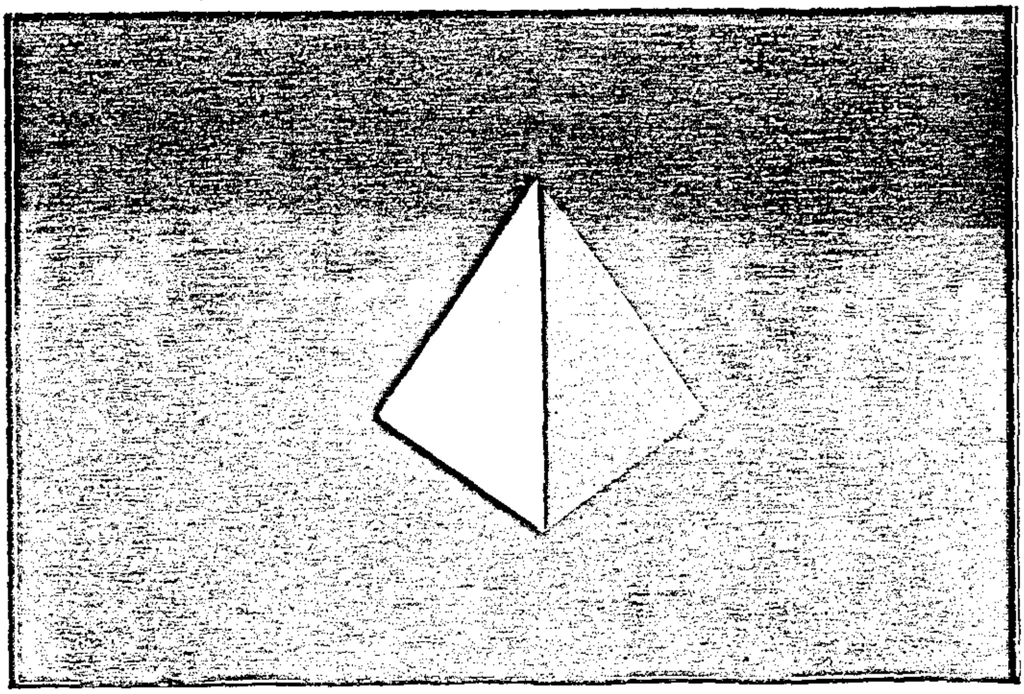
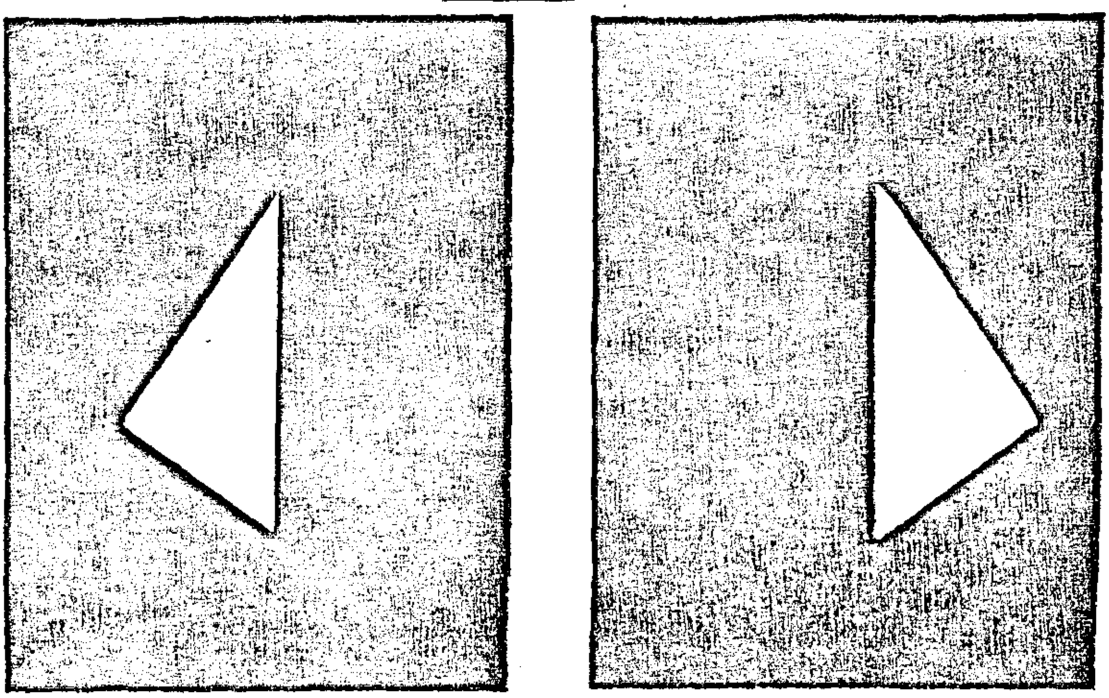


FIGURE 8



(a) (b)

FIGURE 9

```
conn((a,ab,b).
line(ab,8Ø).
slope(ab,89).
sqrline(ab,64ØØ).

conn(b,bc,c).
line(bc,51).
slope(bc,22).
sqrline(bc,26Ø1).

conn(c,ca,a).
line(ca,76).
slope(ca,126).
sqrline(ca,5776).

angle(a,37).
angle(b,67).
angle(c,76).

conn(a,ae,e).
line(ae,61).
slope(ae,55).
sqrline(ae,2721).

conn(e,eb,b).
line(eb,45).
slope(eb,138).
sqrline(eb,2Ø25).

conn(b,ba,a).
line(ba,8Ø).
slope(ba,89).
sqrline(ba,64ØØ).

angle(a,34).
angle(e,97).
angle(b,49).

shape 3D(a,X).
** (top) PROVED : shape 3D(a,tetrahedron) ;
** (top) PROVED : shape 3D(a,other) ;
no.
```

e) *a triangular-prism:-* The object is shown in Fig. 10 and the answers of the recognizer are as follows:-

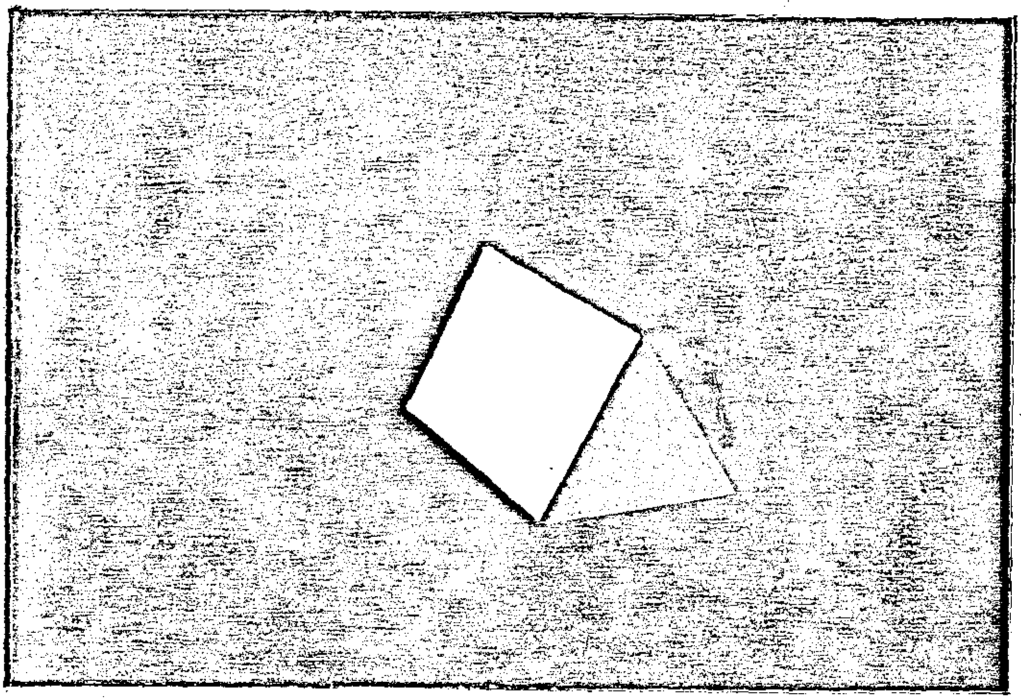


FIGURE 10

```
conn(a,ab,b).              conn(b,bf,f).
line(ab,52).               line(bf,52).
slope(ab,25).              slope(bf,64).
sqrline(ab,27ø4).          sqrline(bf,27ø4).

conn(b,bc,c).              conn(f,fc,c).
line(bc,58).               line(fc,5ø).
slope(bc,118).             slope(fc,175).
sqrline(bc,3364).          sqrline(fc,25øø).

conn(c,cd,d).              conn(c,cb,b).
line(cd,5ø).               line(cb,58).
slope(cd,42).              slope(cb,118).
sqrline(cd,25øø).          sqrline(cb,3364).

conn(d,da,a).              angle(b,54).
line(da,43).               angle(f,69).
slope(da,113).             angle(c,57).

angle(a,88).
angle(b,87).
angle(c,76).
angle(d,1ø9).

shape 3D(a,X).
** (top) PROVED : shape 3D(a,square-pyramid) ;
** (top) PROVED : shape 3D(a,truncated-triangular-pyramid) ;
** (top) PROVED : shape 3D(a,other) ;
```

Because of the perception, parallelogram (abcd) is not represented as such on the 2-D picture and thus the object is not classified as a *triangular-prism*.

d) *a truncated-square-pyramid:-* Fig. 11 shows the actual object and the answers of the recognizer are given below.

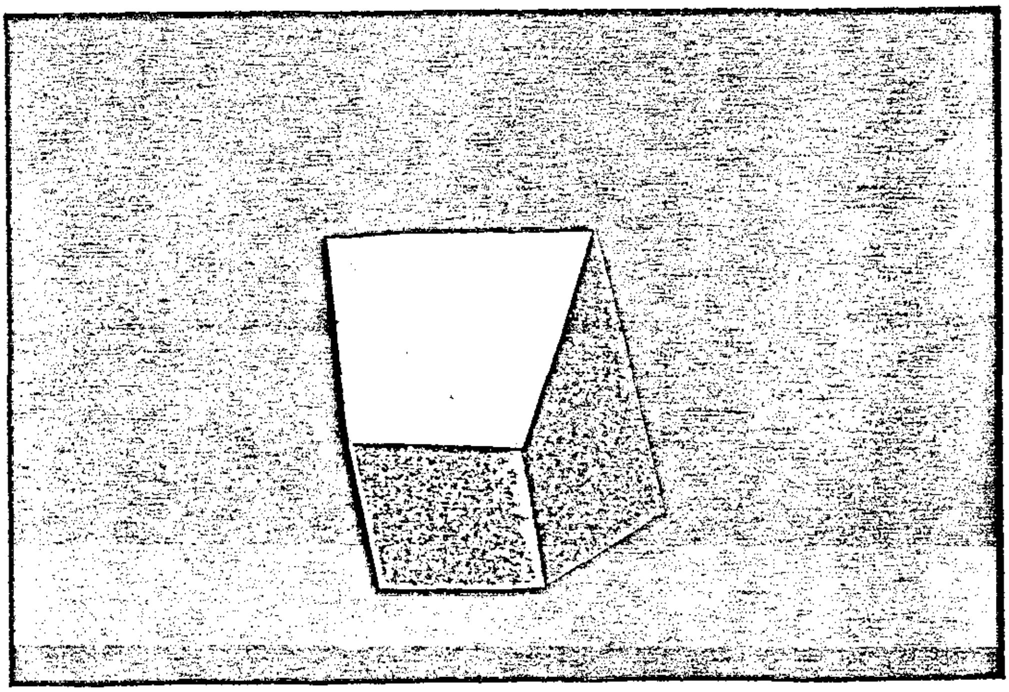


FIGURE 12

```
conn(a,ab,b).          conn(d,dc,d).          conn(b,bf,f).
line(ab,90).           line(dc,50).           line(bf,91).
slope(ab,0).           slope(dc,0).           slope(bf,83).
sqrline(ab,6400).      sqrline(dc,2500).      sqrline(bf,8281).

conn(b,bc,c).          conn(c,cg,g).          conn(j,jg,g).
line(bc,67).           line(cg,42).           line(jg,32).
slope(bc,117).         slope(cg,75).          slope(jg,160).
sqrline(bc,4489).      sqrline(cg,1764).      sqr(jg,1024).

conn(c,cd,d).          conn(g,gh,h).          conn(g,gc,c).
line(cd,50).           line(gh,50).           line(gc,42).
slope(cd,0).           slope(gh,0).           slope(gc,75).
sqrline(cd,2500).      sqrline(gh,2500).      sqrline(gc,1764).

conn(d,da,a).          conn(h,hd,d).          conn(c,cb,b).
line(da,60).           line(hd,42).           line(cb,67).
slope(da,90).          slope(hd,75).          slope(cb,117).
sqrline(da,3600).      sqrline(hd,1764).      sqrline(cb,4489).

angle(a,90).           angle(c,75).           angle(b,34).
angle(b,63).           angle(g,105).          angle(j,103).
angle(c,117).          angle(h,75).           angle(g,85).
angle(d,90).           angle(d,105).          angle(c,138).

shape 3D(a,X).
** (top) PROVED : shape 3D(a,truncated-square-pyramid) ;
** (top) PROVED : shape 3D(a,other) ;
no.
```

From various attempts to recognize 2-D and 3-D shapes the following general conclusions can be drawn:-

a) The 2-D shapes are easier to recognize (because of the high contrast between the main figure and the background) and unless there is slight distortion in the digitized picture, the results are perfect (within the limits of the 2-D recognizer).

b) The 3-D objects are harder to recognize if the lighting conditions are not absolutely right. A common feature of the digitized picture is that the same side may be represented by more than two gray colours, which means that it is taken as two different sides (because sides with gray levels differing by two are considered as different).

c) Objects with fewer sides are easier to recognize than those with more sides. For example a *tetra* with 2 sides is easier to be recognized than a *pyram* (3 sides) or a *pyram1* (4 sides).

SUMMARY - CONCLUSIONS

- The wholeprocess consists of the following procedures:-
  a) Manual set up of the scene and adjustment of hardware.
  b) Programs in Assembly run on the micro, including selection of significant gray levels, isolation of faces, averaging, edging, boundary following and vertex detection.
  c) Programs in C run on the VAX, including verification of real vertices and formation of unit clauses.
  d) Programs in PROLOG run on the VAX, including the 2-D and 3-D recognizers.
- Some suggestions for further development corresponding to the four phases above are:-
  a) Introduction of a *mechanican arm* with *pan* and *tilt* facilities, larger frame, reception of the processed image on the T.V. monitor and modification of the printer's character EPROM.
  b) Spatial differentiation of the picture.
  c) A line drawing routine.
  d) Additional clauses for further classification.
- Compared with PLEX grammars the 3-D recognizer merits in the fact that it uses the same primitives for both 3-D and 2-D shapes.

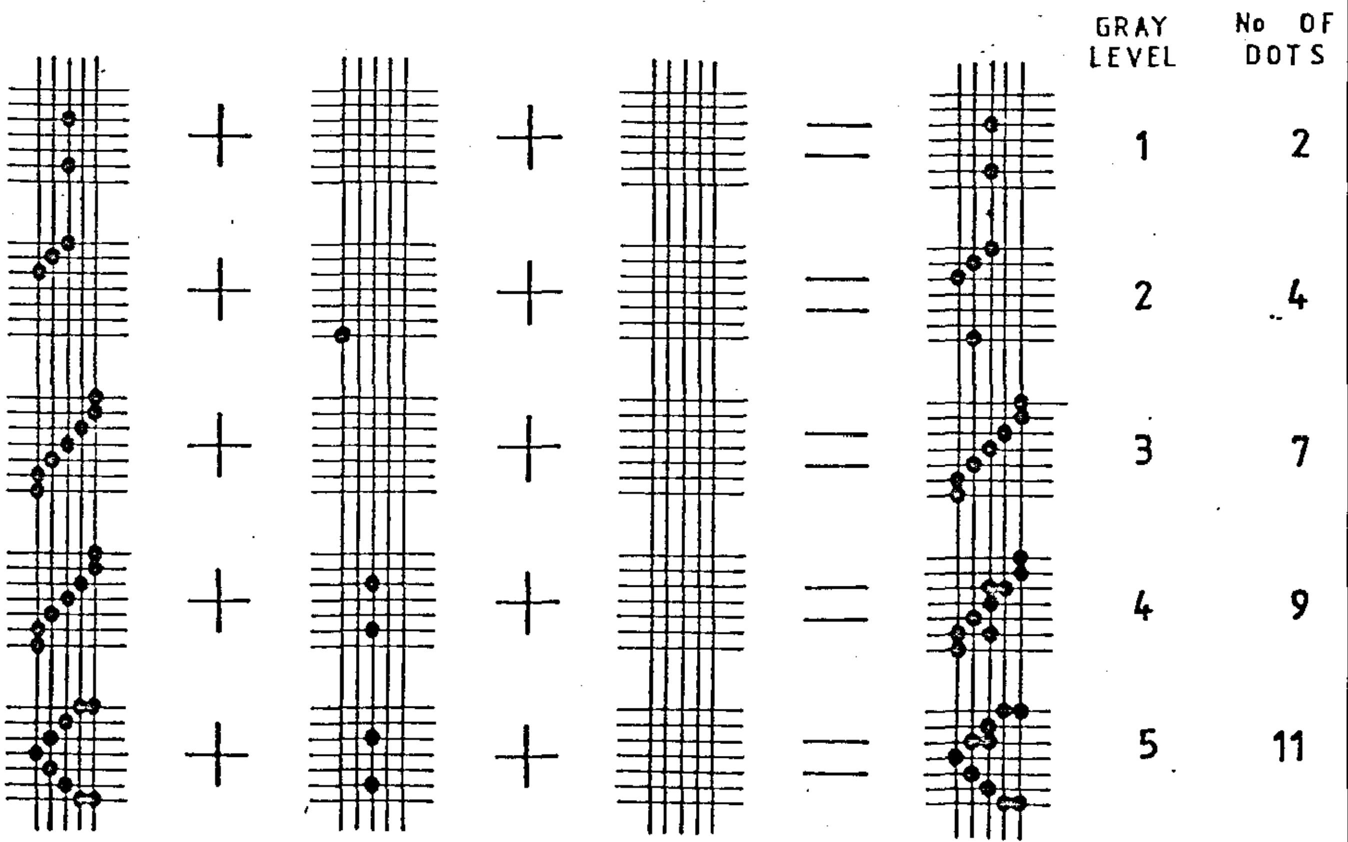
Appendix 1

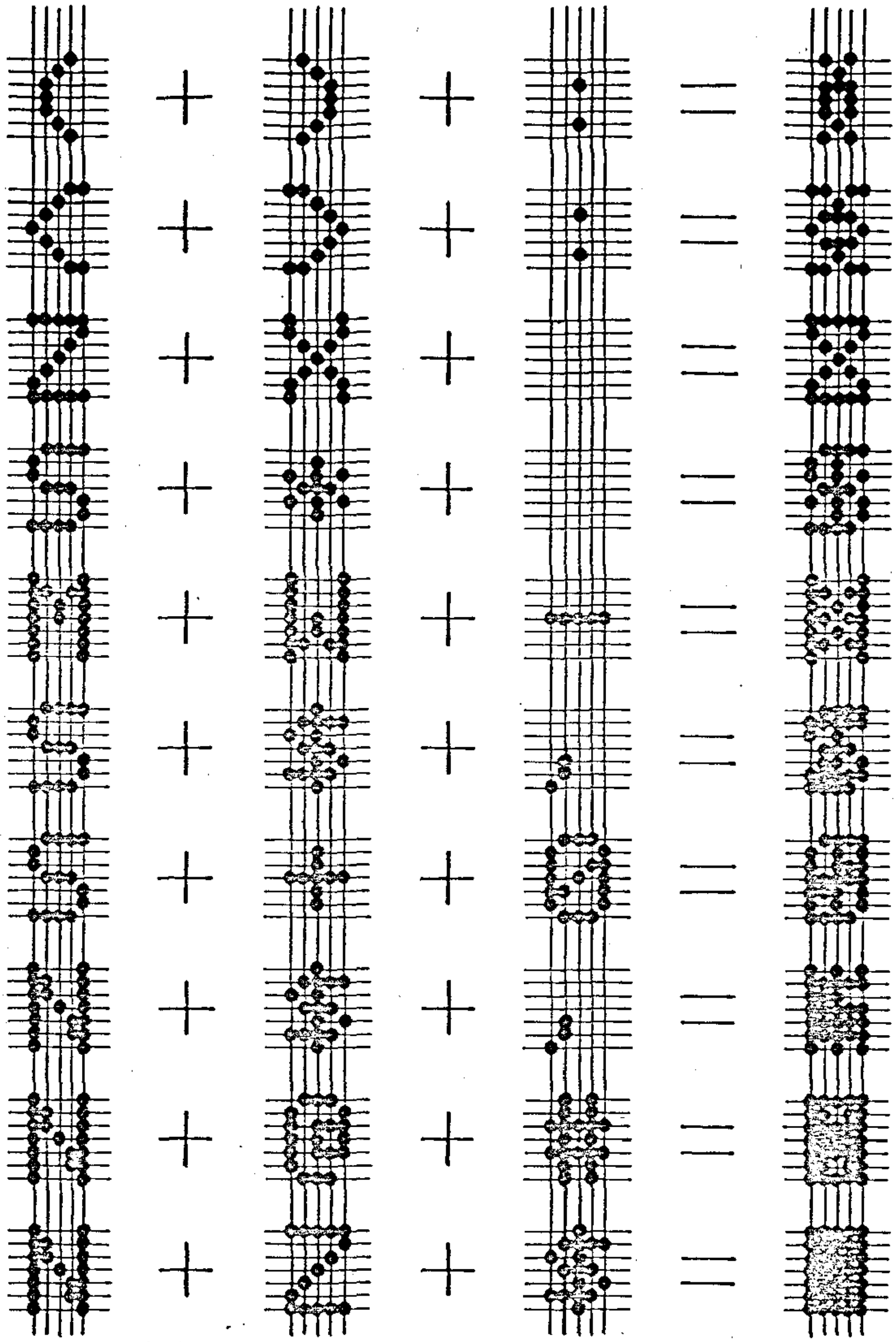
CHARACTER SELECTION FOR IMAGE PRINTING

The visual representation of the 16 gray levels of each pixel-value, is obtained by three successive overprintings of characters taken from the printer's character set. TREND's printing head uses a 5×7 dot matrix, and its character set consists of 64 elements, including *space*. These are 1Ø numerals, 26 letters and 28 symbols. The number of overprintings arises from the fact that 3 is the minimum number of the existing characters that are necessary to be printed one on top of the other in order to obtain the last gray level, which is *black*. This corresponds to a matrix full of dots. Conventionally the first gray level is *white* for space or blank and takes value Ø. The above makes obvious the way the other intermediate levels are obtained. The principle is to fill the grid with enough dots to give the human eye the right impression, according to the gray level that they represent. Another constraint is that the number of dots has to be between Ø and 35 (=5×7):

$$\emptyset \leq \text{no of dots} \leq 35$$

The list of 64 alphanumericals and symbols is given in Fig. A1.2. A first examination shows that some characters are more suitable than others due to their properties of symmetry. For example '+' has four axes and a centre of symmetry while 'F' has none. An obvious method to set up the list would be to find every new combination by adding a constant *step* to the no. of dots which represents the previous one. Since $35 \div 15 = 2.\dot{3}$ the step is determined to be between 2 and 3. This is more or less followed although some times step 1 or 4 is used instead. This is due to the fact that combinations with dots more symmetrically and equally spaced are preferred to others with the same number of dots but less symmetric. For example, for no. 6, represented by 14 dots a combination of '(', ')' and ':' was preferred to say 'K' and two spaces. Finally in very close cases a personal decision was made by placing stripes of different combinations one next to the other. The combination that gave the best impression to a smooth transition from one level to another was preferred. Fig. A1.1 illustrates the 16 combinations.





GRAY LEVEL	No OF DOTS
6	14
7	18
8	19
9	21
10	23
11	24
12	27
13	30
14	33
15	35

fig. A 1.1

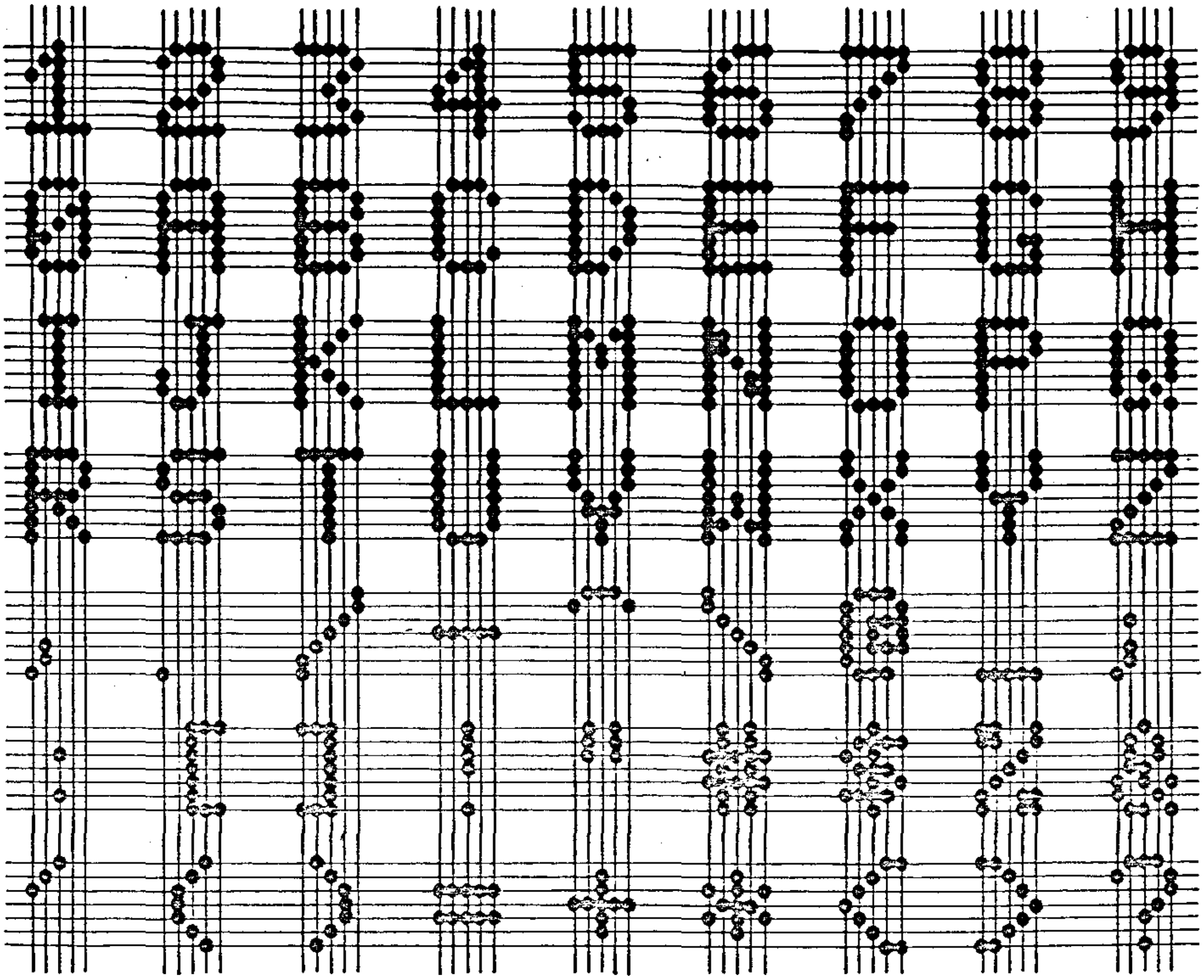


fig. A1.2

Appendix 2

PROGRAMS IN 'ASSEMBLY LANGUAGE' (Z-8∅)

These consist of the following programs:-

MAIN:- This program calls 4 subroutines to perform the preprocessing operations on the digitized picture.

COLL:- This transfers the picture from the framestore of the *ROBOT* to the memory locations 4∅∅∅-7FFF* of the microcomputer system. It first initializes the serial interface by loading its status word with 43 (master reset) and 52 (clockrate divided by 64) and prepares the micro to receive start receiving data by setting the data word (FF∅B) to 1. Then subroutine *INPUT* is called to start inputting data into the micro. The data are not stored at consecutive locations because of the difference in rates discussed in 2.2.2. Thus the first pixel is stored in 4∅∅1 the next in 4∅∅3 etc. until the end of the frame is met (location 8∅∅∅). Then the next pixel

---

*All the addresses and data values mentioned in this chapter are in the hexadecimal numerical system.

goes to 4ØØ2, the other to 4ØØ5 and so on. When the whole frame has been transferred the procedure stops.

INPUT:- This is the subroutine, which loads the pixels into the micro. It first tests the last bit of status word FFØA to see if it is set and if yes, it loads accumulator A with a byte from the data word FFØB.

RCTF:- This subroutine corrects a hardware fault in the *ROBOT*, which reverses the pixel-values from Ø8 to ØF, i.e. gives pixel-value Ø8 for black and vice versa.

PRGM:- This subroutine calls all the procedures responsible for the preprocessing and boundary following, which will be discussed later.

TRND:- This subroutine prints the equivalent of the digitized picture on the TREND printer. First it forms a set-up table with the limit values of the picture to be printed at locations 4Ø and 42 and the base address of the character-set (representing the 16 pixel values) at location 43. Then it calls *PICT* to print the picture. Since this is done in two parts, bottom and top respectively, a new look-up table with limit values is formed. *CRLF* subroutine leaves two blank lines between the two parts.

PICT:- This subroutine performs the three over-printings that form the final visual representation of the digitized picture on a piece of paper. Basically it compares the pixel-value of each pixel with a number between Ø and ØF and calls *OUTP* to print out the appropriate character from the special set stored at locations 8BØØ-8B2F. Before the second and third pass, an offset of 1Ø is added to the location of the first pass so that each time a new character (actually the second and third of the set for this level) is printed. After the third pass, a new line starts and the procedure continues until the whole picture is printed out. The procedure can be modified to print the average of two lines in order to give a more or less square picture (because the height of the character $\approx 2 \times$ its width).

OUTP:- This subroutine outputs one character on the printer everytime it is called. It first tests the first bit of the serial interface status byte if it is set, i.e. a character can be output. If the character is different to control S (13 ASCII) then it outputs it via register B.

A detailed form of the above is given in Listing A.

At this point it should be mentioned that subroutine *PRGM* is chosen in such a way that can be supplied with additional calls of more than one subroutine. At present it calls only *PROG*, which performs all the pre-processing operations on the picture. The subroutines called by *PROC* follow a general form, which is:- At first they are divided into *major* or *primary* and *minor* or *secondary* subroutines. The primary subroutines are longer and call a number of secondary ones to perform their tasks. The latter are generally shorter and can be divided into two categories too. In the first belong those that perform minor operations such as comparisons, additions etc. and they are used to make the primary subroutines easier to follow. In the second belong subroutines that are crucial because they perform major operations such as detection of vertices, determination of coordinates etc. In the following the function of the primary subroutines will be discussed with emphasis only on the important secondary subroutines. It must be made clear that due to four character labels, the same labels have been used more than once. In case of confusing labels the real address can be used as a lead to the right subroutine. In the meanwhile care has been taken that all the subroutines called by the same primary subroutine are grouped together. Finally, apart from the labels of main subroutines *AVRG* and *EDGE*, labels within the range 2ØØØ-2FFF and 3ØØØ-3FFF are irrelevant.

PROC:- This subroutine first calls *WORD* which stores the message *'SEND'* in locations 5∅-53. Then it calls *PRCG* which finds the sums of pixel values different than the background (∅F) and stores them in locations 3∅∅-31D. These locations have been previously zeroized by calling *ZERO* from *WORD*. Next, *SUM* is called to form the sum of all non-background pixels and store it in location 32∅. Then subroutine *SLCT* is called.

SLCT:- This divides the sum in 32∅ by 16, i.e. finds the 6% of the main object pixels and stores it at 32∅. Then it checks every partial sum of the pixel-values against the value in 32∅ *(TEST)* to find out how many of them exceed the 6% of the main object pixels. The ones that satisfy this and differ by two at least are stored in locations 16∅ onwards. The end marker FF is placed at the end of this list.

Then *PROC* calls *MOVE* to save the frame in locations A∅∅∅ -DFFF. It looks for the first pixel-value at 16∅ saves it at 15∅ and if it is not FF - in which case it stops by giving a message *'END'* on the screen *(OUTB)* - it calls the following subroutines:-

ISLT:- This scans the frame and turns to ∅F (black) every pixel with values different than the current value at 15∅ or that plus one. The result is to isolate an area of cells with pixel-values the same or one apart.

LOAD:- This reloads the frame into 4∅∅∅-7FFF and calls *DATA* to set a look-up table of values used by subroutine *EDGE*. These are loaded into locations 1∅∅-14F. Listing B presents all the subroutines discussed above.

AVRG:- This subroutine performs the averaging operation. It calls *INTL* to set the starting address of the operation (in 2∅∅) the starting value of the index register IY (in 2∅2) the final value of IY (in 2∅4) the step for the first address in a new row (in 2∅6) and the step for the last address in a new row (in 2∅8). Finally it calls *OLD* which initializes *EDGE* (will be discussed later) and zeroizes *IY*. IY is used as a row index of the two dimensional array. Then *FRST* sets the top left pixel of the 3×3 window (in 21Ø), the test value for the top right pixel of the window (in 212), the test value for moving the window one row down (in 218), and the test value for the bottom right pixel of the window (in 21A). The above mentioned locations are used to keep the new values of the window as it scans the picture. Then *UPDA* loads register HL (2 bytes) with the current test top right pixel address index register IX with the top left pixel address and IY with the new row value, before the window moves another row down. A little loop finds the sum of the 9 pixels of the window *(SUM)*. Subroutines *LOOP* and *STEP* check the limit values of the window against the test values and *DO* makes the appropriate settings to move to the next row (within the window). Then *MAIN* is called to do the main task. It compares the above found sum with a value decided by the user (here 4B). If the sum is greater than it, ØF is added to the current pixel-value, if it is less then the same quantity is subtracted from the pixel-value. Finally in case of equality the current pixel-value remains unchanged. It can be seen that this routine works in both 2 gray level and 16 gray level occasions, because in the second case the intensification factor can be changed accordingly. Subroutine *SHFL* shifts the pixel-value four places to the left for reasons mentioned in 2.3.3. Subroutine *ONE* checks if the end of a row is reached and if not *ROW* does the appropriate settings to move to the next pixel across. Subroutine *TWO* checks if the end of the frame is reached and if not *CLMN* does the appropriate settings to move the window to the next row. Finally when the end condition is reached subroutine *SHFT* shifts the pixel-value back, four places to the right (2.3.3).

EDGE:- In this a 3×3 window is used like in *AVRG* and thus *INTL* is the same as before. This procedure performs 8 tests in order to find the boundary pixels, thus 8 sets of 5 key values are loaded into locations 1ØØ-14F (see *LOAD* of listing B). These 5 key values are:- the first pixel of the window, the test value to the last of the 3 pixels in each test, the pixel opposite to the 3 in the test, the step (if necessary) to find the last of the 3 pixels, and finally the test value for the starting of a new row. *INIT* saves the address of the first key value (1ØØ) at 21A and 21E, the operand of the first instruction of *STRT* at 21C and the address that contains the address of the first key value at 22Ø. Finally zeroizes pointer 24Ø. *NEW* loads the first 5 addresses of the first set of key value into 21Ø-218, to be used as indirect addressings to the first set of key values *(STRT)*. *ZERO* looks for the first Ø pixel value, masks it *(MASK)* and performs the first test *1,2,3:6* (numbering of pixels in 3×3 window as in Fig. 2.17) via *PIX* (which finds the minimum distance between the pixel values of each of 1,2 and 3 and 6 respectively) and *LOPA* (which determines the sequence of the cells for the first test). *THLD* is used only in the case of 16 pixel value edging. *PNTR* counts the successful tests and in case of two of them the window is moved for another try. *MODF* modifies the pixel, if needed, and *TWIC* does the second test *7,6,5:2*. The next two tests are made by substituting *LOPB* in place of *LOPA* and changing the *JP, IB* by *JP, IIA* respectively. These are *1,8,7:4* and *3,4,5:8*. The rest are combinations of *LOPA* and *LOPC (1,2,8,:5* and *2,3,4:7)* and *LOPA* and *LOPC (8,7,6:3* and *4,5,6:1)* respectively. *TSTA* checks if the end of a row is reached. *WNDA* shifts the window one cell across and *WNDB* a row below. *REPT* uses a number of known subroutines and *CHNG* in order to update the table of key values in the case that not all the tests need to be performed. *TEST* checks if the whole of the tests has been performed. *OLDA* and *OLDB* use *OLD* in order to put *EDGE* back into its original form before a new column, or row starts respectively. *LAST* increases the row index IY by 1 and, *SHFT* is the same as in *AVRG*.

FOLW:- This subroutine follows the boundary of the shape and saves the coordinates of the detected vertices. First it calls *INT* for the usual initializations. The address of the first pixel to be examined is stored at 2ØØ, the address of the last pixel in a row is stored at 2Ø2, the step for the first pixel of a new row is stored at 2Ø4, the last value of the row index IY goes to 2Ø6, the step for the last pixel in a row is kept in 2ØA, and the test address for the end of the procedure is stored at 212. Finally the initial addresses for the vertex array (4ØØ) and the pointer to real-vertex array (48Ø) are kept in 322 and 324 respectively. The first boundary pixel met is marked by placing 11 in it and *NAI* calls *CORD* to find its coordinates and *ADRS* to place a pointer on it. *ZERO* zeroizes addresses 3ØØ-32Ø which will be used by the next subroutines. *CYCL* follows the boundary and will be examined in detail later on. Subroutine *FF* places an end marker FF at the end of the two arrays mentioned above and calls to printout an *'END'* message. *AAA* calls *UNIC* which turns to ØF every marked boundary cell and calls *FOLW* again looking for a new boundary. *TESA* and *TESB* look for the end of a row and the end of the procedure respectively. *WINA* moves the search a pixel across and *WINB* one row below. The procedure ends if no boundary pixel is met.

CYCL:- First of all, locations 3ØØ-31Ø are dedicated to:- 3ØØ: direction of unit (N_U), 3Ø1: length of the unit (L_U), 3Ø2: direction of link (N_L), 3Ø3: mean of length ($\overline{L_U}$), 3Ø5: a flag used in *LINK* (indicates whether the last vertex was real or not), 3Ø8: direction of last vertex, 3ØA: address of previous vertex, 31Ø: address of current vertex. *CYCL* performs the circular search of 3.2.3b. First looks for a boundary cell *(BLAC)*. *RA* to *RH* perform the circular displacement of the pixel. *CYCL* is also called by *DOES* and *DONT* for a look ahead search. In this case a flag is set (reg.C) to indicate a simple exit *(CMPR)*. If the very first vertex is met *(MET)* and the flag is not set the cycle ends, marking the last pixel as a vertex *(CHCK)*. If no boundary value is met after a complete cycle, a recovery operation *(RCVR)* is called. The latter works as in 3.2.3b. In normal cases (flag=∅), vertex detector is called *(YES)*. The main subroutine *VXDT* is illustrated in Fig. 3.23 together with *LINK*, *DONT* and *DOES*. *SAVE* saves the current N_U in 3∅A, and calls *LNTH*. The latter calls *CHSL*, which compares the new L_U with $\overline{L_U}$ and if $|L_U - \overline{L_U}| > \overline{L}_U/4$ the end of the last unit is marked as a vertex *(PREV)*. *AJST* subtracts 8 from values greater than 8 and adds 9 to negative values of N_U so that the cycle is kept. *VRTX* marks a pseudo-vertex by adding 1 to the previous one (stored in 35∅) and saves its coordinates *(CORD)*. *CRTV* marks a certain vertex by adding 11 to the previous one, and saves both *(BOTH)* its coordinates and a pointer to it. Finally *PRVC* marks the previous pseudo-vertex as a certain-vertex when two successive changes of the link occur. Subroutine *FAIL* prints out a message *'FAIL'* when the recovery operator fails to find the best boundary cell.

Listing C contains all the above mentioned subroutines.

LISTING A

```
8800,880C$K
8800  CD  00  89      MAIN:  CALL COLL
8803  CD  00  29             CALL RCTF
8806  CD  10  00             CALL PRGM
8809  CD  40  88             CALL TRND
880C  FF                     RST  38   *
8900,894D$K
8900  11  00  40      COLL:  LD   DE 4000
8903  21  0A  FF             LD   HL FF0A
8906  36  43                 LD   (HL) 43
8908  36  51                 LD   (HL) 51
890A  23                     INC  HL
890B  36  00                 LD   (HL) 00
890D  CD  A0  88      COLA:  CALL INPT
8910  4F                     LD   C A
8911  E6  10                 AND  10
8913  CA  0D  89             JP   Z COLA
8916  C3  2D  89             JP   COLB
8919  CD  A0  88      COLD:  CALL INPT
891C  4F                     LD   C A
891D  E6  F0                 AND  F0
891F  CA  2D  89             JP   Z COLB
8922  E6  10                 AND  10
8924  C2  3E  89             JP   NZ COLC
8927  7B                     LD   A E
8928  E6  7F                 AND  7F
892A  C2  19  89             JP   NZ COLD
892D  79              COLB:  LD   A C
892E  12                     LD   (DE) A
892F  13                     INC  DE
8930  13                     INC  DE
8931  13                     INC  DE
8932  7A                     LD   A D
8933  FE  80                 CP   80
8935  C2  3A  89             JP   NZ COLE
8938  3E  40                 LD   A 40
893A  57              COLE:  LD   D A
893B  C3  19  89             JP   COLD
893E  7B              COLC:  LD   A E
893F  3C                     INC  A
8940  3D                     DEC  A
8941  C2  47  89             JP   NZ COLF
8944  7A                     LD   A D
8945  FE  40                 CP   40
8947  11  00  40      COLF:  LD   DE 4000
894A  C2  2D  89             JP   NZ COLB
894D  C9                     RET   *
88A0,88AA$K
88A0  21  0A  FF      INPT:  LD   HL FF0A
88A3  7E                     LD   A (HL)
88A4  1F                     RRA
88A5  D2  A3  88             JP   NC 88A3
88A8  23                     INC  HL
88A9  7E                     LD   A (HL)
88AA  C9                     RET   *
2900,2982$K
2900  11  FF  7F      RCTF:  LD   DE 7FFF
2903  21  00  40             LD   HL 4000
2906  7E              RLOP:  LD   A (HL)
2907  E6  0F                 AND  0F
2909  FE  08                 CP   08
290B  FA  71  29             JP   M RTST
290E  C2  19  29             JP   NZ RR
2911  E6  F0                 AND  F0
2913  C6  0F                 ADD  A 0F
```

```
2915  77                   LD   (HL) A
2916  C3  71  29           JP   RTST
2919  FE  09         RA  : CP   09
291B  C2  26  29           JP   NZ RB
291E  E6  F0               AND  F0
2920  C6  0E               ADD  A 0E
2922  77                   LD   (HL) A
2923  C3  71  29           JP   RTST
2926  FE  0A         RB  : CP   0A
2928  C2  33  29           JP   NZ RC
292B  E6  0F               AND  0F
292D  C6  0D               ADD  A 0D
292F  77                   LD   (HL) A
2930  C3  71  29           JP   RTST
2933  FE  0B         RC  : CP   0B
2935  C2  40  29           JP   NZ RD
2938  E6  F0               AND  F0
293A  C6  0C               ADD  A 0C
293C  77                   LD   (HL) A
293D  C3  71  29           JP   RTST
2940  FE  0C         RD  : CP   0C
2942  C2  4D  29           JP   NZ RE
2945  E6  F0               AND  F0
2947  C6  0B               ADD  A 0B
2949  77                   LD   (HL) A
294A  C3  71  29           JP   RTST
294D  FE  0D         RE  : CP   0D
294F  C2  5A  29           JP   NZ RF
2952  E6  F0               AND  F0
2954  C6  0A               ADD  A 0A
2956  77                   LD   (HL) A
2957  C3  71  29           JP   RTST
295A  FE  0E         RF  : CP   0E
295C  C2  67  29           JP   NZ RG
295F  E6  F0               AND  F0
2961  C6  09               ADD  A 09
2963  77                   LD   (HL) A
2964  C3  71  29           JP   RTST
2967  FE  0F         RG  : CP   0F
2969  C2  71  29           JP   NZ RTST
296C  E6  F0               AND  F0
296E  C6  08               ADD  A 08
2970  77                   LD   (HL) A
2971  22  0C  02     RTST: LD   (020C) HL
2974  C6  00               ADD  A 00
2976  ED  52               SBC  HL DE
2978  CA  82  29           JP   Z END
297B  2A  0C  02           LD   HL (020C)
297E  23                   INC  HL
297F  C3  06  29           JP   RLOP
2982  C9             END : RET   *
8840,8862*K
8840  21  40  00     TRND: LD   HL 0040
8843  36  7F               LD   (HL) 7F
8845  23                   INC  HL
8846  36  5F               LD   (HL) 5F
8848  23                   INC  HL
8849  36  20               LD   (HL) 20
884B  23                   INC  HL
884C  36  00               LD   (HL) 00
884E  CD  06  8A           CALL PICT
8851  21  40  00           LD   HL 0040
8854  36  5F               LD   (HL) 5F
8856  23                   INC  HL
8857  36  3F               LD   (HL) 3F
```

```
8859  CD  D0  88          CALL CRLF
885C  CD  D0  88          CALL CRLF
885F  CD  00  8A          CALL PICT
8862  C9                  RET   *
8A00,8A72$K
8A00  CD  D0  88   PICT:  CALL CRLF
8A03  1E  80              LD   E 80
8A05  3A  40  00          LD   A (0040)
8A08  57                  LD   D A
8A09  0E  00       PASS:  LD   C 00
8A0B  1A           PIXL:  LD   A (DE)
8A0C  E6  0F              AND  0F
8A0E  00                  NOP
8A0F  00                  NOP
8A10  00                  NOP
8A11  00                  NOP
8A12  00                  NOP
8A13  00                  NOP
8A14  00                  NOP
8A15  00                  NOP
8A16  21  43  00          LD   HL 0043
8A19  46                  LD   B (HL)
8A1A  26  8B              LD   H 8B
8A1C  81                  ADD  A C
8A1D  B0                  OR   B
8A1E  6F                  LD   L A
8A1F  46                  LD   B (HL)
8A20  CD  B0  88          CALL OUTP
8A23  62                  LD   H D
8A24  6B                  LD   L E
8A25  16  FF              LD   D FF
8A27  1E  80              LD   E 80
8A29  19                  ADD  HL DE
8A2A  5D                  LD   E L
8A2B  54                  LD   D H
8A2C  7A                  LD   A D
8A2D  21  41  00          LD   HL 0041
8A30  BE                  CP   (HL)
8A31  C2  0B  8A          JP   NZ PIXL
8A34  7B                  LD   A E
8A35  E6  80              AND  80
8A37  CA  0B  8A          JP   Z PIXL
8A3A  06  8D              LD   B 8D
8A3C  CD  B0  88          CALL OUTP
8A3F  79                  LD   A C
8A40  21  42  00          LD   HL 0042
8A43  BE                  CP   (HL)
8A44  CA  55  8A          JP   Z LINE
8A47  C6  10              ADD  A 10
8A49  4F                  LD   C A
8A4A  3A  40  00          LD   A (0040)
8A4D  57                  LD   D A
8A4E  7B                  LD   A E
8A4F  F6  80              OR   80
8A51  5F                  LD   E A
8A52  C3  0B  8A          JP   PIXL
8A55  06  0A       LINE:  LD   B 0A
8A57  CD  B0  88          CALL OUTP
8A5A  7B                  LD   A E
8A5B  FE  FF              CP   FF
8A5D  CA  6F  8A          JP   Z FIN
8A60  3A  40  00          LD   A (0040)
8A63  57                  LD   D A
8A64  7B                  LD   A E
8A65  E6  7F              AND  FF
```

```
8A67   3C                        INC   A
8A68   00                        NOP
8A69   F6   80                   OR    80
8A6B   5F                        LD    E A
8A6C   C3   09   8A              JP    PASS
8A6F   CD   D0   88      FIN :   CALL  CRLF
8A72   C9                        RET    *
88B0, 88CA$K
88B0   21   02   FF      OUTP:   LD    HL FF02
88B3   7E                OULP:   LD    A (HL)
88B4   1F                        RRA
88B5   D2   C2   88              JP    NC OUTB
88B8   23                        INC   HL
88B9   7E                OUTA:   LD    A (HL)
88BA   E6   7F                   AND   7F
88BC   FE   13                   CP    13
88BE   CA   B9   88              JP    Z OUTA
88C1   2B                        DEC   HL
88C2   7E                OUTB:   LD    A (HL)
88C3   E6   02                   AND   02
88C5   CA   C2   88              JP    Z OUTB
88C8   23                        INC   HL
88C9   70                        LD    (HL) B
88CA   C9                        RET    *
88D0, 88DA$K
88D0   06   8D           CRLF:   LD    B 8D
88D2   CD   B0   88              CALL  OUTP
88D5   06   0A                   LD    B 0A
88D7   CD   B0   88              CALL  OUTP
88DA   C9                        RET    *
```

```
3600,363E#K
3600  CD  B8  35        PROC:  CALL WORD
3603  CD  A9  36               CALL PRCG
3606  CD  4E  36               CALL SUM
3609  CD  B0  37               CALL SLCT
360C  CD  F4  35               CALL MOVE
360F  21  60  01               LD   HL 0160
3612  22  FE  04        PA  :  LD   (04FE) HL
3615  7E                       LD   A (HL)
3616  32  50  01               LD   (0150) A
3619  FE  FF                   CP   FF
361B  CA  38  36               JP   Z END
361E  CD  3C  36               CALL LOAD
3621  3A  50  01               LD   A (0150)
3624  47                       LD   B A
3625  CD  7A  36               CALL ISLT
3628  CD  00  21               CALL AVRG
362B  CD  00  20               CALL EDGE
362E  CD  00  30               CALL FOLW
3631  2A  FE  04               LD   HL (04FE)
3634  23                       INC  HL
3635  CA  12  36               JP   Z PA
3638  CD  E9  35        END :  CALL OUTB
363B  FF                       RST  38   *
363C,364A#K
363C  21  00  A0        LOAD:  LD   HL A000
363F  11  00  40               LD   DE 4000
3642  01  00  40               LD   BC 4000
3645  ED  B0                   LDIR
3647  CD  BD  39               CALL DATA
364A  C9                       RET   *
364E,3679#K
364E  21  00  00        SUM :  LD   HL 0000
3651  11  00  03               LD   DE 0300
3654  1A                SU  :  LD   A (DE)
3655  4F                       LD   C A
3656  13                       INC  DE
3657  1A                       LD   A (DE)
3658  47                       LD   B A
3659  22  0C  02               LD   (020C) HL
365C  ED  53  0E  02           LD   (020E) DE
3660  21  1D  03               LD   HL 031D
3663  C6  00                   ADD  A 00
3665  ED  52                   SBC  HL DE
3667  2A  0C  02               LD   HL (020C)
366A  ED  5B  0E  02           LD   DE (020E)
366E  CA  76  36               JP   Z LST
3671  09                       ADD  HL BC
3672  13                       INC  DE
3673  C3  54  36               JP   SU
3676  22  20  03        LST :  LD   (0320) HL
3679  C9                       RET   *
367A,36A8#K
367A  21  80  40        ISLT:  LD   HL 4080
367D  11  7F  7F        ISA :  LD   DE 7F7F
3680  22  0C  02               LD   (020C) HL
3683  C6  00                   ADD  A 00
3685  ED  52                   SBC  HL DE
3687  CA  A8  36               JP   Z FNS
368A  2A  0C  02               LD   HL (020C)
368D  23                       INC  HL
368E  CD  5E  31               CALL BLAC
3691  CA  7D  36               JP   Z ISA
3694  B8                       CP   B
3695  CA  A3  36               JP   Z ISC
```

```
3698  04                       INC   B
3699  B8                       CP    B
369A  CA  A2  36               JP    Z ISB
369D  36  0F                   LD    (HL) 0F
369F  C3  DB  39               JP    ISD
36A2  05               ISB  :  DEC   B
36A3  36  00           ISC  :  LD    (HL) 00
36A5  C3  7D  36               JP    ISA
36A8  C9               FNS  :  RET   *
39DB,39DC$K
39DB  05               ISD  :  DEC   B
39DC  C3  7D  36               JP    ISA  *

          36A9,3798$K
36A9  21  00  40       PRGG:   LD    HL 4000
36AC  11  7F  7F               LD    DE 7F7F
36AF  22  0C  02       GA   :  LD    (020C) HL
36B2  C6  00                   ADD   A 00
36B4  ED  52                   SBC   HL DE
36B6  CA  98  37               JP    Z FIN
36B9  2A  0C  02               LD    HL (020C)
36BC  23                       INC   HL
36BD  CD  5B  31               CALL  BLAC
36C0  CA  AF  36               JP    Z GA
36C3  FE  00                   CP    00
36C5  C2  D1  36               JP    NZ GB
36C8  ED  4B  00  03           LD    BC (0300)
36CC  03                       INC   BC
36CD  ED  43  00  03           LD    (0300) BC
36D1  FE  01           GB   :  CP    01
36D3  C2  DF  36               JP    NZ GC
36D6  ED  4B  02  03           LD    BC (0302)
36DA  03                       INC   BC
36DB  ED  43  02  03           LD    (0302) BC
36DF  FE  02           GC   :  CP    02
36E1  C2  ED  36               JP    NZ GD
36E4  ED  4B  04  03           LD    BC (0304)
36E8  03                       INC   BC
36E9  ED  43  04  03           LD    (0304) BC
36ED  FE  03           GD   :  CP    03
36EF  C2  FB  36               JP    NZ GE
36F2  ED  4B  06  03           LD    BC (0306)
36F6  03                       INC   BC
36F7  ED  43  06  03           LD    (0306) BC
36FB  FE  04           GE   :  CP    04
36FD  C2  09  37               JP    NZ GF
3700  ED  4B  08  03           LD    BC (0308)
3704  03                       INC   BC
3705  ED  43  08  03           LD    (0305) BC
3709  FE  05           GF   :  CP    05
370B  C2  17  37               JP    NZ GG
370E  ED  4B  0A  03           LD    BC (030A)
3712  03                       INC   BC
3713  ED  43  0A  03           LD    (030A) BC
3717  FE  06           GG   :  CP    06
3719  C2  25  37               JP    NZ GH
371C  ED  4B  0C  03           LD    BC (030C)
3720  03                       INC   BC
3721  ED  43  0C  03           LD    (030C) BC
3725  FE  07           GH   :  CP    07
3727  C2  33  37               JP    NZ GI
372A  ED  4B  0E  03           LD    BC (030E)
372E  03                       INC   BC
372F  ED  43  0E  03           LD    (030E) BC
3733  FE  08           GI   :  CP    08
3735  C2  41  37               JP    NZ GJ
3738  ED  4B  10  03           LD    BC (0310)
```

```
373C  03                    INC  BC
373D  ED  43  10  03        LD   (0310) BC
3741  FE  09         GJ  :  CP   09
3743  C2  4F  37            JP   NZ GK
3746  ED  4B  12  03        LD   BC (0312)
374A  03                    INC  BC
374B  ED  43  12  03        LD   (0312) BC
374F  FE  0A         GK  :  CP   0A
3751  C2  5D  37            JP   NZ GL
3754  ED  4B  14  03        LD   BC (0314)
3758  03                    INC  BC
3759  ED  43  14  03        LD   (0314) BC
375D  FE  0B         GL  :  CP   0B
375F  C2  6B  37            JP   NZ GM
3762  ED  4B  16  03        LD   BC (0316)
3766  03                    INC  BC
3767  ED  43  16  03        LD   (0316) BC
376B  FE  0C         GM  :  CP   0C
376D  C2  79  37            JP   NZ GN
3770  ED  4B  18  03        LD   BC (0318)
3774  03                    INC  BC
3775  ED  43  18  03        LD   (0318) BC
3779  FE  0D         GN  :  CP   0D
377B  C2  87  37            JP   NZ GO
377E  ED  4B  1A  03        LD   BC (031A)
3782  03                    INC  BC
3783  ED  43  1A  03        LD   (031A) BC
3787  FE  0E         GO  :  CP   0E
3789  C2  AF  36            JP   NZ GA
378C  ED  4B  1C  03        LD   BC (031C)
3790  03                    INC  BC
3791  ED  43  1C  03        LD   (031C) BC
3795  C3  AF  36            JP   GA
3798  C9             FIN :  RET  *
39BD,39C8$K
39BD  21  00  22     DATA:  LD   HL 2200
39C0  11  00  01            LD   DE 0100
39C3  01  50  00            LD   BC 0050
39C6  ED  B0                LDIR
39C8  C9                    RET  *
35F4,35FF$K
35F4  21  00  40     MOVE:  LD   HL 4000
35F7  11  00  A0            LD   DE A000
35FA  01  00  40            LD   BC 4000
35FD  ED  B0                LDIR
35FF  C9                    RET  *
35B8,35D1$K
35B8  21  50  00     WORD:  LD   HL 0050
35BB  3E  53                LD   A 53
35BD  77                    LD   (HL) A
35BE  23                    INC  HL
35BF  3E  45                LD   A 45
35C1  77                    LD   (HL) A
35C2  23                    INC  HL
35C3  3E  4E                LD   A 4E
35C5  77                    LD   (HL) A
35C6  23                    INC  HL
35C7  3E  44                LD   A 44
35C9  77                    LD   (HL) A
35CA  23                    INC  HL
35CB  3E  FF                LD   A FF
35CD  77                    LD   (HL) A
35CE  CD  4[illegible]  3[illegible]    CALL ZERO
35D1  C9                    RET  *
```

```
3780,3983#K
37B0  11  60  03          SLCT:  LD    DE 0360
37B3  3E  FF                     LD    A FF
37B5  12                         LD    (DE) A
37B6  2A  20  03                 LD    HL (0320)
37B9  CB  3C                     SRL   H
37BB  CB  1D                     RR    L
37BD  CB  3C                     SRL   H
37BF  CB  1D                     RR    L
37C1  CB  3C                     SRL   H
37C3  CB  1D                     RR    L
37C5  CB  3C                     SRL   H
37C7  CB  1D                     RR    L
37C9  22  20  03                 LD    (0320) HL
37CC  ED  4B  00  03             LD    BC (0300)
37D0  CD  99  37                 CALL  TEST
37D3  F2  D9  37                 JP    P SA
37D6  3E  00                     LD    A 00
37D8  12                         LD    (DE) A
37D9  ED  4B  02  03      SA  :  LD    BC (0302)
37DD  CD  99  37                 CALL  TEST
37E0  F2  FB  37                 JP    P SB
37E3  1A                         LD    A (DE)
37E4  FE  FF                     CP    FF
37E6  CA  F2  37                 JP    Z SAI
37E9  06  01                     LD    B 01
37EB  CD  AB  37                 CALL  ONE
37EE  CA  F8  37                 JP    Z SAII
37F1  13                         INC   DE
37F2  3E  01              SAI :  LD    A 01
37F4  12                         LD    (DE) A
37F5  C3  FB  37                 JP    SB
37F8  CD  A4  37          SAII:  CALL  STOP
37FB  ED  4B  04  03      SB  :  LD    BC (0304)
37FF  CD  99  37                 CALL  TEST
3802  F2  1D  38                 JP    P SC
3805  1A                         LD    A (DE)
3806  FE  FF                     CP    FF
3808  CA  14  38                 JP    Z SBI
380B  06  02                     LD    B 02
380D  CD  AB  37                 CALL  ONE
3810  CA  1A  38                 JP    Z SBII
3813  13                         INC   DE
3814  3E  02              SBI :  LD    A 02
3816  12                         LD    (DE) A
3817  C3  1D  38                 JP    SC
381A  CD  A4  37          SBII:  CALL  STOP
381D  ED  4B  06  03      SC  :  LD    BC (0306)
3821  CD  99  37                 CALL  TEST
3824  F2  3F  38                 JP    P SD
3827  1A                         LD    A (DE)
3828  FE  FF                     CP    FF
382A  CA  36  38                 JP    Z SCI
382D  06  03                     LD    B 03
382F  CD  AB  37                 CALL  ONE
3832  CA  3C  38                 JP    Z SCII
3835  13                         INC   DE
3836  3E  03              SCI :  LD    A 03
3838  12                         LD    (DE) A
3839  C3  3F  38                 JP    SD
383C  CD  A4  37          SCII:  CALL  STOP
383F  ED  4B  08  03      SD  :  LD    BC (0308)
3843  CD  99  37                 CALL  TEST
3846  F2  61  38                 JP    P SE
3849  1A                         LD    A (DE)
```

```
384A  FE  FF                     CP    FF
384C  CA  58  38                 JP    Z SDI
384F  06  04                     LD    B 04
3851  CD  AB  37                 CALL  ONE
3854  CA  5E  38                 JP    Z SDII
3857  13                         INC   DE
3858  3E  04             SDI  :  LD    A 04
385A  12                         LD    (DE) A
385B  C3  61  38                 JP    SE
385E  CD  A4  37         SDII:   CALL  STOR
3861  ED  4B  0A  03     SE   :  LD    BC (030A)
3865  CD  99  37                 CALL  TEST
3868  F2  83  38                 JP    P SF
386B  1A                         LD    A (DE)
386C  FE  FF                     CP    FF
386E  CA  7A  38                 JP    Z SEI
3871  06  05                     LD    B 05
3873  CD  AB  37                 CALL  ONE
3876  CA  80  38                 JP    Z SEII
3879  13                         INC   DE
387A  3E  05             SEI  :  LD    A 05
387C  12                         LD    (DE) A
387D  C3  83  38                 JP    SF
3880  CD  A4  37         SEII:   CALL  STOR
3883  ED  4B  0C  03     SF   :  LD    BC (030C)
3887  CD  99  37                 CALL  TEST
388A  F2  A5  38                 JP    P SG
388D  1A                         LD    A (DE)
388E  FE  FF                     CP    FF
3890  CA  9C  38                 JP    Z SFI
3893  06  06                     LD    B 06
3895  CD  AB  37                 CALL  ONE
3898  CA  A2  38                 JP    Z SFII
389B  13                         INC   DE
389C  3E  06             SFI  :  LD    A 06
389E  12                         LD    (DE) A
389F  C3  A5  38                 JP    SG
38A2  CD  A4  37         SFII:   CALL  STOR
38A5  ED  4B  0E  03     SG   :  LD    BC (030E)
38A9  CD  99  37                 CALL  TEST
38AC  F2  C7  38                 JP    P SH
38AF  1A                         LD    A (DE)
38B0  FE  FF                     CP    FF
38B2  CA  BE  38                 JP    Z SGI
38B5  06  07                     LD    B 07
38B7  CD  AB  37                 CALL  ONE
38BA  CA  C4  38                 JP    Z SGII
38BD  13                         INC   DE
38BE  3E  07             SGI  :  LD    A 07
38C0  12                         LD    (DE) A
38C1  C3  C7  38                 JP    SH
38C4  CD  A4  37         SGII:   CALL  STOR
38C7  ED  4B  10  03     SH   :  LD    BC (0310)
38CB  CD  99  37                 CALL  TEST
38CE  F2  E9  38                 JP    P SI
38D1  1A                         LD    A (DE)
38D2  FE  FF                     CP    FF
38D4  CA  E0  38                 JP    Z SHI
38D7  06  08                     LD    B 08
38D9  CD  AB  37                 CALL  ONE
38DC  CA  E6  38                 JP    Z SHII
38DF  13                         INC   DE
38E0  3E  08             SHI  :  LD    A 08
38E2  12                         LD    (DE) A
38E3  C3  E9  38                 JP    SI
```

```
38E6  CD  A4  37          SHII :  CALL STOR
38E9  ED  4B  12  03      SI   :  LD   BC (0312)
38ED  CD  99  37                  CALL TEST
38F0  F2  05  39                  JP   P SJ
38F3  1A                          LD   A (DE)
38F4  FE  FF                      CP   FF
38F6  CA  02  39                  JP   Z SII
38F9  06  09                      LD   B 09
38FB  CD  AB  37                  CALL ONE
38FE  CA  08  39                  JP   Z SIII
3901  13                          INC  DE
3902  3E  09              SII  :  LD   A 09
3904  12                          LD   (DE) A
3905  C3  0B  39                  JP   SJ
3908  CD  A4  37          SIII:   CALL STOR
390B  ED  4B  14  03      SJ   :  LD   BC (0314)
390F  CD  99  37                  CALL TEST
3912  F2  2D  39                  JP   P SK
3915  1A                          LD   A (DE)
3916  FE  FF                      CP   FF
3918  CA  24  39                  JP   Z SJI
391B  06  0A                      LD   B 0A
391D  CD  AB  37                  CALL ONE
3920  CA  2A  39                  JP   Z SJII
3923  13                          INC  DE
3924  3E  0A              SJI  :  LD   A 0A
3926  12                          LD   (DE) A
3927  C3  2D  39                  JP   SK
392A  CD  A4  37          SJII:   CALL STOR
392D  ED  4B  16  03      SK   :  LD   BC (0316)
3931  CD  99  37                  CALL TEST
3934  F2  4F  39                  JP   P SL
3937  1A                          LD   A (DE)
3938  FE  FF                      CP   FF
393A  CA  46  39                  JP   Z SKI
393D  06  0B                      LD   B 0B
393F  CD  AB  37                  CALL ONE
3942  CA  4C  39                  JP   Z SKII
3945  13                          INC  DE
3946  3E  0B              SKI     LD   A 0B
3948  12                          LD   (DE) A
3949  C3  4F  39                  JP   SL
394C  CD  A4  37          SKII:   CALL STOR
394F  ED  4B  18  03      SL   :  LD   BC (0318)
3953  CD  99  37                  CALL TEST
3956  F2  71  39                  JP   P SM
3959  1A                          LD   A (DE)
395A  FE  FF                      CP   FF
395C  CA  68  39                  JP   Z SLI
395F  06  0C                      LD   B 0C
3961  CD  AB  37                  CALL ONE
3964  CA  6B  39                  JP   Z SMII
3967  13                          INC  DE
3968  3E  0C              SLI  :  LD   A 0C
396A  12                          LD   (DE) A
396B  C3  71  39                  JP   SM
396E  CD  A4  37          SLII    CALL STOP
3971  ED  4B  1A  03      SM      LD   BC (031A)
3975  CD  99  37                  CALL TEST
3978  F2  [illegible]  39         JP   P SN
397B  1A                          LD   A (DE)
397C  FE  FF                      CP   FF
397E  CA  9A  39                  JP   Z SMI
3981  06  0D                      LD   B 0D
3983  CD  AB  37                  CALL ONE
```

```
3986   CA  90  39          JP    Z SMII
3989   13                  INC   DE
398A   3E  0D       SMI  : LD    A 0D
398C   12                  LD    (DE) A
398D   C3  93  39          JP    SN
3990   CD  A4  37   SMII:  CALL  STOR
3993   ED  4B  1C  03  SN : LD  BC (031C)
3997   CD  99  37          CALL  TEST
399A   F2  B5  39          JP    P SO
399D   1A                  LD    A (DE)
399E   FE  FF              CP    FF
39A0   CA  AC  39          JP    Z SNI
39A3   06  0E              LD    B 0E
39A5   CD  AB  37          CALL  ONE
39A8   CA  B2  39          JP    Z SNII
39AB   13                  INC   DE
39AC   3E  0E       SNI  : LD    A 0E
39AE   12                  LD    (DE) A
39AF   C3  B5  39          JP    SO
39B2   CD  A4  37   SNII:  CALL  STOR
39B5   13           SO   : INC   DE
39B6   3E  FF              LD    A FF
39B8   12                  LD    (DE) A
39B9   C9                  RET   *
3799,37A0*K
3799   2A  20  03   TEST:  LD    HL (0320)
379C   C6  00              ADD   A 00
379E   ED  42              SBC   HL BC
37A0   C9                  RET   *
37A4,37A7*K
37A4   1A           STOR:  LD    A (DE)
37A5   00                  NOP
37A6   12                  LD    (DE) A
37A7   C9                  RET   *
37AB,37AF*K
37AB   1A           ONE  : LD    A (DE)
37AC   90                  SUB   B
37AD   FE  FF              CP    FF
37AF   C9                  RET   *
```

LISTING B

```
3000,303D$K
3000  CD  3E  30            FOLW :  CALL INT
3003  2A  00  02            AA   :  LD   HL (0200)
3006  22  12  02                    LD   (0212) HL
3009  7E                            LD   A (HL)
300A  E6  0F                        AND  0F
300C  FE  00                        CP   00
300E  C2  25  30                    JP   NZ BB
3011  C6  11                        ADD  A 11
3013  77                            LD   (HL) A
3014  CD  78  30                    CALL NAI
3017  CD  40  31                    CALL ZERO
301A  06  00                        LD   B 00
301C  CD  12  34                    CALL CYCL
301F  CD  AE  33                    CALL FF
3022  C3  53  3E                    JP   AAA
3025  CD  BD  33            BB   :  CALL TESA
3028  CA  31  30                    JP   Z CC
302B  CD  CC  33                    CALL WINA
302E  C3  03  30                    JP   AA
3031  CD  D9  33            CC   :  CALL TESB
3034  CA  3D  30                    JP   Z FINS
3037  CD  EC  33                    CALL WINB
303A  C3  03  30                    JP   AA
303D  C9                    FINS:   RET   *
303E,3076$K
303E  21  01  40            INT  :  LD   HL 4001
3041  22  00  02                    LD   (0200) HL
3044  21  FE  40                    LD   HL 40FE
3047  22  02  02                    LD   (0202) HL
304A  21  03  00                    LD   HL 0003
304D  22  04  02                    LD   (0204) HL
3050  21  7D  00                    LD   HL 007D
3053  22  06  02                    LD   (0206) HL
3056  21  80  00                    LD   HL 0080
3059  22  0A  02                    LD   (020A) HL
305C  21  7E  7F                    LD   HL 7F7E
305F  22  12  02                    LD   (0212) HL
3062  FD  21  00  00                LD   IY 0000
3066  06  00                        LD   B 00
3068  21  00  04                    LD   HL 0400
306B  22  22  03                    LD   (0322) HL
306E  21  80  04                    LD   HL 0480
3071  22  24  03                    LD   (0324) HL
3074  0E  00                        LD   C 00
3076  C9                            RET   *
30F0,3105$K
30F0  32  30  02            INIT:   LD   (0230) A
30F3  21  00  40                    LD   HL 4000
30F6  22  64  02                    LD   (0264) HL
30F9  21  7F  00                    LD   HL 007F
30FC  22  60  02                    LD   (0260) HL
30FF  21  7F  00                    LD   HL 007F
3102  22  62  02                    LD   (0262) HL
3105  C9                            RET   *
3078,3084$K
3078  22  0C  02            NAI  :  LD   (020C) HL
307B  32  51  03                    LD   (0351) A
307E  CD  85  30                    CALL BOTH
3081  2A  0C  02                    LD   HL (020C)
3084  C9                            RET   *
3085,308B$K
3085  CD  [illegible]  30   BOTH:   CALL CORD
3088  CD  [illegible]  31           CALL ADRS
308B  C9                            RET   *
```

```
3092,30EC#K
3092  ED  43  31  02   CORD:  LD    (0231) BC
3096  FD  22  33  02          LD    (0233) IY
309A  CD  F0  30              CALL  INIT
309D  ED  4B  64  02          LD    BC (0264)
30A1  FD  21  00  00          LD    IY 0000
30A5  DD  21  00  00   AGIN:  LD    IX 0000
30A9  0A               ONE :  LD    A (BC)
30AA  21  50  03              LD    HL 0350
30AD  BE                      CP    (HL)
30AE  C2  B4  30              JP    NZ TEST
30B1  CD  09  31              CALL  STOR
30B4  DD  22  0E  02   TEST:  LD    (020E) IX
30B8  2A  0E  02              LD    HL (020E)
30BB  ED  5B  60  02          LD    DE (0260)
30BF  C6  00                  ADD   A 00
30C1  ED  52                  SBC   HL DE
30C3  CA  CC  30              JP    Z NEWL
30C6  03                      INC   BC
30C7  DD  23                  INC   IX
30C9  C3  A9  30              JP    ONE
30CC  FD  22  22  02   NEWL:  LD    (0222) IY
30D0  2A  22  02              LD    HL (0222)
30D3  ED  5B  62  02          LD    DE (0262)
30D7  C6  00                  ADD   A 00
30D9  ED  52                  SBC   HL DE
30DB  CA  E4  30              JP    Z END
30DE  03                      INC   BC
30DF  FD  23                  INC   IY
30E1  C3  A5  30              JP    AGIN
30E4  ED  4B  31  02   END :  LD    BC (0231)
30E8  FD  2A  33  02          LD    IY (0233)
30EC  C9                      RET   *
3109,3126#K
3109  2A  22  03       STOR:  LD    HL (0322)
310C  DD  22  06  03          LD    (0306) IX
3110  11  06  03              LD    DE 0306
3113  1A                      LD    A (DE)
3114  77                      LD    (HL) A
3115  22  28  03              LD    (0328) HL
3118  23                      INC   HL
3119  FD  22  06  03          LD    (0306) IY
311D  11  06  03              LD    DE 0306
3120  1A                      LD    A (DE)
3121  77                      LD    (HL) A
3122  23                      INC   HL
3123  22  22  03              LD    (0322) HL
3126  C9                      RET   *
312C,313C#K
312C  ED  5B  24  03   ADRS:  LD    DE (0324)
3130  21  28  03              LD    HL 0328
3133  7E                      LD    A (HL)
3134  00                      NOP
3135  00                      NOP
3136  12                      LD    (DE) A
3137  13                      INC   DE
3138  ED  53  24  03          LD    (0324) DE
313C  C9                      RET   *
3140,3157#K
3140  3E  00           ZERO:  LD    A 00
3142  22  0C  02              LD    (020C) HL
3145  21  00  03              LD    HL 0300
3149  36  00           ZA  :  LD    (HL) 00
314A  FE  21                  CP    21
314C  CA  54  31              JP    Z ZB
314F  3C                      INC   A
```

```
3150  23                   INC  HL
3151  C3  48  31           JP   ZA
3154  2A  0C  02    ZB   : LD   HL (020C)
3157  C9                   RET   *
315B,3160$K
315B  7E            BLAC:  LD   A (HL)
315C  E6  0F               AND  0F
315E  FE  0F               CP   0F
3160  C9                   RET   *
3164,3167$K
3164  7E            MET  : LD   A (HL)
3165  FE  11               CP   11
3167  C9                   RET   *
316B,3174$K
316B  32  30  02    CHEC:  LD   (0230) A
316E  78                   LD   A B
316F  FE  08               CP   08
3171  3A  30  02           LD   A (0230)
3174  C9                   RET   *
3178,317B$K
3178  79            CMPR:  LD   A C
3179  FE  00               CP   00
317B  C9                   RET   *
317F,3193$K
317F  79            YES  : LD   A C
3180  FE  00               CP   00
3182  C2  90  31           JP   NZ YE
3185  78                   LD   A B
3186  32  35  02           LD   (0235) A
3189  CD  D4  31           CALL VXDT
318C  3A  35  02           LD   A (0235)
318F  47                   LD   B A
3190  22  12  02    YE   : LD   (0212) HL
3193  C9                   RET   *
3197,31D0$K
3197  2A  12  02    RA   : LD   HL (0212)
319A  23                   INC  HL
319B  C9                   RET
319C  2A  12  02    RB   : LD   HL (0212)
319F  11  81  00           LD   DE 0081
31A2  19                   ADD  HL DE
31A3  C9                   RET
31A4  2A  12  02    RC   : LD   HL (0212)
31A7  11  80  00           LD   DE 0080
31AA  19                   ADD  HL DE
31AB  C9                   RET
31AC  2A  12  02    RD   : LD   HL (0212)
31AF  11  7F  00           LD   DE 007F
31B2  19                   ADD  HL DE
31B3  C9                   RET
31B4  2A  12  02    RE   : LD   HL (0212)
31B7  2B                   DEC  HL
31B8  C9                   RET
31B9  2A  12  02    RF   : LD   HL (0212)
31BC  11  7F  FF           LD   DE FF7F
31BF  19                   ADD  HL DE
31C0  C9                   RET
31C1  2A  12  02    RG   : LD   HL (0212)
31C4  11  80  FF           LD   DE FF80
31C7  19                   ADD  HL DE
31C8  C9                   RET
31C9  2A  12  02    RH   : LD   HL (0212)
31CC  11  81  FF           LD   DE FF81
31CF  19                   ADD  HL DE
31D0  C9                   RET   *
```

```
3412,3591#K
3412  CD  97  31    CYCL:  CALL RA
3415  CD  5B  31           CALL BLAC
3418  CA  3A  34           JP   Z IIIA
341B  FE  00               CP   00
341D  CA  B5  3E           JP   Z CA
3420  CD  64  31           CALL MET
3423  CA  8D  35           JP   Z FIN
3426  C3  3A  34           JP   IIIA
3429  C6  01        IIA :  ADD  A 01
342B  77                   LD   (HL) A
342C  CD  7F  31           CALL YES
342F  06  00               LD   B 00
3431  CD  78  31           CALL CMPR
3434  C2  90  35           JP   NZ LAST
3437  C3  2C  35           JP   IG
343A  CD  6B  31    IIIA:  CALL CHEC
343D  CA  91  35           JP   Z FINL
3440  04                   INC  B
3441  CD  9E  31    IB  :  CALL RB
3444  CD  5B  31           CALL BLAC
3447  CA  69  34           JP   Z IIIB
344A  FE  00               CP   00
344C  CA  BB  3E           JP   Z CB
344F  CD  64  31           CALL MET
3452  CA  8D  35           JP   Z FIN
3455  C3  69  34           JP   IIIB
3458  C6  02        IIB :  ADD  A 02
345A  77                   LD   (HL) A
345B  CD  7F  31           CALL YES
345E  06  00               LD   B 00
3460  CD  78  31           CALL CMPR
3463  C2  90  35           JP   NZ LAST
3466  C3  5E  35           JP   IH
3469  CD  6B  31    IIIB:  CALL CHEC
346C  CA  91  35           JP   Z FINL
346F  04                   INC  B
3470  CD  A4  31    IC  :  CALL RC
3473  CD  5B  31           CALL BLAC
3476  CA  98  34           JP   Z IIIC
3479  FE  00               CP   00
347B  CA  C1  3E           JP   Z CC
347E  CD  64  31           CALL MET
3481  CA  8D  35           JP   Z FIN
3484  C3  98  34           JP   IIIC
3487  C6  03        IIC :  ADD  A 03
3489  77                   LD   (HL) A
348A  CD  7F  31           CALL YES
348D  06  00               LD   B 00
348F  CD  78  31           CALL CMPR
3492  C2  90  35           JP   NZ LAST
3495  C3  12  34           JP   CYCL
3498  CD  6B  31    IIIC:  CALL CHEC
349B  CA  91  35           JP   Z FINL
349E  04                   INC  B
349F  CD  AA  31    ID  :  CALL RD
34A2  CD  5B  31           CALL BLAC
34A5  CA  C7  34           JP   Z IIID
34A8  FE  00               CP   00
34AA  CA  C7  3E           JP   Z CD
34AD  CD  64  31           CALL MET
34B0  CA  8D  35           JP   Z FIN
34B3  C3  C7  34           JP   IIID
34B6  C6  04        IID :  ADD  A 04
34B8  77                   LD   (HL) A
```

```
34B9   CD  7F  31           CALL YES
34BC   06  00               LD   B 00
34BE   CD  78  31           CALL CMPR
34C1   C2  90  35           JP   NZ LAST
34C4   C3  41  34           JP   IB
34C7   CD  6B  31   IIID:   CALL CHEC
34CA   CA  91  35           JP   Z FINL
34CD   04                   INC  B
34CE   CD  B4  31   IE  :   CALL RE
34D1   CD  5B  31           CALL BLAC
34D4   CA  F6  34           JP   Z IIIE
34D7   FE  00               CP   00
34D9   CA  CD  3E           JP   Z CE
34DC   CD  64  31           CALL MET
34DF   CA  8D  35           JP   Z FIN
34E2   C3  F6  34           JP   IIIE
34E5   C6  05       IIE :   ADD  A 05
34E7   77                   LD   (HL) A
34E8   CD  7F  31           CALL YES
34EB   06  00               LD   B 00
34ED   CD  78  31           CALL CMPR
34F0   C2  90  35           JP   NZ LAST
34F3   C3  70  34           JP   IC
34F6   CD  6B  31   IIIE:   CALL CHEC
34F9   CA  91  35           JP   Z FINL
34FC   04                   INC  B
34FD   CD  B9  31   IF  :   CALL RF
3500   CD  5B  31           CALL BLAC
3503   CA  25  35           JP   Z IIIF
3506   FE  00               CP   00
3508   CA  D3  3E           JP   Z CF
350B   CD  64  31           CALL MET
350E   CA  8D  35           JP   Z FIN
3511   C3  25  35           JP   IIIF
3514   C6  06       IIF :   ADD  A 06
3516   77                   LD   (HL) A
3517   CD  7F  31           CALL YES
351A   06  00               LD   B 00
351C   CD  78  31           CALL CMPR
351F   C2  90  35           JP   NZ LAST
3522   C3  9F  34           JP   ID
3525   CD  6B  31   IIIF.   CALL CHEC
3528   CA  91  35           JP   Z FINL
352B   04                   INC  B
352C   CD  C1  31   IG  :   CALL RG
352F   CD  5B  31           CALL BLAC
3532   CA  54  35           JP   Z IIIG
3535   FE  00               CP   00
3537   CA  D5  3E           JP   Z CG
353A   CD  64  31           CALL MET
353D   CA  8D  35           JP   Z FIN
3540   C3  54  35           JP   IIIG
3543   C6  07       IIG     ADD  A 07
3545   77                   LD   (HL) A
3546   CD  7F  31           CALL YES
3549   06  00               LD   B 00
354B   CD  78  31           CALL CMPR
354E   C2  90  35           JP   NZ LAST
3551   C3  CE  34           JP   IE
3554   CD  6B  31   IIIG.   CALL CHEC
3557   CA  91  35           JP   Z FINL
355A   04                   INC  B
355B   CD  C3  31   IH  ..  CALL RH
355E   CD  5B  31           CALL BLAC
3561   CA  83  35           JP   Z IIIH
```

```
3564   FE   00                    CP    00
3566   CA   DF   3E               JP    Z CH
3569   CD   64   31               CALL  MET
356C   CA   8D   35               JP    Z FIN
356F   C3   83   35               JP    IIIH
3572   C6   08           IIH   :  ADD   A 08
3574   77                         LD    (HL) A
3575   CD   7F   31               CALL  YES
3578   06   00                    LD    B 00
357A   CD   78   31               CALL  CMPR
357D   C2   90   35               JP    NZ LAST
3580   C3   FD   34               JP    IF
3583   CD   6B   31      IIIH:    CALL  CHEC
3586   CA   91   35               JP    Z FINL
3589   04                         INC   B
358A   C3   12   34               JP    CYCL
358D   CD   CB   39      FIN  :   CALL  CHCK
3590   C9                LAST:    RET
3591   C3   0A   3F      FINL:    JP    RCVR *
3EB5,3EE2#K
3EB5   CD   A3   3E      CA   :   CALL  STRA
3EB8   C3   29   34               JP    IIA
3EBB   CD   A3   3E      CB   :   CALL  STRA
3EBE   C3   58   34               JP    IIB
3EC1   CD   A3   3E      CC   :   CALL  STRA
3EC4   C3   87   34               JP    IIC
3EC7   CD   A3   3E      CD   :   CALL  STRA
3ECA   C3   B6   34               JP    IID
3ECD   CD   A3   3E      CE   :   CALL  STRA
3ED0   C3   E5   34               JP    IIE
3ED3   CD   A3   3E      CF   :   CALL  STRA
3ED6   C3   14   35               JP    IIF
3ED9   CD   A3   3E      CG   :   CALL  STRA
3EDC   C3   43   35               JP    IIG
3EDF   CD   A3   3E      CH   :   CALL  STRA
3EE2   C3   72   35               JP    IIH *
3EA3,3EB1#K
3EA3   32   00   05      STRA:    LD    (0500) A
3EA5   3E   00                    LD    A 00
3EA8   32   55   03               LD    (0355) A
3EAB   32   65   03               LD    (0365) A
3EAE   3A   00   05               LD    A (0500)
3EB1   C9                         RET   *
39CB,39DA#K
39CB   79                CHCK:    LD    A C
39CC   FE   01                    CP    01
39CE   C8                         RET   Z
39CF   CD   D4   31               CALL  VXDT
39D2   3E   11                    LD    A 11
39D4   32   50   03               LD    (0350) A
39D7   CD   88   30               CALL  BOTH
39DA   C9                         RET   *
3595,35A2#K
3595   4F                OUTP:    LD    C A
3596   21   02   FF               LD    HL FF02
3599   7E                OUTS:    LD    A (HL)
359A   E6   02                    AND   02
359C   CA   99   35               JP    Z OUTS
359F   23                         INC   HL
35A0   73                         LD    (HL) E
35A1   79                         LD    A C
35A2   C9                         RET   *
```

```
35D2,35DC#K
35D2  06  8D          CRLF:   LD    B 8D
35D4  CD  95  35              CALL  OUTP
35D7  06  0A                  LD    B 0A
35D9  CD  95  35              CALL  OUTP
35DC  C9                      RET   *
35DF,35E8#K
35DF  CD  D2  35      OUTA:   CALL  CRLF
35E2  11  50  00              LD    DE 0050
35E5  CD  A6  35              CALL  SEND
35E8  C9                      RET   *
35E9,35F2#K
35E9  CD  D2  35      OUTB:   CALL  CRLF
35EC  11  51  00              LD    DE 0051
35EF  CD  A6  35              CALL  SEND
35F2  C9                      RET   *
35A6,35B4#K
35A6  1A              SEND:   LD    A (DE)
35A7  47                      LD    B A
35A8  FE  FF                  CP    FF
35AA  CA  B4  35              JP    Z FN
35AD  CD  95  35              CALL  OUTP
35B0  13                      INC   DE
35B1  C3  A6  35              JP    SEND
35B4  C9              FN  :   RET   *
33AE,33BC#K
33AE  2A  22  03      FF  :   LD    HL (0322)
33B1  36  FF                  LD    (HL),FF
33B3  2A  24  03              LD    HL (0324)
33B6  36  FF                  LD    (HL) FF
33B8  CD  DF  35              CALL  OUTB
33BB  FF                      RST   38
33BC  C9                      RET   *
33BD,33C8#K
33BD  ED  5B  02  02  TESA:   LD    DE (0202)
33C1  2A  00  02              LD    HL (0200)
33C4  CE  00                  ADD   A 00
33C6  ED  52                  SBC   HL DE
33C8  C9                      RET   *
33CC,33D5#K
33CC  ED  5B  00  02  NINA:   LD    DE (0200)
33D0  13                      INC   DE
33D1  ED  53  00  02          LD    (0200) DE
33D5  C9                      RET   *
33D9,33E8#K
33D9  FD  22  0E  02  TESB:   LD    (020E) IY
33DD  2A  0E  02              LD    HL (020E)
33E0  ED  5B  06  02          LD    DE (0206)
33E4  CE  00                  ADD   A 00
33E6  ED  52                  SBC   HL DE
33E8  C9                      RET   *
33E9,3404#K
33E9  ED  5B  04  02  NINB:   LD    DE (0204)
33F0  2A  00  02              LD    HL (0200)
33F3  19                      ADD   HL DE
33F4  22  00  02              LD    (0200) HL
33F7  2A  02  02              LD    HL (0202)
33FA  ED  5B  0A  02          LD    DE (020A)
33FE  19                      ADD   HL DE
33FF  22  02  02              LD    (0202) HL
3402  FD  23                  INC   IY
3404  C9                      RET   *
```

```
31D4.321F#K
31D4  3A  00  03      VNDT:  LD   A (0300)
31D7  FE  00                 CP   00
31D9  C2  E3  31             JP   NZ VA
31DC  7E                     LD   A (HL)
31DD  32  00  03             LD   (0300) A
31E0  C3  E7  31             JP   VB
31E3  BE              VA  :  CP   (HL)
31E4  C2  EF  31             JP   NZ VC
31E7  3A  01  03      VB  :  LD   A (0301)
31EA  3C                     INC  A
31EB  32  01  03             LD   (0301) A
31EE  C9                     RET
31EF  CD  33  32      VC  :  CALL SAVE
31F2  3C                     INC  A
31F3  CD  20  32             CALL AJST
31F6  BE                     CP   (HL)
31F7  C2  FE  31             JP   NZ VD
31FA  CD  84  32             CALL LINK
31FD  C9                     RET
31FE  3D              VD  :  DEC  A
31FF  3D                     DEC  A
3200  CD  20  32             CALL AJST
3203  BE                     CP   (HL)
3204  CA  1C  32             JP   Z VE
3207  CD  48  32             CALL CRTV
320A  3E  01                 LD   A 01
320C  32  01  03      NEW :  LD   (0301) A
320F  7E                     LD   A (HL)
3210  32  00  03             LD   (0300) A
3213  3E  00                 LD   A 00
3215  32  02  03             LD   (0302) A
3218  32  03  03             LD   (0303) A
321B  C9                     RET
321C  CD  84  32      VE  :  CALL LINK
321F  C9                     RET   *
3220,3232#K
3220  FE  00          AJST:  CP   00
3222  CA  28  32             JP   Z JA
3225  F2  2B  32             JP   P JB
3228  C6  08          JA  :  ADD  A 08
322A  C9                     RET
322B  FE  09          JB  :  CP   09
322D  FA  32  32             JP   M JC
3230  D6  08                 SUB  08
3232  C9              JC  :  RET   *
3233,3246#K
3233  ED  5B  10  03  SAVE:  LD   DE (0310)
3237  ED  53  0A  03         LD   (030A) DE
323B  ED  5B  12  02         LD   DE (0212)
323F  ED  53  10  03         LD   (0310) DE
3243  CD  05  34             CALL LNTH
3246  C9                     RET   *
3405,3410#K
3405  CD  F9  32      LNTH:  CALL CHSL
3408  3E  01                 LD   A 01
340A  32  01  03             LD   (0301) A
340D  3A  00  03             LD   A (0300)
3410  C9                     RET   *
3248,3264#K
3248  3E  00          CRTV:  LD   A 00
324A  32  05  03             LD   (0305) A
324D  22  0C  02             LD   (020C) HL
3250  3A  50  03             LD   A (0350)
3253  E6  F0                 AND  F0
```

```
3255  C6  11                   ADD  A 11
3257  32  50  03               LD   (0350) A
325A  2A  10  03               LD   HL (0310)
325D  77                       LD   (HL) A
325E  CD  88  30               CALL BOTH
3261  2A  0C  02               LD   HL (020C)
3264  C9                       RET   *
3267,3280#K
3267  22  0C  02       VRTX:   LD   (020C) HL
326A  2A  10  03               LD   HL (0310)
326D  7E                       LD   A (HL)
326E  32  08  03               LD   (0308) A
3271  3A  50  03               LD   A (0350)
3274  C6  01                   ADD  A 01
3276  32  50  03               LD   (0350) A
3279  77                       LD   (HL) A
327A  CD  92  30               CALL CORD
327D  2A  0C  02               LD   HL (020C)
3280  C9                       RET   *
      3284,32BA#K
3284  0E  01           LINK:   LD   C 01
3286  47                       LD   B A
3287  3A  02  03               LD   A (0302)
328A  FE  00                   CP   00
328C  C2  A7  32               JP   NZ LA
328F  78                       LD   A B
3290  32  02  03               LD   (0302) A
3293  3A  05  03               LD   A (0305)
3296  FE  00                   CP   00
3298  CA  AB  32               JP   Z LB
329B  CD  A7  37               CALL OLD
329E  CA  AB  32               JP   Z LB
32A1  CD  E3  32               CALL PRVC
32A4  C3  AB  32               JP   LB
32A7  B8               LA  :   CP   B
32A8  C2  B1  32               JP   NZ LC
32AB  CD  4D  37       LB  :   CALL DOES
32AE  C3  B8  32               JP   LE
32B1  78               LC  :   LD   A B
32B2  32  02  03               LD   (0302) A
32B5  CD  78  33               CALL DONT
32B8  0E  00           LE      LD   C 00
32BA  C9                       RET   *
32BE,32CA#K
32BE  22  12  02       SAME:   LD   (0212) HL
32C1  06  00                   LD   B 00
32C3  CD  12  34               CALL CYCL
32C6  3A  02  03               LD   A (0302)
32C9  BE                       CP   (HL)
32CA  C9                       RET   *
32F9,3340#K
32F9  3A  03  03       CHSL:   LD   A (0303)
32FC  FE  00                   CP   00
32FE  C2  0A  33               JP   NZ SA
3301  3A  01  03               LD   A (0301)
3304  32  03  03               LD   (0303) A
3307  C3  35  33               JP   SC
330A  47               SA  :   LD   B A
330B  3A  01  03               LD   A (0301)
330E  90                       SUB  B
330F  FE  [illegible]          CP   [illegible]
[illegible]  F[illegible]  [illegible]  [illegible]   JP   [illegible] SF
[illegible]  CA  1[illegible]  [illegible]            JP   [illegible] SF
3317  ED  [illegible]          NEG
3319  47               SB      LD   B A
```

```
331A  3A  01  03           LD   A (0301)
331D  FE  02               CP   02
331F  CA  24  33           JP   Z SE
3322  CB  3F               SRL  A
3324  CB  3F       SE   :  SRL  A
3326  B8                   CP   B
3327  CA  3F  33           JP   Z SD
332A  F2  3F  33           JP   P SD
332D  CD  CD  32           CALL PREV
3330  C3  3F  33           JP   SD
3333  00                   NOP
3334  00                   NOP
3335  C9           SC   :  RET
333F  3A  03  03   SD   :  LD   A (0303)
3342  47                   LD   B A
3343  3A  01  03           LD   A (0301)
3346  80                   ADD  A B
3347  CB  3F               SRL  A
3349  32  03  03           LD   (0303) A
334C  C9                   RET  *
334D,3374#K
334D  CD  BE  32   DOES:   CALL SAME
3350  CA  67  33           JP   Z DB
3353  3A  00  03           LD   A (0300)
3356  BE                   CP   (HL)
3357  CA  63  33           JP   Z DA
335A  3E  02               LD   A 02
335C  CD  0C  32           CALL NEW
335F  CD  48  32           CALL CRTV
3362  C9                   RET
3363  00           DA   :  NOP
3364  00                   NOP
3365  00                   NOP
3366  C9                   RET
3367  3E  01       DB   :  LD   A 01
3369  32  05  03           LD   (0305) A
336C  3E  02               LD   A 02
336E  CD  0C  32           CALL NEW
3371  CD  57  32           CALL VRTX
3374  C9                   RET  *
3375,33A3#K
3375  CD  BE  32   DONT:   CALL SAME
3378  CA  92  33           JP   Z NA
337B  3A  00  03           LD   A (0300)
337E  BE                   CP   (HL)
3381  C2  92  33           JP   NZ NA
3384  3E  01               LD   A 01
3386  32  01  03           LD   (0301) A
3389  7E                   LD   A (HL)
338A  32  00  03           LD   (0300) A
338D  CD  57  32           CALL VRTX
3390  C9                   RET
3392  3E  02       NA   :  LD   A 02
3394  CD  0C  32           CALL NEW
3397  CD  48  32           CALL CRTV
339A  C9                   RET
339B  3E  02               LD   A 02
339D  CD  0C  32           CALL NEW
33A0  CD  48  32           CALL CRTV
33A3  C9                   RET  *
33A4,33AE#K
33A7  3A  08  03   OLD  :  LD   A (0308)
33AA  BE                   CP   (HL)
33AB  C9                   RET  *
```

```
3EE8,3F07$K
3EE8  21  60  00       FAIL:  LD    HL 0060
3EEB  3E  46                  LD    A 46
3EED  77                      LD    (HL) A
3EEE  23                      INC   HL
3EEF  3E  41                  LD    A 41
3EF1  77                      LD    (HL) A
3EF2  23                      INC   HL
3EF3  3E  49                  LD    A 49
3EF5  77                      LD    (HL) A
3EF6  23                      INC   HL
3EF7  3E  4C                  LD    A 4C
3EF9  77                      LD    (HL) A
3EFA  23                      INC   HL
3EFB  3E  FF                  LD    A FF
3EFD  77                      LD    (HL) A
3EFE  CD  D2  35              CALL  CRLF
3F01  11  60  00              LD    DE 0060
3F04  CD  A5  35              CALL  SEND
3F07  FF                      RST   38  *
3F0A,3FCF$K
3F0A  06  00           RCVR:  LD    B 00
3F0C  3A  65  03              LD    A (0365)
3F0F  FE  00                  CP    00
3F11  C2  9F  3F              JP    NZ RONE
3F14  2A  12  02              LD    HL (0212)
3F17  22  12  05              LD    (0512) HL
3F1A  3A  00  03              LD    A (0300)
3F1D  32  00  05       RCA :  LD    (0500) A
3F20  3A  55  03              LD    A (0355)
3F23  3C                      INC   A
3F24  32  55  03              LD    (0355) A
3F27  3E  01                  LD    A 01
3F29  32  65  03              LD    (0365) A
3F2C  3A  00  05              LD    A (0500)
3F2F  FE  00                  CP    00
3F31  CA  53  3E              JP    Z AAA
3F34  FE  01                  CP    01
3F36  C2  42  3F              JP    NZ RCB
3F39  CD  97  31              CALL  RA
3F3C  22  12  02              LD    (0212) HL
3F3F  C3  12  34              JP    CYCL
3F42  FE  02           RCB :  CP    02
3F44  C2  50  3F              JP    NZ RCC
3F47  CD  9C  31              CALL  RB
3F4A  22  12  02              LD    (0212) HL
3F4D  C3  41  34              JP    IB
3F50  FE  03           RCC :  CP    03
3F52  C2  5E  3F              JP    NZ RCD
3F55  CD  A4  31              CALL  RC
3F58  22  12  02              LD    (0212) HL
3F5B  C3  70  34              JP    IC
3F5E  FE  04           RCD :  CP    04
3F60  C2  6C  3F              JP    NZ RCE
3F63  CD  AC  31              CALL  RD
3F66  22  12  02              LD    (0212) HL
3F69  C3  9F  34              JP    ID
3F6C  FE  05           RCE    CP    05
3F6E  C2  7A  3F              JP    NZ RCF
3F71  CD  B4  31              CALL  RE
3F74  22  12  02              LD    (0212) HL
3F77  C3  CE  34              JP    IE
3F7A  FE  06           RCF :  CP    06
3F7C  C2  88  3F              JP    NZ RCG
3F7F  CD  B5  31              CALL  RF
```

```
3F82  22  12  02          LD    (0212) HL
3F85  C3  FD  34          JP    IF
3F88  FE  07      RCG  :  CP    07
3F8A  C2  96  3F          JP    NZ RCH
3F8D  CD  C1  31          CALL  RG
3F90  22  12  02          LD    (0212) HL
3F93  C3  2C  35          JP    IG
3F96  CD  C9  31  RCH  :  CALL  RH
3F99  22  12  02          LD    (0212) HL
3F9C  C3  5B  35          JP    IH
3F9F  3A  55  03  RONE:   LD    A (0355)
3FA2  FE  01              CP    01
3FA4  C2  B7  3F          JP    NZ RTWO
3FA7  3A  00  03          LD    A (0300)
3FAA  3C                  INC   A
3FAB  CD  20  32          CALL  AJST
3FAE  2A  12  05          LD    HL (0512)
3FB1  22  12  02          LD    (0212) HL
3FB4  C3  1D  3F          JP    RCA
3FB7  3A  55  03  RTWO:   LD    A (0355)
3FBA  FE  02              CP    02
3FBC  C2  CF  3F          JP    NZ REND
3FBF  3A  00  03          LD    A (0300)
3FC2  3D                  DEC   A
3FC3  CD  20  32          CALL  AJST
3FC6  2A  12  05          LD    HL (0512)
3FC9  22  12  02          LD    (0212) HL
3FCC  C3  1D  3F          JP    RCA
3FCF  CD  E8  3E  REND:   CALL  FAIL *
3E53,3E56$K
3E53  CD  2B  3E  AAA  :  CALL  UNIC
3E56  C3  00  30          JP    FOLW *
3E2B,3E4F$K
3E2B  21  80  40  UNIC:   LD    HL 4080
3E2E  11  7F  7F  UA   :  LD    DE 7F7F
3E31  22  0C  02          LD    (020C) HL
3E34  C6  00              ADD   A 00
3E36  ED  52              SBC   HL DE
3E38  CA  4F  3E          JP    Z UFIN
3E3B  2A  0C  02          LD    HL (020C)
3E3E  23                  INC   HL
3E3F  CD  5B  31          CALL  BLAC
3E42  CA  2E  3E          JP    Z UA
3E45  FE  00              CP    00
3E47  CA  2E  3E          JP    Z UA
3E4A  36  0F              LD    (HL) 0F
3E4C  C3  2E  3E          JP    UA
3E4F  C9          UFIN:   RET   *
```

LISTING C

```
2100,2147#K
2100  CD  00  25          AVRG.   CALL INTL
2103  CD  B7  27                  CALL FRST
2106  CD  39  28          AGIN:   CALL UPDA
2109  0E  00                      LD   C 00
210B  CD  D3  27          BEGN:   CALL SUM
210E  CD  DE  27                  CALL LOOP
2111  CA  17  21                  JP   Z IIIA
2114  C3  0B  21                  JP   BEGN
2117  CD  4F  28          IIIA:   CALL STEP
211A  CA  23  21                  JP   Z MAKE
211D  CD  65  28                  CALL DO
2120  C3  0B  21                  JP   BEGN
2123  CD  F4  27          MAKE:   CALL MAIN
2126  CD  7B  28                  CALL ONE
2129  CA  32  21                  JP   Z IIIB
212C  CD  99  28                  CALL ROW
212F  C3  06  21                  JP   AGIN
2132  CD  8A  28          IIIB:   CALL TWO
2135  CA  44  21                  JP   Z FIN
2138  CD  B9  28                  CALL CLMN
213B  FD  23                      INC  IY
213D  FD  22  02  02              LD   (0202) IY
2141  C3  06  21                  JP   AGIN
2144  CD  8A  27          FIN :   CALL SHFT
2147  C9                          RET  *
27B7,27CF#K
27B7  21  00  40          FRST:   LD   HL 4000
27BA  22  10  02                  LD   (0210) HL
27BD  21  03  40                  LD   HL 4003
27C0  22  12  02                  LD   (0212) HL
27C3  21  80  40                  LD   HL 4080
27C6  22  18  02                  LD   (0218) HL
27C9  21  03  41                  LD   HL 4103
27CC  22  1A  02                  LD   (021A) HL
27CF  C9                          RET  *
27D3,27DC#K
27D3  DD  7E  00          SUM :   LD   A (IX+00)
27D6  E6  0F                      AND  0F
27D8  47                          LD   B A
27D9  79                          LD   A C
27DA  80                          ADD  A B
27DB  4F                          LD   C A
27DC  C9                          RET  *
27DE,27F2#K
27DE  DD  23              LOOP:   INC  IX
27E0  DD  22  0E  02              LD   (020E) IX
27E4  22  0C  02                  LD   (020C) HL
27E7  ED  5B  0E  02              LD   DE (020E)
27EB  C6  00                      ADD  A 00
27ED  ED  52                      SBC  HL DE
27EF  2A  0C  02                  LD   HL (020C)
27F2  C9                          RET  *
27F4,272827#K
27F4  0E  0F              MAIN:   LD   C 0F
27F6  ED  5B  00  02              LD   DE (0200)
27FA  FE  4B                      CP   4B
27FC  FA  14  25                  JP   M TT
27FF  C8                          RET  Z
2800  1A                          LD   A (DE)
2801  E6  0F                      AND  0F
2803  81                          ADD  A C
2804  FE  0F                      CP   0F
2806  FA  [illegible]  28         JP   [illegible]
2809  CD  [illegible]  28         CALL SHFT
```

```
2800  C9                          RET
280D  1A               SS   :     LD    A (DE)
280E  E6  0F                      AND   0F
2810  C6  F0                      ADD   A F0
2812  12                          LD    (DE) A
2813  C9                          RET
2814  1A               TT   :     LD    A (DE)
2815  E6  0F                      AND   0F
2817  91                          SUB   C
2818  FE  00                      CP    00
281A  FA  21  28                  JP    M UU
281D  CD  29  28                  CALL  SHFL
2820  C9                          RET
2821  1A               UU   :     LD    A (DE)
2822  E6  0F                      AND   0F
2824  C6  00                      ADD   A 00
2826  12                          LD    (DE) A
2827  C9                          RET   *
2839,284E#K
2839  2A  12  02       UPDA:      LD    HL (0212)
283C  22  16  02                  LD    (0216) HL
283F  DD  2A  10  02              LD    IX (0210)
2843  DD  22  14  02              LD    (0214) IX
2847  FD  2A  02  02              LD    IY (0202)
284E  C9                          RET   *
284F,2861#K
284F  22  0C  02       STEP:      LD    (020C) HL
2852  DD  22  0E  02              LD    (020E) IX
2856  2A  0E  02                  LD    HL (020E)
2859  ED  5B  1A  02              LD    DE (021A)
285D  C6  00                      ADD   A 00
285F  ED  52                      SBC   HL DE
2861  C9                          RET   *
2865,2877#K
2865  2A  0C  02       DO   :     LD    HL (020C)
2868  ED  5B  08  02              LD    DE (0208)
286C  19                          ADD   HL DE
286D  DD  2A  14  02              LD    IX (0214)
2871  DD  19                      ADD   IX DE
2873  DD  22  14  02              LD    (0214) IX
2877  C9                          RET   *
287B,2886#K
287B  ED  5B  10  02   ONE  :     LD    DE (0210)
287F  2A  12  02                  LD    HL (0212)
2882  C6  00                      ADD   A 00
2884  ED  52                      SBC   HL DE
2886  C9                          RET   *
288A,2895#K
288A  2A  02  02       TWO  :     LD    HL (0202)
288D  ED  5B  04  02              LD    DE (0204)
2891  C6  00                      ADD   A 00
2893  ED  52                      SBC   HL DE
2895  C9                          RET   *
2899,28B5#K
2899  2A  10  02       FDN  :     LD    HL (0210)
289C  23                          INC   HL
289D  22  10  02                  LD    (0210) HL
28A0  2A  12  02                  LD    HL (0212)
28A3  23                          INC   HL
28A4  22  12  02                  LD    (0212) HL
28A7  2A  00  02                  LD    HL (0200)
28AA  23                          INC   HL
28AB  22  00  02                  LD    (0200) HL
28AE  2A  1A  02                  LD    HL (021A)
28B1  23                          INC   HL
28B2  22  1A  02                  LD    (021A) HL
                                  RET   *
```

```
28B9.28E4$K
28B9  ED  5B  06  02   CLMN:   LD   DE (0206)
28BD  2A  10  02               LD   HL (0210)
28C0  19                       ADD  HL DE
28C1  22  10  02               LD   (0210) HL
28C4  2A  12  02               LD   HL (0212)
28C7  19                       ADD  HL DE
28C8  22  12  02               LD   (0212) HL
28CB  2A  1A  02               LD   HL (021A)
28CE  19                       ADD  HL DE
28CF  22  1A  02               LD   (021A) HL
28D2  2A  00  02               LD   HL (0200)
28D5  19                       ADD  HL DE
28D6  22  00  02               LD   (0200) HL
28D9  ED  5B  08  02           LD   DE (0208)
28DD  2A  18  02               LD   HL (0218)
28E0  19                       ADD  HL DE
28E1  22  18  02               LD   (0218) HL
28E4  C9                       RET  *
278A,27B3$K
278A  DD  21  00  40   SHFT:   LD   IX 4000
278E  21  00  80       PP   :  LD   HL 8000
2791  DD  7E  00               LD   A (IX+00)
2794  CB  3F                   SRL  A
2796  CB  3F                   SRL  A
2798  CB  3F                   SRL  A
279A  CB  3F                   SRL  A
279C  DD  77  00               LD   (IX+00) A
279F  DD  23                   INC  IX
27A1  DD  22  60  02           LD   (0260) IX
27A5  ED  5B  60  02           LD   DE (0260)
27A9  C6  00                   ADD  A 00
27AB  ED  52                   SBC  HL DE
27AD  CA  B3  27               JP   Z RO
27B0  C3  8E  27               JP   PP
27B3  C9               RO   :  RET  *
```

```
2000,20D4$K
2000  CD  00  25       EDGE:  CALL INTL
2003  CD  49  25       IA  :  CALL INIT
2006  CD  67  25       IB  :  CALL NEW
2009  CD  8F  25              CALL STRT
200C  CD  28  25              CALL ZERO
200F  CA  9F  20              JP   Z IIG
2012  ED  5B  14  02          LD   DE (0214)
2016  CD  B3  25              CALL MASK
2019  CD  C9  25       IC  :  CALL PIX
201C  CD  F0  25              CALL LOPA
201F  CA  25  20              JP   Z ID
2022  C3  19  20              JP   IC
2025  CD  08  26       ID  :  CALL THLD
2028  FA  37  20              JP   M IE
202B  CD  13  26              CALL CNTR
202E  C2  37  20              JP   NZ IE
2031  CD  2E  26              CALL MODF
2034  C3  A4  20              JP   IIH
2037  CD  4E  26       IE  :  CALL TWIC
203A  CA  40  20              JP   Z IF
203D  C3  06  20              JP   IB
2040  21  69  26       IF  :  LD   HL LOPB
2043  22  1D  20              LD   (201D) HL
2046  21  4F  20              LD   HL IIA
2049  22  3B  20              LD   (203B) HL
204C  C3  06  20              JP   IB
204F  CD  67  25       IIA :  CALL NEW
2052  CD  8F  25              CALL STRT
2055  ED  5B  14  02          LD   DE (0214)
2059  CD  B3  25              CALL MASK
205C  CD  C9  25       IIB :  CALL PIX
205F  CD  F0  25              CALL LOPA
2062  CA  68  20              JP   Z IIC
2065  C3  5C  20              JP   IIB
2068  CD  85  26       IIC :  CALL LOPC
206B  CD  C9  25              CALL PIX
206E  DD  2A  0C  02          LD   IX (020C)
2072  CD  08  26              CALL THLD
2075  FA  84  20              JP   M IID
2078  CD  13  26              CALL CNTR
207B  C2  84  20              JP   NZ IID
207E  CD  2E  26              CALL MODF
2081  C3  A4  20              JP   IIH
2084  CD  4E  26       IID    CALL TWIC
2087  CA  8D  20              JP   Z IIE
208A  C3  4F  20              JP   IIA
208D  21  69  26       IIE    LD   HL LOPB
2090  22  60  20              LD   (2060) HL
2093  21  9C  20              LD   HL IIF
2096  22  8B  20              LD   (208B) HL
2099  C3  4F  20              JP   IIA
209C  CD  3A  26       IIF :  CALL LOAD
209F  C6  F0           IIG :  ADD  A F0
20A1  CD  45  26              CALL STOR
20A4  CD  8F  26       IIH :  CALL TSTP
20A7  CA  B9  20              JP   Z IIK
20AA  21  9E  26              LD   HL WNDA
20AD  22  BE  26              LD   (26BE) HL
20B0  CD  B3  26              CALL REPT
20B3  CD  2D  27              CALL OLDA
20B6  C3  03  20              JP   IA
20B9  CD  35  27       IIK    CALL TSTB
20BC  CA  D1  20              JP   Z END
20BF  21  48  27              LD   HL WNDB
```

```
20C2  22  6E  26              LD   (266E) HL
20C5  CD  84  26              CALL REPT
20C8  CD  6E  27              CALL OLDB
20CB  CD  80  27              CALL LAST
20CE  C3  03  20              JP   IA
20D1  CD  8A  27       END :  CALL SHFT
20D4  C9                      RET  *
2500,2525$K
2500  21  81  40      INTL:   LD   HL,4081
2503  22  00  02              LD   (0200) HL
2506  21  00  00              LD   HL 0000
2509  22  02  02              LD   (0202) HL
250C  21  7D  00              LD   HL 007D
250F  22  04  02              LD   (0204) HL
2512  21  03  00              LD   HL 0003
2515  22  06  02              LD   (0206) HL
2518  21  80  00              LD   HL 0080
251B  22  08  02              LD   (0208) HL
251E  CD  0B  27              CALL OLD
2521  FD  2A  02  02          LD   IY (0202)
2525  C9                      RET  *
2549,2563$K
2549  21  00  01      INIT:   LD   HL 0100
254C  22  1A  02              LD   (021A) HL
254F  22  1E  02              LD   (021E) HL
2552  21  90  25              LD   HL 2590
2555  22  1C  02              LD   (021C) HL
2558  21  10  02              LD   HL 0210
255B  22  20  02              LD   (0220) HL
255E  3E  00                  LD   A 00
2560  32  40  02              LD   (0240) A
2563  C9                      RET  *
2567,258B$K
2567  3E  00          NEW :   LD   A 00
2569  2A  1C  02              LD   HL (021C)
256C  ED  5B  1E  02          LD   DE (021E)
2570  47              AA  :   LD   B A
2571  1A                      LD   A (DE)
2572  77                      LD   (HL) A
2573  13                      INC  DE
2574  23                      INC  HL
2575  1A                      LD   A (DE)
2576  77                      LD   (HL) A
2577  78                      LD   A B
2578  FE  04                  CP   04
257A  CA  86  25              JP   Z BB
257D  3C                      INC  A
257E  13                      INC  DE
257F  01  05  00              LD   BC 0005
2582  09                      ADD  HL BC
2583  C3  70  25              JP   AA
2586  13              BB      INC  DE
2587  ED  53  1E  02          LD   (021E) DE
258B  C9                      RET  *
258F,25AF$K
258F  21  FD  7E      STRT:   LD   HL 7EFD
2592  22  10  02              LD   (0210) HL
2595  21  00  7F              LD   HL 7F00
2598  22  12  02              LD   (0212) HL
259B  21  FE  7F              LD   HL 7FFE
259E  22  14  02              LD   (0214) HL
25A1  21  0C  00              LD   HL 000C
25A4  22  16  02              LD   (0216) HL
25A7  21  0A  7F              LD   HL 7F0A
25AA  22  18  02              LD   (0218) HL
25AD  0E  10                  LD   C 10
```

```
2528,252D$K
2528  CD  3A  25        ZERO:  CALL LOAD
252B  FE  0F                   CP   0F
252D  C9                       RET   *
25B3,25BA$K
25B3  CD  BE  25        MASK:  CALL UPDT
25B6  1A                       LD   A (DE)
25B7  E6  0F                   AND  0F
25B9  47                       LD   B A
25BA  C9                       RET   *
25BE,25C5$K
25BE  2A  12  02        UPDT:  LD   HL (0212)
25C1  DD  2A  10  02           LD   IX (0210)
25C5  C9                       RET   *
25C9,25DD$K
25C9  DD  7E  00        PIX :  LD   A (IX+00)
25CC  E6  0F                   AND  0F
25CE  90                       SUB  B
25CF  F2  D4  25               JP   P CC
25D2  ED  44                   NEG
25D4  CD  E1  25        CC  :  CALL MIN
25D7  C3  DD  25               JP   DD
25DA  CD  E1  25               CALL MIN
25DD  C9                DD  :  RET   *
25F0,2604$K
25F0  DD  23            LOPA:  INC  IX
25F2  DD  22  0E  02           LD   (020E) IX
25F6  22  0C  02               LD   (020C) HL
25F9  ED  5B  0E  02           LD   DE (020E)
25FD  C6  00                   ADD  A 00
25FF  ED  52                   SBC  HL DE
2601  2A  0C  02               LD   HL (020C)
2604  C9                       RET   *
2608,260B$K
2608  79                THLD:  LD   A C
2609  FE  0F                   CP   0F
260B  C9                       RET   *
2613,262A$K
2613  32  30  02        CNTR:  LD   (0230) A
2616  3A  40  02               LD   A (0240)
2619  FE  01                   CP   01
261B  C2  23  26               JP   NZ FF
261E  3E  00                   LD   A 00
2620  C3  24  26               JP   GG
2623  3C                FF  :  INC  A
2624  32  40  02        GG  :  LD   (0240) A
2627  3A  30  02               LD   A (0230)
262A  C9                       RET   *
262E,2636$K
262E  CD  3A  26        MODF:  CALL LOAD
2631  C6  00                   ADD  A 00
2633  CD  45  26               CALL STOR
2636  C9                       RET   *
263A,3641$K
263A  ED  5B  00  02    LOAD:  LD   DE (0200)
263E  1A                       LD   A (DE)
263F  E6  0F                   AND  0F
2641  C9                       RET
2645,264A$K
2645  ED  5B  00  02    STOR:  LD   DE (0200)
2649  12                       LD   (DE) A
264A  C9                       RET   *
```

```
264E  32  30  02        TWIC:  LD    (0230) A
2651  3A  50  02               LD    A (0250)
2654  FE  01                   CP    01
2656  C2  5E  26               JP    NZ HH
2659  3E  00                   LD    A 00
265B  C3  5F  26               JP    JJ
265E  3C                HH  :  INC   A
265F  32  50  02        JJ  :  LD    (0250) A
2662  3A  30  02               LD    A (0230)
2665  C9                       RET    *
2669,2681$K
2669  ED  5B  08  02    LOPB:  LD    DE (0208)
266D  DD  19                   ADD   IX DE
266F  DD  22  0E  02           LD    (020E) IX
2673  22  0C  02               LD    (020C) HL
2676  ED  5B  0E  02           LD    DE (020E)
267A  C6  00                   ADD   A 00
267C  ED  52                   SBC   HL DE
267E  2A  0C  02               LD    HL (020C)
2681  C9                       RET    *
2685,268B$K
2685  ED  5B  16  02    LOPC:  LD    DE (0216)
2689  DD  19                   ADD   IX DE
268B  C9                       RET    *
268F,269A$K
268F  ED  5B  18  02    TSTA:  LD    DE (0218)
2693  2A  12  02               LD    HL (0212)
2696  C6  00                   ADD   A 00
2698  ED  52                   SBC   HL DE
269A  C9                       RET    *
269B,26B0$K
269B  2A  10  02        WNDA:  LD    HL (0210)
269E  23                       INC   HL
269F  22  10  02               LD    (0210) HL
26A2  2A  12  02               LD    HL (0212)
26A5  23                       INC   HL
26A6  22  12  02               LD    (0212) HL
26A9  2A  14  02               LD    HL (0214)
26AC  23                       INC   HL
26AD  22  14  02               LD    (0214) HL
26B0  C9                       RET    *
26B4,26CC$K
26B4  CD  49  25        REPT:  CALL  INIT
26B7  CD  67  25        KK  :  CALL  NEW
26BA  CD  8F  25               CALL  STRT
26BD  CD  9B  26               CALL  WNDA
26C0  CD  D0  26               CALL  CHNG
26C3  CD  F2  26               CALL  TEST
26C6  CA  CC  26               JP    Z LL
26C9  C3  B7  26               JP    KK
26CC  C9                LL  :  RET    *
26D0,26EE$K
26D0  3E  00            CHNG:  LD    A 00
26D2  ED  5B  20  02           LD    DE (0220)
26D6  2A  1A  02               LD    HL (021A)
26D9  47                MM  :  LD    B A
26DA  1A                       LD    A (DE)
26DB  77                       LD    (HL) A
26DC  78                       LD    A B
26DD  FE  09                   CP    09
26DF  CA  E8  26               JP    Z NN
26E2  13                       INC   DE
26E3  23                       INC   HL
26E4  3C                       INC   A
26E5  C3  D9  26               JP    MM
26E8  2A  1E  02        NN  :  LD    HL (021E)
26EB  22  1A  02               LD    (021A) HL
26EE  C9                       RET    *
```

```
26F2,2707$K
26F2  ED  53  0A  02    TEST:  LD    (020A) DE
26F6  22  0C  02               LD    (020C) HL
26F9  11  50  01               LD    DE 0150
26FC  C6  00                   ADD   A 00
26FE  ED  52                   SBC   HL DE
2700  ED  5B  0A  02           LD    DE (020A)
2704  2A  0C  02               LD    HL (020C)
2707  C9                       RET   *
270B,2729$K
270B  21  F0  25        OLD :  LD    HL LOPA
270E  22  1D  20               LD    (201D) HL
2711  22  60  20               LD    (2060) HL
2714  21  40  20               LD    HL IF
2717  22  3B  20               LD    (203B) HL
271A  21  8D  20               LD    HL IIE
271D  22  88  20               LD    (2088) HL
2720  ED  5B  0A  02           LD    DE (020A)
2724  3E  00                   LD    A 00
2726  32  50  02               LD    (0250) A
2729  C9                       RET   *
272D,2737$K
272D  CD  0B  27        OLDA:  CALL  OLD
2730  2A  00  02               LD    HL (0200)
2733  23                       INC   HL
2734  22  00  02               LD    (0200) HL
2737  C9                       RET   *
273B,2746$K
273B  2A  02  02        TSTB:  LD    HL (0202)
273E  ED  5B  04  02           LD    DE (0204)
2742  C6  00                   ADD   A 00
2744  ED  52                   SBC   HL DE
2746  C9                       RET   *
2748,276C$K
2748  ED  5B  06  02    WNDB:  LD    DE (0206)
274C  2A  10  02               LD    HL (0210)
274F  19                       ADD   HL DE
2750  22  10  02               LD    (0210) HL
2753  2A  12  02               LD    HL (0212)
2756  19                       ADD   HL DE
2757  22  12  02               LD    (0212) HL
275A  2A  14  02               LD    HL (0214)
275D  19                       ADD   HL DE
275E  22  14  02               LD    (0214) HL
2761  ED  5B  08  02           LD    DE (0208)
2765  2A  18  02               LD    HL (0218)
2768  19                       ADD   HL DE
2769  22  18  02               LD    (0218) HL
276C  C9                       RET   *
276E,277C$K
276E  CD  0B  27        OLDB.  CALL  OLD
2771  ED  5B  06  02           LD    DE (0206)
2775  2A  0A  02               LD    HL (020A)
2778  19                       ADD   HL DE
2779  22  0A  02               LD    (020A) HL
277C  C9                       RET   *
2788,278E$K
2788  FD  23            LAST   INC   IY
278A  FD  22  32  02           LD    (0232) IY
278E  C9                       RET   *
```

Appendix 3

PROGRAMS IN 'C'

These include two main operations:-

a) the Real Vertex Verification and

b) the Formation of Unit Clauses.

The whole program consists of a *main* routine and 16 subroutines. Nine of the subroutines are called by the *main* and the rest of them from other subroutines. In the following, the function of every subroutine is described in brief with special emphasis on their difficult points.

main:- This is the main routine of the program, which basically is responsible for printing out messages according to the various results, and calls the nine *primary* subroutines that do the two operations mentioned at the beginning. The nine subroutines, according to the order they are called are:-

init:- This initializes the character arrays ChA[2Ø],ChB[2Ø],..., ChE[2Ø], by loading them with 'z', and the final *Real Vertex Co-ordinate*

Arrays A[2∅][2],B[2∅][2],...,E[2∅][2] by setting them to ∅. Finally it zeroizes some pointers and flags.

move:- This moves the vertex-array from locations A[2∅][2] to locations B[2∅][2] to C[2∅][2] and so on. This is necessary because locations A[2∅][2] are always loaded by the vertex-array of the new array. The old one has to be stored for later comparisons. A flag f indicates which case to be used.

load:- This loads the new vertex-array into V[51] and sets pointer f accordingly for the next call of *move*.

check:- This checks if the vertex-array V[51] and the pointer-array N[16] have got an end marker -1, and sets flags *F1* and *F2* accordingly.

truvert:- This is the subroutine that checks the distance of every vertex from the equation formed by the real-vertices and accordingly eliminates the ones with distance greater than a certain limit. In doing this it calls three subroutines which are:-

equat:- This forms the equation using the coordinates of the real-vertices, by calculating *tang* and *b*:-

$$tang=y_2-y_1/x_2-x_1 \qquad with\ x_2\neq 1$$
$$b=(x_2*y_1-x_1*y_2)/(x_2-x_1)$$

When $x_2=x_1$ then *tang* is *set* to 1 because it can not be set to infinity and it should not be set to zero by default.

tolernce:- This checks if the distance of every one of the pseudo-vertices is within the prespecified limits. If both $x_2\neq x_1$ and $tang\neq\emptyset$ the tolerance is given by the maximum of the absolute differences between the co-ordinates given by the equation and those of the pseudo-vertex. If $x_2=x_1$ and $tang=\pm\infty$, then the tolerance is given by the absolute difference of the abscissae. Finally if $tang\neq\emptyset$, the tolerance is given by the absolute value of the two ordinates. This is why *tang* was set to 1 in the previous subroutine.

store:- This stores the coordinates of the verified real-vertices into the new vertex-array A[2Ø][2].

newlist:- This creates a new list of pointers N[i], by inserting in the old list pointers to the pseudo-vertices with distance from the equation greater than the limit.

adjacent:- This checks if two adjacent true-vertices within the vertex-array A[2Ø][2] are near enough to be considered as one.

first:- This decides if the first real vertex is to be eliminated due to the case discussed in Section 3.4.2.

classify:- This calculates the primary elements of the 2-D shapes such as angles, lengths of sides etc., and uses the following four sub-routines to prepare the unit clauses for the procedure of recognition.

angle:- This calculates the angles of the 2-D shape according to the formulae of paragraph 3.4.3d.

vertex:- This loads the character array ChA[i] with the appropriate lower-case alphabetic character from c[i].

predicate:- This forms the unit-clauses which are to be used by the PROLOG program and writes them into file *'data'*.

commvert:- This uses subroutine *doit* to check if the 2-D shapes that are faces of a 3-D one have common vertices or not.

doit:- This checks if the distances of every vertex of each 2-D shape from those of everyone of the others are near enough to be considered as common. If so, it uses the same alphabetic character to represent them.

The 'C' programs are presented in the following listing.

```
#include <stdio.h>
double tang,b,tol,x,y,x1,x2,y1,y2,X,Y,T[16],V[51];
double fabs(),sqrt(),atan();
int adm();
int A[20][2],B[20][2],C[20][2],D[20][2],E[20][2];
int f,i,j,k,F1,F2,R,I,An[16],S[16],S2[16],S1[16],N[16];
char c[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m',
            'n','o','p','q','r','s','t','u','v','w','x','y','z'};
char ChA[20],ChB[20],ChC[20],ChD[20],ChE[20];
main()
{       int l,m;
        init();
for(i=0;i(20;i++)
begin: if(f!=0) move();
        if(f)4)
        {       printf("solid with more than 5 sides ,
                        algorithm fails\n");
                goto end;
        }
        load();
        if(f==-1) goto end;
printf("f=%d\n",f);
        check();
        if(F1==1)
        {       printf("2D-shape with more than 15 vertices ,
                        algorithm fails\n");
                goto stop;
        }
        if(F2==1)
        {       printf("too many vertices , augment dimension
                        of V[]\n"):
                goto stop;
        }
        x1=V[0];
        y1=V[1];
        x2=V[N[1]];
        y2=V[N[1]+1];
        I=0;
        for(j=1;(N[j]!=-1);j++)
                truvert();
        store();
        A[I][0]= -1;
        adjacent();
        first();
        classify();
        For(l=0;l(=16;l++)
        {        For(m=0;m(=1;m++)
                        printf("A[%d][%d]  =  %d\n",l,m,A[l][m]);
        }
      goto begin;
end: printf("end of real vertex verification phase ,
               unit clauses in (data)\n");
stop: printf("end of procedure\n");
}

init()
{       For(i=0;i(20;i++)
        {       ChA[i]=ChB[i]=ChC[i]=ChD[i]=ChE[i]='z';
                For(j=0;j(2;j++)
                        A[i][j]=B[i][j]=C[i][j]=D[i][j]=E[i][j]=0;
        }
        f=0;
        [illegible]0;
        [illegible];
```

```
}

move()
{         if(f==4)
        {       for(i=0;i<20;i++)
                {       ChE[i]=ChD[i];
                        ChD[i]=ChC[i];
                        ChC[i]=ChB[i];
                        ChB[i]=ChA[i];
                        for(j=0;j<2;j++)
                        {       E[i][j]=D[i][j];
                                D[i][j]=C[i][j];
                                C[i][j]=B[i][j];
                                B[i][j]=A[i][j];
                        }
                }
        }
        else if(f==3)
        {       for(i=0;i<20;i++)
                {       ChD[i]=ChC[i];
                        ChC[i]=ChB[i];
                        ChB[i]=ChA[i];
                        for(j=0;j<2;j++)
                        {       D[i][j]=C[i][j];
                                C[i][j]=B[i][j];
                                B[i][j]=A[i][j];
                        }
                }
        }
        else if(f==2)
        {       for(i=0;i<20;i++)
                {       ChC[i]=ChB[i];
                        ChB[i]=ChA[i];
                        for(j=0;j<2;j++)
                        {       C[i][j]=B[i][j];
                                B[i][j]=A[i][j];
                        }
                }
        }
        else if(f==1)
        {       for(i=0;i<20;i++)
                {       ChB[i]=ChA[i];
                        for(j=0;j<2;j++)
                                B[i][j]=A[i][j];
                }
        }
        return;
}

load()
{       int i,v[51];
        printf("type coordinates of vertices now\n");
        scanf("%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
                %x%x%x%x%x%x%x%x%x%x%x%x%x",
&v[0],&v[1],&v[2],&v[3],&v[4],&v[5],&v[6],&v[7],&v[8],&v[9],
&v[10],&v[11],&v[12],&v[13],&v[14],&v[15],&v[16],&v[17],&v[18],
&v[19],&v[20],&v[21],&v[22],&v[23],&v[24],&v[25],&v[26],&v[27],);
        scanf("%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%",
&v[28],&v[29],&v[30],&v[31],&v[32],&v[33],&v[34],&v[35],&v[36],
&v[37],&v[38],&v[39],&v[40],&v[41],&v[42],&v[43],&v[44],&v[45],
&v[46],&v[47],&v[48],&v[49],&v[50]);
        printf("type pointers to certain vertices now\n");
        scanf("%x%x%x%x%x%x%x%x%x%x%x%x%x%x",
                &N[0],&N[1],&N[2],&N[3],&N[4],&N[5],&N[6],&N[7],
```

```
        &N[8],&N[9],&N[10],&N[11],&N[12],&N[13],&N[14],&N[15]);
         for(i=0;i<51;i++) V[i]=v[i];
         for(i=0;i<51;i++)
                 if(V[i]!=0.0) printf("V[%d]   =   %5.2f\n",i,V[i]);
          if(N[0]==0) printf("N[0]   =   %d\n",N[0]);
          for(i=0;i<16;i++)
                 if(N[i]!=0) printf("N[%d]   =   %d\n",i,N[i]);
          if(V[0]!=-1) f=f+1;
          else f= -1;
          return;
}

check()
{       F1=1;
        for(i=0;i<16;i++)
        {     if(N[i]==-1) F1=0;
        }
        F2=1;
        for(i=0;i<51;i++)
        {     if(V[i]==-1) F2=0;
        }
        return;
}

equat()
{        X=x2-x1;
         Y=y2-y1;
         if(X!=0)
         {        tang=Y/X;
                  b=(x2*y1-x1*y2)/(x2-x1);
         }
         else
         {        tang=1;
                  b=0;
         }
         return;
}

newlist()
{        int L;
         L=j;
         for( ;N[j]!=-1;j++);
         for( ;j>L;j--)
              N[j+1]=N[j];
         N[j+1]=N[j];
         N[j]=R;
         return;
}

store()
{        int K;
         K=I;
         A[K][0]=x1;
         A[K][1]=y1;
         I=K+1;
         return;
}

tolernce()
{        double Dx,Dy;
         if((X!=0)&&(tang!=0))
         {        y=tang*V[i]+b;
                  x=(V[i+1]-b)/tang;
```

```
                Dy=fabs(y-V[i+1]);
                tol=(Dx<=Dy?Dx:Dy);
printf("Dx=%5.2f,Dy=%5.2f,tol=%5.2f\n",Dx,Dy,tol);
        }
        else if(X==0)
                tol=fabs(V[i]-x1);
        else if(tang==0)
                tol=fabs(V[i+1]-y1);
        return;
}

truvert()
{       int flag;
        double max;
start:  equat();
        max=0.0;
        i=N[j-1]+2;
loop:   if(i<N[j])
        {       tolernce();
                if(tol<=2.0)
                {one:           i=i+2;
                                goto loop;
                }
                flag=1;
                if(tol>=max)
                {       max=tol;
                        x2=V[i];
                        y2=V[i+1];
                        R=i;
                }
                goto one;
        }
        else if(V[i]==-1)
                return;
        else if(flag!=1)
        {       store();
                x1=x2;
                y1=y2;
                x2=V[N[j+1]];
                y2=V[N[j+1]+1];
                return;
        }
        else
        {       newlist();
                flag=0;
                goto start;
        }
}

adjacent()
{       double K1,K2,L1,L2;
        int l,m;
        for(l=0;A[l+1][0]!=-1;l++)
        {       K1=A[l][0]-A[l+1][0];
                K2=A[l][1]-A[l+1][1];
                L1=K1*K1;
                L2=K2*K2;
                if(sqrt(L1+L2)<=2.0*sqrt(2.0))
                {       for(m=l;A[m+1][0]!=-1;m++)
                        {       A[m][0]=A[m+1][0];
                                A[m][1]=A[m+1][1];
                        }
                        A[m][0]= -1;
                }
```

```
          }
          return;
}

first()
{       int No;
        for(i=0;A[i][0]!=-1;i++);
        No=i-2;
printf("No=%d\n",No);
        x1=A[1][0]; y1=A[1][1];
        x2=A[No][0]; y2=A[No][1];
printf('x1=%5.2f,x2=%5.2f,y1=%5.2f,y2=%5.2f\n",x1,x2,y1,y2);
        equat();
        i=0;
        V[i]=A[0][0]; V[i+1]=A[0][1];
        tolernce();
printf("x=%5.2f,y=%5.2f,V[%d]=%5.2f,V[%d+1]=%5.2f\n",x,y,i,V[i],i
printf("tol=%5.2f\n",tol);
        if(tol<=2.0)
        {       for(i=0;A[i][0]!=-1;i++)
                {    A[i][0]=A[i+1][0];
                     A[i][1]=A[i+1][1];
                }
                A[i-2][0]=A[0][0];
                A[i-2][1]=A[0][1];
        }
        else return;
}

classify()
{      double D1,D2;
       for(i=0;(A[i+1][0]!=-1);i++)
       {    D1=A[i][0]-A[i+1][0];
            D2=A[i][1]-A[i+1][1];
            S[i]=sqrt(D1*D1+D2*D2)+0.5;
            T[i]=(D1!=0?(360/(2*3.1416)*(atan(D2/D1))):90);
            if(T[i]<0) T[i]=T[i]+180;
            S1[i]=T[i]+0.5;
            S2[i]=S[i]*S[i];
       }
       R=i;
       angle();
       vertex();
       commvert();
       predicate();
       return;
}

angle()
{    int maxA;
        maxA=A[1][1];
        for(i=1;i<R-1;i++)
              maxA=(maxA>=A[i+1][1]?maxA:A[i+1][1]);
         An[0]=abs(S1[0]-S1[R-1]);
         for(i=1;i<R;i++)
        {     if((R>3)&&((A[i][1]<=A[i-1][1])&&(A[i][1]<=A[i+1][1]))
                         ||((A[i][1]>=A[i-1])&&(A[i][1]>=A[i+1][1])))
                    An[i]=abs(S1[i-1]-S1[i]);
               else if(A[i][1]==maxA)
                    An[i]=abs(S1[i-1]-S1[i]);
               else An[i]=180-abs(S1[i]-S1[i-1]);
        }
       return;
}
```

```

vertex()
{       for(i=0;A[i+1][0]!=-1;i++)
        {     ChA[i]=c[k];
              k=k+1;
        }
}

predicate()
{       FILE *out;
        out=fopen("data","a");
        for(i=0;A[i+1][0]!=-1;i++)
        {       if((A[i+1][0]!=A[0][0])||(A[i+1][1]!=A[0][1]))
                {     fprintf(out,"conn(%c,%c%c,%c).\n",
                              ChA[i],ChA[i],ChA[i+1],ChA[i+1]);
                      fprintf(out,"line(%c%c,%d).\n",
                              ChA[i],ChA[i+1],S[i]);
                      fprintf(out,"slope(%c%c,%d).\n",
                              ChA[i],ChA[i+1],S1[i]);
                      fprintf(out,"sqrline(%c%c,%d).\n",
                              ChA[i],ChA[i+1],S2[i]);
                }
                else
                {     fprintf(out,"conn(%c,%c%c,%c).\n",
                              ChA[i],ChA[i],ChA[0],ChA[0]);
                      fprintf(out,"line(%c%c,%d).\n",
                              ChA[i],ChA[0],S[i]);
                      fprintf(out,"slope(%c%c,%d).\n",
                              ChA[i],ChA[0],S1[i]);
                      fprintf(out,"sqrline(%c%c,%d).\n",
                              ChA[i],ChA[0],S2[i]);
                }
        }
       for(i=0;i<R;i++)
             fprintf(out,"angle(%c,%d).\n",ChA[i],An[i]);
       fclose(out);
        return;
}

commvert()
{     if(E[0][0]!=0)
      {      doit(A,B,ChA,ChB);
             doit(A,C,ChA,ChC);
             doit(A,D,ChA,ChD);
             doit(A,E,ChA,ChE);
             doit(B,C,ChB,ChC);
             doit(B,D,ChB,ChD);
             doit(B,E,ChB,ChE);
             doit(C,D,ChC,ChD);
             doit(C,E,ChC,ChE);
              return;
       }
       else if(D[0][0]!=0)
       {     doit(A,B,ChA,ChB);
             doit(A,C,ChA,ChC);
             doit(A,D,ChA,ChD);
             doit(B,C,ChB,ChC);
             doit(B,D,ChB,ChD);
             doit(C,D,ChC,ChD);
             return;
       }
       else if(C[0][0]!=0)
       {     doit(B,C,ChB,ChC);
             doit(A,B,ChA,ChB);
```

```
                doit(A,C,ChA,ChC);
                 return;
            }
            else if(B[0][0]!=0)
            {       doit(A,B,ChA,ChB);
                    return;
            }
            else return;
}

doit(W,U,ChW,ChU)
        int W[20][2],U[20][2];
        char *ChW,*ChU;
{       int K1,K2,L1,L2;
        for(j=0;U[j+1][0]!=-1;j++)
        {    for(i=0;W[i+1][0]!=-1;i++)
               {       K1=W[i][0]-U[j][0];
                       K2=W[i][1]-U[j][1];
                       L1=K1*K1;
                       L2=K2*K2;
                       if(sqrt(L1+L2)<=2.0*sqrt(2.0))
                       ChW[i]=ChU[j];
               }
        }
        return;
}
```

Appendix 4

PROGRAMS IN 'PROLOG'

These are the two recognition programs *2-D Recognizer* and *3-D Recognizer*. The following is a listing of the two programs, which have been presented in detail in Chapter 5.

```
init2D:-retractall(atrian(A,B,C,D,E,F)),retractall(aquadril(G,H,I,J,K,L,M,N)).

shape(X,triangle):-trian(X,S,Y,T,Z,U).
trian(X,S,Y,T,Z,U):-conn(X,S,Y),conn(Y,T,Z),conn(Z,U,X),not(atrian(X,S,Y,T,Z,U)),not(atrian(Y,T,Z,U,X,S)),
                    not(atrian(Z,U,X,S,Y,T)),assert(atrian(X,S,Y,T,Z,U)).

shape(X,isosceles):-init2D,trian(X,S,Y,T,Z,U),equalpair(S,T,U).
equalpair(X,Y,Z):-equalline(X,Y).
equalpair(X,Y,Z):-equalline(Y,Z).
equalpair(X,Y,Z):-equalline(X,Z).
equalline(X,Y):-line(X,M),line(Y,N),equal1(M,N).
equal1(X,Y):-N is X,M is Y,N=M,!.
equal1(X,Y):-(X)Y),!,N is X-Y,N=(2.
equal1(X,Y):-(X(Y),!,N is Y-X,N=(2.

shape(X,equilateral):-init2D,trian(X,S,Y,T,Z,U),equalline(T,U),equalline(S,T).

shape(X,right-angled):-init2D,trian(X,S,Y,T,Z,U),right(S,T,U).
right(X,Y,Z):-sqrline(X,L),sqrline(Y,M),sqrline(Z,N),pythag(L,M,N).
pythag(X,Y,Z):-equal2(X,Y+Z).
pythag(X,Y,Z):-equal2(Y,Z+X).
pythag(X,Y,Z):-equal2(Z,X+Y).
equal2(X,Y):-N is X,M is Y,N=M,!.
equal2(X,Y):-X)Y,!,N is X-Y,N=(4.
equal2(X,Y):-X(Y,!,N is Y-X,N=(4.

shape(X,obtuse-angled):-init2D,trian(X,S,Y,T,Z,U),obtuse(S,T,U).
obtuse(X,Y,Z):-sqrline(X,L),sqrline(Y,M),sqrline(Z,N),obttheor(L,M,N).
obttheor(X,Y,Z):-N is X,M is Y+Z,N)M,!,P is N-M,P)4.
obttheor(X,Y,Z):-N is Y,M is X+Z,N)M,!,P is N-M,P)4.
obttheor(X,Y,Z):-N is Z,M is X+Y,N)M,!,P is N-M,P)4.

shape(W,quadrilateral):-quadril(W,R,X,S,Y,T,Z,U).
quadril(W,R,X,S,Y,T,Z,U):-conn(W,R,X),conn(X,S,Y),conn(Y,T,Z),conn(Z,U,W),(X\==Z),(Y\==W),
                          not(conn(W,O,Y)),not(conn(X,Q,Z)),not(conn(U,P,W)),not(conn(Z,V,X)),
                          not(aquadril(W,R,X,S,Y,T,Z,U)),not(aquadril(X,S,T,Y,Z,U,W,R)),
                          not(aquadril(Y,T,Z,U,W,R,X,S)),not(aquadril(Z,U,W,R,X,S,Y,T)),
                          assert(aquadril(W,R,X,S,Y,T,Z,U)).

shape(W,non-convex-quadrilateral):-nonconvquad(W,R,X,S,Y,T,Z,U).
nonconvquad(W,R,X,S,Y,T,Z,U):-init2D,quadril(W,R,S,Y,T,Z,U),nonconvex(W,X,Y,Z).
nonconvex(W,X,Y,Z):-angle(W,K),angle(X,L),angle(Y,M),angle(Z,N),two180(K,L,M,N).
two180(W,X,Y,Z):-N is W+X+Y+Z,N(360,!,R is 360-N,R)3.

shape(W,trapezium):-trapez(W,R,X,S,Y,T,Z,U).
trapez(W,R,X,S,Y,T,Z,U):-init2D,quadril(W,R,X,S,Y,T,Z,U),((paral(R,T),not(paral(S,u)));
                         (paral(S,U),not(paral(R,T)))).
paral(X,Y):-slope(X,M),slope(Y,N),equalslope(M,N).
equalslope(X,Y):-X)Y,!,N is X-Y,N=(3.
equalslope(X,Y):-X(Y,!,N is Y-X,N=(3.
equalslope(X,Y):-N is X,M is Y,N=M,!.

shape(isosceles-trapezium):-trapez(W,R,X,S,Y,T,Z,U),((equalline(R,T),not(paral(S,U)));
                            (equalline(S,U),not(paral(R,T)))).
```

```
shape(W,parallelogram):-paralgrm(W,R,X,S,Y,T,Z,U).
paralgrm(W,R,X,S,Y,T,Z,U):-init2D,quadril(W,R,X,S,Y,T,Z,U),paral(R,T),paral(S,U).

shape(W,rectangle):-rectan(W,R,X,S,Y,T,Z,U).
rectan(W,R,X,S,Y,T,Z,U):-paralgrm(W,R,X,S,Y,T,Z,U),rect(R,S).
rect(X,Y):-slope(X,L),slope(Y,M),right(L,M).
right(X,Y):-X)Y,!,N is X-Y,90=N.
right(X,Y):-X(Y,!,N is Y-X,90=N.
right(X,Y):-X)Y,!,N is X-Y,N(90,!,R is 90-N,R=(3.
right(X,Y):-X(Y,!,N is Y-X,N(90,!,R is 90-N,R=(3.
right(X,Y):-X)Y,!,N is X-Y,N)90,!,R is N-90,R=(3.
right(X,Y):-X(Y,!,N is Y-X,N)90,!,R is N-90,R=(3.
right(X,Y):-X=Y,!,R is 90.

shape(W,square):-squar(W,R,X,S,Y,T,Z,U).
squar(W,R,X,S,Y,T,Z,U):-rectan(W,R,X,S,Y,T,Z,U),equalline(R,S).

shape(W,rhombus):-rhomb(W,R,X,S,Y,T,Z,U).
rhomb(W,R,X,S,Y,T,Z,U):-paralgrm(W,R,X,S,Y,T,Z,U),equalline(R,S).

shape(N,other):-init2D,not(trian(A,X,B,Y,C,Z)),not(quadril(R,S,T,U,V,W,Q,P)).
```

```
init2D:-retractall(atrian(A,B,C,D,E,F)),retractall(aquadril(G,H,I,J,K,L,M,N)).

trian(X,S,Y,T,Z,U):-conn(X,S,Y),conn(Y,T,Z),conn(Z,U,X),not(atrian(X,S,Y,T,Z,U)),not(atrian(Y,T,Z,U,X,S)),
                    not(atrian(Z,U,X,S,Y,T)),assert(atrian(X,S,Y,T,Z,U)).
equalline(X,Y):-line(X,M),line(Y,N),equall(M,N).
equall(X,Y):-N is X,M is Y,N=M,!.
equall(X,Y):-(X)Y),!,N is X-Y,N=(2.
equall(X,Y):-(X(Y),!,N is Y-X,N=(2.

quadril(W,R,X,S,Y,T,Z,U):-conn(W,R,X),conn(X,S,Y),conn(Y,T,Z),conn(Z,U,W),(X\==Z),(Y\==W),
                          not(conn(W,Q,Y)),not(conn(X,Q,Z)),not(conn(U,P,W)),not(conn(Z,V,X)),
                          not(aquadril(W,R,X,S,Y,T,Z,U)),not(aquadril(X,S,T,Y,Z,U,W,R)),
                          not(aquadril(Y,T,Z,U,W,R,X,S)),not(aquadril(Z,U,W,R,X,S,Y,T)),
                          assert(aquadril(W,R,X,S,Y,T,Z,U)).

paralgrm(W,R,X,S,Y,T,Z,U):-init2D,quadril(W,R,X,S,Y,T,Z,U),paral(R,T),paral(S,U).
paral(X,Y):-slope(X,M),slope(Y,N),equalslope(M,N).
equalslope(X,Y):-X)Y,!,N is X-Y,N=(3.
equalslope(X,Y):-X(Y,!,N is Y-X,N=(3.
equalslope(X,Y):-N is X,M is Y,N=M,!.

rectan(W,R,X,S,Y,T,Z,U):-paralgrm(W,R,X,S,Y,T,Z,U),rect(R,S).
rect(X,Y):-slope(X,L),slope(Y,M),right(L,M).
right(X,Y):-X)Y,!,N is X-Y,90=N.
right(X,Y):-X(Y,!,N is Y-X,90=N.
right(X,Y):-X)Y,!,N is X-Y,N(90,!,R is 90-N,R=(3.
right(X,Y):-X(Y,!,N is Y-X,N(90,!,R is 90-N,R=(3.
right(X,Y):-X)Y,!,N is X-Y,N)90,!,R is N-90,R=(3.
right(X,Y):-X(Y,!,N is Y-X,N)90,!,R is N-90,R=(3.
right(X,Y):-X=Y,!,R is 90.

squar(W,R,X,S,Y,T,Z,U):-rectan(W,R,X,S,Y,T,Z,U),equalline(R,S).

rhomb(W,R,X,S,Y,T,Z,U):-paralgrm(W,R,X,S,Y,T,Z,U),equalline(R,S).

notshape:-not(trian(A,B,C,D,E,F)),not(quadril(G,H,I,J,K,L,M,N)).

shape3D(A,tetrahedron):-tetra(A,B,C,D).
tetra(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(C,V,B,W,D,U),!,notshape,not(other1(A,B,C,D)).
tetra(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(A,V,C,U,D,W),!,notshape,not(other1(A,B,C,D)).
tetra(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(A,V,D,U,B,W),!,notshape,not(other1(A,B,C,D)).

shape3D(A,tetrahedron):-tetra1(A,B,C,D).
tetra1(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(A,R,D,S,B,T),trian(A,U,C,V,D,W),!,notshape,not(other1(A,B,C,D)).
tetra1(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(A,R,C,S,D,T),trian(A,E,B,F,D,G),trian(C,U,B,V,D,W),!,
                 notshape,not(other1(A,B,C,D)).
tetra1(A,B,C,D):-init2D,trian(A,X,B,Y,C,Z),trian(A,R,D,S,B,T),trian(A,E,D,F,C,G),trian(B,U,D,V,C,W),!,
                 notshape,not(other1(A,B,C,D)).

other1(A,B,C,D):-conn(I,J,K),(K\==A),(K\==B),(K\==C),(K\==D).
```

```
shape3D(A,square-pyramid):-pyram(A,B,C,D,E).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,W,C,U,D,V),trian(A,X,D,Y,E,Z),(E\==B),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,U,D,V,B,W),trian(A,X,C,Y,E,Z),(D\==E),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,U,D,V,B,W),trian(A,X,E,Y,D,Z),(C\==E),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(C,U,B,V,D,W),trian(C,X,D,Y,E,Z),(A\==E),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,U,D,V,B,W),trian(C,X,B,Y,E,Z),(E\==D),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,E,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(C,U,B,V,D,W),trian(A,X,C,Y,E,Z),(E\==D),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,U,C,V,D,W),trian(D,X,C,Y,E,Z),(B\==E),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(A,U,D,V,B,W),trian(D,X,E,Y,B,Z),(C\==E),!,
                   notshape,not(other2(A,B,C,D,E)).
pyram(A,B,C,D,E):-init2D,trian(A,R,B,S,C,T),trian(C,V,B,U,D,W),trian(B,X,E,Y,D,Z),(A\==E),!,
                   notshape,not(other2(A,B,C,D,E)).

shape3D(A,square-pyramid):-pyram1(A,B,C,D,E).
pyram1(A,B,C,D,E):-init2D,trian(A,S,B,T,C,U),trian(A,I,C,J,D,K),trian(A,L,D,M,E,N),trian(A,P,E,Q,B,R),
                   quadril(B,W,C,X,D,Y,E,Z),!,notshape,not(other2(A,B,C,D,E)).
pyram1(A,B,C,D,E):-init2D,trian(A,S,B,T,C,U),trian(C,I,B,J,D,K),trian(C,L,D,M,E,N),trian(E,P,A,Q,C,R),
                   quadril(E,W,A,X,B,Y,D,Z),!,notshape,not(other2(A,B,C,D,E)).
pyram1(A,B,C,D,E):-init2D,trian(A,S,B,T,C,U),trian(B,I,A,J,D,K),trian(B,L,D,M,E,N),trian(B,P,E,Q,C,R),
                   quadril(A,W,D,X,E,Y,C,Z),!,notshape,not(other2(A,B,C,D,E)).

shape3D(A,square-pyramid):-tetra(A,B,C,D).

other2(A,B,C,D,E):-conn(I,J,K),(K\==A),(K\==B),(K\==C),(K\==D),(K\==E).

shape3D(A,square-pyramid):-truntripyr(A,B,C,D,E).

shape3D(A,triangular-prism):-sqrprism(A,B,C,D,E,F).

shape3D(A,truncated-triangular-pyramid):-truntripyr(A,B,C,D,E).
truntripyr(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,quadril(A,R,D,S,E,T,B,U),!,notshape,not(other2(A,B,C,D,E)).
truntripyr(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,quadril(A,R,C,S,E,T,D,U),!,notshape,not(other2(A,B,C,D,E)).
truntripyr(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,quadril(B,U,D,R,E,S,C,T),!,notshape,not(other2(A,B,C,D,E)).
truntripyr(A,B,D,C,E):-init2D,quadril(A,R,B,S,C,T,D,U),!,trian(D,X,C,Y,E,Z),!,notshape,not(other2(A,B,C,D,E)).
truntripyr(A,B,C,D,E):-init2D,quadril(A,R,B,S,C,T,D,U),!,trian(B,X,E,Y,C,Z),!,notshape,not(other2(A,B,C,D,E)).

other3(A,B,C,D,E,F):-conn(I,J,K),(K\==A),(K\==B),(K\==C),(K\==D),(K\==E),(K\==F).

shape3D(A,truncated-triangular-pyramid):-truntripyr1(A,B,C,D,E,F).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(C,W,B,X,E,Y,F,Z),trian(D,K,C,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(A,W,D,X,E,Y,F,Z),trian(D,K,C,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(D,W,C,X,E,Y,F,Z),trian(C,K,B,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(B,W,A,X,E,Y,F,Z),trian(C,K,B,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
```

```
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(C,W,B,X,E,Y,F,Z),trian(B,K,A,L,E,M),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(A,W,D,X,E,Y,F,Z),trian(B,K,A,L,F,M),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(D,W,C,X,E,Y,F,Z),trian(A,K,D,L,F,M),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),!,quadril(B,W,A,X,E,Y,F,Z),trian(A,K,D,L,E,M),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,quadril(A,W,C,X,D,Y,E,Z),quadril(B,K,A,L,E,M,F,N),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,quadril(B,W,A,X,D,Y,E,Z),quadril(C,K,B,L,E,M,F,N),!,
                              notshape,not(other3(A,B,C,D,E,F)).
truntripyr1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,quadril(C,W,B,X,D,Y,E,Z),quadril(A,K,C,L,E,M,F,N),!,
                              notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,triangular-prism):-triprism(A,B,C,D,E).
triprism(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,paralgrm(A,R,D,S,E,T,B,U),!,notshape,not(other2(A,B,C,D,E)).
triprism(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,paralgrm(A,R,C,S,E,T,D,U),!,notshape,not(other2(A,B,C,D,E)).
triprism(A,B,C,D,E):-init2D,trian(A,X,B,Y,C,Z),!,paralgrm(B,U,D,R,E,S,C,T),!,notshape,not(other2(A,B,C,D,E)).
triprism(A,B,D,C,E):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,trian(D,X,C,Y,E,Z),!,notshape,not(other2(A,B,C,D,E)).
triprism(A,B,C,D,E):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,trian(B,X,E,Y,C,Z),!,notshape,not(other2(A,B,C,D,E)).

shape3D(A,triangular-prism):-trirpism1(A,B,C,D,E,F).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(C,W,B,X,E,Y,F,Z),trian(D,K,C,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(A,W,D,X,E,Y,F,Z),trian(D,K,C,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(D,W,C,X,E,Y,F,Z),trian(C,K,B,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(B,W,A,X,E,Y,F,Z),trian(C,K,B,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(C,W,B,X,E,Y,F,Z),trian(B,K,A,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(A,W,D,X,E,Y,F,Z),trian(B,K,A,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(D,W,C,X,E,Y,F,Z),trian(A,K,D,L,F,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),!,paralgrm(B,W,A,X,E,Y,F,Z),trian(A,K,D,L,E,M),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,paralgrm(A,W,C,X,D,Y,E,Z),paralgrm(B,K,A,L,E,M,F,N),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,paralgrm(B,W,A,X,D,Y,E,Z),paralgrm(C,K,B,L,E,M,F,N),!,
                          notshape,not(other3(A,B,C,D,E,F)).
trirpism1(A,B,C,D,E,F):-init2D,trian(A,R,B,S,C,T),!,paralgrm(C,W,B,X,D,Y,E,Z),paralgrm(A,K,C,L,E,M,F,N),!,
                          notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,truncated-square-pyramid):-trunsqrpyr(A,B,C,D,E,F).
trunsqrpyr(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),quadril(C,W,B,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
trunsqrpyr(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),quadril(B,W,A,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
trunsqrpyr(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),quadril(A,W,D,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
trunsqrpyr(A,B,C,D,E,F):-init2D,quadril(A,R,B,S,C,T,D,U),quadril(D,W,C,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).

other4(A,B,C,D,E,F,G):-conn(I,J,K),(K\==A),(K\==B),(K\==C),(K\==D),(K\==E),(K\==F),(K\==G).
```

```
shape3D(A,truncated-square-pyramid):-trunsqrpyr1(A,B,C,D,E,F,G).
trunsqrpyr1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),quadril(B,K,A,L,E,M,F,N),quadril(C,R,B,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
trunsqrpyr1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),quadril(A,K,D,L,E,M,F,N),quadril(B,R,A,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
trunsqrpyr1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),quadril(D,K,C,L,E,M,F,N),quadril(A,R,D,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
trunsqrpyr1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),quadril(C,K,B,L,E,M,F,N),quadril(D,R,C,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).

shape3D(A,square-prism):-sqrprism(A,B,C,D,E,F).
sqrprism(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),paralgrm(C,W,B,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
sqrprism(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),paralgrm(B,W,A,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
sqrprism(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),paralgrm(A,W,D,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
sqrprism(A,B,C,D,E,F):-init2D,paralgrm(A,R,B,S,C,T,D,U),paralgrm(D,W,C,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,square-prism):-sqrprism1(A,B,C,D,E,F,G).
sqrprism1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),paralgrm(B,K,A,L,E,M,F,N),paralgrm(C,R,B,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),paralgrm(A,K,D,L,E,M,F,N),paralgrm(B,R,A,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),paralgrm(D,K,C,L,E,M,F,N),paralgrm(A,R,D,S,F,T,C,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,quadril(A,W,B,X,C,Y,D,Z),paralgrm(C,K,B,L,E,M,F,N),paralgrm(D,R,C,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),quadril(B,K,A,L,E,M,F,N),paralgrm(C,R,B,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),quadril(A,K,D,L,E,M,F,N),paralgrm(B,R,A,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),quadril(D,K,C,L,E,M,F,N),paralgrm(A,R,D,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),quadril(C,K,B,L,E,M,F,N),paralgrm(D,R,C,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(B,K,A,L,E,M,F,N),quadril(C,R,B,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(A,K,D,L,E,M,F,N),quadril(B,R,A,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(D,K,C,L,E,M,F,N),quadril(A,R,D,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
sqrprism1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(C,K,B,L,E,M,F,N),quadril(D,R,C,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).

shape3D(A,parallelepiped):-sqrprism(A,B,C,D,E,F).

shape3D(A,parallelepiped):-paralepd1(A,B,C,D,E,F,G).
paralepd1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(B,K,A,L,E,M,F,N),paralgrm(C,R,B,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
paralepd1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(A,K,D,L,E,M,F,N),paralgrm(B,R,A,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
paralepd1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(D,K,C,L,E,M,F,N),paralgrm(A,R,D,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
paralepd1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(C,K,B,L,E,M,F,N),paralgrm(D,R,C,S,F,T,G,U),!,
                                 notshape,not(other4(A,B,C,D,E,F,G)).
```

```
shape3D(A,rhomboid):-rhombid(A,B,C,D,E,F).
rhombid(A,B,C,D,E,F):-init2D,rhomb(A,R,B,S,C,T,D,U),rhomb(C,W,B,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rhombid(A,B,C,D,E,F):-init2D,rhomb(A,R,B,S,C,T,D,U),rhomb(B,W,A,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rhombid(A,B,C,D,E,F):-init2D,rhomb(A,R,B,S,C,T,D,U),rhomb(A,W,D,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rhombid(A,B,C,D,E,F):-init2D,rhomb(A,R,B,S,C,T,D,U),rhomb(D,W,C,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,rhomboid):-rhombid1(A,B,C,D,E,F,G).
rhombid1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(B,K,A,L,E,M,F,N),rhomb(C,R,B,S,F,T,G,U),!,
                              notshape,not(other4(A,B,C,D,E,F,G)).
rhombid1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(A,K,D,L,E,M,F,N),rhomb(B,R,A,S,F,T,G,U),!,
                              notshape,not(other4(A,B,C,D,E,F,G)).
rhombid1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(D,K,C,L,E,M,F,N),rhomb(A,R,D,S,F,T,G,U),!,
                              notshape,not(other4(A,B,C,D,E,F,G)).
rhombid1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(C,K,B,L,E,M,F,N),rhomb(D,R,C,S,F,T,G,U),!,
                              notshape,not(other4(A,B,C,D,E,F,G)).

shape3D(A,rectangular-parallelepiped):-rectparalep(A,B,C,D,E,F).
rectparalep(A,B,C,D,E,F):-init2D,rectan(A,R,B,S,C,T,D,U),rectan(C,W,B,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rectparalep(A,B,C,D,E,F):-init2D,rectan(A,R,B,S,C,T,D,U),rectan(B,W,A,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rectparalep(A,B,C,D,E,F):-init2D,rectan(A,R,B,S,C,T,D,U),rectan(A,W,D,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
rectparalep(A,B,C,D,E,F):-init2D,rectan(A,R,B,S,C,T,D,U),rectan(D,W,C,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,rectangular-parallelepiped):-rectparalep1(A,B,C,D,E,F,G).
rectparalep1(A,B,C,D,E,F,G):-init2D,rectan(A,W,B,X,C,Y,D,Z),paralgrm(B,K,A,L,E,M,F,N),paralgrm(C,R,B,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,rectan(A,W,B,X,C,Y,D,Z),paralgrm(A,K,D,L,E,M,F,N),paralgrm(B,R,A,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,rectan(A,W,B,X,C,Y,D,Z),paralgrm(D,K,C,L,E,M,F,N),paralgrm(A,R,D,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,rectan(A,W,B,X,C,Y,D,Z),paralgrm(C,K,D,L,E,M,F,N),paralgrm(D,R,C,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),rectan(B,K,A,L,E,M,F,N),paralgrm(C,R,B,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),rectan(A,K,D,L,E,M,F,N),paralgrm(B,R,A,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),rectan(D,K,C,L,E,M,F,N),paralgrm(A,R,D,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),rectan(C,K,B,L,E,M,F,N),paralgrm(D,R,C,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(B,K,A,L,E,M,F,N),rectan(C,R,B,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(A,K,D,L,E,M,F,N),rectan(B,R,A,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(D,K,C,L,E,M,F,N),rectan(A,R,D,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).
rectparalep1(A,B,C,D,E,F,G):-init2D,paralgrm(A,W,B,X,C,Y,D,Z),paralgrm(C,K,B,L,E,M,F,N),rectan(D,R,C,S,F,T,G,U),!,
                                   notshape,not(other4(A,B,C,D,E,F,G)).

shape3D(A,rectangular-parallelepiped):-rhombid1(A,B,C,D,E,F,G).
```

```
shape3D(A,cube):-cube(A,B,C,D,E,F).
cube(A,B,C,D,E,F):-init2D,squar(A,R,B,S,C,T,D,U),squar(C,W,B,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
cube(A,B,C,D,E,F):-init2D,squar(A,R,B,S,C,T,D,U),squar(B,W,A,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
cube(A,B,C,D,E,F):-init2D,squar(A,R,B,S,C,T,D,U),squar(A,W,D,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).
cube(A,B,C,D,E,F):-init2D,squar(A,R,B,S,C,T,D,U),squar(D,W,C,X,E,Y,F,Z),!,notshape,not(other3(A,B,C,D,E,F)).

shape3D(A,cube):-cube1(A,B,C,D,E,F,G).
cube1(A,B,C,D,E,F,G):-init2D,squar(A,W,B,X,C,Y,D,Z),rhomb(B,K,A,L,E,M,F,N),rhomb(C,R,B,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,squar(A,W,B,X,C,Y,D,Z),rhomb(A,K,D,L,E,M,F,N),rhomb(B,R,A,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,squar(A,W,B,X,C,Y,D,Z),rhomb(D,K,C,L,E,M,F,N),rhomb(A,R,D,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,squar(A,W,B,X,C,Y,D,Z),rhomb(C,K,B,L,E,M,F,N),rhomb(D,R,C,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),squar(B,K,A,L,E,M,F,N),rhomb(C,R,B,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),squar(A,K,D,L,E,M,F,N),rhomb(B,R,A,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),squar(D,K,C,L,E,M,F,N),rhomb(A,R,D,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),squar(C,K,B,L,E,M,F,N),rhomb(D,R,C,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(B,K,A,L,E,M,F,N),squar(C,R,B,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(A,K,D,L,E,M,F,N),squar(B,R,A,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(D,K,C,L,E,M,F,N),squar(A,R,D,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).
cube1(A,B,C,D,E,F,G):-init2D,rhomb(A,W,B,X,C,Y,D,Z),rhomb(C,K,B,L,E,M,F,N),squar(D,R,C,S,F,T,G,U),!,
                      notshape,not(other4(A,B,C,D,E,F,G)).

shape3D(A,cube):-rhombid1(A,B,C,D,E,F,G).

shape3D(A,other):-tetra(A,B,C,D).

shape3D(A,other):-tetra1(A,B,C,D).

shape3D(A,other):-pyram(A,B,C,D,E).

shape3D(A,other):-pyram1(A,B,C,D,E).

shape3D(A,other):-truntripyr(A,B,C,D,E).

shape3D(A,other):-truntripyr1(A,B,C,D,E,F).

shape3D(A,other):-triprism(A,B,C,D,E).

shape3D(A,other):-triprism1(A,B,C,D,E,F).

shape3D(A,other):-trunsqrpyr(A,B,C,D,E,F).

shape3D(A,other):-trunsqrpyr1(A,B,C,D,E,F,G).
```

```
shape3D(A,other):-not(tetra(A,B,C,D)),not(tetra1(A,B,C,D)),
                  not(pyram(A,B,C,D,E)),not(pyram1(A,B,C,D,E)),
                  not(truntripyr(A,B,C,D,E)),not(truntripyr1(A,B,C,D,E,F)),
                  not(triprism(A,B,C,D,E)),not(triprism1(A,B,C,D,E,F)),
                  not(trunsqrpyr(A,B,C,D,E,F)),
                  not(trunsqrpyr1(A,B,C,D,E,F,G)),
                  not(sqrprism(A,B,C,D,E,F)),not(sqrprism1(A,B,C,D,E,F,G)),
                  not(paralepd1(A,B,C,D,E,F,G)),
                  not(rhombid(A,B,C,D,E,F)),not(rhombid1(A,B,C,D,E,F,G)),
                  not(rectparalep(A,B,C,D,E,F)),
                  not(rectparalep1(A,B,C,D,E,F,G)),
                  not(cube(A,B,C,D,E,F)),not(cube1(A,B,C,D,E,F,G)).
```

Appendix 5

AN APPLICATION OF THE AVERAGING-INTENSIFICATION OPERATOR

A spin-off of this project was the application of the averaging-intensification operator on digitized pictures. The Department of Mechanical Engineering working on digitized pictures taken from vibrating surfaces, needed a technique to obtain a clearer picture showing the fringes caused by vibration. The equipment that they used was similar to this described in Chapter 2 of this project, i.e. a camera to take the picture, a digitizer, and a tele-screen for projection. The only difference was that the result of the processed picture could be seen on the same screen.

So far the technique used was the *voting* technique in a 16×16 window. The results were not very satisfactory, because the window was very large and the middle pixel was modified according to a rather ununiform area of a not very clear - grainy - picture. Actually the modified picture turned out to be as grainy as the original. Of course the main objective was to distinguish the *'dark'* fringes from the rest of the picture, in order to illustrate the effect of the vibration.

The suggested method was that of intensification with constant threshold in a 3×3 window. This time the result was slightly better but not very encouraging, because the fringes were still not distinctly shown. But after a second and third successive intensifications with the same threshold, the dark fringes started appearing due to the fact that the gray levels were pushed towards the two extremes. Eventually after a number of applications only *black* and *white* pixels remained. A simple - two level - averaging then was enough to smoothen the picture.

Fig. A5.1 shows the original digitized picture. The fringes are easy for a human to detect but not for a computer. Fig. A5.2, the picture has been intensified with a constant threshold 7. The *black* area of the fringes and the bolt in the middle are more distinct now. Fig. A5.3, the picture after some successive intensifications show clearly now the existance of the fringes (upper half) compared with the original (bottom half).

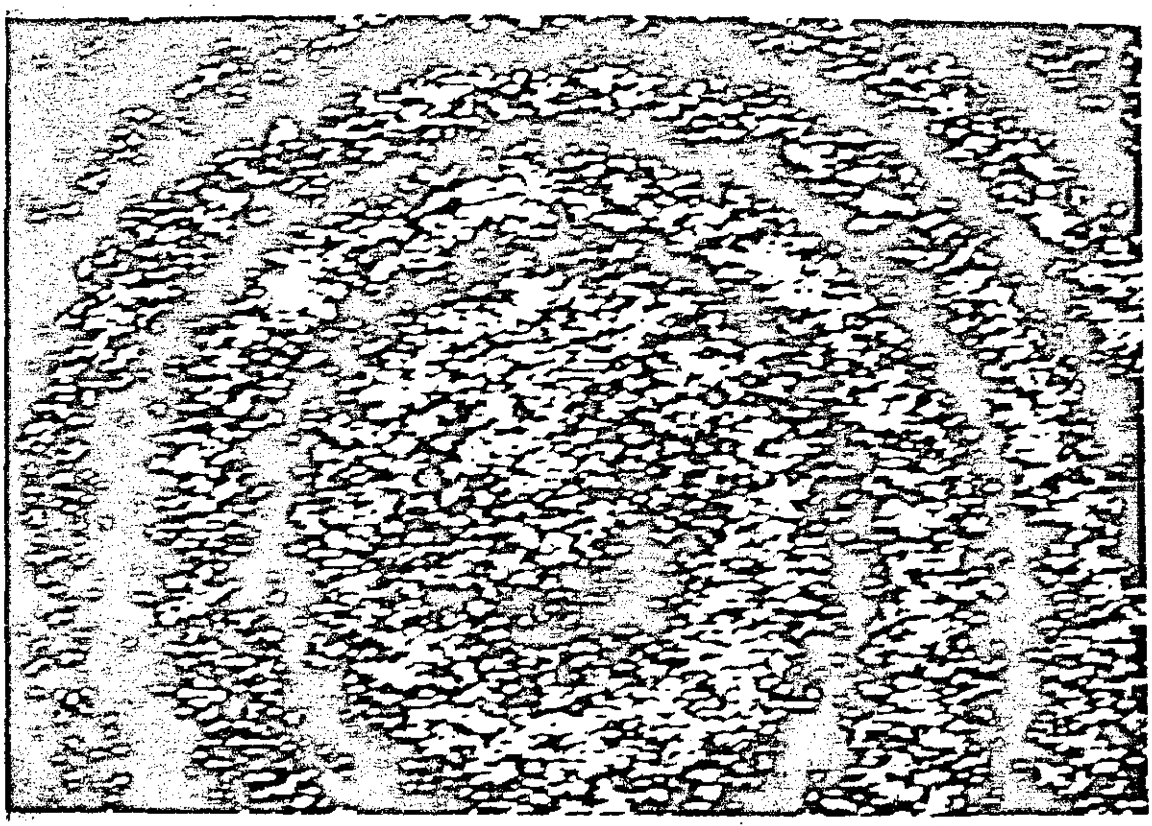


FIG. A5.1

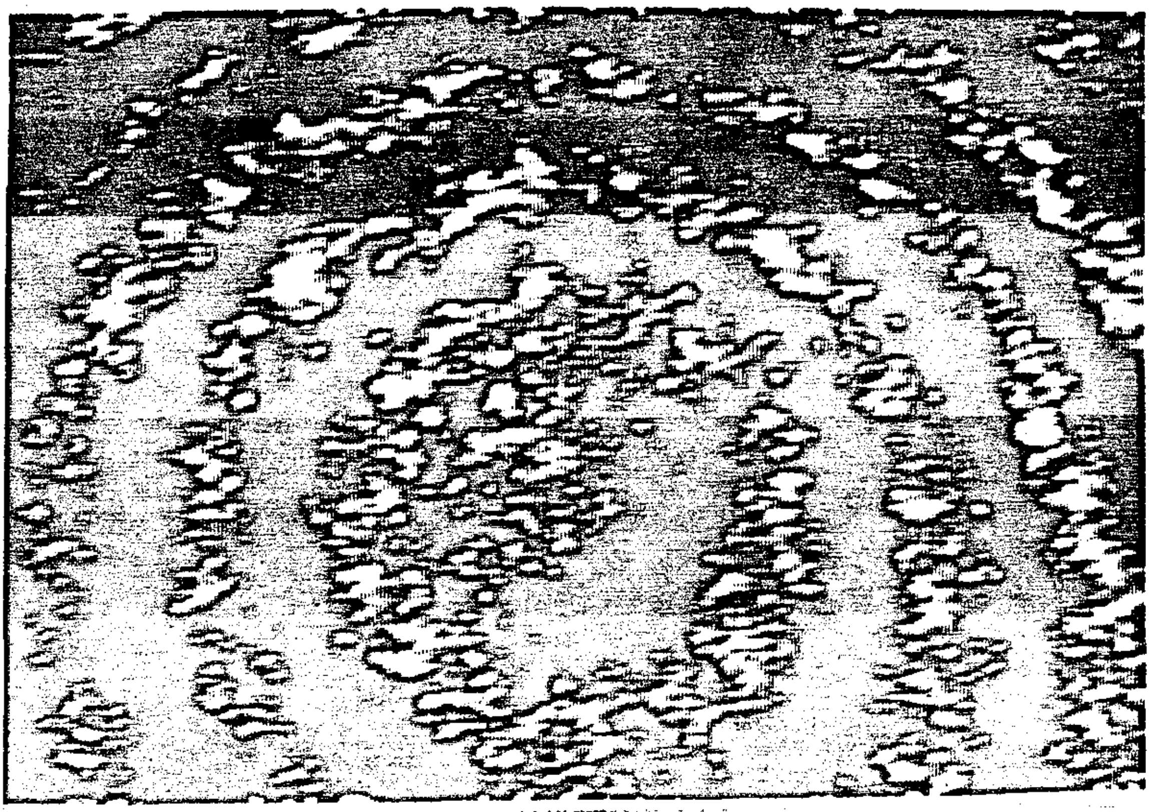


FIG. A5.2

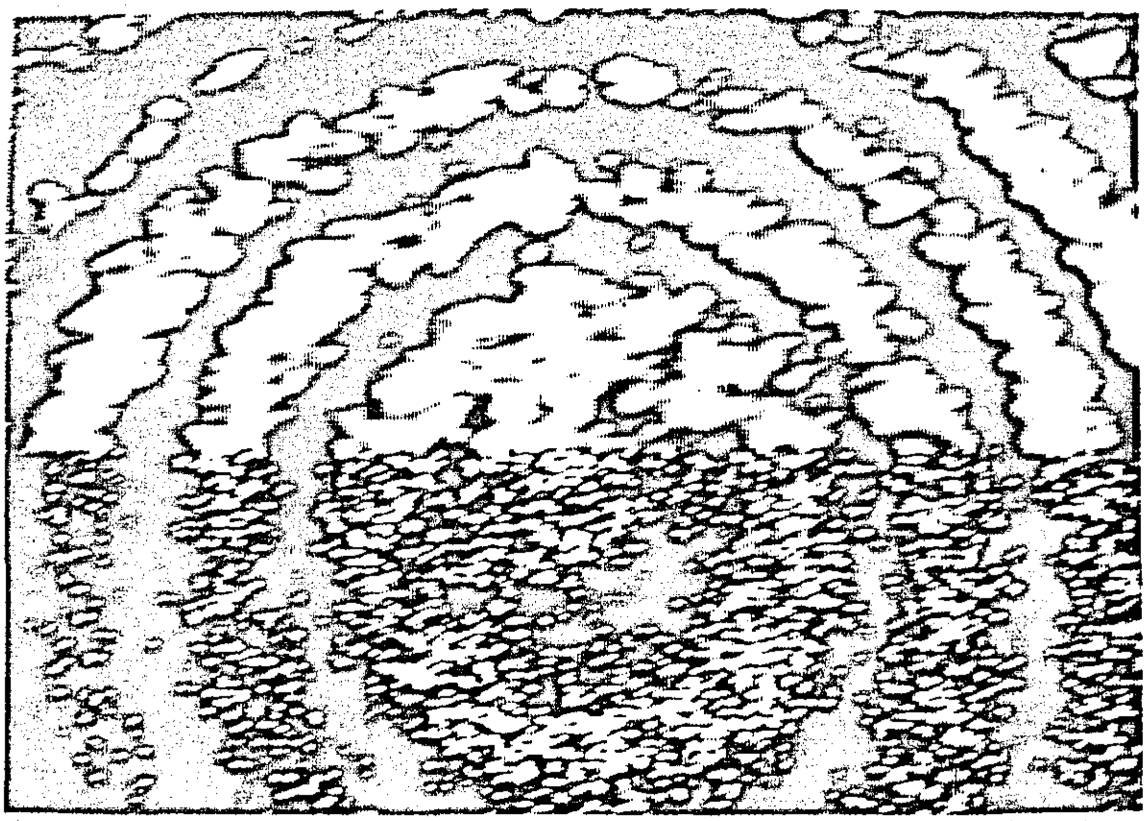


FIG. A5.3