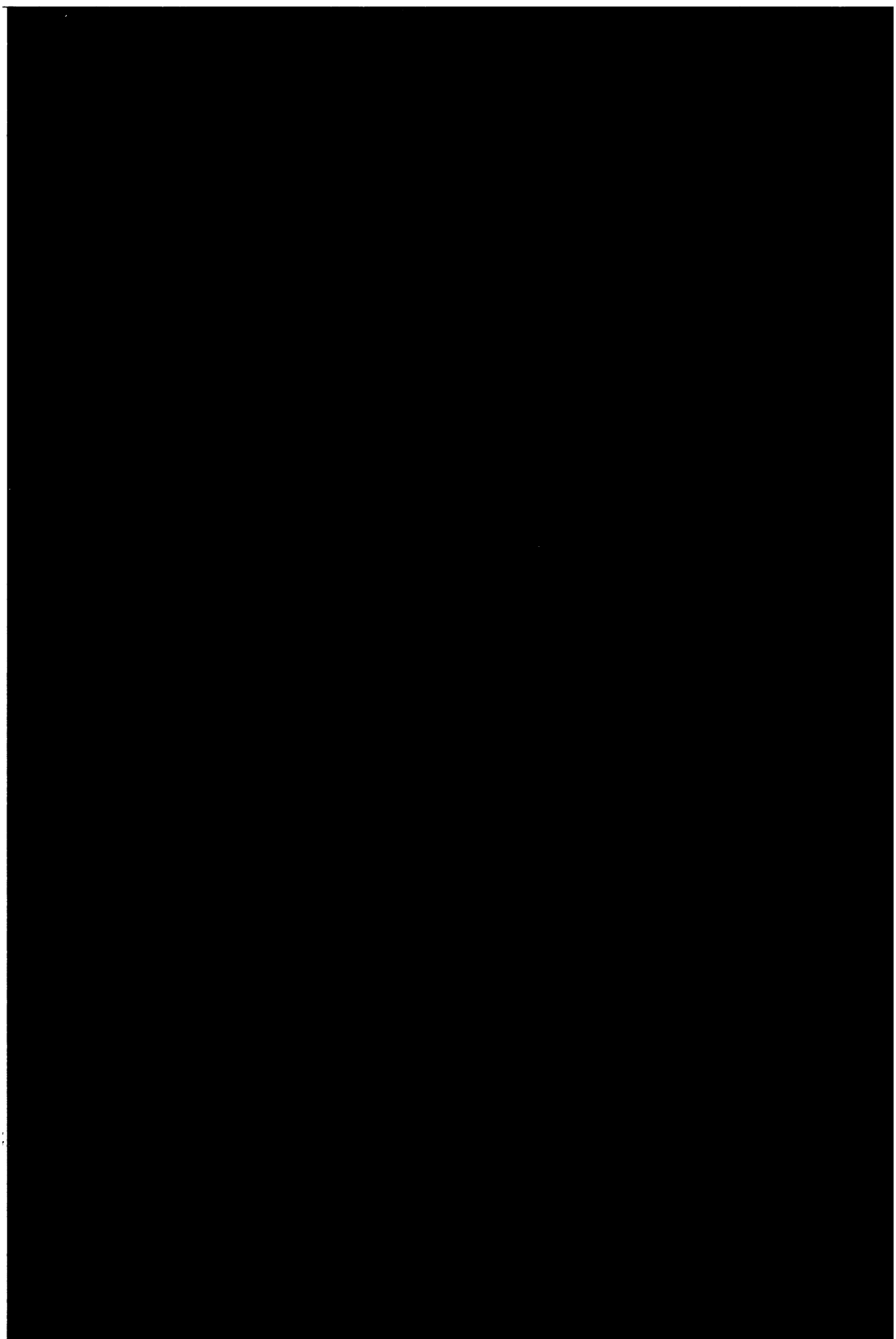


This blank page was inserted to preserve pagination.

SOME ASPECTS OF PATTERN RECOGNITION BY COMPUTER

by

ADOLFO GUZMÁN - ARENAS

B. S., Instituto Politécnico Nacional (México), 1965;

Submitted in partial fulfillment of the
requeriments for the degree of
Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February, 1967.

Signature of Author.....
Department of Electrical Engineering.
January 15, 1967.

Certified by.....Marvin Minsky.....
Thesis Supervisor.

Accepted by.....Truman S. Gray.....
Chairman, Departmental Committee on Graduate Students.

SOME ASPECTS OF PATTERN RECOGNITION BY COMPUTER

by

Adolfo GUZMAN Arenas

Submitted to the Department of Electrical Engineering on
January 15, 1967, in partial fulfillment of the requeriments
for the degree of Master of Science.

ABSTRACT

A computer may gather a lot of information from its environment in an optical or graphical manner.

A scene, as seen for instance from a TV camera or a picture, can be transformed into a symbolic description of points and lines or surfaces. This thesis describes several programs, written in the language CONVERT, for the analysis of such descriptions in order to recognize, differentiate and identify desired objects or classes of objects in the scene. Examples are given in each case.

Although the recognition might be in terms of projections of 2-dim and 3-dim objects, we do not deal with stereoscopic information.

One of our programs (Polybrick) identifies parallelepipeds in a scene which may contain partially hidden bodies and non-parallelepipedic objects. The program TD works mainly with 2-dimensional figures, although under certain conditions successfully identifies 3-dim objects. Overlapping objects are identified when they are transparent.

A third program, DT, works with 3-dim and 2-dim objects, and does not identify objects which are not completely seen.

Important restrictions and suppositions are: (a) the input is assumed perfect (noiseless), and in a symbolic format; (b) no perspective deformation is considered.

A portion of this thesis is devoted to the study of models (symbolic representations) of the objects we want to identify; different schemes, some of them already in use, are discussed.

Focousing our attention on the more general problem of identification of general objects when they substantially overlap,we propose some schemes for their recognition, and also analyze some problems that are met.

Thesis Supervisor: Marvin L. Minsky

Title: Professor of Electrical Engineering.

ACKNOWLEDGMENTS

The author wishes to express his appreciation to the Vision Group of the Artificial Intelligence Project at Project MAC, M.I.T., and in particular to Professor M. L. Minsky and Dr. S. A. Papert.

Some of the programs described in this thesis were done by the author while working at Computer Corporation of America, Cambridge, Mass.

The author was partially supported by a grant from the Instituto Nacional de la Investigacion Cientifica (Mexico).

Work reported herein was also supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

TABLE OF CONTENTS

CHAPTER		PAGE
	ABSTRACT	ii
	ACKNOWLEDGMENTS	iii
	TABLE OF CONTENTS	iv
I	EXPOSITION	1
II	PROBLEMS FOUND	3
	Occlusion	4
	Degenerated positions	5
	Hidden lines	5
	Spurious and bad regions	6
	Duplicated regions	8
	Shadows	9
	The α-λ transformation	10
	Spurious points	12
III	PREPROCESSING	13
	Equal intensity contours	14
	Line followers	15
	Stacks of succesive transformations	17
	Regions.	19
	Syntactical analysis of figures	20
IV	POLYBRICK	
	Introduction	22
	Input to the program	22
	Format of the answer	24
	The programs	24
	CUBES2 LISP: Use of neighborhood	24
	Shortcomings of CUBES2	25
	What to erase	25
	CUBS LISP: Classification of the corners	28
	Shortcomings of CUBS	33
	CUBA LISP: Differentiating among parallel cubes	33
	Collinearity is not sufficient	35
	Numbering the Y's. Unit distance	35
	Preprocessing	36
	Localization of the cubes	36
	CUBE LISP	37
	Recognition of cubes in a picture which also contains other objects	38
	Some examples	39

V TD 43

The figure description language FDL-1 44
Introduction 44
The language 44
Elementary figure 45
Properties 47
Attachment of properties 48
Open variables in properties 49
Composition of figures 50
Definition: single names 50
Definition of new properties 52
The program 54
What the program does 54
Examples 57
Example 1. P27 57
Example 2. SQUARE 60
Example 3. XS 60
Example 4. CHEMIS 66

VI DT 67

An example of recognition 68
Restrictions 68
The scene 70
Example 70
The model 71
Example 72
The results 74
Example 1. EX2 74
Example 2. FIG2 77

VII MODELS 80

2-dimensional representation of 3-dim models 81
Models written in edge-notation 81
Models written in region-notation 82
Multiple models for the same object 83
First approach: Multirepresentation 84
Second approach: two-dimensional patterns 85
Uni-dimensional (CONVERT-type) patterns 88
2-dim models 89
Differences and similarities between the two representation of models 91
2-dim patterns (CONVERT-type models) 92
Semi-models 93
Union of patterns 94
A different representation 95
Third approach: three dimensional transparent models (edge-representation) 96
3-dim models projected into 2-dim patterns 98
Numerical models 98
Curved objects 99

VIII DISCUSSION OF SOME SCHEMES FOR RECOGNITION 100

One-to-one matching 101
Implementation 101
Linear weighting 102
Weights 102
Sub-weights 103
Conclusions 104
The generalized region (gregion) approach 106
Generalities 106
Definitions 107
Marking the boundary of regions 109
Unreliability of DOTS 111
Classification of regions 112
Merging regions into gregions 113
Comparison: the job of the matcher 114

REFERENCES 116

CHAPTER I. E X P O S I T I O N

The goal.- Given a scene, as seen for instance from a TV camera or a picture, it is desired to analyze it in order to recognize, differentiate and identify desired objects or classes of objects (i. e., patterns) in it.

The problem.- A picture, scene or view is read with the help of an optical device and stored as an array of light intensities in the memory of the computer. The ultimate goal will be to understand this information, that is, to identify, separate and position the different objects or bodies belonging to the scene(s). The demands of information will vary: sometimes we will be interested in knowing if an object is seen in the scene or not, while at other times we may require a complete description of the scene, including information on relative support and (3-dim) position of the different components. Hence it is clear that the recognizer will need an additional input to specify the nature of the question that the program is to answer by analyzing the scene.

Some work has been done by the author, specifically in the area of "recognition" (see below). This thesis describes the general problem, its difficult points, possible solutions, and specific attempts by the author and also by some others.

The work is divided in two parts: preprocessing, which converts the input into symbolic data, and recognition, which studies these data and, with the help of a model of the object we are searching for, finds all instances of that object in the scene in question.

The different chapters of this thesis.- The array containig the scene is swept and transformed by the preprocessor (chapter 3), which converts the picture in a more compact (and perhaps symbolic) form of information. Sometimes a syntactical analysis (end of chapter 3) of this data is enough to recognize the objects we are interest in. In general, the problems found (chapter 2) require the use of more sophisticated weapons. Very often it is necessary to specify a model (chapter 7) of the objects we want to find, and a considerable part of this thesis (chapter 7) is devoted to the different models and their characteristics. Some schemes for recognition are proposed and discussed in chapter 8, using the models of chapter 7 and assuming we have the scene preprocessed as in chapter 3; problems are taken into account as in chapter 2. Finally, three particular schemes were implemented, and are described in chapters 4, 5 and 6.

Contents.

Chapter 1. Exposition (just done).

Chapter 2. Problems found

Chapter 3. Preprocessor.

Chapter 4. Polybrick.

Chapter 5. TD

Chapter 6. DT

Chapter 7. Models

Chapter 8. Discussion of some schemes for recognition.

CHAPTER II. P R O B L E M S F O U N D

This chapter will list a number of important problems present for any 3-dim recognition system. Some of these problems are discussed in the chapters which cover Polybrick, TD and DT; others are only mentioned here or slightly discussed.

Solutions, approachs and lines of thought are given when available. In particular, some of the problems encountered by the recognizer are treated in the chapter about models (chapter 7).

These problems generally fall in two cathegories: are either general, or caused by the particular method or approach. It should also be mentioned that this chapter makes a description, rather than an evaluation, of some of the ways to solve the problems found.

No hardware difficulties are discussed.

Occlusion.- Since objects in the scene may be partially behind anothers, the recognizer has to be able to find instances of a given object even when only a part of it is actually seen in the picture.

Small parts of an object. If an object is totally occulted in the scene except for a small part of it, identification becomes difficult and ambiguous. Here, the recognizer could use context or statistical information[1] to resolve the ambiguity, or to report the small part as being a portion of one of several possible objects. Note that this problem is one of lack of enough information.

For instance, Polybrick[2] --somewhat arbitrarily-- decides to identify as cubes (parallelepipeds) corners of the form A B C D (see fig. 'AMBIGUOUS').

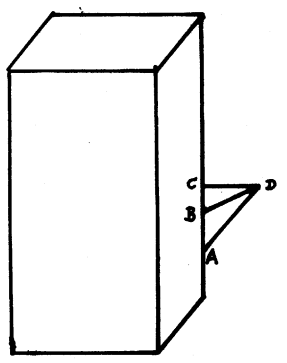


Fig. 'AMBIGUOUS'. The corner A B C D may belong to several objects of different shapes.

[1] As done, for instance, in W. W. Bledsoe and I. Browning [2].

[2] See chapter 4 of this thesis.

Degenerate positions.- Probably most recognizers will fail to identify the objects in figure 'CONES' as cones.

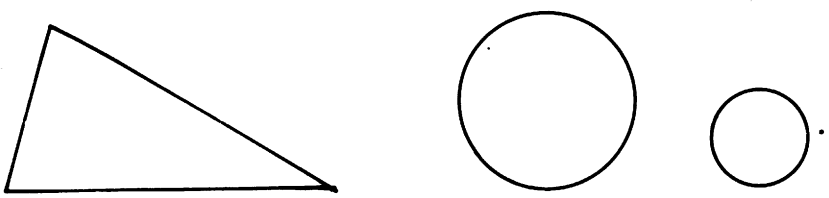


Fig. 'CONES'. Degenerated positions are difficult to deal with.

Fig. 'CONES' also shows that there are degenerate positions with non-zero probabilities. A major finesse will be required from the preprocessor and its surface functions (see chapters 3 and 6) in order to get the hint "this is a degenerated case". Other kind of information may also help: shading, shadows, knowledge of support structure, etc. If the recognizer has no idea that it is dealing with this case, it can do little to identify correctly the body, unless the frequency of these cases is such that special software is devoted to them. Once the recognizer suspects a degeneracy, the special machinery is used upon it.
Heuristic: watch out for isolated single regions surrounded by background or not otherwise explained.

Hidden Lines.- In chapter 7, talking about transparent or 3-dimensional models, we assert that we must know what lines or regions of an object are hidden by the same object, with respect to the different views of such a body.

Perspective.- Parallel lines are no longer parallel, but they converge at the horizon... Accurate measurements[1] have to be made if we want tc use this information in order to know the position of the objects with respect to the observer.

Polybrick, DT and TD ignore this problem, under the assumption that we are working with small objects and/or far from them.

Spurious regions.- In using different surface-functions or predicates for finding good regions (see in chapter 3, the section "the 'summer vission group' approach"), there may be overlapping among the found regions, duplicated regions, bad regions, etc. (discussed below).

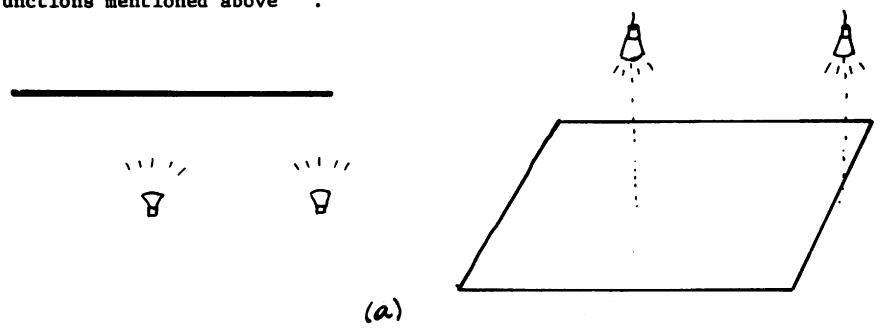
Bad regions.- Almost any surface function will occasionally find a region which is considered "bad", in the sense that it does not match exactly with the outline of the face to which it corresponds. It is well known, for instance, that intensity countour levels of a scene do not follow closely the outlines of the objects [21]; over a flat surface, they get ugly distributions, as fig. 'LEVELS' indicate.

Problems to be solved by the executive or the recognizer are what function to employ in each particular case, or else how to decide if the produced region is acceptable. Feedback between the recognizer and the preprocessor (see "the generalized region approach" in chapter 8) is needed at this point. Read also chapter 7 to see how models get involved.

It will be a good idea to have a stack of functions useful in particular conditions; their utility could be further increased if we are able to compose them, that is, to apply one function to the result of

another. Chapter 3 talks about this.

Also, it will result worthwhile to have an easy framework to test different predicates manually, in order to collect the stack of functions mentioned above[1].



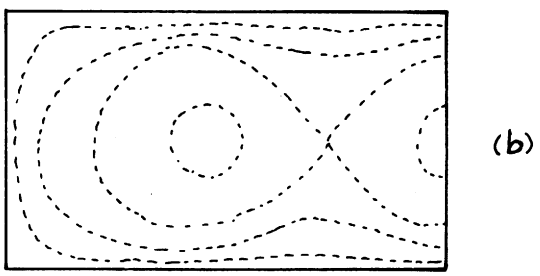


Fig. 'LEVELS'. (a) A plane is illuminated by two concentrated sources. (b) Equal intensity curves. If we use a cutoff value of intensity to identify the region, our result will have little resemblance with a rectangle.

[1] Towards this direction is EYE [30].

Overlapping regions.- The big repertory of functions (in the "region approach, chapter 8) suggests that the same region could be found more than once and, more over, that different functions when applied to the same face or zone will in fact return different regions; if there is little discrepancy in the boundary of two regions which otherwise have the same center of figure, extension, etc., we could conclude that they are the same and keep the more reliable one.

If two regions overlap considerably but one is significantly bigger than the other, we may suppose that they are composed of smaller regions, and that we should subdivide them, using a more delicate surface function. Our hypothesis could be tested by taking the intersection of these two regions and sending an speciallized feature-seeker (see chapter 3) to find out if the region formed by the intersection could be detected in a different manner.

Duplicated regions.- These are overlapping region s whose discordance is small. Since their boundaries are not exactly equal, we still have to find a criterion for choosing the best boundary.

Regions which are not there.- (Highlights, shadows, reflections, etc.) A number of regions will be found which are not 'actually' there. Small regions caused by camera noise can be eliminated due to their smallness[1]. As we point out in the paragraph "bad regions", wrong surface functions are

---

[1] This is done by Larry Krakauer [21]. Note, however, that his program is not designed with the idea of finding "good" regions.

the main responsable of these monstrosities.

It is possible that the curved surface of the cylinder in figure 'CYLIN' be reported as two, if we use as function the constancy of intensity, and even if we use the constancy of variation (first derivative). The recognizer should be aware of this possibility.

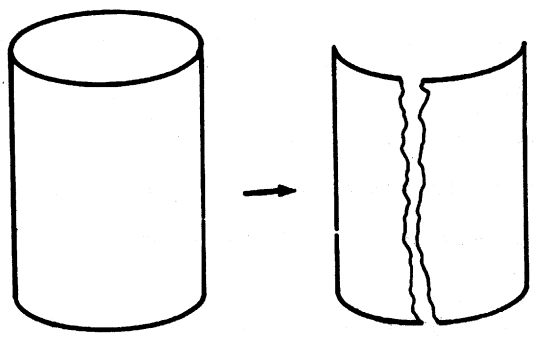


Fig. 'CYLIN'. Example of regions which are not there. If the surface function is simple enough, two regions may be found where there is only one.

Non-interesting regions.- When seeing a box with small letters written in its sides, or a piece of wood showing its grain, the preprocessor tends to find a multitude of small regions, which are un-interesting to the recognizer. We have to recognize them as "non-important" because of their smallness, regularity, or some more appropiate property, and to use their position to help to construct the 'real' region containing them --the interesting one-- , and finally to ignore them.

Shadows.- Shadows over the surfaces of bodies in process of identification complicate this task, although they may reveal information about the shape

of surfaces. A good way of discriminate them is to use as surface function the composition of the light, that is, the ratio of some color to the total lumens/m^2, instead of the plain intensity (assuming we have color perception).

Spurious Lines.- A line-follower[1] could get trapped into spurious lines, and the same applies to the region finder (see chapter 3), when it has to work with noisy input.

Such spurious lines can be eliminated by their short length, and on a higher level by the fact that they do not "fit" into the boundary of a shape for which there is good, independent, evidence.

The α-λ transformation.- The following scheme[2] is useful in detecting undesired lines when dealing with rectilinear bodies. Given an array containing elementary segments (a small number of points plus a direction associated to them), as indicated in figure 'LITTLE SEGMENTS', we associate with each segment a pair of numbers, α is the angle that this segment forms with the x-axis, and λ is the distance of the (extended) line from the origin.

That is to say, we convert the figure to an array of points (see figure 'CLUSTERING'). In the α-λ space, points which fall close together

(1)The following programs are typical line-followers, and are confronted by the mentioned problem: 1. Sides 21 [10]; 2. Polygon detector [23].

(2)This scheme is used by Nilsson at Stanford Research Institute in the visual part of their robot.

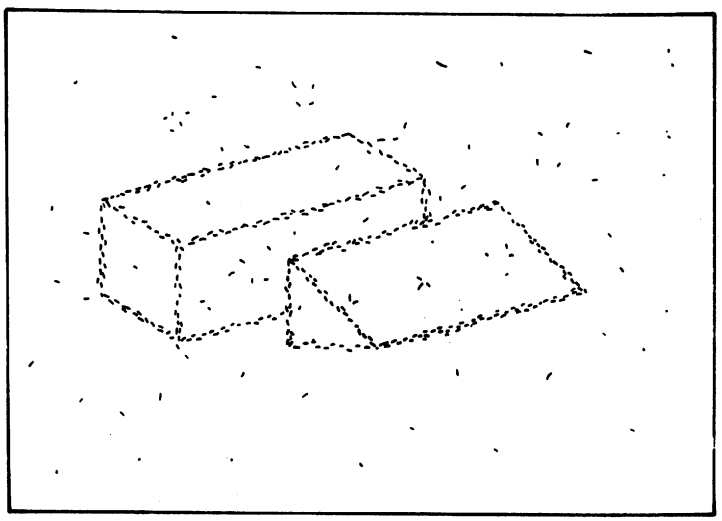


Fig. 'LITTLE SEGMENTS'.
A scene after processing by a gradient operation. Irrelevant segments have to be taken out.

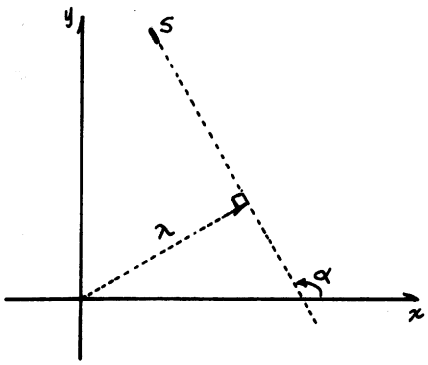


Fig. α-λ. A little segment S is represented by the pair(α, λ).

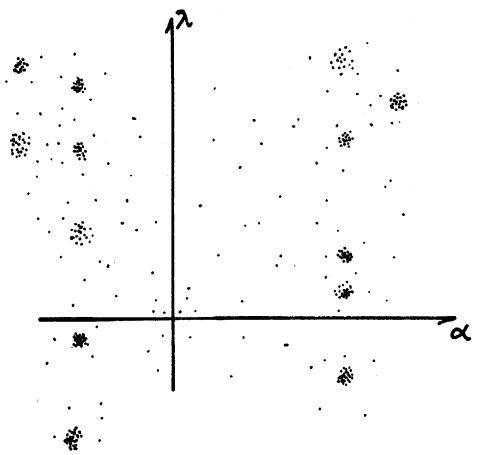


Fig. 'CLUSTERING'. Clouds with the same α are parallel sides.

are over the same line, so that frequency count will eliminate the spurious segments, as desired. Clouds with the same α are parallel lines, and this fact could be used in order to look for parallel lines.

Smooth curved lines could also be detected by this method, if we use a fancier criteria for the detection of clusters[1].

Spurious Points.- Often enough, after the application of some surface function (again, we are assuming the region-approach) to some part of the scene, the result will exhibit some isolated or irrelevant points, which have to be elliminated from the region. These points could be swept out by averaging and then having a treshold. There are also the so-called noise-eliminators, line-thinners, and so on, widely used in the field of character-recognition[2]. Disturbances coming from noise in the camera could be mitigated by reading the same spot several times and averaging. This technique wastes time.

Range of brightness.- Whem comparing brightnesses, it is advisable to use their ratios, or differences between logarithms. This helps make the system invariant to changes in illumination levels.

---

(1) Work in this area is: 1. Evan L. Ivie. PhD Thesis [18].
2. Probably a conventional pattern clasifier will do it. See N. J. Nilsson [26].

(2) A good collection of references is in [9]. See there [1].

CHAPTER III. P R E P R O C E S S I N G

This chapter will cover some schemes that might be used to preprocess pictures before we can use our "symbolic description" recognition methods.

The preprocessor is responsible for taking the scene as read into memory --generally as an array of numbers which correspond to the intensity or brightness of the points in the picture, scene, film, etc.-- and transforming it into a smaller but more usable amount of information, usually as a highly organized description, in symbolic format perhaps, of points, regions, lines, surfaces.

The main goal of a preprocessor is to throw away as much information as it can, while at the same time to keep the relevant facts in an organized structure. Most of them perform a local operation over a point and its neighbors, producing an output that depends only in the values of the intensity in a small neighborhood.

EQUAL INTENSITY CONTOURS (global threshold algorithms)

The CNTOUR program.- This program [21] plots an intensity relief map of an image which is read from the vidisector camera (TV-B) attached to the PDP-6 computer. For high-contrast images, it produces something like a line drawing.

A contour is a set of closed curves enclosing all those points in an image whose intensity is greater than a specified threshold. These contours correspond to the contours of a relief map, and not to the boundaries of an object. Thus, except for high-contrast pictures, equal intensity contours do not match (do not follow closely) the boundaries of a region.

Local threshold.- Something may be gained if, instead of cutting at a pre-set global threshold, we make a histogram (fig. 'HISTOGRAM') or frequency count of the scene under consideration. If this is a sharp-contrast figure, significant peaks will be found, and then we may put thresholds in the valleys (see fig. 'HISTOGRAM').

The output of these programs has to be fed to a line-fitter, in order to get numbers, slopes, etc., out of the lines.

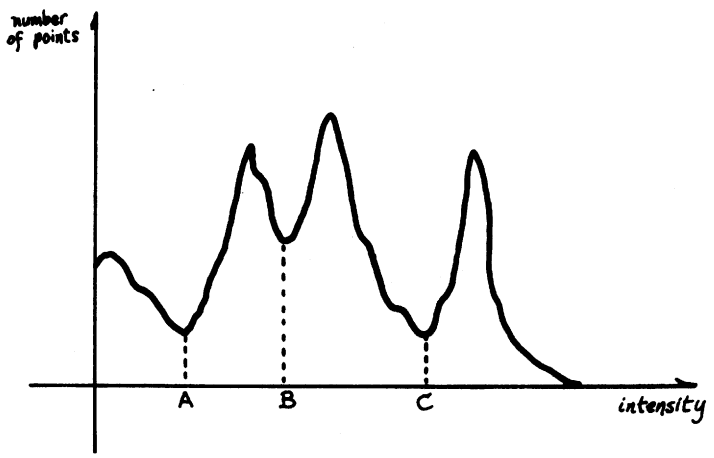


Fig. "HISTOGRAM". Local thresholds could be put at A, B and C.

LINE FOLLOWERS

When there exists a sharp contrast between the different surfaces, it is usually possible to follow fairly well the boundaries of two zones which differ greatly in brigthness, using what is called a line follower. This program sends a probe that travels the scene looking for a place where the intensity changes abruptly, and then travels along this change or discontinuity; in order to achieve this, we may think of it (the probe) as having two legs, and each one is kept in a different zone. This should be taken only as a pictorical description.

The output of a line fo-lower is a set of lines, which often has to be processed slightly more, in order to elliminate very small lines, and in order to merge several fairly collineal segments[1]. For instance, it may be difficult to find the exact ubication of the corners (place

where two lines intersect); one can instead follow the lines until near the corner and then, after the complete set of lines is found, use an analytical interpolation to find the "best" corners(2).

From the many line followers that exist, we will present two.

Sides 21.- This program [10] for the PDP-6 computer uses a box with a zone of tolerance (see fig. 'BOX'), the width and length of the box being functions of the length of the portion of line already found and of the noise present in the picture. It uses CORNS(2). The program searches for the maximum gradient, the relative location of the maximum gradient

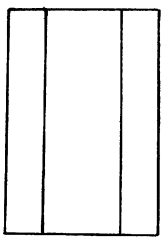


Fig. 'BOX'. The innermost rectangle is termed the acceptance box; the outer two rectangles are collectively termed the looking box. When tracking, the less sharply defined the edge is, the wider the box must be to successfully track it.

with respect to the box is also known to the program, and this information is subsequently used to steer the box. If the line is within the acceptance box, the program considers that the correct location of the line has been found, and thus will track further. In the case of a noisy edge, the box is widened as a function of how far the maximal gradient goes astray. 90% of the points must be inside the acceptance box before the box will extend. When the program believes it has arrived at a corner, it will move the box in several directions to find the possible emerging lines.

---

(1) See at this respect 'spurious lines' in chapter 2; more detailed information is found in Cyclops-1[24].

(2) CORNS [32] is a program that does this.

Finding the edges of a polygon.- Mann [23] uses also an edge detector in order to find the different sides of a rectilinear polygon; it first searches in a rectangular grid, until it observer an abrupt change in the intensities. Noting this point, it continues until the intensity is stable again. It returns the average of these points as the edge.

STACKS OF SUCCESIVE TRANSFORMATIONS(1)

We may transform pictures, that is, arrays of intensities (array of numbers) into new arrays of numbers which, generalizing, could also be considered as pictures. So, we would have functions which transform pictures into pictures, and we could stack them , that is, compose them.

The following table shows some of the different possibilities.

FROM	TO			
	Intensity pictures	point pictures	line pictures	symbolic descriptions
Intensity pictures	averaging. gradient. laplacian. region finder. color.	threshold operations. Kirsch package.	contours. edge detectors.	
Point pictures		smoothing. noise eliminators. Kirsch PAX.	line fitters.	
Line pictures			projections. line fitters.	White's program.
Symbolic description	Martin's display.	Display programs.	Display programs.	TD. DT. Models to patterns compiler.

In this table, the lower triangle ◺ corresponds to display; the upper triangle ◹ , to preprocessing.

Now, a preprocessing may be considered as a path between the upper left square and the lower right square; typically, we transform intensity pictures into intensity pictures several times, and then apply a transformation from intensity pictures to point pictures, etc. (see diagram below). Nevertheless, due to the fact that "local preprocessing" is expensive (time consuming), the preprocessing could be under 'global' control --more complicated loops would appear in the diagram below-- so that only difficult regions are transformed much. This is discussed somewhat in 'the generalized regions approach', in chapter 8.

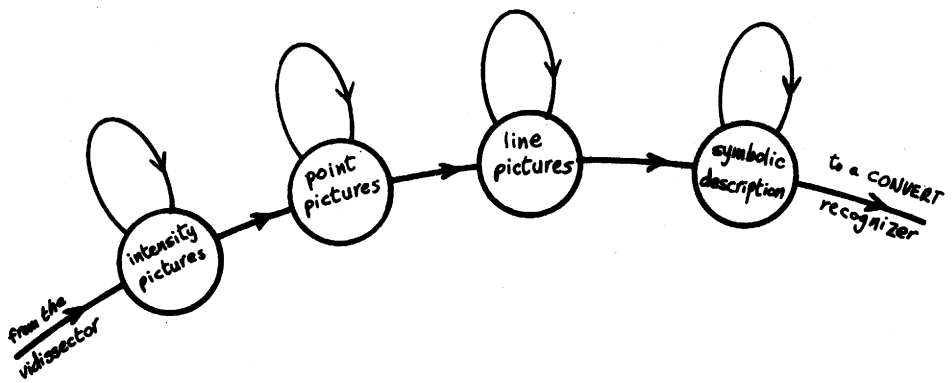


Fig. 'GRAPH'. Typical flow-chart of a preprocessor.

(1) I got this idea from T. Marill.

SOME OTHER METHODS.

Laplacian. Gradient. They are local operations. See [5].

Ridge detectors.- See [29].

Logical (boolean) preprocessing.- See Kirsch [20].

A. Rosenfeld and J. L. Pfaltz.
J. ACM 13, 4; pp. 471-494 (oct 66).

Hodes line follower.- See [17].

REGIONS. THE "SUMMER VISION GROUP" APPROACH[27].

A program[33] sweeps the array containing the scene and collects sets of points satisfying a given predicate; these sets are called regions, and roughly correspond to the different faces of the objects. It is entirely possible, but it is undesirable, that two or more regions will be reported as one. To find good regions is not a trivial task.

Another program[22] will drive the region-finder, supplying 'good' predicates; the boundaries will be sorted and "smoothed", and the bad regions elliminated and/or merged.

A further preprocessing will then be done[37, 38], interpolating straight lines, segments of curves, etc., until finally each region is described by a set of properties, in the so-called region-notation (see chapter on models).

This input is the one which the recognizer (for instance, TD[16]) will use.

Syntactical analysis of figures.- We cite the following references:

1. Ledley, R. S., and Wilson, J. B. Automatic Programming languages translation through syntactical analysis. Comm. ACM 5, No. 3. March 1962.

2. Ledley, R. S., Rotolo, L. S., Belson, M., Jacobsen, J. Pattern recognition studies in the biomedical sciences. SJCC, 1966, vol. 28, p. 411-430.

3. Narasimhan, R. Labeling schemata and syntactic descriptions of pictures. Inform. Control 7 (1964), 151-179.

4. Hodes, L. [17].

5. Cyclops-1 [24].

CHAPTER IV. P O L Y B R I C K

I am presenting in this chapter a description and discussion of Polybrick. This is a program that recognizes 3-dimensional parallelepipeds (solids limited by 3 pair of parallel planes), using as data 2-dimensional orthogonal projections.

Under the name CUBE LISP, a version of this program is running in the CTSS 7094 Time Sharing System of Project MAC, MIT. A more complete description and a listing of the program is found in a MAC memo [13].

Polybrick is written in the CONVERT language [14].

Introduction.- The programs contained in this chapter solve the following problem:

A scene contains noise free parallelepipeds without perspective effects, but partially occulting one to others. Extraneous rectilinear objects other than parallelepipeds may be present.

Problem: what parallelepipeds (hereafter called sloppily "cubes") are there and where they are (in the 2-dim picture)?

An answer is considered bad when it misses some cube, or if it confuses some. On the other hand, ambiguous cubes or partially-identified ones should be reported as such.

The program should also give the position of such cubes, to the extend such information is available.

Input to the program.- Eventually the program will read its data directly from the world[1]. Right now, the picture is transformed (by hand) to a list of corners and points of intersection (real or virtual) of lines, and their --two dimensional-- coordinates in the picture, together with its nearest adjacent points.

For example, the input associated with fig. '#CUBE' is

(A (B F) B (A G C) C (B D) D (G E C) E (F D) F (A E G) G (F B D))
[1, 2] [2,1] [4,1] [4, 2] [3, 3] [1, 3] [2, 2]

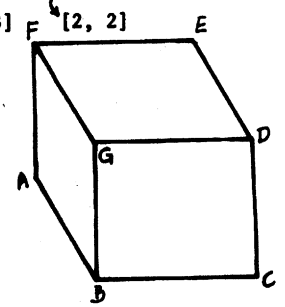
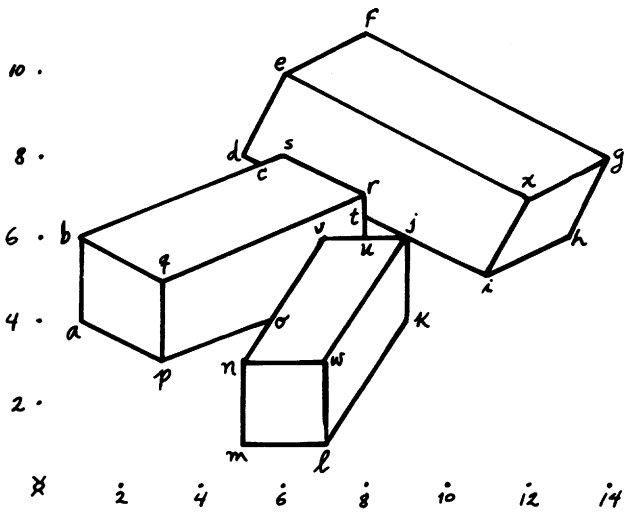


Fig. '#CUBE'. A cube showing its vertices.

(1) See chapter on preprocessing.



Vertex	Coordinate X	Y	neighbors
A	1	4	B P
B	1	6	A Q C
C	$5\frac{4}{9}$	$7\frac{7}{9}$	B D S
D	5	8	E C
E	6	10	D X F
F	8	11	E G
G	14	8	X H F
H	13	6	I G
I	11	5	X H J
J	9	6	I T U WK
K	9	4	J L
L	7	1	K W M
M	5	1	N L
N	5	3	O W M
O	5.7213	4.0909	P V
P	3	3	A Q O
Q	3	5	B P R
R	8	7	Q S T
S	6	8	C R
T	8	6.5	R U J
U	8	6	T V J
V	7	6	U D
W	7	3	N J L
X	12	7	I G E

Fig. GORDO

To its right is its description list, the input to the program.
The numbers are stored in the property list of each vertex.

Format of the answer. We use the CONVERT processor and apply the function cube (in the file "CUBE LISP") to the picture GORDO (in file "gordo"). Here is the operation in CTSS.

```
      load (( a cube gordo))
        (CERO UNDECLARED)
        (CERO UNDECLARED)
        NIL
                                        e (cubs gordo)
( THERE ARE AT LEAST 3 OR 3 CUBES)
( CUBE 1  IS(N ( W O M) W (N J L) L (M K W))  )
( CUBE 2 IS (I (J H X) G (F X H) X (E G I) E (X F D))  )
( CUBE 3 IS (P (A O Q) R (S Q T) Q (B R P) B (Q C A))  )
```

THE PROGRAMS.

They are written in CONVERT, a pattern-driven symbolic transformation language [14], and we will discuss here the following:

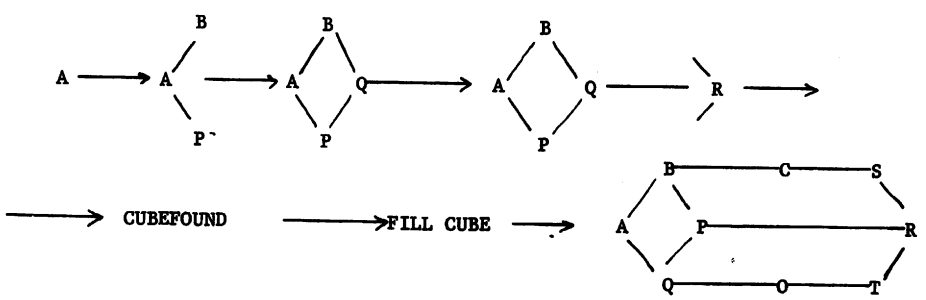
CUBES2	LISP	000	Original,uses continuity.
CUBS	LISP	000	Partitions the set into disjoint classes
CUBA	LISP	000	Final version; uses the unit distance method.
CUBE	LISP	000	Breaks ≺ into ≺≺ (not conected)

The last one is the one currently in use, but it is interesting to talk about all of them.

CUBES2 LISP Use of neighbourhood .

If a corner (≺) is found, we look for a parallelogram (◇) which has that corner (we use here the information about which points are joined to which); as usual, solid arrows in the flow chart indicate the direction of success; broken ones, the direction of failure.

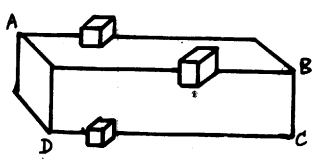
For example, in fig GORDO, CUBES2 proceeds in this way :



Now it tries again : it finds all the 3 cubes.

Shortcomings of CUBES2

The scheme just presented gives an idea of the power or weakness of CUBES2. It is able to find connnected cubes; for example, it solves fig GORDO and fig COMMON, but it fails to find A B C D in the figure at the right because it is formed of two disconnected parts (disconnected it the sense that, in order to go from one part A D to the other B C, we have to cross other cubes).



What to erase and what to leave

Once all the points of a cube are found, we have to delete it from the picture, in order to process the remainder. Or, if you do not want to delete points, you still have to mark them as "already processed". This process is explained with COMMON, the example in fig "COMMON".

Once the cube K J I W U V F G H is found, we delete these points from the graph. The point G, for example, is safely deleted, since its neighbors ,F,H and W also belong to the encountered cube.

What to erase and what to leave

Once all the points of a cube are found, we have to delete it from[1] the picture, in order to process the remainder. Or, if you do not want to delete points, you still have to mark them as "already processed". This process is explained with COMMON, the example in fig. 'COMMON'. (page 34).

Once the cube K J I W U V F G H is found, we delete these points from the graph. The point G, for example, is safely deleted, since its neighbors, F,H, and W also belong to the encountered cube. But F, for example , is still not deleted, since it has as neighbors point outside (not belonging to) the actual cube. Therefore, one pass through the graph eliminates all the lines arriving at points in the cube; for example,F*(E*C* K) is transformed to F* (E* C*), since K was in the cube. In this way we delete the line F* -K, if we also make the transformation from K(U J F*) to K(U J).

Another pass looks for points of the form W ( ), that is, points "isolated" (not connected to anything else), and deletes them.

The first pass is done with the CONVERT rule

[(XXX (YYY U ZZZ) WWW) (XXX (*REPT* ((YYY ZZZ) WWW)))]

where we define U as "member of CUBEJUSTFOUND".

The second pass --deletion of isolated points-- is done with

[(XXX X( ) YYY) (XXX (*REPT* (YYY)))].

In this way points shared by several cubes (like K) are preserved. But not the lines; for example, the line K-U is erased (fig. 17."CHANGE"), because it belonged to the cube K J I W U V F G, even if it also belongs to the cube M N P D E F V T.

In general, there is no way to predict such an event, since the

(1) Canaday [4] treats this problem in a similar way.

second cube has not yet been found, and therefore there is no way to tell what its parts are. We sill discuss this point later.

In general, this is not a serious defect, but see the example TRICKY, fig TRICKY.

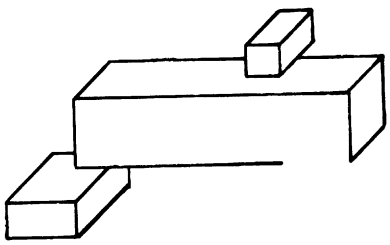
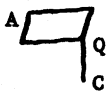
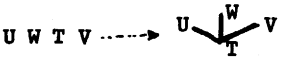


Fig. COMMON after erasing cube K J I W U V F G.

CUBS LISP. Classification of the corners.

We want to be able to recognize "disconnected" cubes; roughly speaking, the problem is this: in some way I manage to know that A Q C



looks like it is going to be a cube (see also fig HIDDEN) , so I would like to look for a corner of the form in the direction Q C. That corner happens to be U W T V 

at the bottom, but in order to find it I have to continue the line Q C for a while, and stop after finding W T, which is the continuation.

We could use the scheme of trying to extend all lines that seem to be stopped--like QC,TW --, making the picture somewhat transparent. Also, when looking for corner T, we could extend slowly the line Q C, and every 2 millimeters or so ask : Have I hit a point yet ?

Instead of that, we use the opposite approach: look for the points (corners) which exist, and see which of them may be continuations of Q C. But it would be better not to look at all of them, but just to the most promising ones. That is what CUBS does.

The vertices may be CORNERS,Y's,T's or ANY's.

The program classifies the vertices of the picture into several categories :

CORNERS; With this name we denote vertices at which two lines arrive, for example U ,A, I, r*, etc. in HIDDEN (fig.).

Y's : Three lines meeting at a point, two of them colinear; B*,W, K*, L*, M* .

ANY's: Vertices having more than 3 lines.

What the program CUBS does is divides the vertices into CORNERS, Y's,T's and ANY's. The Y's are also classified into classes, according to

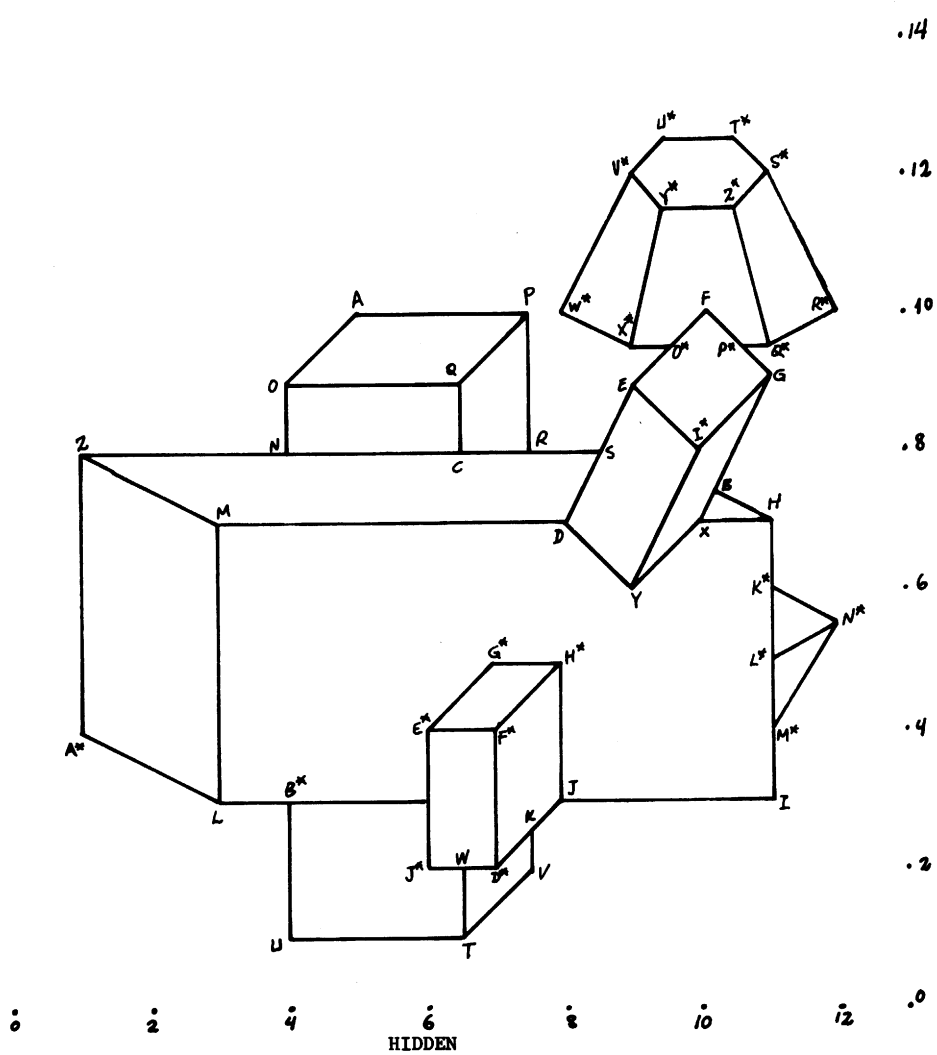
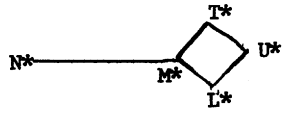


Fig The cubes A Q V U T and A* H are disconnected.

the slope of its sides

After this, all the Y's of a particular cube can be found in a given class; if it happens that there are no parallel cubes, like in STICKS (fig.), then you simply print the classes, because each class contains exactly one cube.

That is not the complete solution. There is more to be said, of course. When a single class contains just one vertex, such as G or M* (STICKS, fig .) ,it may or may not be part of a cube. CUBS make further analysis and depending upon the kind of vertices attached to the lines forming that Y, an acceptance or rejection is made. For information purposes, a message "FALSE CUBE FOUND" is issued.

For example, analyzing the points attached to H, XM and F, the "Y" G is accepted as a cube; analyzing the points N*, T* and Z*, the point M* is rejected, that is to say, 

is not part of a cube.

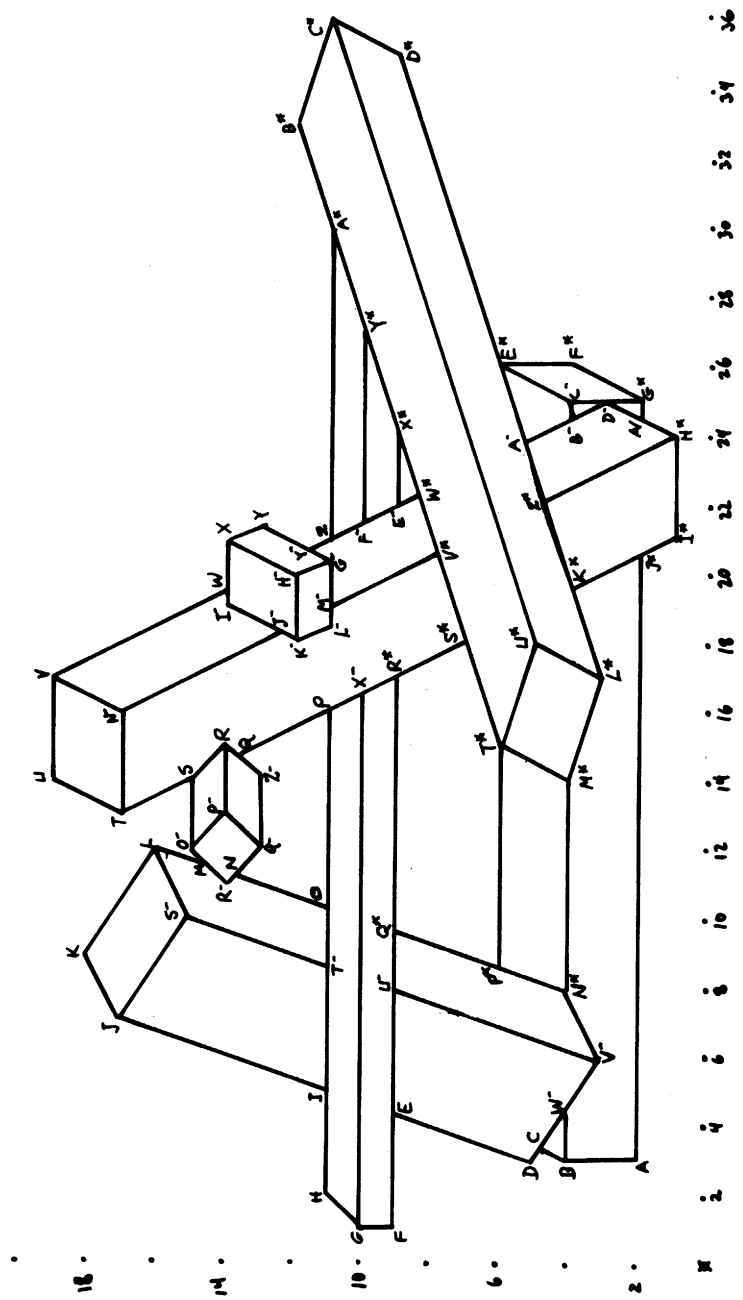


Fig. STICKS

This is the solution for STICKS (last page), as the program CUBS does it:

```
(CORNERS == ZM (QM Q) RM (M N) LM (KM MM) IM (W JM) I* *H*
J*) F* (E* G*) D* (C* E*) B* (C* A*) Y (X YM) U (T V) K (J
L) H (G I) F (G E) D (C E) A (B J*))

(TES = A/ (H* DM G*) YM (GM Z Y) XM (P R* G) WM (C B VM) UM
(Q* E VM) TM (SM I O) MM (LM GM V*) JM (KM IM NM) FM (Z EM
Y*) EM (W* FM X*) BM (AM CM DM) AM (Z* BM E*) Z* (K* AM H*)
Y* (FM X* A*) X* (EM W* Y*) W* (EM X* V*) V* (S* MM W*) S*
(R* T* V*) R* (Q* S* XM) Q* (UM P* R*) P* (N* T* Q*) K* (L*
Z* J*) J* (K* I* A) A* (Z Y* B*) Z (YM A* FM) W (IM V X) Q
(ZM P R) P (Q XM O) O (N TM P) N (RM QM O) M (RM OM L) I (H
J TM) E (F D UM) C (D B WM))

(FALSE CUBE ( 0.30000002E1  0.5e0  -0.0  N* (VM P* M*)))

(FALSE CUBE ( 0.2E1  0.0  -0.3333333 M* (T* L* N*)))

(CUBE 1 IS (U* (T* L* C*) L* (U* M* K*) C* (B* U* D*)))

(CUBE 2 IS (KM (JM HM LM) HM (KM X GM) GM (MM HM YM) X (W Y HM)))

(CUBE 3 IS (NM (T V JM) H* (A/ Z* I*) V (U NM W) T (S NM U)))

(FALSE CUBE (0.0  -0.1E1  -0.2E1  S (T OM R)))

(CUBE 4 IS (QM (N PM ZM) PM (OM QM R) OM (M PM S) R (S PM Q)))

(CUBE 5 IS (VM (WM N* UM) SM (J L TM) L (SM K M) J (I SM K)))

(CUBE 6 IS (G F H XM))

(CUBE 7 IS (CM (BM DM E*) G* (DM F* A/) B (A WM C)))

(ANYS = DM (BM CM A/ G*) T* (S* U* M* N*) E* (AM D* CM F*))
```

We print, as additional information, the CORNERS and the T's. Note that only a small part of each cube is printed; for example, of the long horizontal cube, only vertices G, F, H and XM are printed. It is not difficult to "fill" the cube, as CUBES2 does[1], but CUBS does not do that,

---

[1] See Polybrick memorandum [13].

if for no other reason, because we already know how to do it, so it is just a matter of adding that part of the program.

Also, CUBS does not use any information about CORNERS; we will need it in more complicated cases.

Shortcomings of CUBS.

I think the most serious one is that it is unable to make recognition among parallel cubes, for example cubes A Q U T and G* F* J* H* in fig. 'HIDDEN' (page 29) are confused and reported as just one, since they lie in the same class. A better (or worse) example is fig. 'COMMON' (page 34), where all the four cubes are parallel, and the program thinks there is just one. Also, the program does not check for length of edges.

Let us not get angry at CUBS. It is obvious that the program is incomplete, and it is also obvious what should be done.

The main good idea in CUBS is that, by dividing the cubes into classes, we transform the problem of finding all the cubes, into the problem of finding the cubes in a given class, in which all of them are parallel. This approach also solves the disconnectivity problem.

CUBA LISP. Differentiating among parallel cubes.-

The program just discussed takes a figure and separates the cubes into classes, each one of them containing parallel cubes. For example, in fig. 'HIDDEN' (page 29), the cubes A Q V U T and E* J* D* J H* are parallel. We would like to differentiate among them. Here we use the collinearity among two vertices; for example, Q and T are collineal

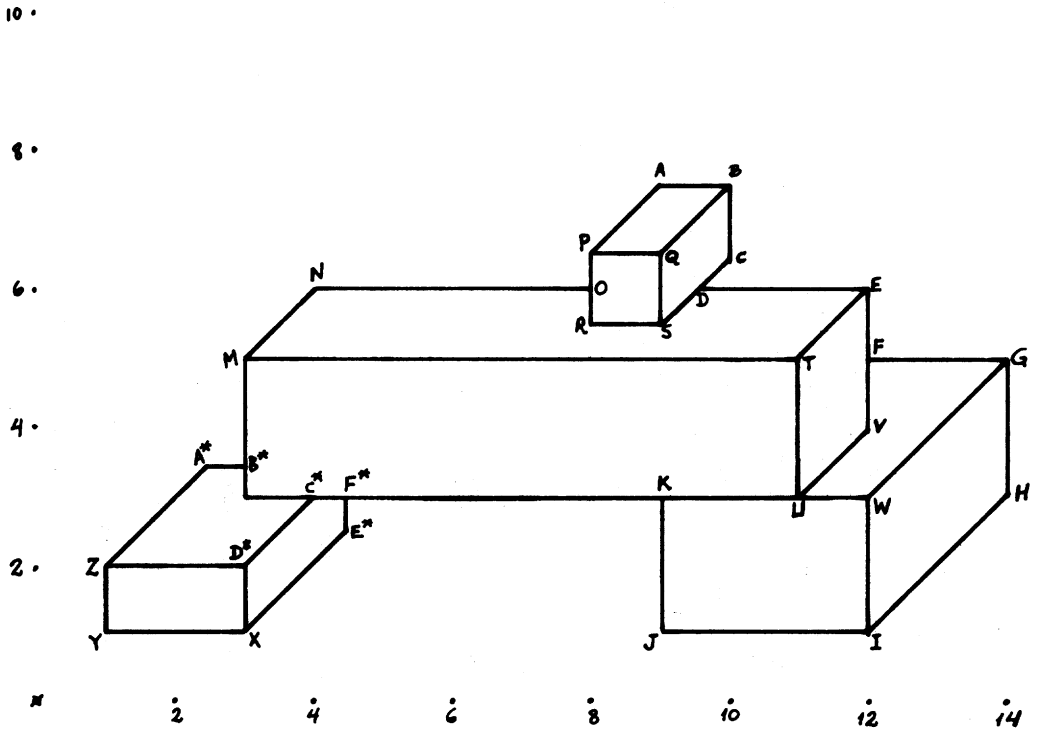
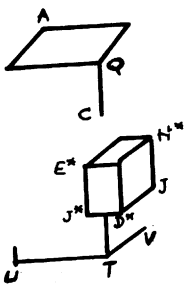


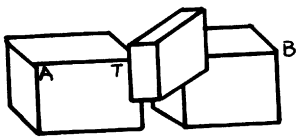
Fig. 'COMMON'. A scene analyzed b y Polybrick.

--figure at the right-- , but Q and D* are not, so Q and D* can not form a cube.



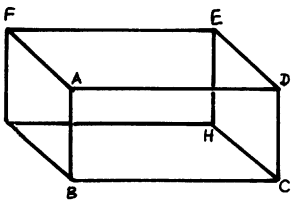
Also, we do not want to compare Q with all the vertices of its same class in order to select the possible ones; it seems that a further classification of vertices of the same class is desirable.

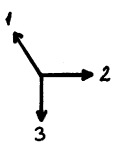
Collinearity is not sufficient. For example, vertices A and B --see figure below-- are collineal, and still do not form a cube; therefore, we will select all the vertices colineal to A in the direction AT and (if there are some) select the appropiate one.



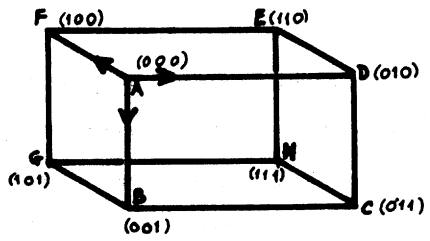
Numbering the Y's. Unit distance vertices.-

TAke a cube, pick any vertex and establish the three directions of its lines, as done in the figure. Now, examine for each vertex, the lines which depart from it. For vertex A, all its lines depart in the positive directions ↖ , → and ↓ ; therefore, is (+ + +) or (0 0 0). For vertex B, line B G is ↖ (+)
line B C is → (+)
line B A is ↑ (-) ;





therefore, vertex B is (+ + -) or (0 0 1) or 1. When we finish this process, our cube now looks like this:



This numbering scheme is independent of the starting vertex (0 0 0) and of the directions which are considered positive.

Connected vertices are unit-distant, that is, their binary words differ in exactly one bit. Vertices which are 2 units apart lie on the diagonal of the faces (A E, A G, B H, etc.) and vertices lying in opposite extremes of the diagonals of the cube are 3 units apart, for example F (1 0 0) and C (0 1 1).

Pre-processing(1).- The pre-processing done in CUBA is more complicated than the one done in CUBS.

Vertices are divided into CORNERS, T's, Y's and ANY's (as before);

1. CORNERS are divided according to the slope of the sides.
2. T's are divided according to the slope of the top and the slope of the tail.
3. Y's are divided into classes, according to slopes.

   In each class, vertices are divided according to the unit distance concept. If certain vertex happens to be the first of a given class, the number (0 0 0) is assigned to it.

Localization of the cubes.- A second part of CUBA applies to each class of Y's the following process:

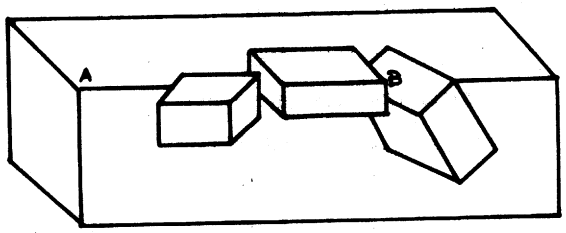
1. A vertex is selected and the program tries to attach to it a cube, if possible; therefore, its unit-distance vertices are looked for [if the vertex in question has number (x_1, x_2, x_3), only subclasses $(\bar{x}_1, x_2, x_3)$, $(x_1, \bar{x}_2, x_3)$ and $(x_1, x_2, \bar{x}_3)$ are searched(*)];

(1) (*) footnotes in next page.

a vertex has to pass the test for collinearity and, if several are found, the closest is chosen. It turned out that these 3 test are still not sufficient; for example, B is

(1) unit-distant from A
(2) colineal
(3) the closest

and still A - B is not (part of) a cube. In relation with this, see also fig. 'TOWER'.



2. We apply to the vertices found in (1.) the same process (1.), up to a certain depth.

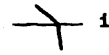
3. The cube formed in this way is accepted if it has two or more vertices; if it has one, as N* (K* L* M*) in fig. 'HIDDEN', page 29, we check the extreme points [K*, L* and M* in the example], as explained in CUBS.

A fancier program should say, after finding a cube such as N*: "I am not sure it is really a cube, but it looks like one". This comment can be inserted in this part of the program.

4. Accepted cubes are reported and their vertices erased from the subclasses where they were found, and the whole process is applied again to the next vertex of the subclass.

5. When a subclass (or a class) is empty, the next one is searched.

CUBE LISP.

Is the program currently in use; in addition to what CUBA does, it also breaks vertices of the type  in two Y's: and .

---

(1) This should not be confused with the kind of preprocessing of chapter 3.

(*) $\bar{x} = \begin{cases} 1 \text{ if } x = 0 \\ 0 \text{ if } x = 1 \end{cases}$

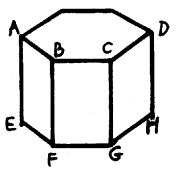
Recognition of Cubes in a Picture which also contains other Objects.-

In the presence of non-cubic objects, an effort is made by the program to see cubes in them; if none is found, these objects are simply ignored. A good example is fig. 'HIDDEN', page 29, where the truncated pyramid is ignored, but only after several "false cubes" found in it. The output is the following:

(FALSE CUBE (Z* (Q* Y* S*)))
(FALSE CUBE (Y* (V* Z* X*)))
(FALSE CUBE (X* (W* O* Y*)))
(FALSE CUBE (S* (T* Z* S*)))
(FALSE CUBE (Q* (Z* P* R*)))
(CUBE 1 IS (N* K* L* M*))
(FALSE CUBE (X (H Y B)))
(FALSE CUBE (J (I K H*)))
(CUBE 2 IS (H* (G* F* J) E* (F* G* C*) F* (E* H* D*) D* (W K F*)))
(CUBE 3 IS (P (A Q R) O (Q A N) Q (O P C) T (U V W)))
(CUBE 4 IS (L (A* B* M) Z (M N A*) M (Z D L) H (B X K*)))
(FALSE CUBE (B* (Y* U* W*)))
(CUBE 5 IS (Y (D X I*) G (P* I* B) I* (E G Y) E (I* O* S)))
(FALSE CUBE (D (Y M S)))

SOLUTION TO H I D D E N

If instead of a pyramid we put an hexagonal prism, it will recognize in it the "cubes" A B C E F G and B C D F G H !



As you see, CUBE is not very successful in a foreign environment. A more general program should be more careful about accepting candidates which look good.

Some Examples.

We have already shown several figures which the program analyzes correctly; they are COMMON (page 34), GORDO (page 23), HIDDEN (page 29), STICKS (page 31). Some of them, like HIDDEN (page 29) are somewhat complicated, since they involve parallel cubes, disconnected cubes, 1-corner cubes, extraneous objects, etc.

I would like to present now a couple of examples, TRICKY (fig. 'TRICKY'), and WHAT? (fig. 'WHAT?'), where the answer is ambiguous (non unique). The program does its best, and its answers are acceptable but, in general, CUBE is not designed to solve optical illusions.

```
load ((a cube tricky))
(CERO UNDECLARED)
(CERO UNDECLARED)
NIL
                e (cubs tricky)

(THERE ARE 2 OR 1 CUBES)

(CUBE 1 is (M (B D L) B (M C A)) )

(CUBE 2 IS (J (K I P) P (L H J)) )

(FALSE CUBE (O (H L D))

(CUBE 3 IS (F (E N G) N (D F H)) )
```

CUBE accepts the 3 exterior cubes and rejects O (H L D). Now we apply it to the scene WHAT?:

```
(CUBE 1 IS (O (P Y X) X (Q W O) Q (X R P)) )
(CUBE 2 IS (S (D T R) D (S E C)) )
(CUBE 3 IS (Q (B R P) B (Q C A)) )
(CUBE 4 IS (M (Y L Z) K (J Z L) Z (W K M)) )
(CUBE 5 IS (H (G U I) U (T H J)) )
(CUBE 6 IS (M (Y N Z) Y (M O W) O (N Y X)) )
(FALSE CUBE (V (J R T)))
(FALSE CUBE (C (R B D)))
```

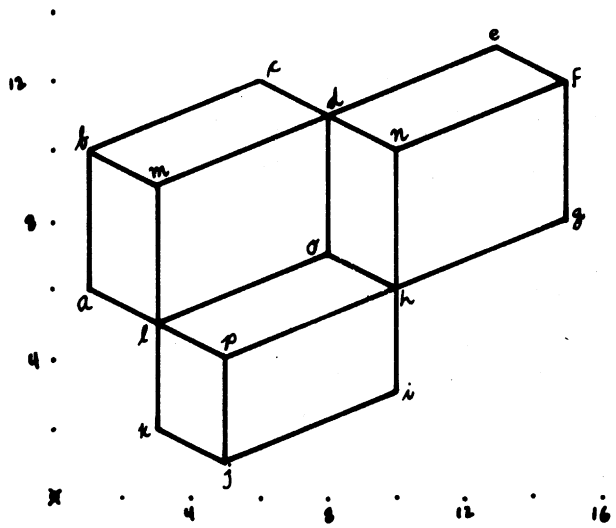


Fig. 'T R I C K Y'.

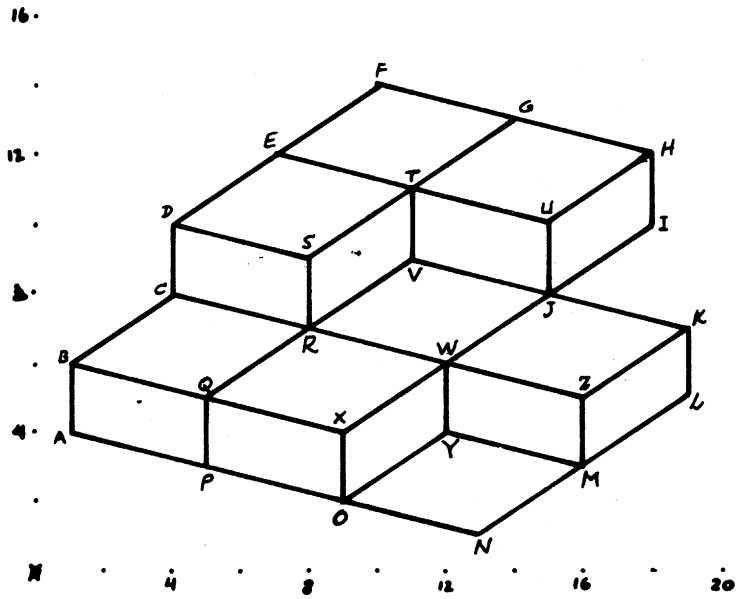


Fig. 'W H A T ?'.

These are the results for WHAT? (page 40). 6 cubes are found; M Y O W is accepted, but J V R T is not. This is (see figure 'WHAT?') certainly a possibility; otherwise, how does one explain with cubes the presence of lines ON and NM ?

The next page shows the scene 'TOWER'. All the cubes but one are correctly identified; cubes C* T and P I are [con]fused and they appear in the answer as only one, namely C* N* A* I. This is because we do not use information about lines; lines P* Q and R* T (see page 42) will solve the problem.

```
e (cubs tower)
(THERE ARE AT LEAST 3 OR 2 CUBES)
(CUBE 1 IS (A* (X I X*) X* (Z O A*)) )
(CUBE 2 IS (X* (Z O U*) U* (H S X*)) )
(CUBE 3 IS (V (C T* B) F* (T* C C*) T* (F* V N*) N* (C* R T*)) )
(CUBE 4 IS (N* (C* R P*) A* (X I W*)) )
(CUBE 5 IS (F (Y M N) N (G* L F)) )
(CUBE 6 IS (N (G* L K) K (D* G N)) )
(CUBE 7 IS (K (D* G J) J (H* V* K)) )
(CUBE 8 IS (U* (H S F) F (Y M U*)) )
(FALSE CUBE (W* (D Y* A*)))
(CUBE 9 IS (E (W Q* J*) E* (J* B R) U (B J* Q*) J* (E* U E)) )
(FALSE CUBE (E (W Q* T)))
(CUBE 10 IS (W* (D Y* A) I* (A Q D) R* (P* A S*) A (I* R* W*)) )
(FALSE CUBE (P* (R* Q N*)))
(FALSE CUBE (B (U E* V)))
```

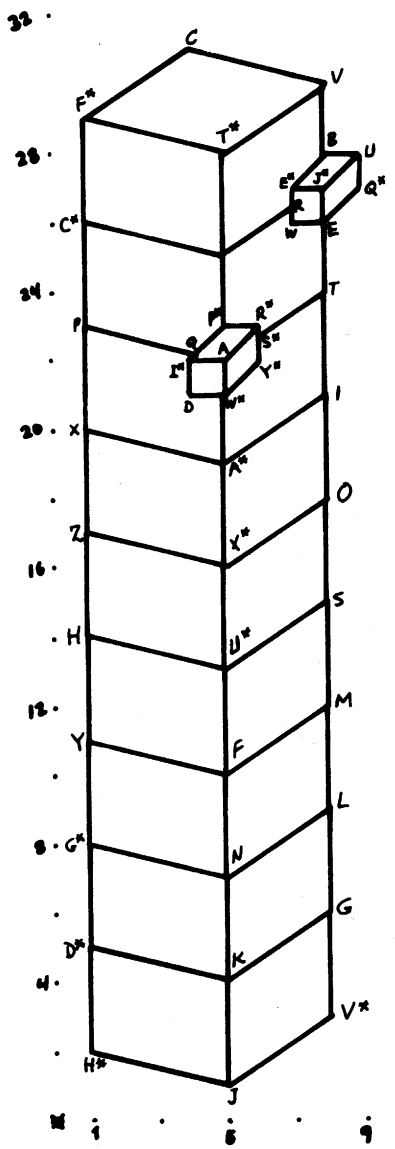


Fig. 'T O W E R'. Vertices such as K (D* N G J) having 4 connected points, two of which (J and N) are collineal, get decomposed in two Y's: K (D* N G) and K (D* J G).

CHAPTER V. T D

This chapter describes a program[1], written in the CONVERT language, and run in the IBM 7094 system, called TD. The function of TD is to accept a scene expressed as a symbolic expression (cf. chapter 3) and a model (cf. chapter 7), expressed in the same notation, and to find all instances of this model in the scene.

The symbolic notation for expressing scenes and models uses a language called FDL-1 (figure description language-one). The notation restricts the present application of the system to scenes and models which are made of straight line segments. Models can be described independent of position, size, rotation, reflection, etc.

The program TD is particularly well-suited for non-overlapping figures. Overlapping figures are identified only when they are transparent.

Either two or three-dimensional models and scenes can be represented in our notation. Furthermore, the program TD will handle three dimensional scenes and models as readily as two-dim ones. That is, we can compare 2-dim scenes with 2-dim models, or 3-dim scenes with 3-dim models (both cases describable in FDL-1); this last case is rather rare, due to the difficulty to get 3-dim scenes to analyze. On occasion, it is possible correctly to analyze a scene which is the two-dimensional representation of a 3-dim scene by using only two-dimensional models.

---

[1] TD was developped by the author at Computer Corporation of America, Cambridge, Mass., under contract AF 19(628)-5914.

SECTION I. The Figure Description Language FDL-1.

Introduction.

The figure description language FDL-1 is a language in which figures and models may be represented in a symbolic notation. For the purposes of FDL-1 we restrict ourselves to figures and models which are made of straight-line segments.

Formally, there is no distinction in the language between a figure and a model. Informally, we will use the term "figure" to mean a certain specified picture (which may or may not be fixed in position), whereas we will use the term "model" to refer to a class of pictures, such as the class of squares for example, or the class of chemical formulae containing one benzine ring.

THE LANGUAGE

Points.

Points are the building blocks of further structures. A point is represented by an atom. Example: A
B
COC are points (see fig. 'POINTS').

Vertex.

A vertex is a point followed by a (non-ordered) list of points, called the neighbors of such a point. A vertex has no repeated neighbors,

A.

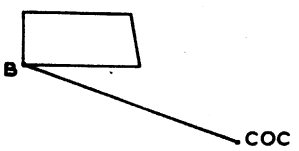


Fig. 'POINTS'. The simplest element of a figure is a point.

and is not a neighbor of itself.

Examples: A (B C D) is a vertex; B, C, and D are the neighbors of A.

A (B D C) is a vertex.

A (B C C) is not a vertex.

A (B C A D) is not a vertex.

Elementary Figure.

A list of vertices satisfying the following constraints is called an elemantary figure:

1. Each point mentioned occurs exactly once as a vertex.

2. Neighborhood is a symmetric property; that is, the occurrence of ... A (... B ...) ... in the figure means that ... B (... A ...) ... must also be present.

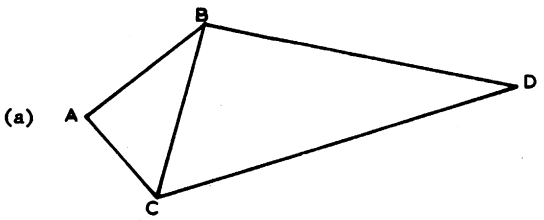
Elementary figures are sometimes called the connection matrix or connection list. The order in which the vertices are mentioned is irrelevant.

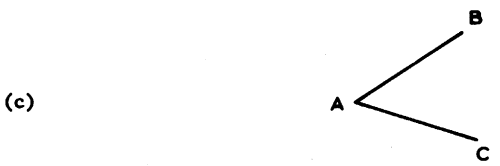
Example: (A (B C) B (C D A) D (B C) C (A B D))

is an elementary figure. See figure below.

Note that the vertex A (B C) is different from the figure (A (B C) B (A) C (A)). See figure 'THREE'.

(a) elementary figure
(b) point A.
(c) vertex A (B C).





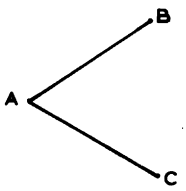


Fig. 'THREE'. The elementary figure (A (B C) B (A) C (A)).

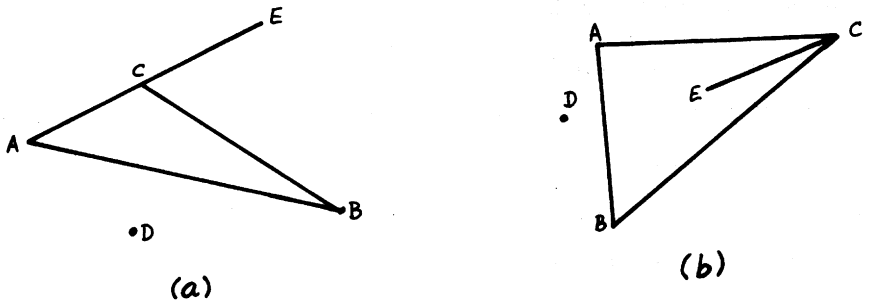
Example: (A () B () C ()) is an elementary figure (see figure below).

Example: ((A (B C) M (N S))) is <u>not</u> an elementary figure.

Example: (A B (C) D (116. 0.563)) is <u>not</u> an elementary figure.

This elementary figure shows the fact that the neighbors of a given point P are the other vertices to which lines from P are drawn. Compare with figure 'THREE'.

Example: (A (B C) C (B A E) D () B (C A) E (C)) is an elementary figure (see figure below).



Two figures (a) and (b) may be described by the same elementary figure in FDL-1: (A (B C) C (B A E) D () B (C A) E (C)).

Properties.

Elementary figures describe only the topology of the connection between the different vertices of the object[1]; in order to characterize further the scene or model in question, we modify (we restrict, in fact) this topological skeleton by specifying properties which the figure has (to have).

A property is an ordered list whose first atom has been declared to be of the type "property name" (i. e., it is not a point), and whose remaining elements are atoms representing points, or numerical constants. A property is simply a predicate, i.e., an expresion with open variables, such that the expression becomes T or F upon substitution for the variables.

Examples: (LENG C B 4.0)

---

(1) Evans [8] uses a similar scheme for the representation of his figures.

(ANGLE A B C 75)

Attachment of Properties to Figures.

Given a figure, a new figure may be formed by attaching to it a collection of properties, using the connector 'where'. An example will illustrate the syntax.

(A (B C) B (A C) C (A B)) (1)

(SLOPE A B 3.0) (LENG C A 5.0) (2)

Expression (1) represents an elementary figure with three vertices; expression (2) represents two properties.

A new (non-elementary) figure may be formed by saying:

((A (B C) B (A C) C (A B)) where ((SLOPE A B 3.0) (LENG C A 5.0))) (3)

In the example, (1) is any triangle, and (3) is any triangle with a side of length 5.0 and the adjacent one with slope 3.

Example: ((M (N R) Q (N R) N (M Q) R (M Q)) where
((LENG N R 8.5)
(SLOPE M Q 0.0)))

represents a quadrilateral with an horizontal diagonal, the other being 8.5 units long.

Therefore: A figure may be formed by a list containing 3 elements:

1. a figure
2. the connective where
3. a list of properties.

Properties Containing Open variables.

The properties (SLOPE A B X)

(SLOPE M N X)

where X is not a point, but an explicitly declared "open variable", are interpreted as saying that the slope of AB is X and the slope of MN is also X, whatever X may be. In short, the lines AB and MN are parallel. In general, open variables are used when we do not want to commit ourselves to specific values, but insist only that the value be the same each time the variable is encountered.

In order to distinguish open variables from points and property names, open variables are declared as such using VARIABLES, which is a special property. Thus, the expression

(VARIABLES ALPHA THETA CAMOTES ....)

defines the atoms ALPHA, THETA, CAMOTES, ..., to be open variables. These variables are considered open only with respect to the figure which they modify. An example of a figure containing open variables is as follows:

((P (Q R) Q (P R) R (P Q)) where

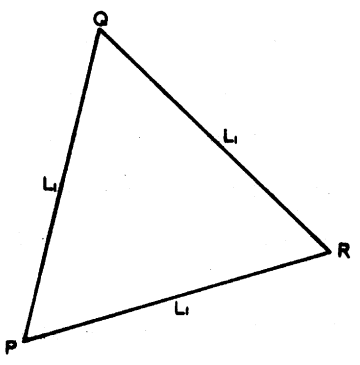
((LENG P Q L1)

(LENG Q R L1)

(LENG P R L1)

(VARIABLES L1)))

This figure represents an equilateral triangle. A second description may be:



```
(( (P (Q R)  Q (P R)  R (P Q)) where
   ((ANGLE P Q R ALPHA)
    (ANGLE R P Q ALPHA)
    (VARIABLES ALPHA)))  where
       ((ANGLE P R Q ALPHA)
        (ANGLE P Q R ALPHA)
        (VARIABLES ALPHA)) )
```

Composition of Figures.

Boolean connectors may be used to form new expressions. For example:

```
(=OR= (A (B C) B (A C) C (A B))
      (A (B C) B (A D) D (B C) C (A D)) )
```

is a representation of a triangle or a certain type of quadrilateral.

Definitions: Single Names.

An operator is now introduced which allows us to give to a whole figure a single name. This operator is

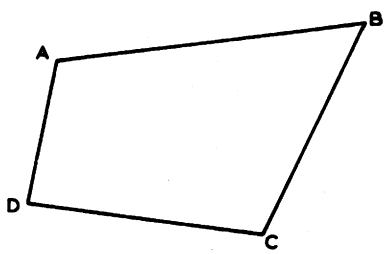
```
(=DEF=  name  fig)
```

where name is an atom not previously used as either a point, an open variable, or a property name, and fig is a figure. After such a statement has been executed, name and fig are completely equivalent and interchangeable.

=DEF= allows us to set single atoms to stand for whole figures. For example, assume we have a quadrilateral, i. e.,

```
(A (B D) B (A C) C (B D) D (A C))
```

Fig. 'QUADRILATERAL'. A model.



We may define the atoms "QUADRILATERAL" and "QUAD" both to represent that quadrilateral as follows.

```
(=DEF= QUADRILATERAL  (A (B D)  B (A C)  C (B D)  D (A C)))
(=DEF= QUAD  QUADRILATERAL).
```

A parallelogram may now be defined as a quadrilateral having two equal,

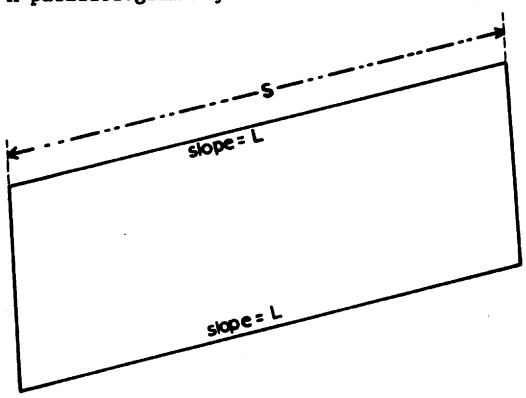


Fig. 'PARALLELOGRAM'. Parallelogram defined with the help of the model of figure 'QUADRILATERAL'.

non-adjacent, and parallel sides, such as AB and DC (not CD). For example:

```
(=DEF= PARALLELOGRAM (QUAD where
                        ( (SLOPE A B S)
                          (SLOPE D C S)
                          (LENG D C L)
                          (LENG A B L)
                          (VARIABLES A L) )) )
```

We may now define a rectangle as a parallelogram having one right angle:

(=DEF= RECTANGLE (PARALLELOGRAM where

((ANGLE D A B 90^{o}))))

Of course, we could have made some of the definitions in a different manner; for example,

(=DEF= PARALLELOGRAM ((A (B D) B (A C) C (B D) D (A C)) where

((SLOPE A B X) (SLOPE A D Y)

(SLOPE D C X) (SLOPE B C Y) (VARIABLES X Y))))

The above defines a parallelogram to be a quadrilateral with opposite sides parallel two by two.

A rectangle may be defined as a parallelogram constrainted to have its two diagonals of equal length, as follows.

(=DEF= RECTANGLE (PARALLELOGRAM where

((LENG A C Z)

(LENG D B Z) (VARIABLES Z))))

Note that properties are not attached to lines, but to figures; for instance, (LENG A C Z) is a well-defined property, even when the figure does not contain line AC(*).

Definitions of New Properties.

New properties may be defined at will. In order for the recognition program to take proper action in regard to the new properties, these must be defined before use in terms of LISP functions.

A property (P A_1 A_2 ... A_{n-1} A_n) is handled by producing a call to

(*)This feature is considered important by Sutherland in Sketchpad [34].

the LISP function (P A_1' A_2' ... A_{n-1}'), where A_i' is the expression obtained by replacing A_i by its value (with respect to TD); the value of this LISP function is then computed and compared with A_n', yielding a match or a failure[1]; furthermore, when the mentioned value is T, this comparison does not take place, and TD handles this case as if a successful match were occurred.

For example, suppose we want to define the property EQUILATERAL, function of m sides A_1 B_1; A_2 B_2; ...; A_m B_m; and we will say that a figure has such property if $|\text{length } A_i\ B_i|$ falls within $\pm$ EPS of the average length $\overline{AB} = \dfrac{\sum_i \text{length } A_iB_i}{\sum_i i}$

where EPS is some pre-specified tolerance. We must write a LISP function of name EQUILATERAL, of 2m (not 2m+1) arguments, whose value is, for instance, YES if the arguments [whose values are the points forming the sides of the figure in question] fulfill the appropiate requeriments; this function should check its arguments to see if some of them is not a point, in whose case should return T; otherwise (failure), its value must be different from YES or T. One could then write

```
(=DEF= SQUARE ( (A (B D) B (A C) C (B D) D (A B)) where
      ((EQUILATERAL  A B  B C  C D  D A  YES)) ))
```

The user is able to define properties as complicated as he wishes, since properties are functions (predicates) of several variables, the variables being the (coordinates of the) vertices, and the values which obtain different UAR variables: slopes of lines, distances, etc. Therefore, arbitrarily complex restrictions may be specified, and models can have fairly elaborate properties or constraints between its different elements.

(1) This comparison is done by RESEMBLE (see CONVERT [14]).

SECTION II. The Program

What the Program does

As already mentioned, the recognition program called TD accepts a description of a scene expressed in the notation of section I and a description of a model expressed in the same notation. It will produce, as output, the instances of the given model in the given scene.

TD operates in several modes which are set by 'switches'. There are three switches called EXACT, ALL and SYMMETRIC.

EXACT can have one of two values: T and (). A value of T for EXACT is the normal mode. In this mode, an object will be said to match the model only

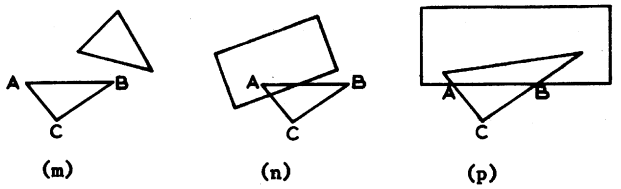


Fig. γ-16. The small triangle A B C in (p) is not recognized when EXACT is T, but it will be when EXACT is ().

if the vertices of the object have exactly the same number of lines as are specified for the corresponding vertices in the model. Thus, ABC will be recognized as a triangle in fig. γ-16-m and in fig. γ-16-n, but not in fig. γ-16-p.

When the value of EXACT is (), the object will be recognized as matching a model even if there are more lines occurring at a given vertex than those

specified in the model. Thus, ABC in fig.γ-16-p will also be recognized as a triangle.

The next switch is called ALL. It takes one of three values: T, () and 69. The normal setting is T. In this state, the program will identify a certain portion of the scene, erase that portion, and then operate on the remainder. The program terminates when the scene no longer contains parts which may be identified as the model in question.

Under the setting (), the program stops after having identified the first instance of the model.

Under the setting 69, the program will do an exhaustive analysis of the scene in terms of the given model. Thus, for example, in fig. 'ALL' the program will find two rectangles in the setting T, one rectangle in the setting (), and 12 rectangles in the setting 69. The 12 are all the permutations of the three rectangles, A B C D, N M L K and S N R D.

The third switch is called SYMMETRIC. It has two values: T and (). The setting T is to be used when the current model is, in fact, symmetric. In this case, the program will operate faster than in the mode (), the normal mode. In case the model is not symmetric and the switch is set to T, the program will not behave correctly. See examples below.

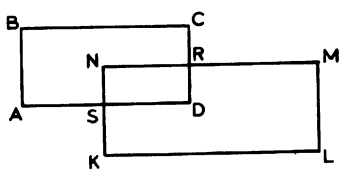


Fig. 'ALL'. Twelve rectangles.

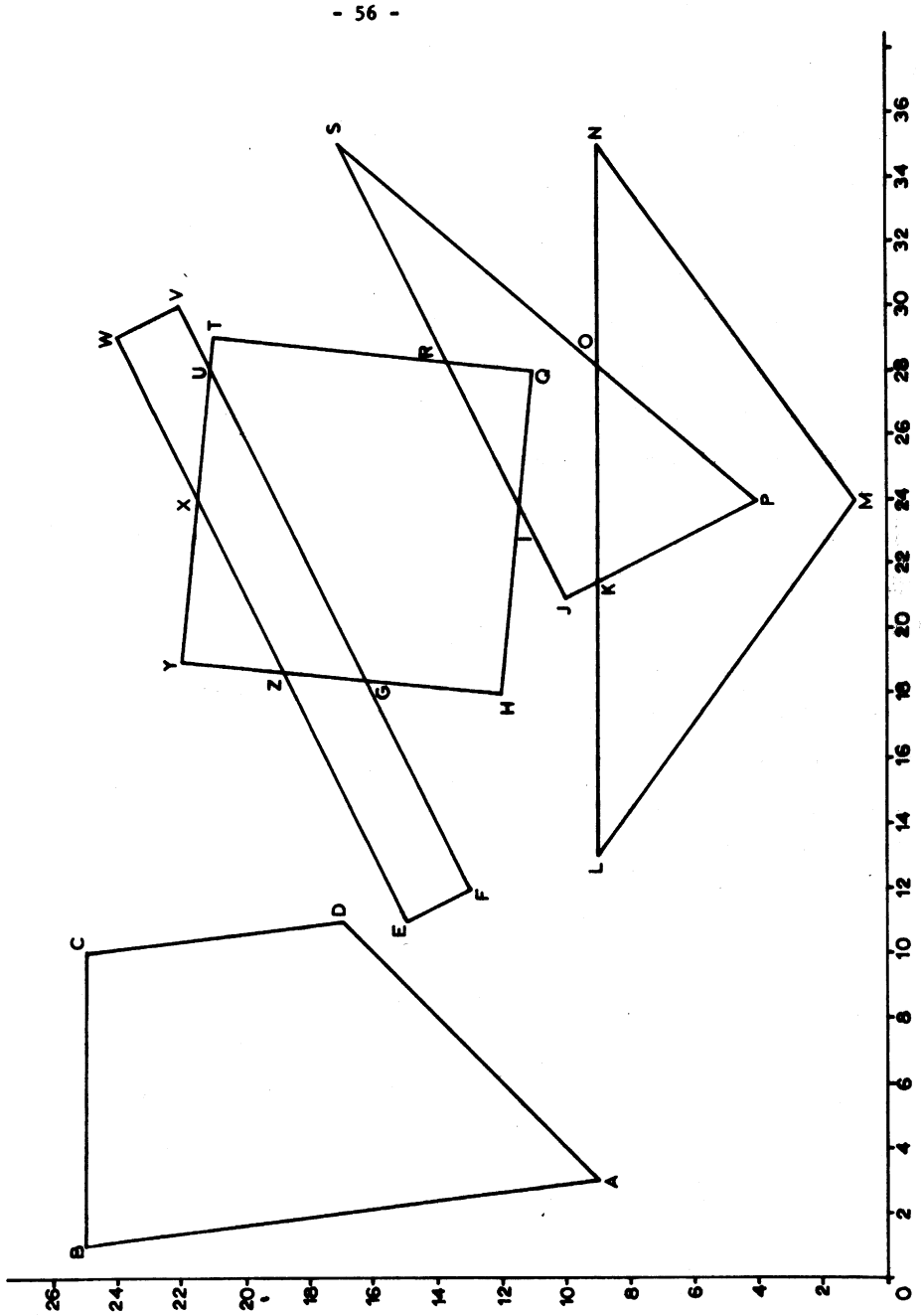


Fig. 'P27'. Scene analyzed in example 1.

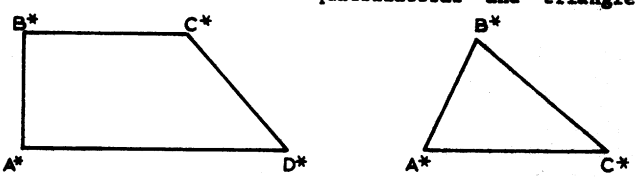
EXAMPLES

Example 1. P27.

We have shown (fig. 'P27) in the page 56 a scene we want to analyze. The symbolic description of such a scene, that is, the way it is assimilated by the program, is the following:

```
          (LAMBDA (A B C) (PUT A B (LLENA C)))
(P27 SCENE (A 3. 9. (B D) B 1. 25. (A C) C 10. 25. (B D)
D 11. 17. (A C) E 11. 15. (F Z) F 12. 13. (E G) G 18.421052
16.210525 (F Z U H) H 18. 12. (G I) I 23.833333 11.416667
(H R Q J) J 21. 10. (I K) K 2.5 9. (J O P L) L 13. 9.
(K M) M 24. 1. (L N) N 35. 9. (O M) O 28.230769 9. (P K S N)
P 24. 4. (K O) Q 28. 11. (R I) R 28.263158 13.631578 (I T S Q)
S 35. 17. ( R O) T 29. 21. (U R) U 28.166666 21.083333
(X V T G) V 30. 22. (U W) W 29. 24. (X V) X 24. 21.5          (Y W U Z)
          Y 19. 22. (X Z) Z 18.68421 18.84210 (E Y X G)    ))
PUT (QUADRILATERAL MODEL ( A* (B* D*)  B* (A* C*)  C* (B* D*)
D* (A* C*) ))
PUT (TRIANGLE MODEL (A* (B* C*) B* (A* C*) C* (A* B*)  ))
```

The last three rows define the models "quadrilateral" and "triangle".



Models "quadrilateral" and "triangle" used in the analysis of fig. 'P27' (page 56).

We ask the program to look first for triangles, then for quadrilaterals:

```
(TRIANGLE 1 IS (J S P))
(TRIANGLE 2 IS (L N M))
((O (P K S N)) (O (R I)) (R (I T S Q)) (T (U R)) (U (X V T
G)) (V (U W)) (W (X V)) (X (Y W U Z)) (Y (X Z)) (Z (E Y X G))
(A (B D)) (B (A C)) (C (B D)) (D (A C)) (E (F Z)) (F (E G))
(G (F Z U H)) (H (G I)) (I (H R Q J)) (K (J O P L)))
(QUADRILATERAL 1 IS (A B D C))
(QUADRILATERAL 2 IS (E F W V))
(QUADRILATERAL 3 IS (Y T H Q))
((R (I T S Q)) (S (R O)) (U (X V T G)) (X (Y W U Z)) (Z (E
```

```
Y X G)) (G (F Z U H)) (I (H R O J)) (J (I K)) (K (J O P L))
(L (K M)) (M (L N)) (N (O M)) (O (P K S N)) (P (K O)))
```

Now we change EXACT to (). When we look for quadrilaterals, the answer is

```
(QUADRILATERAL 1 IS (A B D C))
(QUADRILATERAL 2 IS (E F Z G))
(QUADRILATERAL 3 IS (L O M N))
(QUADRILATERAL 4 IS (U X V W))
((H (G I)) (I (H R Q J)) (J (I K)) (K (J O P L)) (P (K O))
(Q (R I)) (R (I T S Q)) (S (R O)) (T (U R)) (Y (X Z)))
```

note that the analysis is consistent with the setting EXACT = () and

ALL = T; namely, we cannot identify other quadrilaterals in the remainder.

This answer was one of several possible matches, which could be discovered

by reordering the vertices of the scene and applying TD again, or by making

ALL = 69.

Looking for all (in the 69 sense) triangles:

```
cset (symmetric nil)  cset (all 69)   cset (exact nil)
td (triangle p27)

      (TRIANGLE 1 IS U Y G)
      (TRIANGLE 2 IS Y U G)
      (TRIANGLE 3 IS Q R I)
      (TRIANGLE 4 IS  R Q I)
      (TRIANGLE 5 IS P O K)
      (TRIANGLE 6 IS O P K)
      (TRIANGLE 7 IS N L M)
      (TRIANGLE 8 IS  L N M)
      (TRIANGLE 9 IS K P O)
      (TRIANGLE 10 IS O K P)
      (TRIANGLE 11 IS P K O)
      (TRIANGLE 12 IS S J P)
      (TRIANGLE 13 IS K O P)
      (TRIANGLE 14 IS J S P)
      (TRIANGLE 15 IS I R Q)
      (TRIANGLE 16 IS R I Q)
      (TRIANGLE 17 IS Q I R)
      (TRIANGLE 18 IS I Q R)
           QUIT
```

(the program was stopped and did not finish).

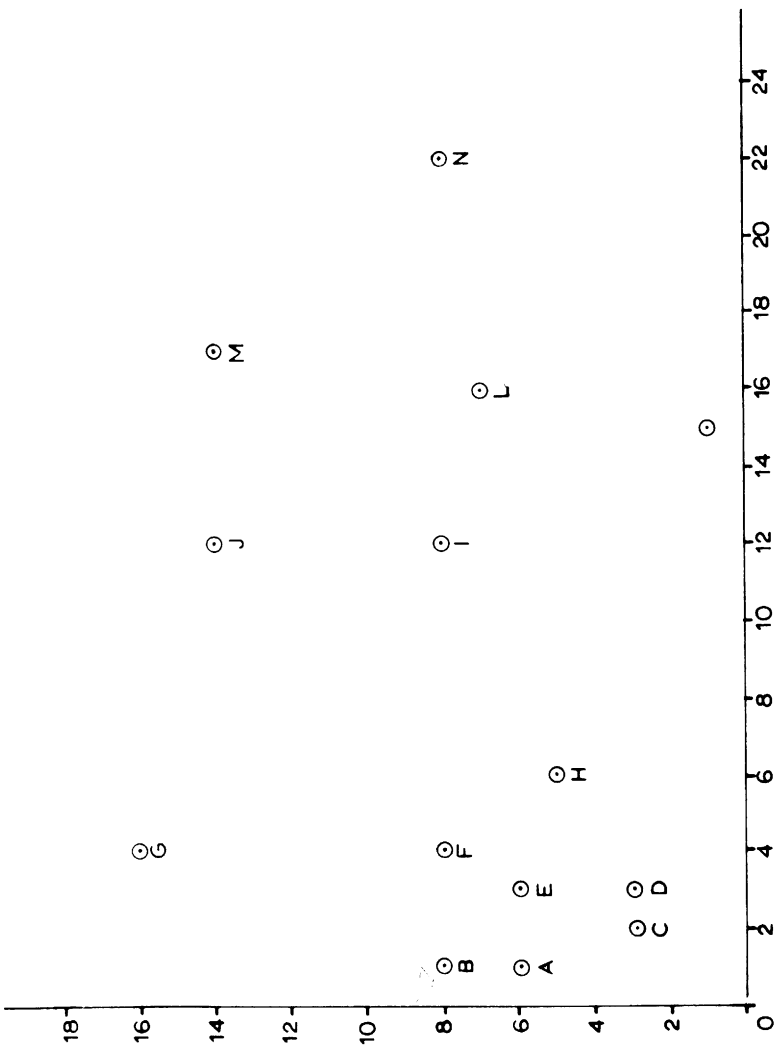


Fig. 'S Q U A R E'.
Find all the squares in this picture.

Example 2. 'S Q U A R E'.

(see fig. 'SQUARE', page 59). The symbolic scene is

```
(LAMBDA (A B C) (PUT A B (LLENA C)))
(SQUARE SCENE (A 1. 6. () B 1. 8. () C 2. 3. () D 3. 3. ()
E 3. 6. () F 4. 8. () G 4. 16. () H 6. 5. () I 12. 8. ()
J 12. 14. () K 15. 1. () L 16. 7. () M 17. 14. ()
N 22. 8. ()     ))
```

We will look for squares here, in the sense of sèts of four points which could be located at the corners of a square; the model in question is

```
(PUT (SQUARE MODEL
( (M* () O* () N* () P* () )   WHERE
(LENG M* N* L1) (LENG P* O* L1) (LENG N* O* L2) (LENG M* P* L2)
     (ANGLE N* M* O* A1) (ANGLE M* N* P* A1) (ANGLE P* M* O* A1)
(ANGLE O* N* P* A1) (VARIABLES L1 L2 A1)   )))
```

The answer is

```
(SQUARE 1 IS (A D F H))
(SQUARE 2 IS (K M C G))
```

Example 3. 'X S'.

We will now analyze a three dimensional scene (see fig. 'XS', page 61), or rather, to a 2-dim view of a 3-dim scene. We are interested in objects of a shape as "X " (see fig. 'EQUIS'.)

The operation is:

```
CSET (SYMMETRIC ())

TD (EQUIS XS)

(EQUIS 1 IS (I J H A F G E D C B))
(EQUIS 2 IS (K D1 L S O M N P Q R))
(EQUIS 3 IS (Z A1 U B1 V T Y W X C1))
NIL
```

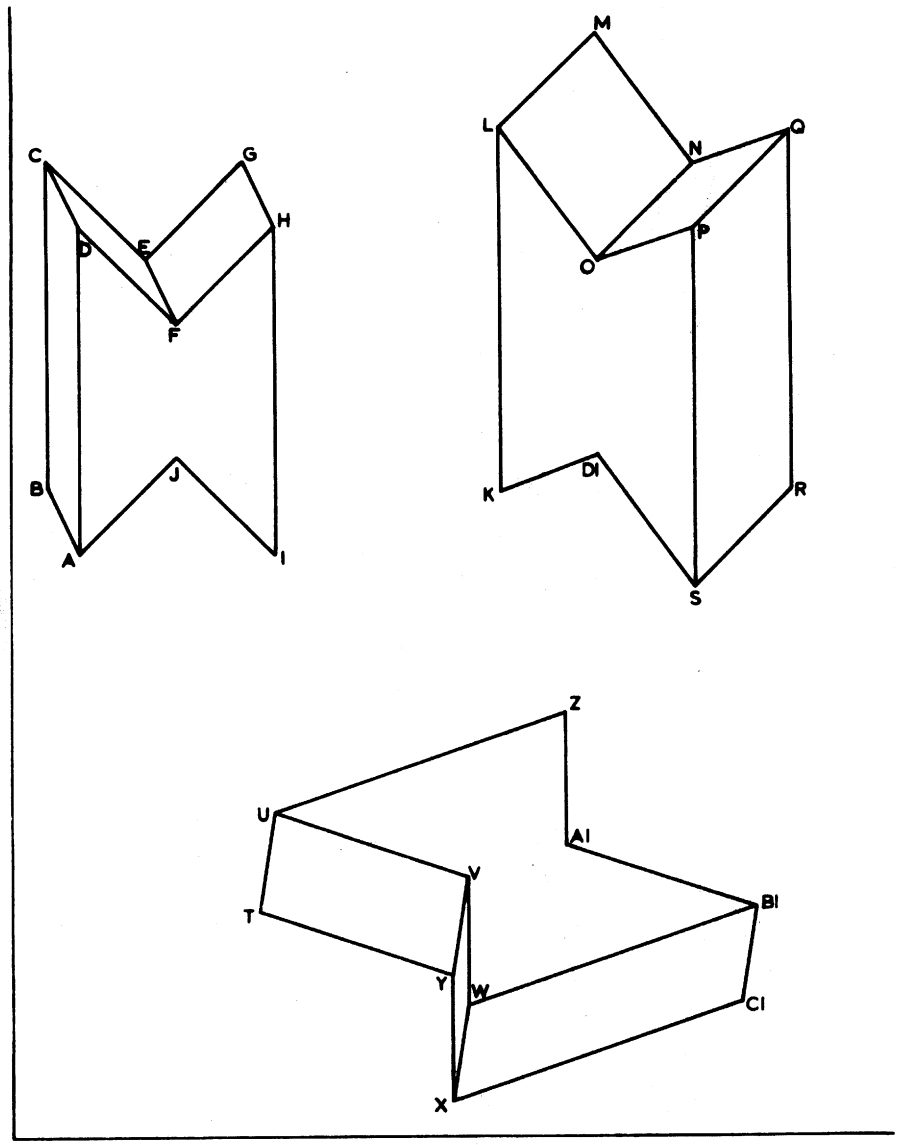


Fig. "X S".
Scene for example 3. Model is called 'X' (see fig. 'EQUIS')

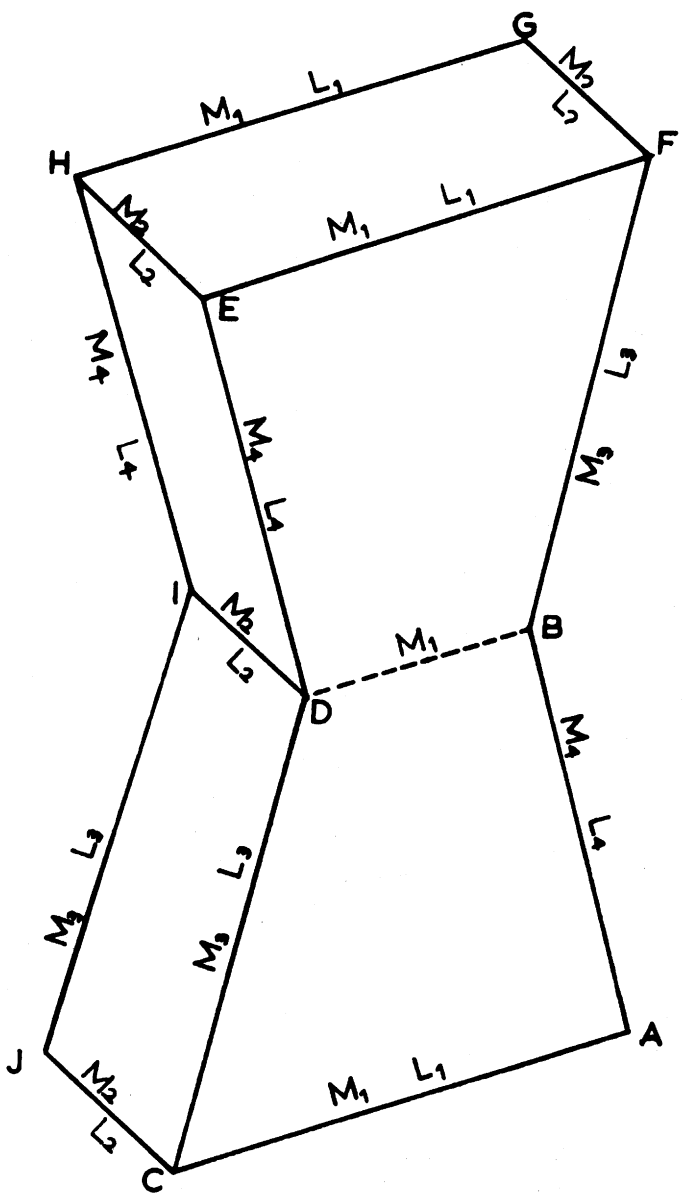


Fig. 'EQUIS'. A model
for fig. 'XS'.

The NIL at the bottom is the remainder of the scene, as allways. In this case is an empty remainder, i. e., the scene consisted only of the searched object (EQUIS).

Nevertheless, when we look for the object EQUIS (page 62) in the figure below, the program fails to identify it.

This is due to the fact that the two dimensional representations are different. Chapter 7 discusses this in detail.

A solution to this is to define a model (like the block in question) as one of several models; FDL-1 has an =OR= for this effect.

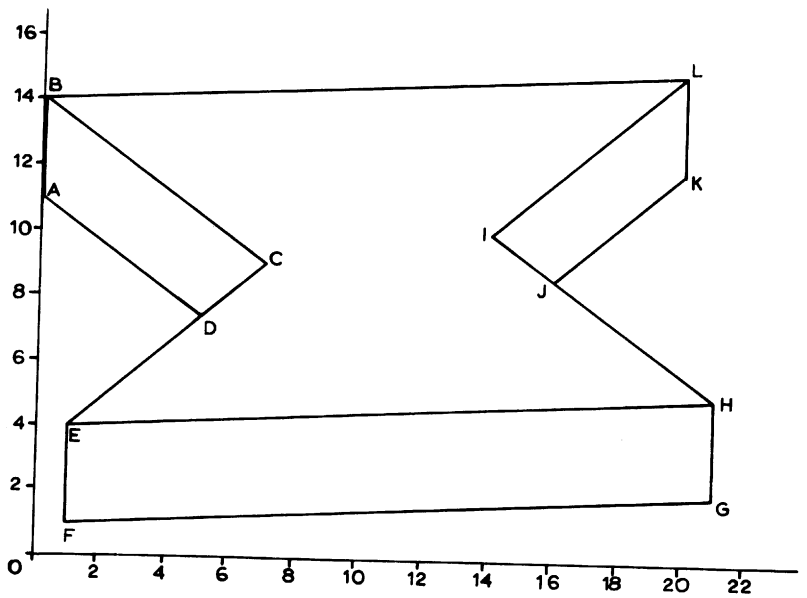


Fig. d24. The model EQUIS (page 62) is inadequate for the identification of this drawing.

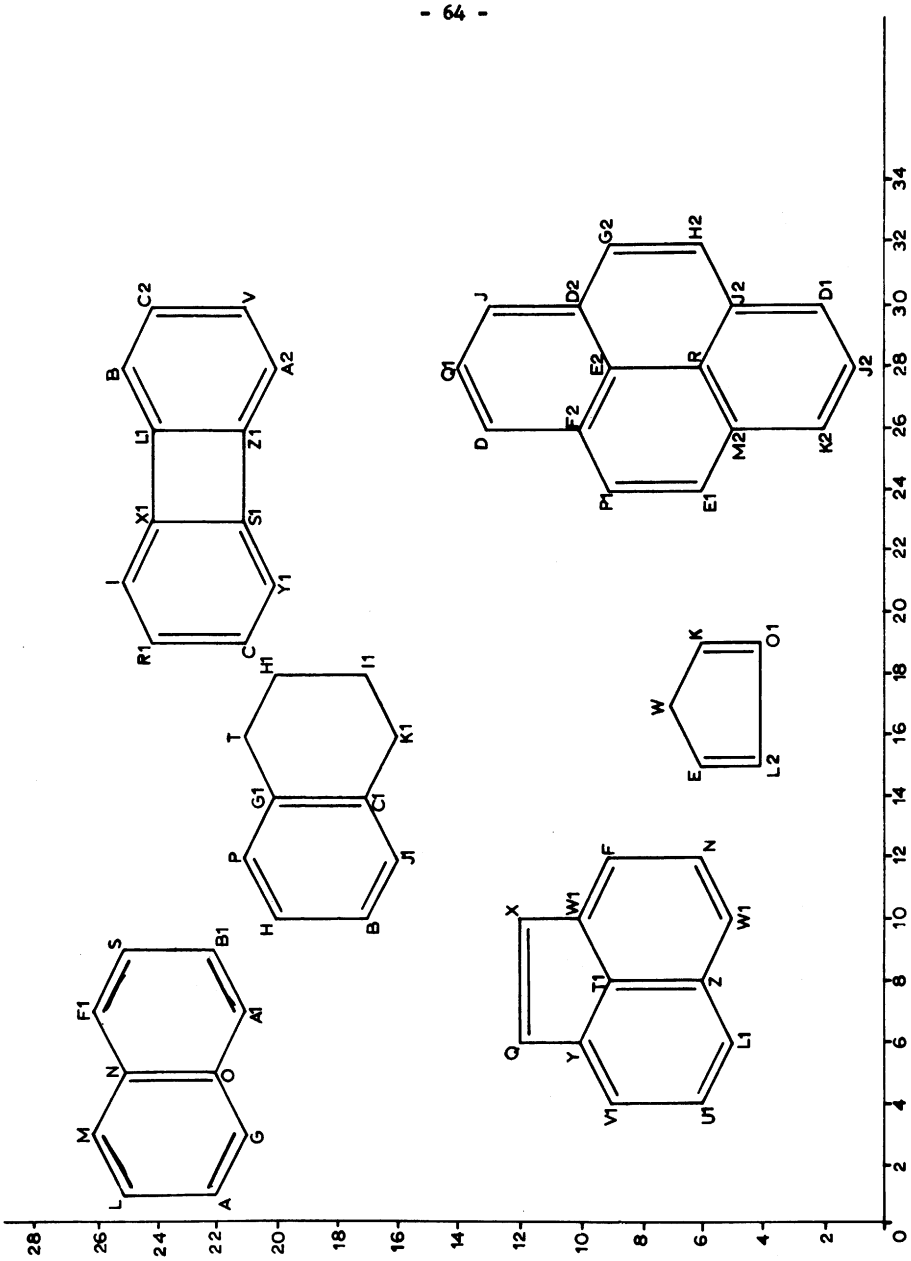


Fig. 'C H E M I S'. This scene was analyzed by TD using the models in the next page. (see example 4).

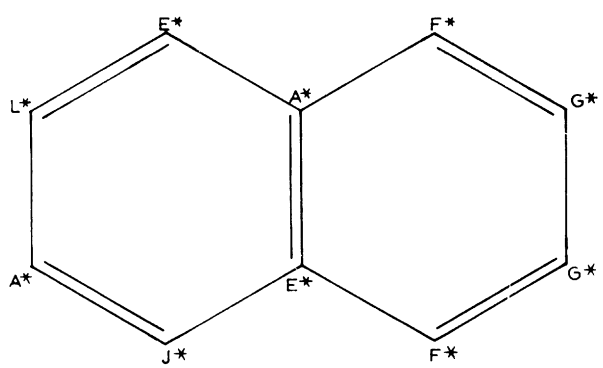


Fig. 'N A P H T A L E N E'. ▲ model.

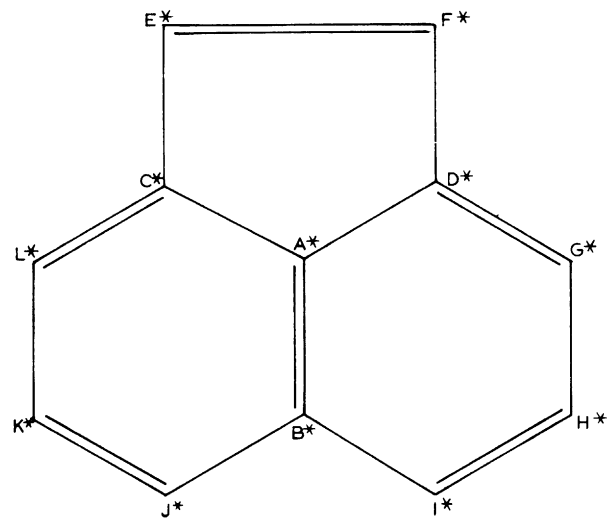


Fig. 'A C E N A P H T Y L E N E'. ▲ model.

Example 4. C H E M I S

--see figures "N A P H T A L E N E" (page 65), ACENAPHTYLENE (page 65), and CHEMIS (page 64).--

Several chemical compounds were looked in CHEMIS, page 64. The results are given below.

```
(NAPHTALENE 1 IS (N O M F1 A1 G B1 S L A))
((P (H G1)) (O (Y X)) (R (E2 M2 I2)) (T (G1 H1)) (U (X1 B2
Z1)) (V (C2 A2)) (W (E K)) (X (Q W1)) (Y (V1 T1 Q)) (Z (t1
L1 N1)) (C1 (J1 K1 G1)) (D1 (I2 J2)) (E1 (M2 P1)) (G1 (P T
C1)) (H1 (T1 I1)) (I1 (H1 K1)) (J1 (B C1)) (K1 (C1 I1)) (L1
(U1 Z)) (M1 (F N1)) (N1 (Z M1)) (O1 (L2 K)) (P1 (E1 F2)) (Q1
(D J)) (R1 (I C)) (S1 (X1 Y1 Z1)) (T1 (Y W1 Z)) (U1 (V1 L1))
(V1 (Y U1)) (W1 (T1 X F)) (X1 (S1 I U)) (Y1 (C S1)) (Z1 (U
S1 A2)) (A2 Z1 V)) (B2 (U C2)) (C2 (B2 V)) (D2 (E2 G2 J))
(E2 (F2 D2 R)) (F2 (P1 E2 D)) (G2 (D2 H2)) (H2 (G2 I2)) (I2
(R H2 D1)) (J2 (K2 K1)) (K2 (M2 J2)) (L2 (O1 E)) (M2 (E1 R
K2)) (B (H J1)) (C (R1 Y1)) (D (F2 Q1)) (E (W L2)) (F (W1 M1))
(H (B P)) (I (R1 X1)) (J (D2 Q1)) (K (W O1)))

(ACENAPTHTYLENE 1 IS (T1 Z Y W1 L1 N1 V1 Q X F M1 U1))
((X1 (S1 I U)) (Y1 (C S1)) (Z1 (U S1 A2)) (A2 (Z1 V)) (B2 ... etc, etc.
```

CHAPTER VI. D T

The present chapter describes a program, written in CONVERT, and run in the PDP-6 computer of Project MAC, M. I. T., which recognizes objects in a scene. Two imputs to the program determine its behavior and response:

1. The scene to be analyzed, which is entered in a symbolic format, called the region-format, somewhat different from FDL-1.
2. A symbolic description --the model-- of the class of the objects we want to identify in the scene.

Given a set of model s of the objects we want to locate, and a scene or picture, the program will identify in it all those objects or figures which are similar to one of the models, provided they appear complete in the picture (i. e., no partial occlusion or hidden parts). Recognition is independent of position, orientation, size, etc.; it strongly depends on the topology of the model.

Important restrictions and supositions are:

(a) the input is assumed perfect --noiseless-- and highly organized.
(b) more than one model is in general required for the description of one object.
(c) partially seen objects may appear in the scene, but only objects which appear unobstructed are recognized.

Work is continuing in order to drop restriction (c) and to improve (a).

A more complete description of DT is found in a Project MAC memorandum[16].

Relation of DT with other parts of this thesis.- DT represents the implementation of a different approach to recognition; it works with regions, instead of lines, as TD does. It is general, and may recognize any model (within its limitations), instead of only parallelepipeds, as Polybrick. Its models are discussed in chapter 7. DT needs improvement to deal with partially occluded objects.

An example of recognition.- This chapter describes DT, a program which, given an scene (such as 'EXAMPLE2') and a model (such as 'CUBE'), will identify all 'parallelepipeds' present in 'EXAMPLE2'. In this case, parallelepipeds 1 and 3 are found; parallelepiped 2 is partially hidden and is not recognized. Both the scene and the model are in symbolic format.

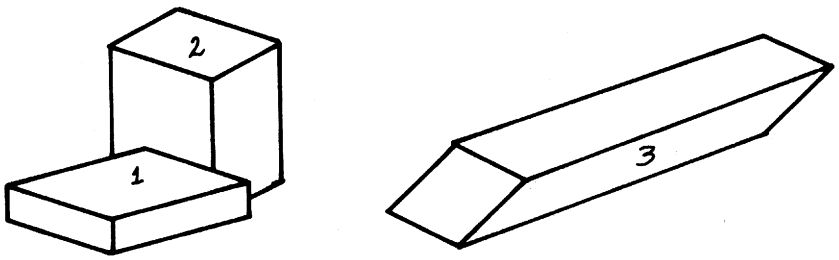


Fig. 'EXAMPLE2'. Three parallelepipeds.

Restrictions: In this first experimental system we will live with the following constraints:

1.- Noiseless data is suposed, i. e., the scene must be accurately described by its symbolic representation. Also, the set of shapes assumed is small, so that we need not worry about heuristic efficiency in algorithms.

2.- Whenever a 3-dim object gives rise to several (2-dim) projections which are topologically different, all these need to be presented as models in order to cover the possible cases. The recognizer has an OR feature for this effect. For instance, fig. 'L' has the same object in four different positions, requiring 3 or possibly 5 models of an 'L' to identify all. The exact number depends on the particular models in question and their "dont-care" conditions, which may depend on what other objects in the world have to be distinguished.

3.- Only objects which are totally seen are recognized. Partially

occluded or hidden parts or bodies may be present in the picture but the occulted objects will not be identified. For instance, parallelepiped 2

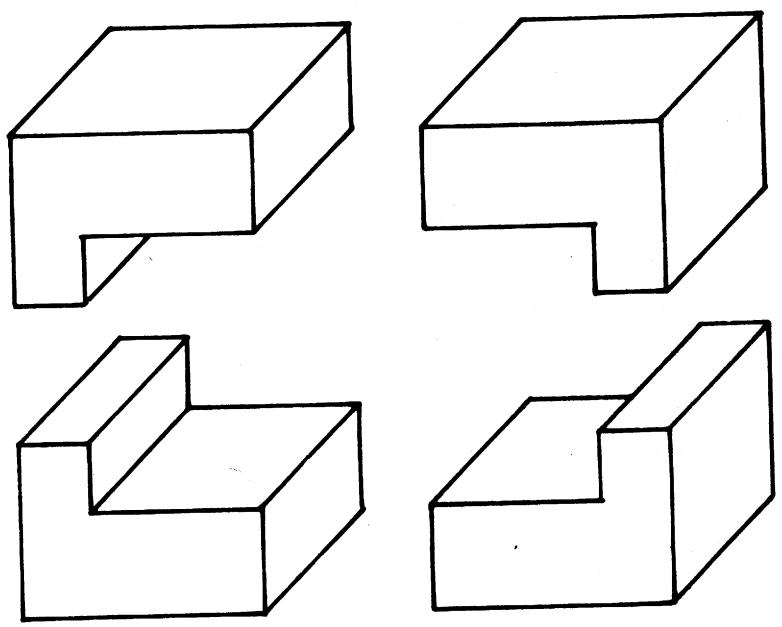


Fig. 'L'
The same object in four different positions, all of which differ in the topology of its two dimensional projection over the plane of the drawing.

in fig. 'EXAMPLE2' was not found. Our current work will help to relax this last restriction, and also restriction (1). The reader unfamiliar with progress in that direction can see references 4, 13, 24 and 29 for some earlier work of that kind.

4.- In the present program we assume orthogonal projections. Later we will considere finite perspective. For small visual angles, a simple tolerance should suffice for most cases, but for large visual angles we will have to use other methods.

THE SCENE

Informally a scene (picture) is a collection of regions (projections of faces); a region is described by an ordered collection of segments (lines or curves), and these have several properties.

A scene is represented by an atom which has under the entry 'regions' a list of the regions composing it; for instance (see fig. 'BOTTLES'), the atom BOTTLES is a scene for which

(GET (QUOTE BOTTLES) (QUOTE REGIONS)) = (A B C D E F G H I J K L M Z)

In this case the regions of 'BOTTLES' are A, B, ... , M, Z.

A region is an atom which has in its property list the entries NEIGHBOR, SHAPE, and possibly others. A region corresponds to a surface or face in the scene, except that it is treated 2-dimensionally; i. e., in fig. 'EXAMPLE2', the upper face of the eraser A B C E L is composed of two regions, namely B and L.

Example.- In the property list of region M (figure 'BOTTLES') we find:

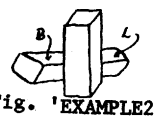


Fig. 'EXAMPLE2'

NEIGHBOR (L Z) (L and Z are limitrophe regions with M)

SHAPE ELLIPSE

At present, the shapes of regions can only be atoms; this is a severe restriction since may be too much to require that the preprocessor recognize region M (fig. 'BOTTLES') as an ellipse or region A (fig. EX2) as a paralle logram. In the models, the shapes are also atoms. This restriction will be abandoned eventually, but now is observed.

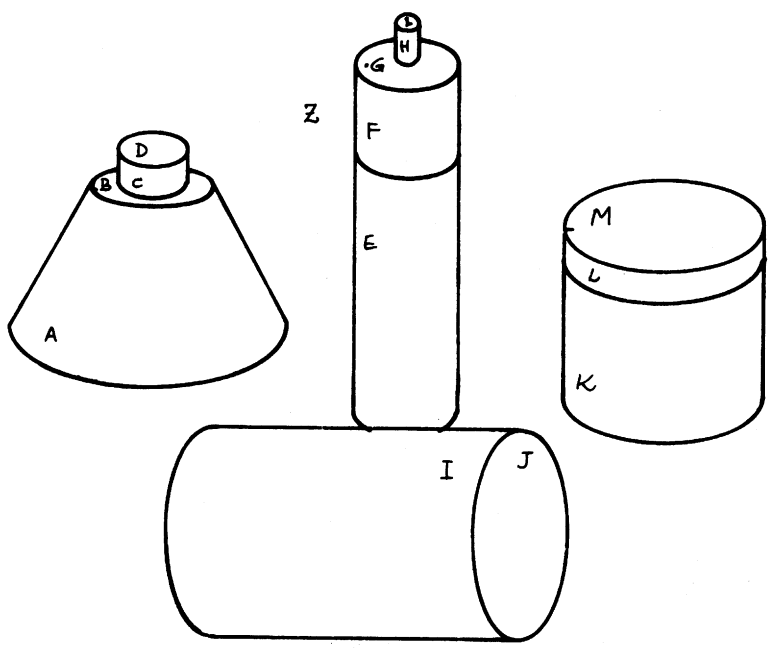


FIG. 'BOTTLES'
An scene composed of regions A, B, ... , L, M, Z.

THE MODEL

A model is an atom which contains in its property list, under the entry

'REGIONS', a list of the following form:

a) the first element of such a list is an atom, the name of the region, as far as the model is concerned.
b) Each of the remaining elements of such a list is a property; specifically, is either a list (NEIGHBOR ...)
or a list (SHAPE ...). More complicated properties will be used when objects start getting more complicated.

A model is composed of regions, with properties inter-relating them.

Given an object, there is a large number of models which correctly describe it.

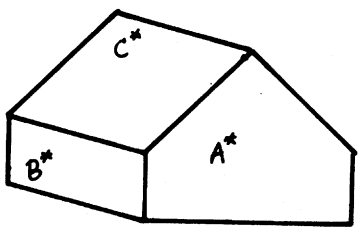


Fig. 'H O U S E'. A model.

Example. The model 'HOUSE' is written in this way (see fig. 'HOUSE'):

HOUSE

(in its property list, we find:)

REGIONS ((A* (NEIGHBOR B*) (NEIGHBOR C*) (SHAPE PENTAGON))
(B* (NEIGHBOR A*) (NEIGHBOR C*) (SHAPE PARALLELOGRAM))
(C* (NEIGHBOR A*) (NEIGHBOR B*) (SHAPE PARALLELOGRAM)))

What this list means is that HOUSE is composed of three regions, namely A*, B* and C*; and that A* is neighbor of B* and C*, etc.

More over, it says the shapes of A* (pentagon), B* (parallelogram) and C* (parallelogram). Additional properties could be inserted here.

The names A*, B*, etc., given to the different faces, have no importance, they act as dummy variables (UAR or 'undefined' variables in CONVERT); the names such as PARALLELOGRAM, PENTAGON, etc., given to the shapes, are

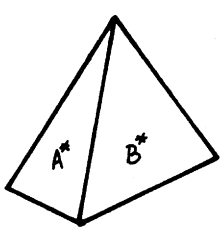


Fig. 'PYRAMID'

crucial, since they are going to be compared by equality with the corresponding names in the property list of the regions of the scene. Note that the models we are using are not "cathegorical" -- they do not contain enough information (usually) to reconstruct the object.

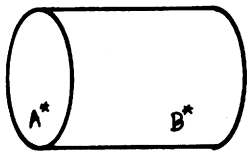
Example.- PYRAMID (see fig. 'PYRAMID') is a model written as

```
(DEFPROP PYRAMID ((A* (NEIGHBOR B*) (SHAPE TRIANGLE))
                  (B* (NEIGHBOR A*) (SHAPE TRIANGLE))) REGIONS)
```

--but also see fig. 'PYRAM1'.--

Fig. 'CYLINDER'. A model.

```
(DEFPROP CYLINDER
    ((A* (NEIGHBOR B*) (SHAPE ELLIPSE))
     (B* (NEIGHBOR A*)
         (SHAPE (I C I D))) )
                          REGIONS)
```



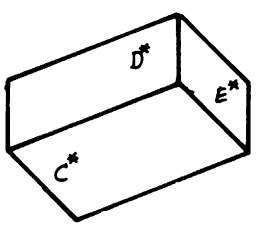
Remark: Note how we describe B*'s shape as

(SHAPE (I C I D)), i. e., as (straight, convex, straight, concave).

Example.- A cube (parallelepiped) is described as

```
(DEFPROP PARALLELEPIPED ((C* (NEIGHBOR E*) (NEIGHBOR D*) (SHAPE PARALLELOGRAM))
                         (D* (NEIGHBOR C*) (NEIGHBOR E*) (SHAPE PARALLELOGRAM))
                         (E* (NEIGHBOR D*) (NEIGHBOR C*) (SHAPE PARALLELOGRAM)))
        REGIONS)
```

Fig. 'CUBE'. A model.
It is really a parallelepiped.



THE RESULTS

We will present now several examples of scenes analyzed by DT, the program, in the PDP-6 computer. The symbol γ marks the lines typed by the user.

```
γ     CONV 4                                   Bring the CONVERT processor from
                                               tape 4.
γ     (UREAD DT LISP 5 ↑Q ↑W)                   Load the file containing DT,
                                                  the recognizer.
γ     (UREAD EX2 LISP ↑Q ↑W)                    Bring the scene EX2 into
                                                  memory (see fig. 'EX2').
γ     (UREAD MOD2 LISP ↑Q ↑W) (IOC V)           Load the models
      (v)
γ     (DT (QUOTE CUBE) (QUOTE EX2))             Look for 'CUBES' in 'EX2".
      (CUBE 1. IS (A B C))                          (see fig. 'EX2').
      (CUBE 2. IS (J L M))
      (D E F G H I K N O P Q R S T U V W X Y Z)  Remaining of scene.

γ     (DT (QUOTE CYLINDER) (QUOTE EX2))         Look for cylinders (see fig.
      (CYLINDER 1. IS (E D))                      'CYLINDER').
      (CYLINDER 2. IS (G F))
      (A B C H I J K L M N O P Q R S T U V W X Y Z)  Remaining of scene.

γ     (DT (QUOTE HOLLOWCYLINDER) (QUOTE EX2))
      (HOLLOWCYLINDER 1. IS (T U S))
      (A B C D E F G H I J K L M N O P Q R V W X Y Z)

γ     (DT (QUOTE HOLLOWBRICK) (QUOTE EX2))
      (HOLLOWBRICK 1. IS (N O P Q R))           See fig. 'HOLLOWBRICK'.
      (A B C D E F G H I J K L M S T U V W X Y Z)
```

We define DD, a FEXPR that suppresses the QUOTEs:

```
γ     (DEFPROP DD (LAMBDA (A) (DT (CAR A)
γ                                 (CADR A)))   FEXPR)
      DD
γ     (DD HOLLOWBRICK EX2)                      Compare with above.
      (HOLLOWBRICK 1. IS (N O P Q R))           Good. Let us see other example.
```

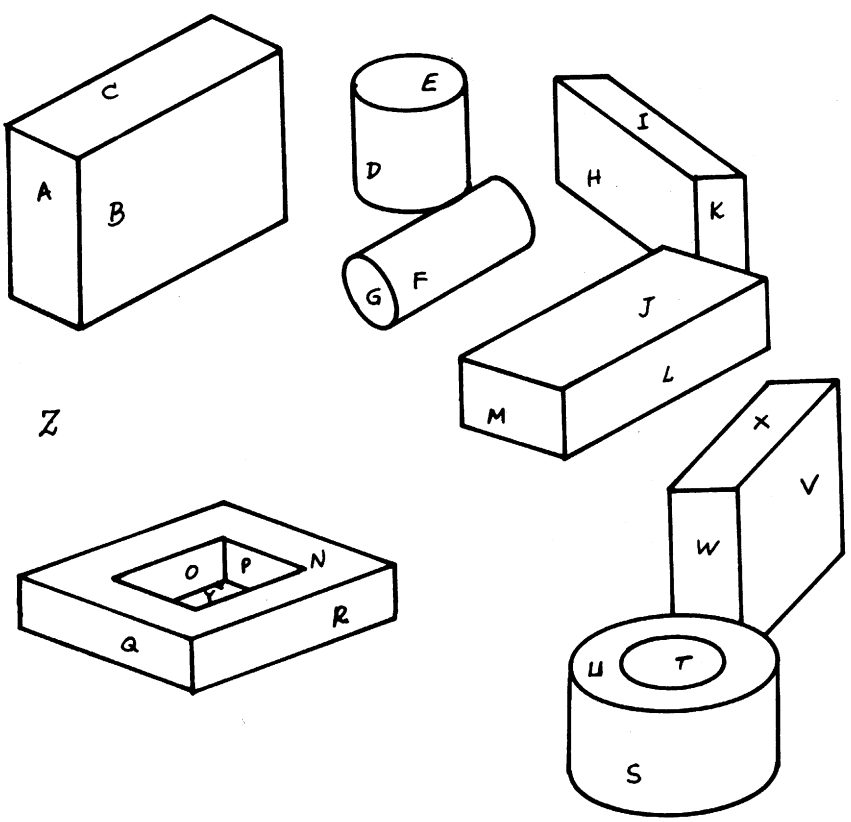


Fig. 'E X 2'. A scene.

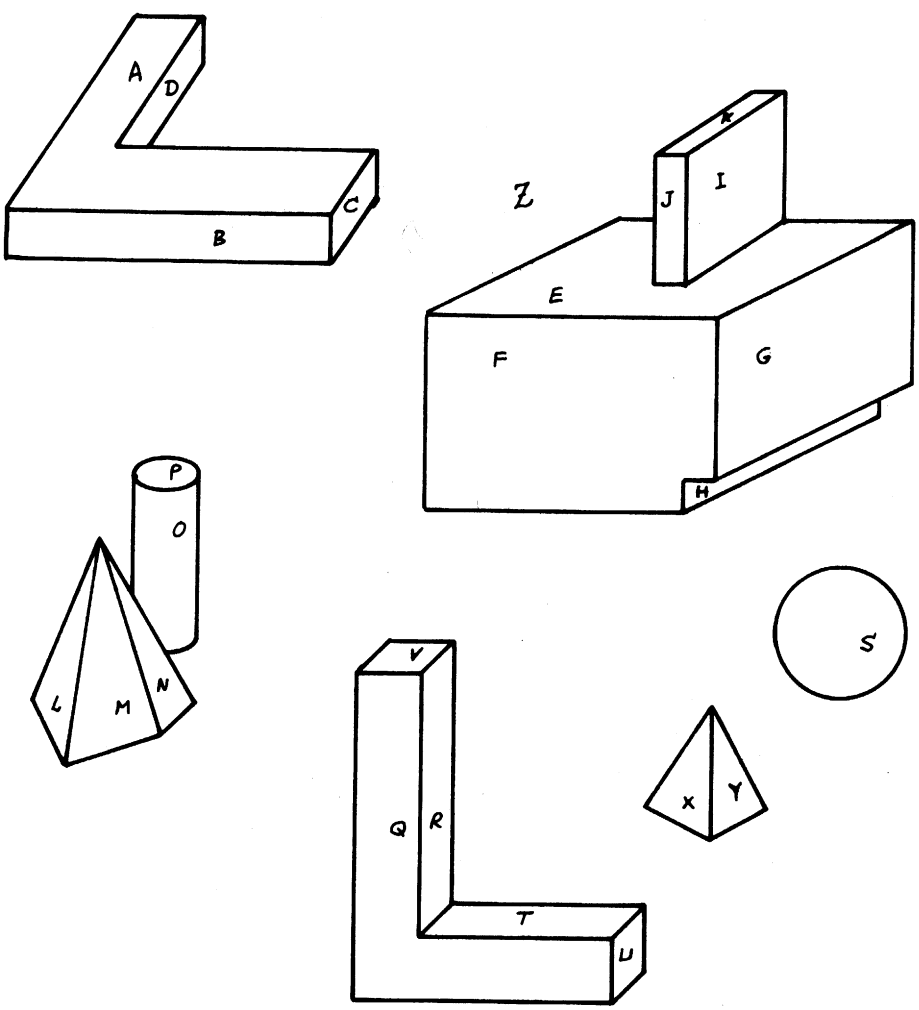


Fig. 'F I G 2". A scene

We analyze now FIG2 (see fig. 'FIG2') with DT.

```
γ   (DD PYRAMID FIG2)                          Looking for PYRAMIDs (see model
    (PYRAMID 1 IS (L M))                        in fig. 'PYRAMID'). DD is like
    (PYRAMID 2 IS (X Y))                        DT, but it is an FEXPR (cf.[28]).
    (A B C D E F G H I J K N O P Q R S T U V W Z)
```

Note that pyramid L M N is not reported as such, but only LM is reported or recognized. Why is this? Because the model 'PYRAMID' (see fig. 'PYRAMID') is composed of <u>two</u> triangles. Also, it is in the nature of our algorithm that L M prevents recognizing M N. In order to get L M N, we define PYRAM1 as a pyramid which has <u>three</u> visible triangular faces:

```
γ   (DEFPROP PYRAM1   ((A* (NEIGHBOR B*) (SHAPE TRIANGLE))
γ                      (B* (NEIGHBOR A*) (NEIGHBOR C*) (SHAPE TRIANGLE))
γ                      (C* (NEIGHBOR B*) (SHAPE TRIANGLE)) )    REGIONS)
    PYRAM1                                  See fig. 'PYRAM1'.
```

Now we apply this model to scene FIG2:

```
γ   (DD PYRAM1  FIG2)
    (PYRAM1  IS (L M N))
    (A B C D E F G H I J K O P Q R S T U V W X Y Z)  Aja. Only one is found.
                                            Correct. Only one pyramid with
                                            three visible faces is present in FIG2.
```

What we really want is to define a pyramid as something which shows either two or three triangular faces; so,

```
γ   (DEFPROP  PYR  (OR  PYRAMID  PYRAM1)  REGIONS)
    PYR                                     The last model of an OR, PYRAM1
                                            in this case, is searched first.
```

At this moment, PYR is a model which stands for either or

```
γ   (DD  PYR  FIG2)
    (PYRAM1 1 IS (L M N))
    (PYRAM1 2 IS (X Y))                     Good. Two objects were found to
    (A B C D E F G H I J K O P Q R S T U V W Z)   match with PYR: (L M N)
                                                 and (X Y). See fig. FIG2.
```

What would have been happened if we define PYR in the reverse order? Let us define

```
γ   (DEFPROP  PYR  (OR  PYRAM1  PYRAMID)  REGIONS)
    PYR
```

The last model in the OR list, PYRAMID in this case, is searched first. The answer is:

```
γ   (DD  PYR  FIG2)
    (PYRAMID 1  IS (L M))
    (PYRAMID 2  IS (X Y))                    Two objects matched with PYRAMID:
    (A B C D E F G H I J K N O P Q R S T U V W Z)   (X Y) and (L M); after
                                             this, no object was found
                                             to match with PYRAM1.
```

Conclusion: Order in the models is important, so long as we leave things to the normal CONVERT matching algorithm.

```
γ   (DD  PYR  FIG3)
    NIL                                      FIG3 is an empty scene.

γ   (DD  CYLINDER  FIG2)
    (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)  No cylinders.
                                             Cylinder P O is partially occulted,
                                             so is not found.

γ   (DD  CUBE  FIG2)
    (CUBE 1  IS (I J K))
    (A B C D E F G H L M N O P Q R S T U V W X Y Z)

γ   (DD  ANGLE  FIG2)
    (ANGLE 1 IS (D A B C))
    (E F G H I J K L M N O P Q R S T U V W X Y Z)
```

Angle is a model described in the next page (see fig. 'ANGLE'). Angle Q V R T U was not found because has a different form (its two dimensional projection has a different topology from model 'ANGLE'; namely, has 5 faces or regions, and 'ANGLE' only 4).

Angle E F G H was not found because it is partially occulted.

```
γ   (DD  SPHERE  FIG2)
    (SPHERE 1 IS (S))
```

Some models.

```
(DEFPROP ANGLE ((A* (NEIGHBOR B*) (SHAPE FUNNY))
                (B* (NEIGHBOR A*) (NEIGHBOR C*) (NEIGHBOR D*) (SHAPE ELE))
                (C* (NEIGHBOR B*) (NEIGHBOR D*) (SHAPE PARALLELOGRAM))
                (D* (NEIGHBOR B*) (NEIGHBOR C*) (SHAPE PARALLELOGRAM)) )
                                                 REGIONS)
```

This is the model for ANGLE. See figure in next page. Angle was used in FIG2.

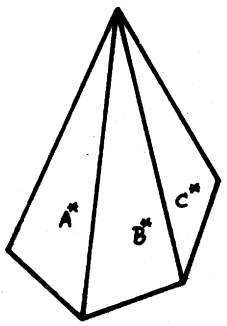


Fig. 'PYRAM1". A model.

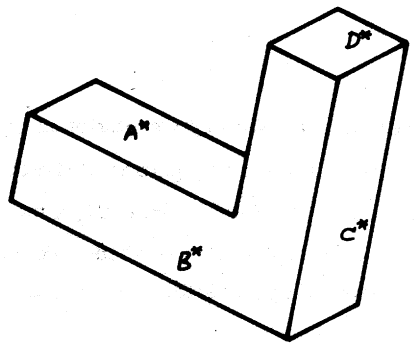
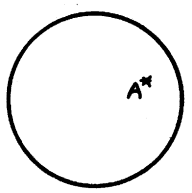


Fig. 'ANGLE'. A model.

```
(DEFPROP PYRAM1 ((A* (NEIGHBOR B*) (NEIGHBOR C*) (SHAPE TRIANGLE))
                 (B* (NEIGHBOR A*) (NEIGHBOR C*) (SHAPE TRIANGLE))
                 (C* (NEIGHBOR B*) (SHAPE TRIANGLE)))      REGIONS)

(DEFPROP SPHERE
     ((A* (NEIGHBOR ==) (SHAPE CIRCLE)))
                 REGIONS)
```

Fig. 'SPHERE'. A model.



CHAPTER VII. MODELS

A model is a written representation of an object that we want to identify. Models are mainly used for recognition of the object they represent; they are similar to patterns in CONVERT. Generally, a model can represent a large class of objects.

We have already talked about models in TD (see notation FDL-1) and in DT; the purpose of this chapter is to discuss them more systematically.

2-dim representation of 3-dim models.

2-dim models are capable of representing either two or three dimensional objects. This is possible since, in analyzing a scene or a picture, we may considere a 3-dim object as a 2-dim portion of the picture formed by several 2-dim regions (surfaces). In describing the model, the inter-connection of the vertices of the object is given, plus additional properties or constraints between different features (points, corners).

We will simultaneously talk about two types of such a representation; in one of them (see fig. 'PARAL'), a whole 3-dim object is described by the structure of its edges, as used in TD and Polybrick (chapters 5 and 4)[15, 13], and is called edge-representation or notation. The other type uses regions as building blocks of models; it is called the region-representation or format, and is the one used by DT (chapter 6) [16] and some of the vision group programs [22, 33, 38].

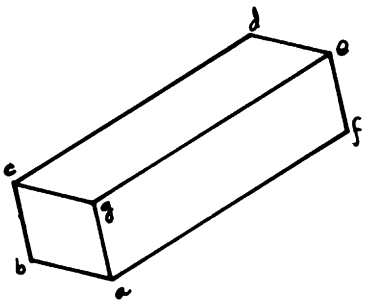


Fig. 'PARAL'. Representation of a parallelepiped as a 3-dim model.

Models written in edge-notation.- We give as example the parallelepiped of figure 'PARAL', which may be represented (written) as

(a (b g f) b (c a) c (d g b) d (e c) e (f g d) f (e a) g (c e a))

plus the additional properties

(slope b a m1) (length b a L1)
(slope c g m1) (length c g L1)
(slope d e m1) (length d e L1)
(slope b c m2) (length b c L2)

(1)See the FDL-1 language in chapter 5.

```
(slope a g m2)    (length a g L2)
(slope f e m2)    (length f e L2)
(slope c d m3)    (length c d L3)
(slope g e m3)    (length g e L3)
(slope a f m3)    (length a f L3)
```

Plus the additional [pseudo]property

(variables m1 m2 m3 L1 L2 L3)

which indicates (see fig. 'PARAL' again) that the symbols m1, ..., L3 are dummy variables that may have any value, the only restriction being that this value be the same for each occurrence of the symbol. Variables which behave in this form are called bound variables in logic, and UAR (undefined variables) in CONVERT. Under this convention, we see that

(slope b a m1)

(slope c g m1)

(slope d e m1)

means three parallel lines.

Properties such as (slope b a m1) are in general function-predicates which have as arguments vertices and undefined variables; the user may define arbitrary (LISP) properties, which represent constraints on the figure or object that the model has to match.

Models written in region-notation.- Surfaces (faces) are given names, and the neighborhood relation between them is indicated; in addition, each region has a description of its shape, pretty much in edge notation. Fig. 'PARALE' (page) is described in this way.

Vertices are treated as being two-dimensional, that is, the coordinates with respect to the (frame of the) picture or scene are used; coordinates are then those of the projection over the plane of the drawing; all the

points of a model are coplanar, so z-coordinate is not indicated.

Multiple models for the same object.- A three dimensional object will, in general, have more than one 2-dim representation; for instance, the body with an L shape shown in figure 'ELE' will have three or four (5?) models, according to the position from where you are seeing it.

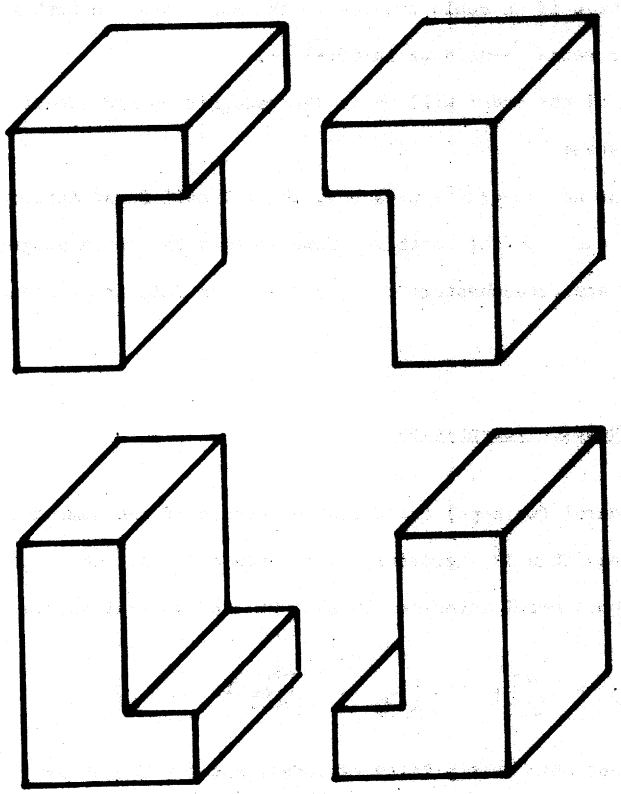


Fig. 'ELE'. A three dimensional object has four or more different representations as a model, if the model contains only 2-dim information about the (relative) position of the vertices.

Our goal in this chapter is, using the representations for models that we just described, to develop a notation in which models will be easily expressed; to investigate different conventions regarding to the model attached to a given object; that is, given an object, how will we write its model?

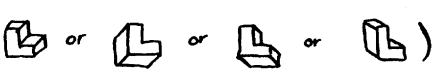
It would be nice if we could express in the same notation both the model and the figure or scene --as done in FDL-1 --.

The main use of the model will be in the recognition and identification of objects in a scene.

I will present now several approaches, which I call First Approach, Second Approach, etc., to this problem. Some of them have been programmed, tested, thought, etc.(see chapters 4, 5, and 6), and this is so indicated when they are.

First Approach; Multirepresentation.

Since, in general (we hope) the different models of the same object will be just a few (less than 5? Certainly, less than a dozen), we could define a complex (compound) model compossed by the OR of several simple ones, such as

ELE = ()

That is: we accept this multiplicity of models and try to get used to it. The program DT works in this way, using 2-dim models of 3-dim objects; the description of the models is made in terms of regions, instead of lines. In section of chapter 6 we see an example of this kind of recognition-identification (pages 77-78): the way DT finds pyramids, in the figure 'FIG2'.

The program TD also works in this way (cf. chapter 5), using 2-dim models of 3-dim objects; the descrpition of the model is made in terms of lines --instead of regions-- pretty much the way we have been describin models in this section. See the way TD recognizes 'Xs' in figure 'EQUIS' (page 61). Note also that, if we do not define the complete set of different models of an object, we run the risk to fail to recognize the object (see figure d24, page 63) when is found in the scene in some positions.

Conjecture: If for each object we write all its possible models we may run out of storage. May be not, may be fairly simple objects may be represented by just a few 2-dim models.

Order in the models.- When our model is compound, that is, when we have
MOD = (OR MOD1 MOD2 ... MODm)
then the recognition is done (we are talking of programs DT and TD) from right to left: we find all the instances of model MODm in the scene, and erase them; then all instances of model MOD_{m-1}, etc. In this way, rare representations of the object could be included in the OR list, to be used only when more usual models have failed.

Second Approach: Two-dimensional Patterns.-

This could also be considered as an extension to the first approach. Properties, defined by the user, may be very complicated functions of the (coordinates of the) vertices [cf. FDL-1, chapter 5], value of slopes, distances, etc. Nevertheless, these properties are attached to a mesh of connections specifying the seen edges, which is topollogically

invariant[1]. We would like to be able to have ways to specify variations, modifications and additions to this network, in a rich and systematic way. Using solution 1 (first approach) is not enough: suppose I want to define a STAR as an object having an arbitrary (bigger than 3) number of equal "peaks", equally distributed, as in fig. 'STARS'.

Fig. 'STARS'. Different objects which could fall under the same generalized model.

We do not want to say STAR = (STAR1 or STAR2 or STAR3 or ...)

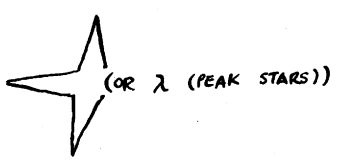
We would like to say

STAR = (PEAK PEAK PEAK STARS)

where STARS = (λ or (PEAK STARS))

λ is the null string.

More pictorically, and more informally too,

S T A R = 

P E A K =

How good could this approach get?

If we were going to specify patterns for a lineal string (or for an S-expression), this would be the approach we would take. Observe how easy is

---

[1] It is interesting to note here that Evans [8] used essentially this kind of representation, but he used few attached properties.

to say in CONVERT

(== == ==) for any list containing exactly three elements;

(X === X), with X as UAR variable,

for any list with two or more elements, the first equal to the last, but otherwise totally arbitrary;

(EVEN) PAT ((*OR* () (== == EVEN)): this definition of the fragment EVEN makes possible for (EVEN) to stand for a list with an even number of elements, but arbitrary otherwise.

Probably an extension of this notation will allow us to specify two-dimensional patterns, in a CONVERT-like manner, which will then be used for matching, i. e., for recognition of objects in a scene. In order to achieve this, we have to specify

syntax
- --- the primitive constituents (primitive patterns) of our notational language.
- --- the ways new patterns are formed from patterns.

semantics --- the way the matching or identification is carried out; that is, what a 2-dim pattern stands for.

Non-trivial problems to solve are also:

- --- to find a good written representation of the patterns.
- --- the internal (machine) representation of the patterns.
- --- the interpreter (recognizer) for such 2-dim patterns; i. e., the algorithm which the machine will use in order to carry the match or comparison, expressed in a meaningful language(*).

Incidentally, in this last point questions of efficiency in time (speed of execution), efficiency in space (size of program + magnitude of intermediate swell[1] + extent of data), efficiency in use (easiness of writing,

(*) that is, in a language which the machine is able to understand/execute.

(1) phrase used by Tobey [35].

understanding, modifying and debugging a program, model or pattern), etc., have to be considered --the so-called implementation details-- . For the moment, through the remaining of this chapter, we will not worry about implementation and we will use for written representation of the patterns a mixture of line drawing and atomic symbols, as we already did with STAR (cf. page 86).

Let me point out briefly the syntax and semantics of CONVERT-patterns, that is, linear --unidimensional-- strings of symbols, and then I will do the same for 2-dim.

UNIDIMENSIONAL (CONVERT-type) PATTERNS.

Terminal Patterns. () stands for ()

== stands for or matches any S-expression.

=ATO= matches any atom.

A some other atom, if it does not appear with a definition in the dictionary, stands for itself; it will match only with an identical atom.

==== matches with any fragment such that the remainder of the pattern finds an acceptable match with the remaining of the expression under comparisson.

There is a way to define boolean combination of patterns.

Definitions can be done in several ways in the dictionary, and we may define a single atom to represent a whole pattern, this last being either an S-expression or a fragment.

Recursive definitions are possible.

Concatenation: the pattern (P1 P2 ... Pm) where P_i are patterns, stands for a list of m elements (E1 E2 ... Em) such that each element E_i is represented (is matched) by the corresponding pattern P_i.

A way exists to isolate subparts of a pattern and to have them available for future analysis or for other purposes.

2-dimensional Models.- TD and Polybrick use models where the connection matrix is in terms of the vertices, as ilustrated in fig. 'PARAL'; that is, atoms represent points. The notation called FDL-1 (chapter 5) is also developed with this convention.

A model as used by TD is a list of the form

(connectionlist 'WHERE' properties)

where connectionlist is a list of points and neighbors (example refers to figure 'PARALLELOGRAM'): (A (B D) B (A C) C (B D) D (A C))

properties is a list of properties:

((slope b c m1) (slope a d m1) (slope b a m2) (slope c d m2) (variables m1 m2))

DT and the summer-vision group programs [12, 16, 22, 27, 33, 37, 38], on the other hand, are using regions (faces) as elementary constituents among which the relations of neighboorhood are specified; in models for DT, atoms represent regions. For instance, the same figure 'PARAL', a parallelepiped, is described as

```
( (A* (NEIGHBOR B*)
      (NEIGHBOR C*)
      (SHAPE PARALLELOGRAM))
  (B* (NEIGHBOR C*)
      (NEIGHBOR A*)
      (SHAPE PARALLELOGRAM))
  (C* (NEIGHBOR B*)
      (NEIGHBOR A*)
      (SHAPE PARALLELOGRAM)) )
```

As we see, 'SHPE' indicates the shape of the region; in this case, the

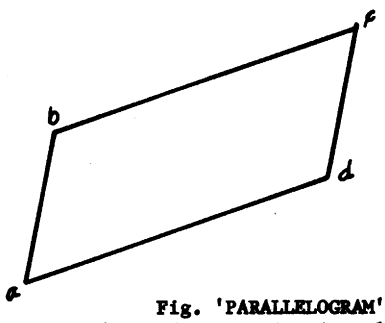


Fig. 'PARALLELOGRAM'.
A model in edge-notation is a list of three elements: the connectionlist, the conjunction 'WHERE' and a list of properties. See FDL-1 in chapter 5.

shape is an atom, 'PARALLELOGRAM'; in general, it will be a list of points and segments; undefined variables are local for each model, but not for each region: if two regions of the same model mention the same undefined variable, this atom will in fact represent the same quantity, but if the same variable is mentioned in two models, no relation holds between them. In this way, slopes, lengths, etc., are transmitted between regions.

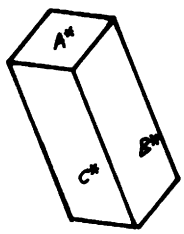


Fig. 'PARALE'.
Model of a parallelepiped.

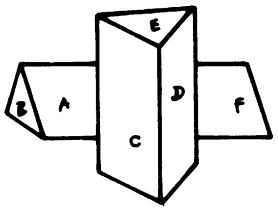


Fig. 'WEDGES'. A scene.
The regions A and F have to be fused together and the result to match a region-model with shape 'parallelogram'.

Differences and Similarities between the two representations of models.-

Both representations use symbolic descriptions of an object, suitable for comparison or recognition (matching); the edge-representation (as in TD) is easier to understand and contains less redundant information; the representation by regions (as in DT) is more cumbersome to read; it has more repetitions of information.

There is a good advantage in using the representation by regions of a model: the comparison is made using bigger "elementary units", so the resulting program is less complicated (compare the sizes of TD and DT); I also believe the match is done faster, because the tree has less branches.

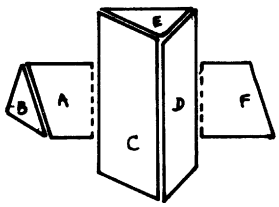


Fig. 'WEDGES...'
False segments are found in regions A and F; they are the dotted ones.

Other advantage of the region-representation seems to be evident when dealing with two distinct regions that are really one; for instance, in figure 'WEDGES' we have to realize that regions A and F are really continuation one of the other, and that the union[1] of both will form

[1] Not really union, because one has to assume the hidden part...

a gregion (general region) that will match a region-model having the shape 'parallelogram'; this being the case, if the scene is represented in terms of regions, it is easier to identify regions A and F as 'mergeable' and to construct from them the gregion A F. Of course, this is also possible, but cumbersome, when scene 'WEDGES' is represented in FDL-1 format, that is, by edges.

Another advantage in using the region-representation of a scene is that it allows one to talk, for each region, of 'spurious' boundaries, that is, boundaries that do not actually belong physically to the region, but are the result of superpositions. DOTS is the name of a program that analyzes each region and tries to determine, using the information about T-joints (terminology explained in chapter 8), which boundaries are 'false' and marks them with 'dots'.

A given segment may be 'false' with respect to one region, but 'true' with respect to the neighbor region; this is a property of a pair region-segment.

For instance, DOTS converts figure WEDGES into figure 'WEDGES...'

TWO-DIMENSIONAL PATTERNS (CONVERT-type models).

A terminal patter is a model with no special marks, with or without properties. (In this section, we generally refer to the edge-notation, as TD uses it, but our remarks apply also to any other representation of a model; we will use a mixture of line drawings and atomic symbols as written representation of models).

Up to this point, all the models have been of the type 'terminal patterns',

with the exception of the (OR MOD1 MOD2 ...) model.

(OR MOD1 MOD2 ... MODn). This pattern will match with a figure if this figure matches one of the models MOD_i; the first (rightmost) that matches is accepted, no more are tried.

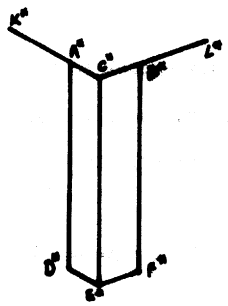


Fig. 'LEG'. A semi-model. The tying points, A* and B* here, have additional lines or edges that connect the semi-figure that this semi-model represents, to a bigger figure.

Semi-models.- With this name are designated patterns that are terminal patterns, except that they are joined to a bigger figure by some points, the tying points. In fig. 'LEG', A* and B* are tying points.

A tying point (like B* in fig. 'LEG') must have as neighbor, in addition to the specified 'normal' points (C* and E*), a point L*, nothing of which is known. K* is also a'missing neighbor' of the tying point A*.

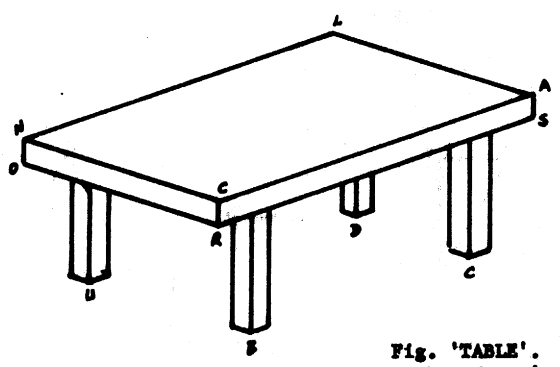


Fig. 'TABLE'.
It has five 'LEGS'.

TD is capable of handling semi-models; for instance, when we look for LEGS in scene 'TABLE', five legs are found: U , B , C , D , (R O N C A S) .

The way to specify a semi-model in TD notation is simply to avoid talking, in the connection-matrix, about the points K* and L* :

(A* (K* C* D*) B* (C* L* F*) C* (A* B* E*) D* (E* A*) E* (D* F* C*)
F* (E* B*))

Note that we mention that A* has K* as neighbor, but we do not say which are the neighbors of K*.

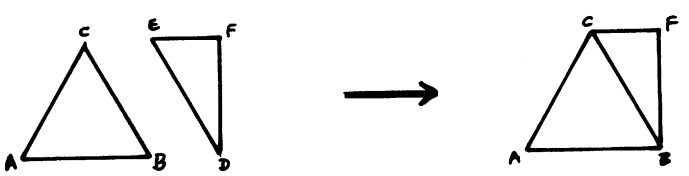
Properties may use the coordinates of K* and L* (the "missing points"); for instance, we could ask for the same slope between lines K* - A* and A* - C* (see again fig. 'LEG').

Union or Concatenation of Patterns.- New patterns are formed from old ones by soldering together some of their points; see =TIE= statement in FDL-1. The pattern has the form

(=TIE= PAT1 N1 PAT2 N2 ... PATm Nm (union1 union2 ... unionk))

For instance, the following figure may be described as

```
(=TIE= (A (B C) B (A C) C (A B)) (1)
       (E (D F) D (E F) F (E D)) (2)
       ( (C 1 TO E 2)
         (B 1 TO D 2) ) )
```



This feature is not implemented in TD. Originally we proposed to do it by forming a new terminal pattern which would be equivalent to (=TIE='...) but simpler than it.

The former pattern would be converted (by TD, at some stage) into

(A (B C) B (A C F) F (C B) C (A B F)) plus properties. Since we can tie a figure to itself several times, renaming of the vertices has to be done; we use here the GENSYM capabilities of LISP and CONVERT.

Comment: It looks like this way of concatenation is easy but messy.

A different representation.- Daniel Conrad(1) is interested in the generation of recursive figures. A line is represented as a series of n points that are r units apart lying in the direction d, or (n, r, d).

A figure is a dictionary of cycle 2 of the form

(N V1 F1 V2 F2 V3 F3 ... Vn Fn)

V1 is the first line in the figure. Since each line or edge of a figure should end at a vertex, it may be referred to as a vertex.

F1 is a figure built on the vertex B1. It in turn has the same form (N' V1' F1' V2' F2' ... Vm' Fm') as the larger figure of which F1 is a subfigure.

V2 is the next vertex. It will always have its tail on the tip of the previous vertex.

F2 is the subfigure of V2 ...

These figures are plotted in the printer.

A different representation.- William Martin [25] also displays figures, this time in the scope of the PDP-6.

Several others [19, 29, 34] have symbolic representations for the

(1) Planar figures and LISP functions to manipulate them are described in [6]; the use of CONVERT to construct these figures is explained in [7].

purpose of constructing figures. I will not discuss their work here.

So far we have seen two approaches to the use of models for identification. A third will be presented now.

Third Approach: 3-dim transparent models (edge-representation).- These models can be considered as a 3-dim wired structure, wires corresponding to edges, plus properties establishing restrictions between vertices, slopes, lengths, etc. For instance, a tetrahedron will be modelled as follows (see fig. 'TETRAHEDRON'):

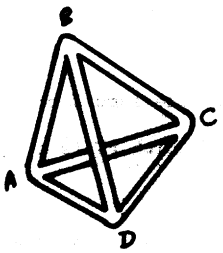


Fig. 'TETRAHEDRON'. This is a three- dimensional model; it is seen here resting on a table.

```
((A (B C D) B (A C D) C (A B D) D (A B C))  where
    ((length a b nl) (length a c nl) (length a d nl)
     (length b c nl) (length b d nl) (length c d nl)
     (variables nl)) )
```

The vertices of these models have 3 coordinates; properties now refer to coplanarity, etc.

We only need one of these 3-dim models to describe completely an object; the problem is how to compare this 3-dim model against a 2-dim scene.

A possible way, which we began to think about for a while, was to direct the machine to use information of which lines may possibly be occulted, given that certain others are already seen.

That is, the model should contain enough information, or the program should be written in such a way, that after having identified some lines, it would be possible to predict or know which lines of the 3-dimensional model are necessarily hidden, so as not to look for them.

For instance, suppose we are looking for 'Fs" (see fig. 'F') in a scene, and we have already found lines K-A, A-B, B-G, B-C, C-D; then the program would recognize that line E-F would be occluded, and would not try to find it.

The trouble with this approach seems to be the sophistication of the program necessary in order to "predict" the lines which are going to be occulted and the lines that are required to be present in the scene. This situation could be somewhat alleviated if the user supplies --as part of the model-- for each point of view the list of visible and invisible lines (or regions) from that position. Instead of a true-false dichotomy for visibility, we could have several cathegories: visible - partially visible -- invisible - ; possibly others, e. g., all - or - none.

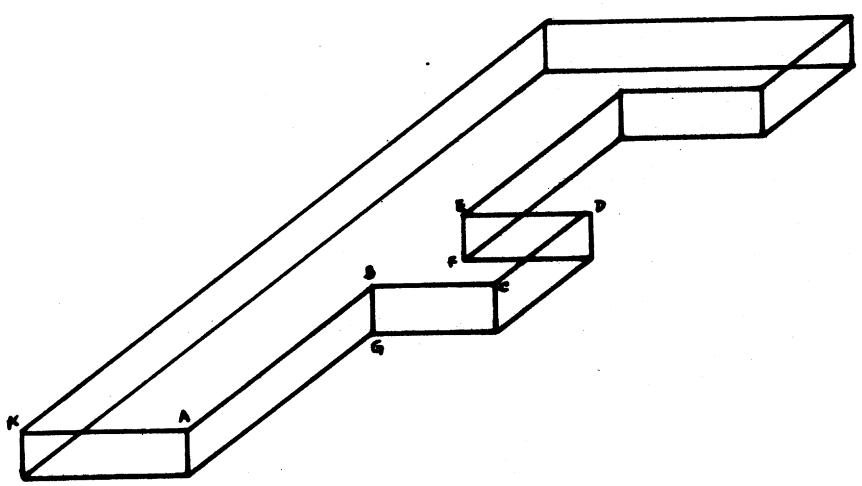


Fig. ' F '. A transparent 3-dim model.

3-dim models projected into 2-dim patterns.- In the last parragraphs we have explained the idea of a 3-dim model with associated lists, one for each direction of view, these lists containing information about the regions or lines visible from that particular line of sight, and the lines which the object itself makes invisible from that direction of view.

It could be possible for a program to produce itself this lists, that is, when comparing the scene with the 3-dim model, to use the results of this comparisson in order to get the best "line of sight";best in the sense that, if we see from that position our 3-dim model, we will obtain a 2-dim projection that would closely resemble that part of the scene under comparison, if the model and the object are really the same. The scene will drive the construction. In this way, we are producing the 2-dim model as meeting the requirements of the scene[(1)], but at the same time the model we should produce has to be a projection of the 3-dim model of the object we want to identify, so recognition is achieved.

Perhaps the main difficulty in this approach is the fact that very little is known about symbolic projections. Also the amount of computation might be large.

Numerical Models.- Roberts [29] uses 3-dim models; these are numerical in nature, and are represented by lists of tied blocks connected in rings. See the Coral language [34] for this ring structure.

Each model block is tied to lists of its points, lines and surfaces.

(1) When we eventually finish the construction of the 2-dim model, matching against the corresponding part in the scene will be easy, and could be reduced to a simple check-up, since we tailor the model to produce (some of) the regions in the scenee.

Curved Objects.- Objects containing curve edges are represented by the same kind of models we described for rectilinear objects. We have now more than one kind of segments joining two points, and some notation must be used for them[1].

When the surfaces adopt sophisticated curvatures and inflexions, the kinds of models we have described will be inexact. There are major conceptual problems to be faced if we are to find really good models for intricately-curved surfaces. We can perhaps take a gloomy comfort in the fact that humans are very weak (much weaker than they think!) in their mental ability to deal with such things.

---

[1] A straightforward representation of curve line segments is given by White [38].

CHAPTER VIII. DISCUSSION OF SOME SCHEMES FOR RECOGNITION.

The following subject is treated in this chapter: assume that a preprocessor (see chapter 3) has transformed a scene into a line drawing or a set of regions, and that a symbolic description of them is available. Independently or otherwise[1], the computer has also in its memory a collection of lists or patterns called models (see chapter 7), which define objects or classes of objects we want to find or recognize. We discuss here some algorithms which, using as data the symbolic description of the scene and the models of the objects we are interested in, goes ahead and finds them. Chapter 2 talks about some of the problems that we expect to meet.

Some of these algorithms or variations have already been put into practice; see chapters on TD, DT and Polybrick.

---

[1] For instance, learning is discussed in Cyclops-2 [3].

The One-to-one matching scheme.

Under this schema, identification of a given object by means of a model is done only if all the features present in the model are also present in the object --in the scene--; that is, partially occulted bodies are not identified, unless the model in question specifically has as don't-care conditions the face or lines missing in the scene.

DT effectuates mainly this kind of matching or recognition. It has to care essentially for finding the right re gions, having the required neighbors, erasing the identified bodies, and repeating again.

TD is more sophisticated, being able to identify overlapping transparent objects. The proper vertices are searched for, and com plicated binding-restorations have to be made to account for failures and not-yet-defined properties. See chapter 5.

Polybrick does not effectuate a one-to-one matching.

Evans' identification program [8] makes first a one-to-one matching between two figures, using one of them as a model, but it has provisions to abandon this mode (he "weakens" the requirements) if neccessary. In a complicated sense it, too, has to 'account for failures' and it has a set of scoring systems to decide which of a number of matching attempts has the 'least amount of failure'.

Implementation.- In DT and TD, the model to be matched one-to-one to the scene is converted to a CONVERT pattern, then definitions are added to the dictionary, and the pattern is handed to the CONVERT processor, which executes

it. In this way, we avoid the double interpretation which would occur if we keep both model and scene in their original format and use a program to scan the model, chose a feature (a region, a line) and search the scene for it; then, scan the model more, select another feature and search the scene looking for a match for it; scan the model some more, etc.

LINEAR WEIGHTING

Weights.- Given a model, we attach coefficients to each of its parts, and also assign a number or threshold to the entire model, as it is indicated in figure 'WEIGHTS', where the coefficients or weights are assigned to lines.

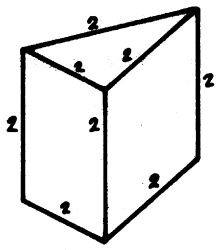


Fig. 'WEIGHTS'. Coefficients of 2 are assigned to each line of this model; the total sum is 16. If we set the threshold = 10, then we allow three lines to be missing.

The weight of a given feature represents the relative value of this feature; in fig. 'WEIGHTS' all lines have the same value, 2. Therefore, the total weight of this model is 16; the threshold value is set to a lower number, say 10. The recognizer is instructed to try to match each feature of the model with the corresponding feature of the scene, as in one-to-one matching, but in addition it accepts matches even if the features do not agree --that is, even in a normal failure-- . At the end, we have a match of value v, this number being the sum of the weights of the features which <u>did</u> agree.

We reject the match if v is smaller than the threshold of the given model.

Important features have big numbers; the nearer the threshold to the total value of the model, the more "strict" we are with our matches.

This scheme is a majority consensus; to the extent that it works its success is due to the fact that random lines will have a low probability of being aligned so as to match some model. That is, "if some figure looks enough like a cube, it has to be a cube."

Linear weighting is easy to implement, but it is weak in differentiating between two slightly different models.

Sub-weights.- When using the region-notation, an improvement can be made if we assign also weights to the segments that form the boundary of a region. We will have now two thresholds: one for accepting a face or a region when the lines found for it(*) are enough to overweight the threshold for the region; another for accepting a collection of regions to form an object.

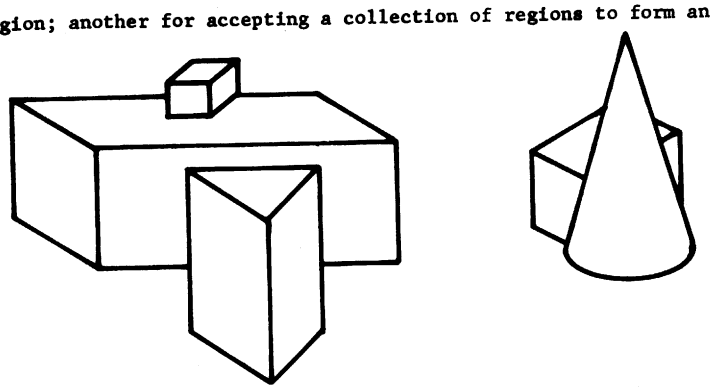


Fig. 'LINEAL'. A face is good if 3 out of 4 of its lines are seen; a body is good if 3 of 4 of its faces (regions) are seen. Under this 75 °/o criteria, only the cube behind the cone fails to be recognized.

(*) Bad matches are those when the complete line is missing, or it goes in the wrong direction. They contribute with a weight of 0 to the total value of the match.

Note that even disconnected bodies, such as the parallelepiped in figure 'COPY' can be identified -- at least one of its parts, and from this will not be hard to find the remaining, using a merger like 'FIT' (next section).

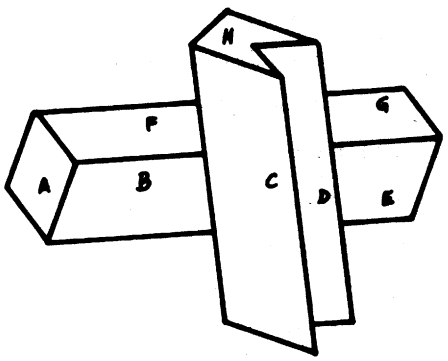


Fig. 'COPY'. Under the 75 $^{o}/_{o}$ criteria (see fig. 'LINEAL'), face A of the parallelepiped matches completely; faces F and B match with 3/4 of success, and are accepted. Therefore, parallelepiped A B F is found. Under 66 $^{o}/_{o}$ of success, we will still find another parallelepiped: G and E are faces of 3/4 of success, and parallelepiped G E is one with one face missing: 2/3.

But then things start to get complicated: unless our program be quite sophisticated, face H is (more or less) according to linear weight, a parallelogram, and so are faces C and D, so figure C D H will be taken as a parallelepiped also.

Conclusions.- We can go a considerable distance with simple methods as linear weighting; if we decide to use another method, this last one has to do better than linear weighting.

This observation has application not only here; often enough in the field of artificial intelligence, people is tempted to choose complicated but anthropomorphic programs; let them do so, provided that

their algorithms work better than simpler 'machine oriented' methods. Perhaps the best thing would be to use these simple schemes as some sort of heuristic that guides and complements the more powerful (but more expensive in time) tools.

Despite its simplicity, linear weighting[1] has the disadvantage of not being easily extendable. To be sure, we could develop more complex schemes that use linear weighting[2] as the main tool but, in the light of this added sophistication, the weights would probably become for us more a nuisance than a help. And the cause is clear: evidence in pro or in contra does not behave linearly; more interaction among the different facts is neccessary in order to arrive to a sound conclusion than the simple 'majority votation' implied by the weights and the threshold. When you have easy ways to get information, linear weight will find its way in. When we have to use more powerful tools and sophisticated methods to extract relevant facts, we usually need sharper "combining" tools for making partial conclusions, if for no other reason then because these better tools are more expensive (time-consuming), so have to be driven with care and with a more detailed knowledge of the prevailing situation.

Nevertheless, linear weighting has important uses[1].

Linear weight puts a lower limit of performance which more sophisticated programs have to excede if they want to be called "good".

---

[1] One of the most succesful users of weights is Samuel [31].

[2] A proposed refinement of [31] is done by Griffith [11] [A new machine-learning technique applied to the game of checkers. MAC-M-299 (AI memo 94). March 66].

THE GENERALIZED REGION (GREGION) APPROACH

Generalities.- This section refers to a more involved approach to the recognition problem; we suppose that several models are available, in region-notation, and that the regions of the scene to be analyzed have been found 'correctly', that is, the symbolic description is an accurate, exact specification of it. We want to find in that scene all instances of a given object.

The general procedure is as follows: each region in the scene is analyzed and some of the segments or sides of its boundary are marked (with dots); these new regions are then classified and merged, and then comparison (matching) begins with the model. The model guides the classification and merging, so that there is not a clear cut between the matching and the merging; during these processes, difficulties may suggest the inadequacy of the data, so that (1) a new redotting or (2) a new preprocessing(*) of the given region can be performed, this time with indication of what to look for and how.

Matching is done at a high level (I am using CONVERT) and, when some feature is not there, like a line, for instance, we call to FIT to extend the region, or we try to find a reason for this omission; or, as we said, perhaps FIT does not trust any more the data and decides to get new one, etc.

All this section is devoted to the explanation of what was just said.

The programs that mark (with dots) the regions and classify them are written but undebugged; it is my idea to finish this work and implement the approa ch described in this section, or an extension of it; originally, this thesis was going to be about analysis of scenes using the algorithm

(*) Which, in turn, may originate new merging, etc.

which I am about to describe but, since the programs are unfinished ...

Before talking about the gregion approach, we give some

Definitions.

BOUNDARY.- (or SHAPE) of a region. Counterclockwise ordered list of segment (lines) and vertices that separate the region from others.

VERTEX.- Informally, point where two or more segments meet. Formally, point where the slope of a line is discontinuous, multivaluee (?) or has a maximum or minumum --inflection point--.

Vertices are inflection points, points where two straight segments or one curve and one straight meet, or more than two segments encounter each other. They do not need to correspond to 3-dim vertices, although some of them do.

Y-JOINT.- Or simply 'Y'. Vertex with three rectilinear segments. (see chapter 4 on Polybrick for the use of Y's).

T-JOINT.- Y-joint with exactly two segments having the same slope. More generally, vertex with 3 segments, two of which are rectilinear and colineal.

SEGMENT.- The finite part of a line between two points in the line (usual definition).

SCENE.- Internal representation of visual or graphical data. The information that a scene contains is going to be secured by the process of recognition or identification of objects. Frequently, I mean by a SCENE a collection of data as above, but organized in a symbolic format.

MODEL.- A representation of an object or body, used mainly for recognition purposes, and generally in a non-numeric format (see in chapter 5 the notation FDL-1 for models; see chapter 7 and the region-notation for models).

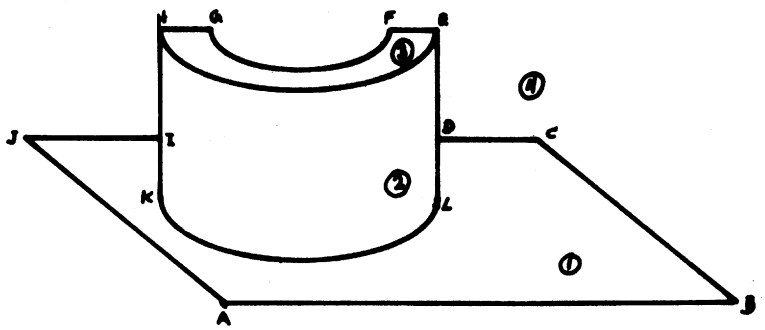


Fig. 'DEFINITIONS'. We identify the following entities:
SCENE: One, called 'definitions'.
REGION: four regions: 1, 2, 3, 4. Region 4 is the background.
BOUNDARY: each region has one.
Boundary of region 3: (point H, segm. HE, point E, segm EF, point F, segm FG, point G, segm GH, point H)
VERTEX: A, B, ..., K, L.
T-JOINT: I, D. (and, in a more general manner, H, E).
Y-JOINT: D, I. E is not because EH is curved.
SEGMENT: Each region has a set of them.
Segments of region 3: HE, EF, FG, GH.

REGION.- A simple closed curve of a scene or a model.

DOTTED REGION.- A region processed by 'DOTS' (a program); a region where some of its segments are 'false' and therefore, dotted.

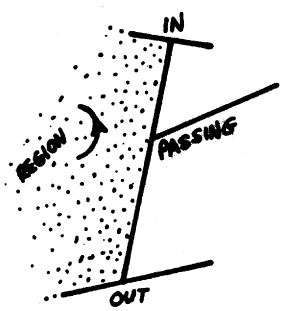
GREGION.- (generalyzed region). The merging of two or more dotted regions or gregions, which belong to the same face of a 3-dim (or 2-dim) body but that, due to occlusion, are disconnected in the scene, as done by'FIT" (a program).

OBJECT.- (or BODY). A mass of matter distinct from other masses (usual definition).

FACE.- Any of the surfaces that bound a geometric solid. (usual definition).

Marking the boundary of regions .- Each region of the scene in question receives the treatment specified in this paragraph.

Each one of the vertices of the boundary is analysed, looking for 'T' joints ; with respect to a region, T-joints may be of one of three classes : Out, In or Passing



Once identified, the program called DOTTS marks, erases, or as I prefer to say, puts dots to chains of segments between OUT-T's and IN-T's; for instance, fig 'PATCH' becomes 'PATCH ..." The doted segments are 'false' ones, in the sense that they do not belong to the region, but are occassioned by overlapping or occlusion. See also fig. 'WEDGES..." .(page 91)

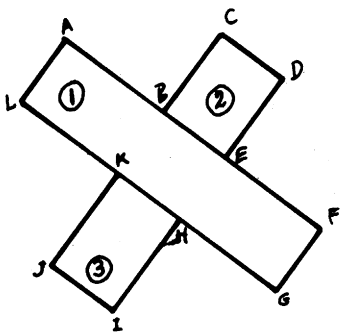


Fig. 'PATCH'. A scene composed of four regions

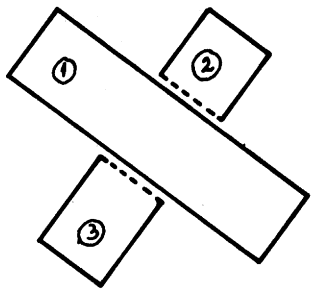


Fig. 'PATCH..." T- joints are found and dots are placed in some of the sides of regions 2 and 3

We erase the 'false segments with the purpose of facilitating the comparissons which have to be made.

Unreliability of DOTTS.- In some cases, the information obtained analyzing the T-joints is not enough to determine completely the 'false' segments of a given region ; that is, we can not completely put dots to all the false segments. In figure 'CARDS', two answers are acceptable, which in turn correspond to two possible identifications of the scene.

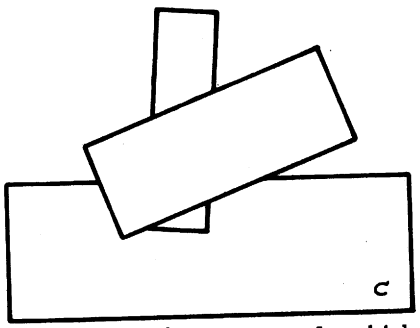
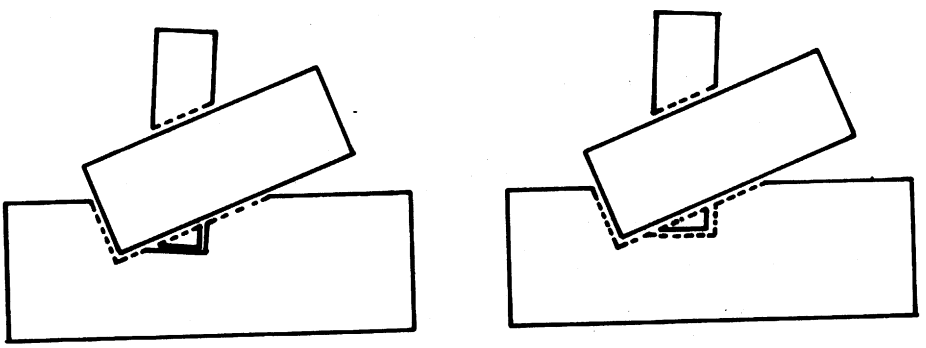
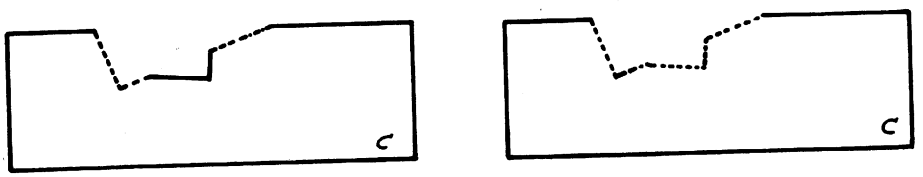


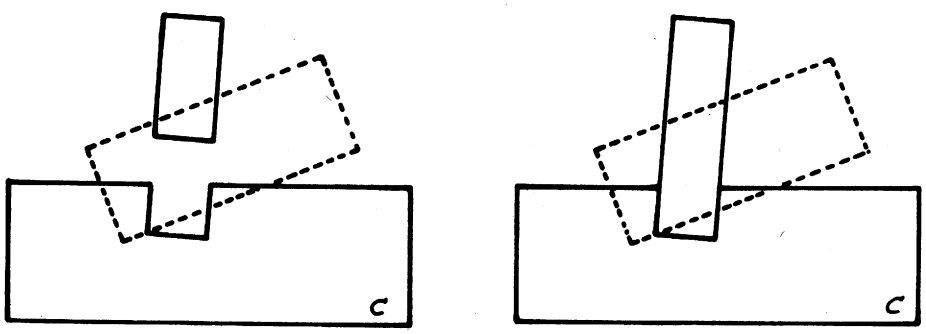
Fig. 'CARDS'. An scene for which the identification of 'false' boundaries for region C is not unique



Two possible identifications for Fig. 'CARDS'



Two different gregions for region C.



If we remote the uppermost card, the interpretation of the remaining two cards is different depending on the gregion for C.

Probably this problem will not be serious, since we will, in general, not be dealing with thin objects.

Classification of the regions.- So far, we have converted the regions to dotted-regions, using the program DOTS. Now, these dotted regions are classified in one of the two following forms, the decission depending if there are just a few models to identify (a), or if there is a fair amount (b).

(a) We examine the shapes of the regions of the models, and let us say we find m different shapes s_1, s_2, ..., s_m. Now, for each dotted-region of the scene, we compute a vector $[p_1\ p_2\ p_3 \ldots p_m]$, where p_i is the probability that the region in question has shape s_i. The p_i are not strictly probabilities, but have a small range, say, 0, 1, ..., 5. The idea is to make these computations fast, even if they are somewhat unreliable.

(b) If the number of models we try to find in the scene is not small, instead of (a), it will be better to work with predetermined shapes S_1, S_2, ..., S_k, and to compute for them the probability vector as before.

Shapes S_i should be "standard" ones, such as parallelogram, ellipsoi-

dal, long, large, small, rounded, fragmentary (when the region is really part of a bigger but disconnected region), mess, etc. Classes do not need to be disjoint.

These computations are done by examining the boundary of each dotted region, using fast 'rules of thumb', such as: how long is the longest segment compared to the average segment length; ratio of curved to non-curved sides; total number of sides; parallel sides, etc.

I may say that the purpose of this pass is to "become familiar" with the data (with the scene); that is, to have a fair idea of how things look, where are situated the large regions, etc.

Perhaps a better idea is to make this classification pass <u>after</u> the merging of regions into gregions; when the number of regions in a scene is small, use this pass <u>before</u> merging the regions, otherwise merge first and then classify.

Merging regions into gregions.- Two dotted regions having among them certain relationships --certain configuration of sides(*), as used in Polybrick when trying to find the "next" vertex-- are grouped or merged into a more general region, called <u>gregion</u>. Gregions are also joined under similar criteria into gregions. For instance, in fig. 'PATCH...", the regions 2 and 3 will be merged. This operation is done by the skeleton-program 'FIT', and is facilitated by the fact that the 'false' sides or segments have been erased; they do not influence this concatenation.

A question arises: Do we want to make the merging independent of the shapes of the regions of the model or not? If we choose independency, the algorithm will be simpler to implement, and faster. If merging is governed or influenced by the matching algorithm --that is, when we have

information about which region is being compared in this moment--, then we will have a more powerful merging, but it will be slower and, beyond a point, we will be exploring branches of the tree with very low probability of success.
Let me explain it: If two regions more or less fit (they are 'mergeable'), except in a portion, the comparator or matcher should be called to see if it finds a third region --the missing linkage--; in looking for this third region, the matcher will probably call again to the merger, because it has a candidate for the third region that "almost" matches, except that ... etc. My point is: if two gregions are not mergeable after a few attempts, they will never be mergeable. It is like having two somewhat distant pieces when we are working in a jigsaw puzzle, and try to find the link between them recursively. May be it will work, if the pieces are not far apart; otherwise, other ways to work the puzzle will generally be easier.
Conclusion: Interaction between the merger and the matcher will be carried up to a certain (shallow) depth.

Under good reasons, 'FIT' (the merger) may not give full credit to some segment, that is, it could question its autenticity or fidelity; for instance, see fig. 'CYLIN' in chapter 2. The surface function which obtained it will be seen, and perhaps the preprocessor will be given a new (hopefully improved) function to reanalyze the region; also, special feature-seekers, line followers, etc., could be used at this point.

Comparison: the job of the matcher.- As in DT and TD, the part of the program that effectuates the match has two inputs: a model and a scene. This time, the scene is composed by gregions, since it has been treated alredy by DOTS

and 'FIT'.

Matching is dona at high level (I am using CONVERT), comparing feature after feature --that is, segments, the right neighbors, etc.-- When some part of a region is missing --say a line--, we (1) call to FIT to extend the gregion, or (2) we try to justify the absence of the line; for instance, a dotted side means either (a) end of the gregion or (b) thisgregion is expandable or mergeable, since a dotted or false segment indicates that the gregion is partially occluded; or (c) indicates that data is not reliable and FIT will call the preprocessor; or (d) the comparator or matcher says "this object is not found in this scene."

Unlike TD or DT, matching will be done here interpreting the model and trying to find in the scene the required features. It looks to me that the process is complicated enough, and that the formation, as in TD or DT, of a CONVERT pattern from the model in question will be very complicated.

The comparisson program (and FIT also) will use the information contained in the probability vector, pretty much in a conventional way: when looking for a parallelogram, will analyze first the regions with big chances of being parallelogram. We must realize that the probability vectors may be wrong, since they contain information that was gathered in a quick manner. Other thing that may go wrong is that FIT merged two gregions that should not be merged; at some point the program has to realize this, and undo the consequences of its mistake.

Since this is expensive in time, it may pay to have a cautious merger.

As pointed out before, 'DOTS' may also (rarely) fail.

REFERENCES

1. Andrews, M. C., "Multifont Print Recognition", In (9).

2. Bledsoe, W. W. and Browning, I., "Pattern Recognition and Reading by Machine", 1959 Proceedings of the Eastern Joint Computer Conference, pp. 225-232. Also in (9).

3. Bloom, B. H. and Marill, T., Cyclops-2: A Computer System That Learns to See, Tech Report TR65-RD1-1, Computer Corporation of America, Cambridge, Mass. (July 1965).

4. Canady, R. H., The Description of Overlapping Figures, M.S. Thesis, Electrical Engineering Dept., M.I.T., 1962.

5. Chatten, J. B. and Teacher, C. F., "Character Recognition Techniques for Address Reading", In (36).

6. Conrad, D., LISP Functions for Generating and Plotting Figures, Program Note No. 2, Centro Nacional de Calculo, IPN. México, 1964.

7. Conrad, D., "CONVERT Functions for Generating Figures", Ciencias de la Informacion y Computacion 1, 1(june 1966), pp. 63-83, Universidad Nacional Autonoma de México.

8. Evans, T. G., A Program for the Solution of a Class of Geometric-Analogy Intelligence-Test Questions, Air Force Report AFCRL-64-88, November 1964, Cambridge, Mass. (Also available as a Ph.D. Thesis, M.I.T. Electrical Engineering Dept.)

9. Fischer, G. L., Optical Character Recognition, Spartan Books, 1962.

10. Greenblatt, R. and Holloway, J., Sides 21, Project MAC Memorandum MAC-M-320 (AI Memo 104-2), M.I.T., August 1966.

11. Griffith, A. K., A New Machine-Learning Technique Applied to the Game of Checkers, Project MAC Memorandum MAC-M-299 (AI Memo 94), M.I.T., March 1966.

12. Guzmàn, A. and Sussman, G., A Quick Look to Some of Our Programs, AI Internal Memo V-104-1, Project MAC, M.I.T., July 1966.

13. Guzmàn, A., Polybrick: Adventures in the Domain of Parallelepipeds, Project MAC Memorandum MAC-M-308 (AI Memo 96), M.I.T. May 1966.

14. Guzmàn, A. and McIntosh, H. V., "CONVERT", Communications of the ACM 9, 8 (August 1966),pp. 604-615. (Also available as Project MAC Memorandum MAC-M-316 (AI Memo 99), June 1966.

15. Guzmán, A., Scene Analysis Using the Concept of Model, TR-12, Computer Corporation of America, Cambridge, Mass. (in preparation).

16. Guzman, A., A Primitive Recognizer of Figures in a Scene, (in preparation).

17. Hodes, L., Machine Processing of Line Drawings, M.I.T. Lincoln Laboratory Report 54G-0028, March 1961.

18. Ivie, E. L., Search Procedures based on Measures of Relatedness Between Documents, Project MAC Technical Report MAC-TR-29 (M.I.T. Ph.D. Thesis), June 1966.

19. Johnson, T., Sketchpad III: 3-D Graphical Communication with a Digital Computer, S.M. Thesis, Mechanical Engineering Dept., M.I.T., 1963.

20. Kirsch, R. A., et al, "Experiments in Processing Pictorial Information with a Digital Computer", Proceedings of the Eastern Joint Computer Conference, Vol. 12, 1957, pp. 221-229.

21. Krakauer, L., A Description of the CNTOUR Program, Project MAC Memorandum MAC-M-335 (AI Memo 104-7), M.I.T., November 1966.

22. Lamport, L., Summer Vision Programs, Project MAC Memorandum MAC-M-332 (AI Memo 104-6), M.I.T., October 1966.

23. Mann, W. F., Finding the Edges of a Polygon in the Field of View, Tech Report No. 10, Computer Corporation of America, Cambridge, Mass., March 1966.

24. Marill, T., et al, "CYCLOPS-1: A Second-Generation Recognition System", Proceedings of the Fall Joint Computer Conference, Vol. 24, 1963, pp. 27-33,

25. Martin, W., A Step-by-step Computer Solution of Three Problems in Non-numerical Analysis, Project MAC Memorandum MAC-M-323 (AI Memo 101), M.I.T., July 1966.

26. Nilsson, N. J., Learning Machines, McGraw Hill, 1965.

27. Papert, S., The Summer Vision Project, AI Memo 104-1, Project MAC, M.I.T., July 1966.

28. PDP-6 LISP, Project MAC Memorandum MAC-M-313 (AI Memo 98), M.I.T., June 1966.

29. Roberts, L. G., "Machine Perception of Three-Dimensional Solids", Optical and Electro-optical Information Processing, M.I.T. Press.

30. Samson, P., EYE, AI Programs Memo 103-4, Project MAC, M.I.T., December 1966.

31. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers", Computers and Thought (Feigenbaum and Feldman, editors), McGraw Hill.

32. South, R., CORNS, AI Memo 103-3, Project MAC, M.I.T., October 1965.

33. Sussman, G., Untitled report in preparation.

34. Sutherland, I., Sketchpad: A Man-Machine Graphical Communication System, Tech Report 296, M.I.T. Lincoln Lab, 1963.

35. Tobey, R. G., "Experience with FORMAC Algorithm Design", Communications of the ACM 9, 8(August 1966), p. 590.

36. Uhr, L.(ed), Pattern Recognition, John Wiley and Sons, 1966.

37. White, J., Additions to Vision Library, AI Internal Memo V-104-2, Project MAC, M.I.T., August 1966.

38. White, J., Figure Boundary Description Routines for the PDP-6 Vision Project, Project MAC Memorandum MAC-M-328 (AI Memo 104-5), September 1966.

CS-TR Scanning Project
Document Control Form

Date : 12 / 11 / 95

Report # LCS-TR-37

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 134 (134-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☒ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-04) UNNUMBERED TITLE PAGE, ii-vi, 1-118	
(119-134) SCANCONTROL COVER, FUNDING AGENT, DOD TRGT'S (3)	

Scanning Agent Signoff:
Date Received: 12/11/95 Date Scanned: 12/28/95 Date Returned: 12/28/95

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE

Some Aspects of Pattern Recognition by Computer

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

M.S. Thesis, Electrical Engineering, February 1967

5. AUTHOR(S) *(Last name, first name, initial)*

Guzmán-Arenas, Adolfo

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February 1967	124	38

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Office of Naval Research, Nonr-4102(01) b. PROJECT NO. Nr -048-189 c. RR 003-09-01 d.	MAC-TR-37 (THESIS) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D. C. 20301

13. ABSTRACT A computer may gather a lot of information from its environment in an optical or graphical manner, if a TV picture of a scene is transformed into a symbolic description of points and lines, or surfaces. This thesis describes several programs, written in the language CONVERT, for analyzing such descriptions in order to recognize, differentiate, and identify desired objects or classes of objects in a scene. Examples are given in each case.

Important restrictions and suppositions are: a) input is assumed perfect (noiseless) and in a symbolic format; b) no perspective deformation is considered. A portion of this thesis is devoted to the study of models (symbolic representations) of the objects we want to identify, and different schemes, some of them already in use, are discussed.

Focusing attention on the more general problem of identifying general objects when they substantially overlap, some schemes are proposed for such recognition, and some concurrent problems are analyzed.

14. KEY WORDS

Machine-aided cognition	On-line computer systems	Symbolic manipulation
Multiple-access computers	Pattern recognition	Time-sharing
Object identification	Real-time computer systems	Time-shared computer systems

DD FORM 1 JAN 64 1473 (M.I.T.)

UNCLASSIFIED

Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 12/28/1995

M.I.T. Libraries
Document Services