20

Analysis of Curved Line Drawings Using Context and Global Information

A. Guzman

Department of Machine Intelligence and Perception
University of Edinburgh

Abstract

We describe the analysis of visual scenes consisting of black on white drawings formed with curved lines, depicting familiar objects and forms: houses, trees, persons, and so on; for instance, drawings found in coloring books.

The goal of such analysis is to recognize (by computer) such forms and shapes when present in the input scene; that is, to name (correctly) as many parts of the scene as possible: finger, hand, girl, dance, and so on. Complications occur because each input scene contains several such objects, partially occluding each other and in varying degrees of orientation, size, and so on.

The analysis of these line drawings is an instance of 'the context problem', which can be stated as 'given that a set (a scene) is formed by components that locally (by their shape) are ambiguous, because each shape allows a component to have one of several possible values (a circle can be sun, ball, eye, hole) or meanings, can we make use of *context information* stated in the form of *models*, in order to single out for each component a value in such manner that the whole set (scene) is consistent or makes global sense?'

Thus, shape drastically limits the values that a component could have, and further disambiguation is possible only by using global information (derived from several components and their inter-relations or inter-connections) under the assumption that the scene as a whole is meaningful.

This paper proposes a way to solve 'the context problem' in the paradigm of coloring book drawings.

We have not implemented this approach; indeed, a purpose of this paper is to collect criticisms and suggestions.

INTRODUCTION

1.1 Statement of the problem

An input picture is read into a computer. We would like to analyze it.

1.1.1 *The input picture*

The input picture consists of a line drawing (black curved thin lines on white paper) containing familiar objects [figure 1(a)]; one could think of drawings in coloring books for children. The objects forming the picture should be drawn correctly and accurately: no intentional distortions, caricatures, or humanizations of animals (figure 2) will be allowed.

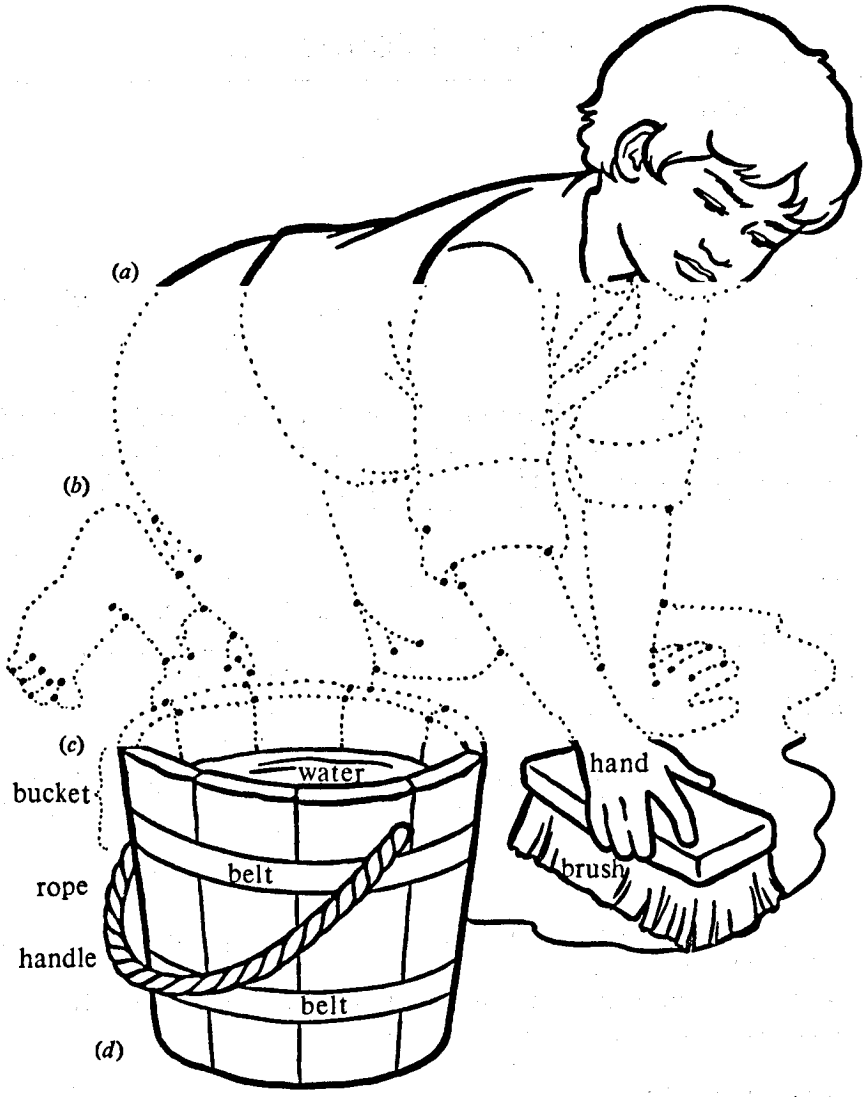


Figure 1. (a) typical input scene; (b) initial representation of scene in the memory; (c) showing singular points heavily marked; (d) description of objects present in the scene.

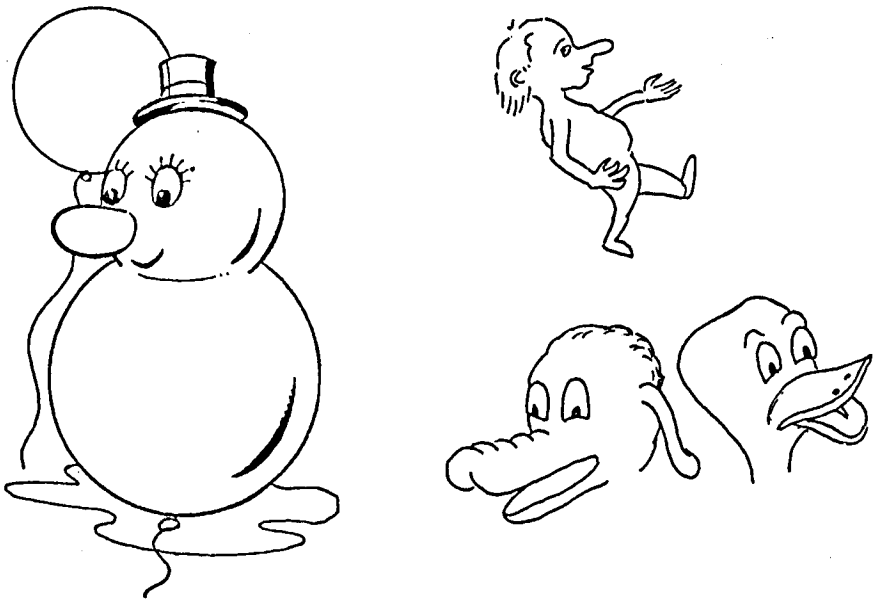


Figure 2. Line drawings containing distortions, caricatures, comic strips, humanized animals, and so on, will not be accepted.

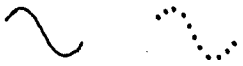
Thus, it can be said that the class of input pictures we want to analyze is that found in coloring books, except distortions.

We could think of a person looking at the input data [figures 1(a) or 1(b)] and saying: there is a straight line from point (30, 40) to point (67, −18.5), two curved lines above that big zig-zag line, which runs from point (30, 15) to . . . .

Representation in the machine. The input picture [figure 1(a)] is stored initially in the memory of the computer as a collection of black points (specified by their two-dimensional coordinates) closely spaced along each black line [figure 1(b)].

We will assume that:

(a) The points are uniformly spaced along the lines of the drawing. Thus,



(b) Spacing between points is close enough to keep necessary structure and form details. That is, we will not worry about resolution, quantization, and so on. Alternative assumptions could be employed (but they are not).

1.1.2 *The result or output*

The desired output is a description of the object constituents of the scene. That is, names are found for each region or part of the scene. We could think

of a person looking at the drawing and saying: there is a tree there, a big rock, and a boy, an armadillo behind that coconut tree, with two mimsy borogoves. . . .

For instance, for the scene as depicted in figure 1(b) the result is as depicted in figure 1(d).

1.1.3 *The context problem*

The problem we are trying to solve is the Context Problem, which can be stated in general words as

> 'given that a set (a scene) is formed by components that locally (by their shape) are ambiguous, because they can have one of several possible values (a circle = sun, ball, eye, hole) or meanings, can we make use of *context information* (nose, eyes, mouth, occurs often) stated in the form of models* in order to assign to each component a value such that the whole set (scene) is consistent or makes global sense?'

By analyzing each component, we come to several possible interpretations of such component, and further disambiguation is possible only by using global information (information derived from several components, or by the inter-connection or inter-relation between two or more components), under the assumption that the scene as a whole 'makes global sense' or is 'consistent'.

1.1.4 *Example*

By local analysis (by observing the shape of each of the following regions) we conclude that certain shapes can be interpreted only as certain objects (*see* figure 3). (For instance, the shape of an oblong suggests as reasonable for it the names, stick, box, arm, leg, handle, but suggests as unreasonable the names, sun, mouth, horses.) The shape of an object drastically limits the things that it could be.

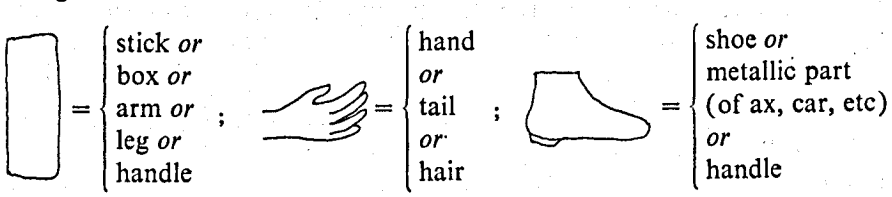


Figure 3

Also, we have information that the combinations shown in figure 4 (called *models*) are usual, that they occur with frequency.

If we use the above knowledge or information to analyze the scene, as shown in figure 5, the local shape of each region allows the interpretation in roman lettering. Thus, we have several reasonable values for each part of our scene.

* By model we mean a generalized description of an object or a class of objects, with certain parameters left unspecified.

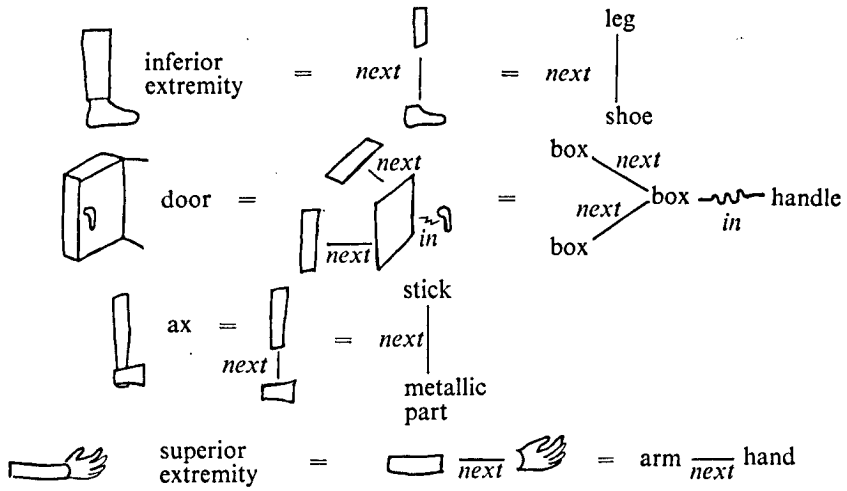


Figure 4

Now, by taking into account the models door, ax, and so on, we further find the interpretations in italic lettering in figure 5. That is to say, the models have suggested that

box *or* stick *or* arm *next* hand *or* tail *or* hair *or* leg *or* handle

should be

arm *plus* hand

which is a superior extremity (an instance of something commonly found in our scenes), while discarding interpretations such as

box *plus* tail

which, although consistent with the individual shapes of its regions, are not suggested or are not in agreement with the models used.

Also, in

stick		metallicpart		ax
or	*plus*	*or*	*equals*	*or*
leg		shoe		inferior extremity

we have still not decided between ax and inferior extremity; further disambiguation could be done by

(1) introducing a *shape* difference between shoe and metallic part (that is, drawing them more accurately so that their shapes will be different), or between stick and leg; or by

(2) introducing additional context information by introducing the additional models shown in figure 6.

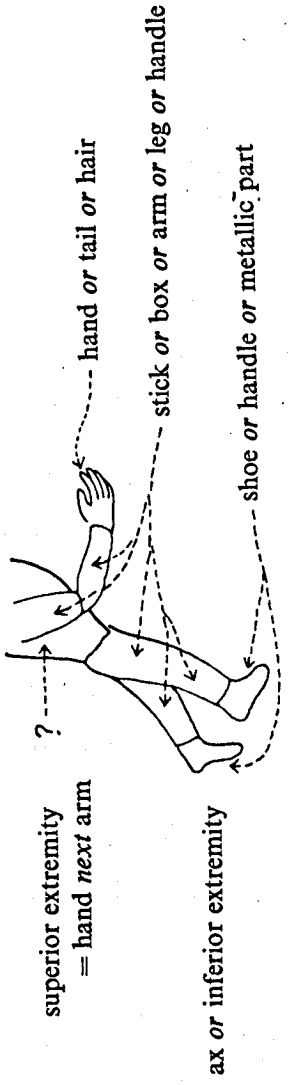


Figure 5

1.2 An example from natural language

To translate a sentence to a different language, it is necessary to disambiguate it by finding the precise meaning of each word.

Almost every word has more than one meaning:

KICK { KICK1 – to hit with feet, . . .
KICK2 – emotion obtained from drugs, . . .

BALL { BALL1 – formal dance
BALL2 – sphere
BALL3 – . . .

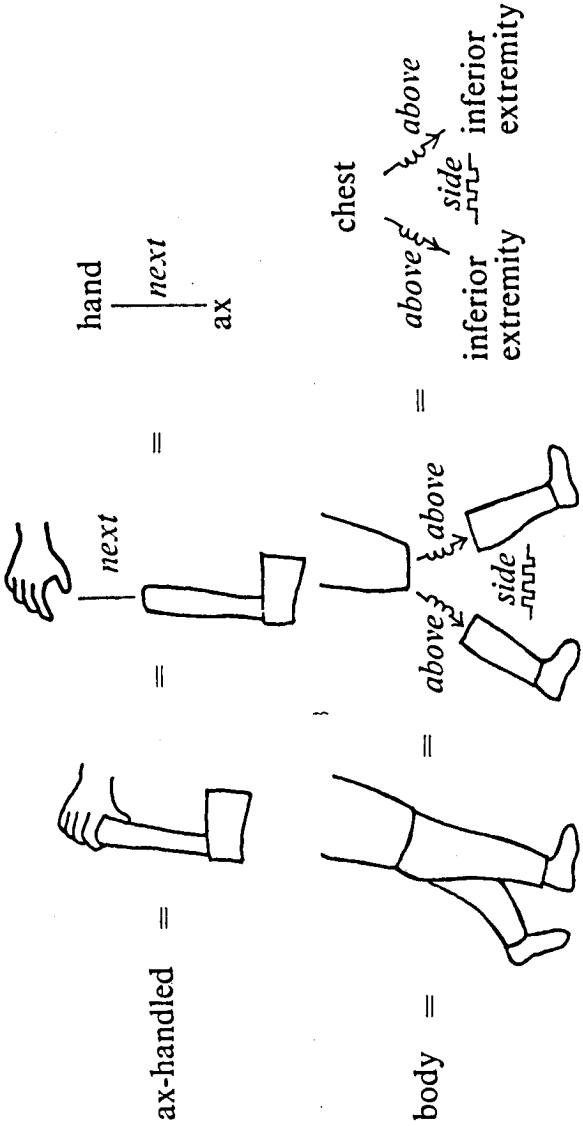


Figure 6

That is, the 'shape' of the word (KICK begins with K, followed by I, . . .) restricts its meaning to KICK1 or KICK2.

With the help of the shape, the sentence (or scene)

THE BOYS KICK THE BALL

becomes

THE BOYS KICK1 THE BALL1
or KICK2 or BALL2
or BALL3

We also have the models

'SOCCER-ACTIVITY'=KICK1 acts upon BALL2;
'HOW-TO-KICK'=KICK1 uses SHOES3
'DANCING'=SHOES3 necessary for BALL1
MUSIC4 necessary for BALL1
'DRUGGING'=KICK2 uses DRUG2

Thus these models restrict our choices in the sentence to the consistent assignment

THE BOYS KICK THE BALL
THE BOYS KICK1 THE BALL2

That is, the models have rejected as inconsistent (or not suggested by the models) the interpretation or assignment

THE BOYS KICK2 THE BALL2

although the meaning of the words according to their shape is satisfied.
Note. The models are specified in terms of relations* among the parts; these will also change from one context problem to the next (figure 7).

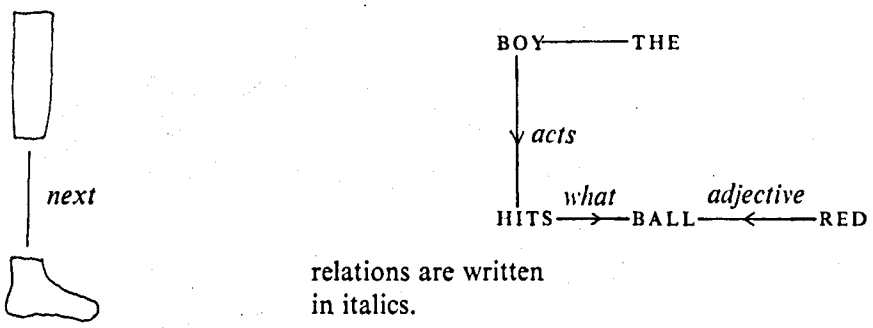


Figure 7. Models and relations are different for different paradigms of Context Problems.

1.3 General description of the method

Given a scene of which we wish to make an interpretation consistent with a given set of models, we need:
(1) To encode the scene in a form more suitable for shape comparison and extraction or relations to be used for context analysis.
(2) To describe the models necessary for the same purpose.
(3) To analyze the encoded scene against the descriptions of the models, in order to find the assignment of values to parts of the scene which best agree with the models and the relations associated to them.

We proceed to give here a brief explanation of each part, while the rest of this paper will give the pertinent details.

* We will have both relations such as (*next* leg, foot)=True, and quasi-relations such as (*thin* rectangle)=0·7. It is also a good idea to have degrees of nearness, of containment, and so on, as used by Barrow and Popplestone (1971).

1.3.1 *Encoding procedure*

(1) Find singular points
(2) Give to each line a name with a direction
(3) Encode each directed line
(4) Encode each region as a set of lines (take care of regions with trees inside).
(5) To save code, encode special regions and lines: wiggly lines, sticks, and so on.

1.3.2 *Model description*

(1) Describe each model in terms of *metaregions* (model regions) or lines (*metalines*) and relations among them, as shown in figure 8 (take care of size and relative orientation among parts of the model.) A metaregion could have one of several values from a small set, for instance, metaregion $R_1 =$ ▭ could be box, arm, handle, stick, leg; this set is known for each metaregion, but the shape of the metaregion is unable to distinguish the exact value of that metaregion, context being necessary.

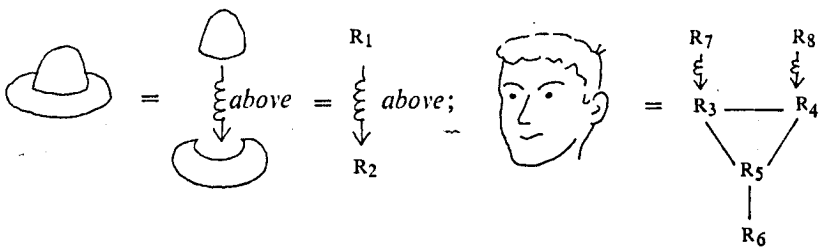


Figure 8

(2) Encode each metaregion and each metaline making use of the encoding procedure of section 1.3.1.

1.3.3 *Analysis*

To analyze a scene, proceed as follows (execute the following algorithms or programs):

(1) *Encode scene.* When this is finished, we have each region of the scene encoded. Inter-relations between them can be derived, but are not derived yet. Note that we could derive now all relations, like Evans (1970) does, but we choose not to do it because there are too many.

(2) *Describe models.* When this is finished, we have each metaregion and metaline of the models properly described (encoded). The problem now is how to determine what regions of the scene match with what metaregions (and metalines) according to shape. In principle, we could test all pairs (metaregion, region) to see if they match. Too inefficient. Heuristics needed (or a better way) described below.

Take care of regions that are separated:
We do this by describing them counterclockwise, just as in Guzman (1971) the occluded case for models matching with polyhedra.

(3) *Match regions to metaregions.* Find for each region of the scene the metaregion of closest shape. In this way, we find for each region of the scene the small set of values allowed by its shape.

How do we implement this? (a) For each region and each metaregion, find a vector of 'cheap' features or properties, and use this vector in a 'plausible move generator' which suggests pairs (metaregion, region) with a good chance of matching. (b) Final comparison is done using the encoded descriptions of the lines of the boundary of the region against the encoded descriptions of the metalines of the boundary of the metaregion; that is, they match if congruent.

(4) *Use models to find a unique assignment of values to regions.* Given a metaregion, find which models possess it; pick one and see the requirements that the context imposes in neighboring and related regions; see if these are satisfied in scene. Choose a different model if not satisfied.

2. ENCODING PROCEDURE

We have exposed in section 1 the general method to use for analysis of a scene; it begins by encoding the scene, which at this moment is a collection of closely-spaced points along lines [figure 1(b)], in a form suitable for further processing.

The encoding procedure consists of the following steps:

(1) find singular points;
(2) give to each line a name with a direction;
(3) encode each directed line;
(4) encode each region as a set of lines;
(5) to save code, encode special regions and lines.

The goal is to convert the scene to some string of characters (its description); models are also capable of being described in this way (and we will do so later).

We consider a scene as formed by several regions (although lines 'standing alone' are allowed in the description when they cannot be treated as part of a region. Details are given in section 2.4) which in turn are formed by curved lines, which are described in a manner independent of their size, orientation, and position, that is, pertaining only to their *shape*, plus additional information about the size, orientation, and position of each line in the scene.

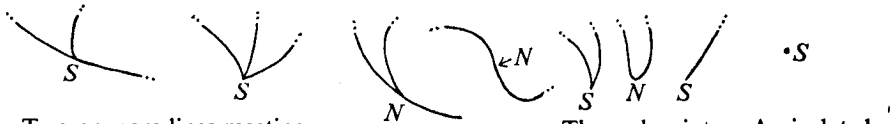
The main part (3) of the encoding procedure thus deals with the encoding of lines. But we have to find these lines first [parts (1) and (2)].

Shape of a curve. Two curves have the same shape if they can be made congruent by (a) a translation of the origin; (b) a rotation of the axes; and (c) a dilation of the axes, where both axes get expanded the same amount.

Two curves have the same shape (alternative definition) if the function that gives the radius of curvature as a function of s', the normalized distance along the curve ($s'=[s$, the true distance along the curve$]/[\max s]$), is the same for both.

2.1 Find singular points

A singular point possesses several values for the slopes of the line(s) impinging in it; it corresponds roughly to the notion of vertex. We also consider the end points of single lines as singular points as shown in figure 9. For instance, figure 1(c) shows the singular points of the diagram.



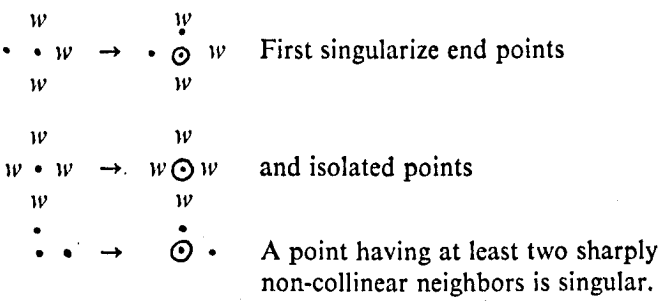
Two or more lines meeting at one point, with different slopes

The end point of a line

An isolated point

Figure 9. Singular points (S). Points marked with N are *not* singular points, but normal points.

Since our scene is composed of points along lines [figure 1(b)], the productions shown in figure 10 will find all singular points. Note that inflexion points are generally not singular points.



• is a black point, w a white (blank) point, ⊙ is singular (and black).

These are isotropic productions, • • w means  and and etc.

Figure 10

2.2 Give to each line a name with a direction

The singular points are extreme points of lines; give to each line a different name (identifier, atom, variable name) as follows:

(a) If the two extremes of the line are not the same singular point, arbitrarily consider one extreme as its beginning, the other as its end, and give a

name (say a) to the directed line, beginning→end; the line going in reverse sense is $\bar{a}$.

(b) If the two extremes touch, assign a counterclockwise sense to the line. $\bar{b}$ is thus b reversed.

(c) Tangency cases are considered as shown in figure 11.

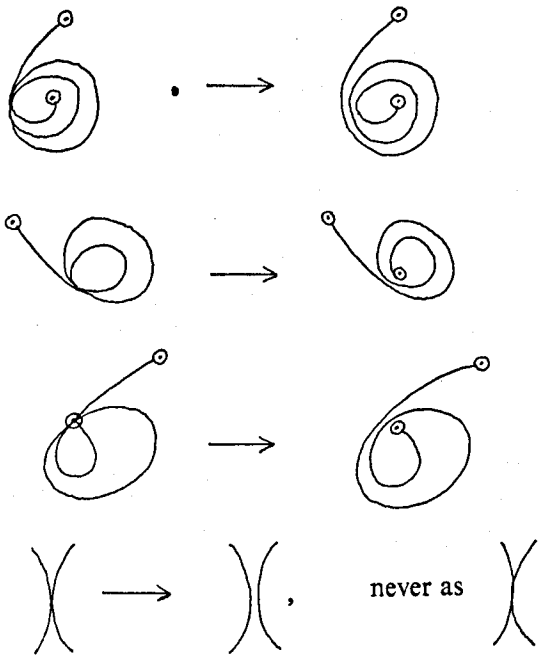


Figure 11

2.3 Encode each directed line

In figure 1(b), lines are specified by many points (each point known by its two-dimensional coordinates) and it is desired to have a more economical description of lines.

This description should specify as clearly as possible the shape of the line. Ideally, this description is not more than $f(s')$, the function that gives the curvature as a function of arc length (cf. the 'shape of a curve' above). We avoid $f(s')$ because it is difficult to obtain some spatial feeling for the curve described by a given $f(s')$. Nevertheless, we note that $f(s')$ is independent of size, position, and orientation of the curve. (*See* figure 12.)

Arriving at the encoding we use. Basically, we want to observe the slope of the directed line – as shown in figure 13. We also want to record the points where such slopes are obtained:

former figure	0	3	0	3	2	1	0	0	0
	(3, 5)	(5, 4)	(8, 2)	(9, 2)	(10, 1)	(9, 0)	(7, 0)	(4, 1)	(3, 0)

This is ambiguous for the figure with an infinite number of points with slope 0; which will we record?

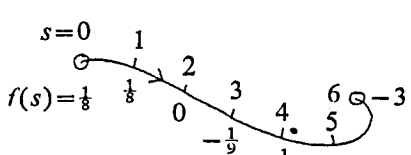


Figure 12

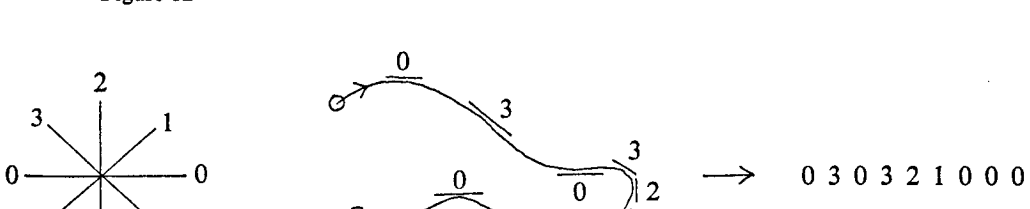


Figure 13. Note that slopes are unoriented.

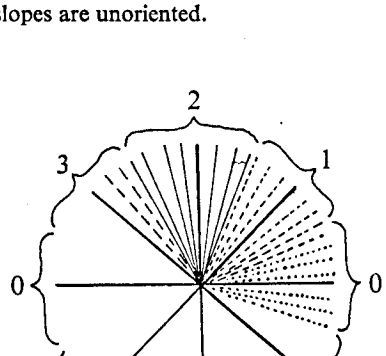


Figure 14

To solve this, we
(a) extend the definition of slope 0 – to and similarly for the others, as shown in figure 14.
(b) Select as the point to record together with slope *i* the midpoint of all those points having such slope *i*. (*See* figure 15.)

Note that the slope of point (60, 50) falls within the range of slope 0, that is, between +tan 22·5° and −tan 22·5°; thus it may very well not be exactly 0.

Then, to make the coding insensible to rotation, we select the most common slope and rename it 0, modifying correspondingly the names of the other slopes. (We are choosing as the horizontal or 0 slope the axis of the curve.)

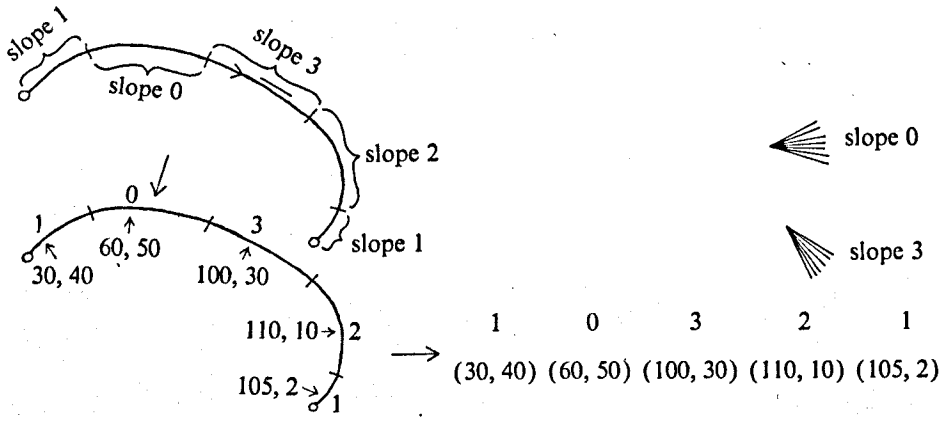


Figure 15

Unfortunately, the scheme is not really insensible to orientation; and for the same line we have a one-slope code in one case, and a two-slope code when we rotate it slightly, as shown in figure 16. To avoid this, we decide to find the axis (the most common slope) first and then encode.

With this experience, we now proceed to describe the procedure we choose. [Other ways to encode a line or a scene are given in Freeman (1961), Narasimhan (1969), and Montanari (1968).]

The *procedure* to encode each directed line is as follows:

(1) *Select the main axis of the curve.* Select an axis, to be considered the horizontal axis (slope 0) of the curve, that with the most common slope.

(2) *Code slopes and midpoints.* Walking along the directed line, take note of slopes and midpoints where they occur [as in (b) above].

(3) *Normalize the curve.* Choose an origin and normalize coordinates of points so as to have maximum coordinates (1, 1).

2.3.1 *Select the main axis of the curve*

If each curve carries its own 'main' axis with it, then we could code the curve (with respect to that axis) independently of rotation in two-dimensional space. This axis must be such that small modifications to the curve be unlikely to shift the axis drastically.

Each axis can have one of 64 possible directions, within a 180° span. To find out the likelihood of axis i to be the main axis of curve C, we add up the individual contributions of segments of C to the axis. Then we select as main axis that with largest likelihood (largest total contribution).

A *contribution* of a segment is the length of its (unsigned) projection over the axis i; in this way a segment that is parallel to axis i votes 100 per cent (of its own length) in favor of i [contributes 100 per cent of its length to i], and a perpendicular segment votes 0. (*See* figure 17.)

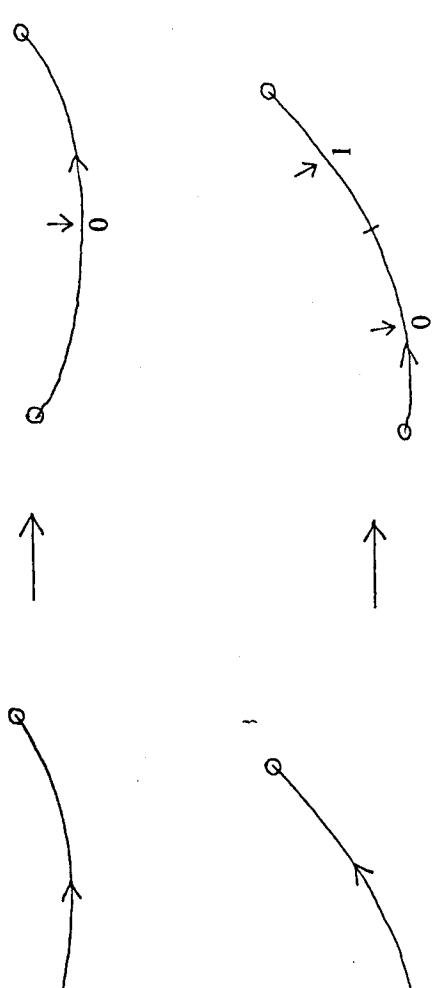


Figure 16

For a segment with slope 0, its contribution (as a percentage of its length) to the different axes j is shown in figure 18(a). This curve is nothing else but $|\cos\theta|$.

Other curves could be used; $a=\cos^4\theta$ diminishes more sharply the contribution to axes which are not appreciably parallel to the segment, while $c=\sqrt[4]{(|\cos\theta|)}$ gives small contribution only to axes almost orthogonal to the segment [*see* figure 18(b)]. We could also use an 'empirical' curve such as d, in figure 18(c).

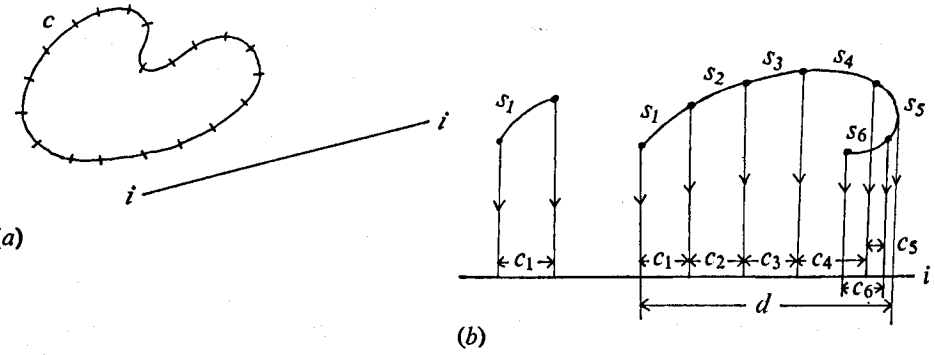


Figure 17

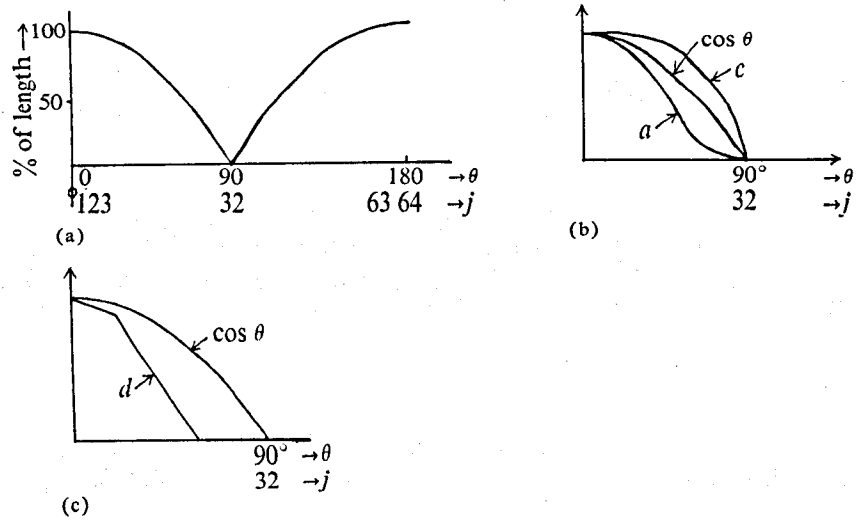


Figure 18

To find the main axis of a curve, find the total contribution of this curve to each of the 64 axes, and select the one with largest total contribution.

We give an example with only 4 axes and the $|\cos\theta|$ law, as shown in figure 19. The largest total contribution (12.1) corresponds to axis 1, which is chosen as the main axis of the curve.

In general the main axis found in this way will not coincide with the axis produced by the two points furthest apart (a, b), or with the longest side of the rectangle with smallest area that totally contains the curve, as shown in figure 19(b).

A practical way to select the main axis is with the help of table 1. To do this, we travel counterclockwise around the curve, and if a segment has slope number i, we add the ith row of the table to the row of buckets (initially empty); for instance, if a given segment has slope number 1, we go to row 1

and add 0·98 to bucket 0, add 1·00 to bucket 1, ..., add 0·01 to bucket 32, ..., add 0·96 to bucket 63 (if the segment has length $\neq 1$, multiply these coefficients by the length, before adding). Thus, we make a vector addition for each segment; when we finish with all segments of the curve we simply select the most filled bucket (largest value) as the main axis.

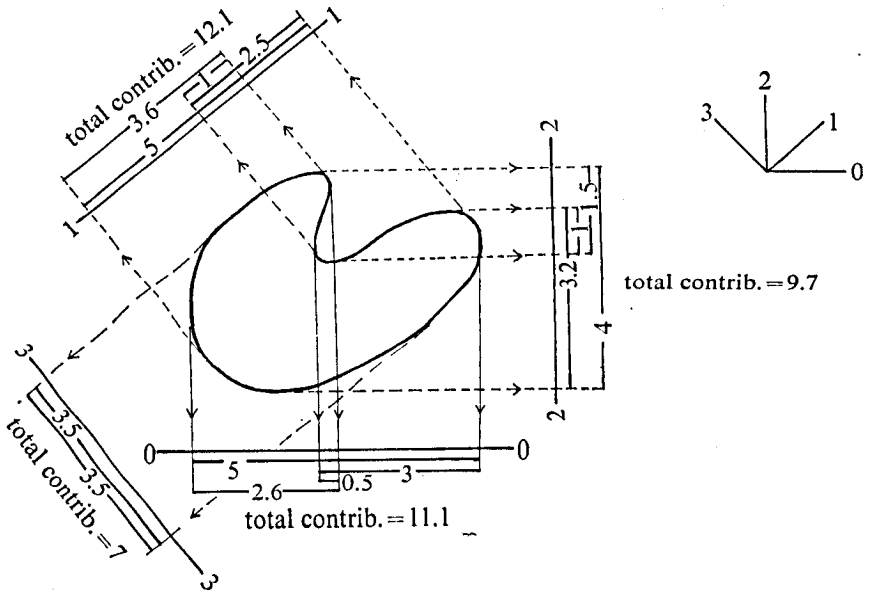


Figure 19

Table 1

	$\theta =$	0°			90°	
	$j =$	0	1	2	32	63
0	—	1.00	0.98	0.96	0.00	0.98
1	—	0.98	1.00	0.98	0.01	0.96
2	—	0.96	0.98	1.00		
		.	.	.	.	.
		.	.	.	.	.
		.	.	.	.	.
32	|	0.00	0.01		1.00	0.01
		.	.	.	.	.
		.	.	.	.	.
		.	.	.	.	.
63	—					1.00

Since the contribution curve is symmetric about 90° (cf. section 2.3), we can reduce the table to a size 32×32. [Size of table for selection of main axis: If we consider the table as a function $f(i, j)$ where i is the slope number of the segment and j that of the axis, define then $g(|i-j|) \triangleq f(i, j)$ $i, j \in [0, 63\}$ and it is only necessary to store the 64 values $g(0)=1{\cdot}00$, $g(1)=0{\cdot}98$, ..., $g(63)=0{\cdot}98$. Moreover, the function $h(|i-j|-321) \triangleq f(i, j)$ requires only to store the 33 values $h(0)=0$, $h(1)=0{\cdot}01$, ..., $h(30)=0{\cdot}96$, $h(31)=0{\cdot}98$, $h(32)=1{\cdot}0$.]

The graph of total contribution *v.* axis number (figure 20) is a feature of the curve in question; we could save the 64 numbers to use when comparing with other curves.

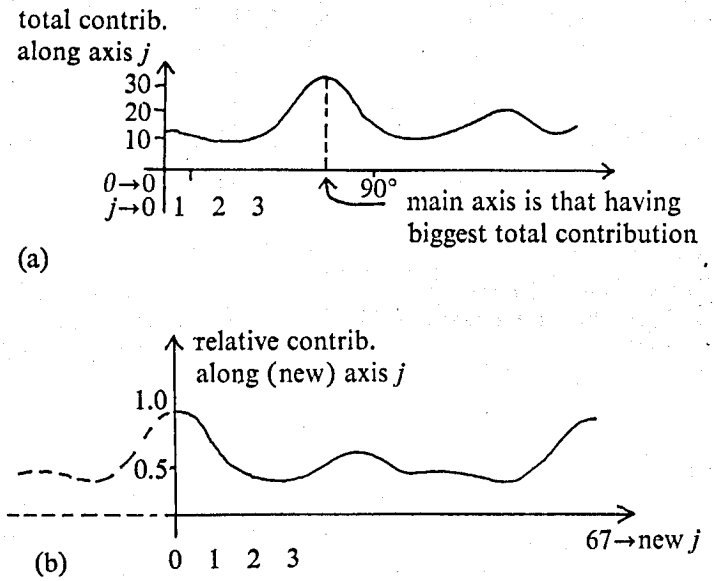


Figure 20

For two curves to have the same shape it is necessary that they have the same total contributions at the same θs (or js); thus we store the values collected in the buckets:

(1) normalized first by dividing by the largest value, so that max (total contribution normalized)=1; and

(2) normalized by renaming the 64 axes giving new name 0 to the main axis.

The graph in figure 20(a) is normalized as shown in figure 20(b) and these 64 values are stored.

If this graphic possesses several maxima a, b of nearly equal value, we could expect small deformations to the original curve to shift the main axis from a to b. It is therefore important to save this 64-vector to have an idea about which curves are prone to this shift. Thus, the main axis of the curve has been selected, in a way independent of the orientation of the curve.

2.3.2 *Code slopes and midpoints*

To do this

(1) Assign to the main axis the slope number 0; this determines the ranges for slopes 1, 2, and 3.

To any segment a slope is thus assigned; as shown in figure 21.

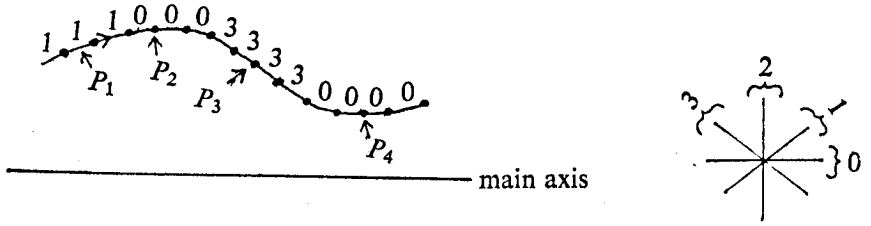


Figure 21

(2) Find the midpoints of strings of segments with the same slope number. The length of a string of segments equals sum of individual length of segments; the midpoint is that point on the string equidistant to its ends.

Example. P_i in figure 21 are the midpoints of the string of segments.

(3) Make an alternating list of midpoints (its two-dimensional coordinates: these coordinates are with respect to the original coordinates system of the scene, and are in no way affected by the choice of main axis for the curve) and slopes attached to them.

$$\underbrace{(30, 40)}_{P_1}\ 1\ \underbrace{(34{\cdot}5, 42)}_{P_2}\ 0\ \underbrace{(38, 38)}_{P_3}\ 3\ \underbrace{(41{\cdot}7, 37)}_{P_4}\ 0$$

where *1* is the slope at P_1; *0* the slope at P_2; *3* the slope at P_3; and *0* the slope at P_4. This list begins with a point, and goes in the direction of the directed curve.

2.3.3 *Normalize curve*

The above string of points and slopes is substantially the sought description of the curve, but we need to make it independent of curve size and origin of coordinates. For this, we choose a new coordinate system as follows:

(1) Draw the smallest box containing the curve, with sides parallel and perpendicular to the main axis, as in figure 22.

(2) Choose as new X axis the main axis, and as new Y axis the axis perpendicular to the main axis.

(3) Choose as new origin one of the corners of the box such that (a) the curve falls in the first quadrant, that is, all new coordinates will be non-negative, and (b) $x \times y$, the vectorial product of $+x$ and $+y$, comes out of the paper towards the reader.

In our example, we are left with A and B as candidates for the origin. Between these two, the final origin is selected by one of the following conditions (we do not know which, yet): (i) that origin which is closest to the

center of gravity of the curve. If we choose this condition, save the center of gravity as a feature to compare later. (ii) Jerry Evans' method.

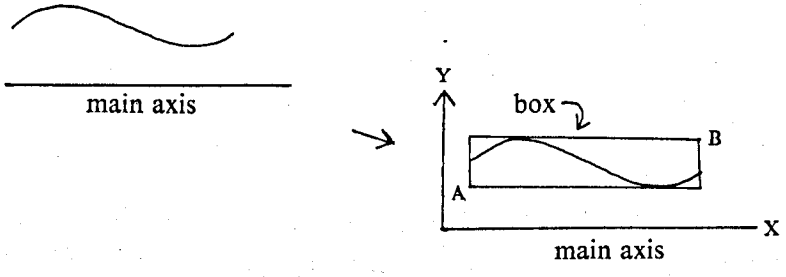


Figure 22

(4) Re-scale (contract or expand) the new axis to give the box unitary dimensions. Thus, the sides of the box have now length 1, and the unit length in the X direction is different from that in the Y direction.

(5) Now, re-compute the coordinates of the midpoints of the alternating list of 2.3.2(3) to refer them to this new origin and re-scaled axes; this will involve:

a translation of origin of A or B;

a rotation of axes to agree with box;

a scaling of Y axis and X axis to give to corners of the box the coordinates (0, 0), (0, 1), (1, 0), (1, 1).

Note that the new Y axis is perpendicular to the new X axis.

After we do this, the alternating list of 2.3.2(3) becomes

(,) *1*	( ) *0*	( ) *3*	( ) *0*	
new coord for P_1	new coord for P_2	new coord for P_3	new coordinates for P_4	slope at P_4

where *1* is the slope at P_1; *0* the slope at P_2; *3* the slope at P_3; and *0* the slope at P_4.

The slopes do not change since they refer to the main axis.

(6) Finally, the code for the line is the above alternating list plus the initial (singular) point of the segment, referred to the new axes; the final (singular) point of the segment, referred to the new axes; and information about how to go from new to original coordinate system, that is, information about: true orientation (instead of 0 orientation) of the main axis and hence of curve; true size (instead of unit box); true position [instead of (0, 0)].

For our example, the code for that line is

c=((0·3, 0·2) (0·8, 0·7) 4·5 6·2 (32, 28) 17 P_1 1 P_2 0 P_3 3 P_4 0)

initial point — final point

where origin (0, 0) is located at point (32, 28) in original space; alternating list (L) as above; contains only shape information (that is, P_1 1 P_2 0 P_3 3 P_4 0; x is to be multiplied by the factor 4·5 to obtain correct x-size; y is to be multiplied by 6·2 to obtain correct y-size, and main axis is really slope number 17).

(*Note*: if C is circular, instead of the initial point, put the mark CIR, and also place CIR instead of the final point. This will indicate that the alternating list should really be considered as a circular list.)

We associated with this code the name given [in section 2.2.(a)] to the directed line; the code for $\bar{C}$ is obtained from C by reversing the initial and final points and the alternating list, thus:

$$\bar{C}=((0{\cdot}8, 0{\cdot}7)\ (0{\cdot}3, 0{\cdot}2)\ 4{\cdot}5\ 6{\cdot}2\ (32, 28)\ 17\ P_4\ 0\ P_3\ 3\ P_2\ 0\ P_1\ 1)$$

What have we done? We are able to code the shape of a curve; to do this we normalize the curve to make it horizontal (that is, along its main axis), then we expand it or contract it to make it fill a unit box; then we bring this box to the origin (0, 0).

2.4 Encode each region as a set of lines

We have described a procedure to encode each line joining singular points. From these descriptions we could regenerate the original scene. Moreover, the scene is not restricted to contain exclusively 'regions'. (A *region* is simply a closed curve, or a surface with a boundary, an inside, and an outside.) In former programs, this was a requirement, and 'illegal' scenes (scenes not meeting this requirement) would indicate that the scene could not be constructed by photographing or projecting a collection of three-dimensional polyhedra.

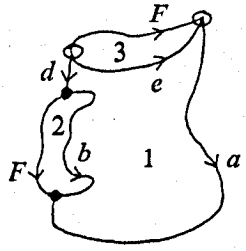
(a) (b) (c)

Figure 23

An illegal scene is detected if by deleting a line you are able to increase the number of connected components; (cf. Guzman 1968, page 217, 1970). Examples are shown in figure 23.

Now a scene which is 'illegal' is also acceptable, for instance an apple, with 'trees' inside [*see* figure 23(c)]. We do not know if we will use the concept of 'region' as much as before, but just in case we do, we will specify a way to encode regions in terms of (already coded) lines.

Coding a region is simple, you go around counterclockwise and write down the names of the lines as you find them, as shown in figure 24. We understand that the code for a region is circular, so that code for region 1 could have been $\bar{e}\ d\ b\ \bar{a}$; $d\ b\ \bar{a}\ \bar{e}$; or $b\ \bar{a}\ \bar{e}\ d$.



region name		code for that region
1	=	$\bar{a}$ $\bar{e}$ d b
2	=	c $\bar{b}$
3	=	e $\bar{f}$

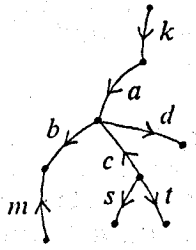
Figure 24

Coding of illegal regions

A tree of lines is coded as shown in figure 25(a). A region with a tree hanging is coded as shown in figure 25(b). If there are several trees at a node, list them counterclockwise, first the interior ones, as shown in figure 25(c).

Note. There are many more cases. This section on coding of illegal regions needs more work; it may be inadequate as it is now.

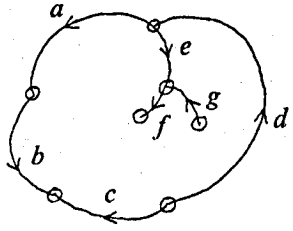
$(k \quad a \quad \underbrace{(b \quad \bar{m}) \quad (\bar{c} \quad (s) \quad (t)) \quad (d)}) \quad$ first k, then a, then a node with three branches, listed counterclockwise



a tree is a list of branches

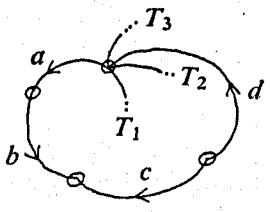
(a)

$d \quad a \quad b \quad \bar{c}$ – region without tree
$d \quad (e \quad (f) \quad (\bar{g})) \quad a \quad b \quad \bar{c}$ – with tree
where $(e \; (f) \; (\bar{g}))$ is the tree between d and a



(b)

$d \quad (\quad) \quad (\quad) \quad (\quad) \quad a \quad b \quad \bar{c}$
tree T_1 tree T_2 tree T_3



(c)

Figure 25

2.5 Encode special regions and lines

Although the encoding just described is good for any scene composed of curved and straight lines, it could save code to have special ways to encode repetitive shapes.

Stick regions. A region formed by two long parallel (not necessarily straight) lines classifies as a stick; to code the stick we code its skeleton plus information about length-to-width ratio. An example is shown in figure 26.

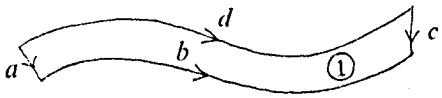


Figure 26. Normal coding for region 1 is $1 = a\ b\ \bar{c}\ \bar{d}$; alternative coding, recognizing that it is a stick, is 1=(STICK 0·05*b*), where 0·05 is the ratio of width : length and *b* is the code for the skeleton of 1, which is essentially the code for *b* or *d*.

Should we use $a\ b\ \bar{c}\ \bar{d}$ or (STICK 0·05 *b*) as code for region 1? Until we know better, we will retain both.

Wiggly lines. The wiggly line in figure 27(a) has been coded by the general coding procedure as the (independent) lines *a*, *b*, ..., *k*. An alternative coding is shown in figure 27(b). Regions containing repeated lines are shown in figure 28.

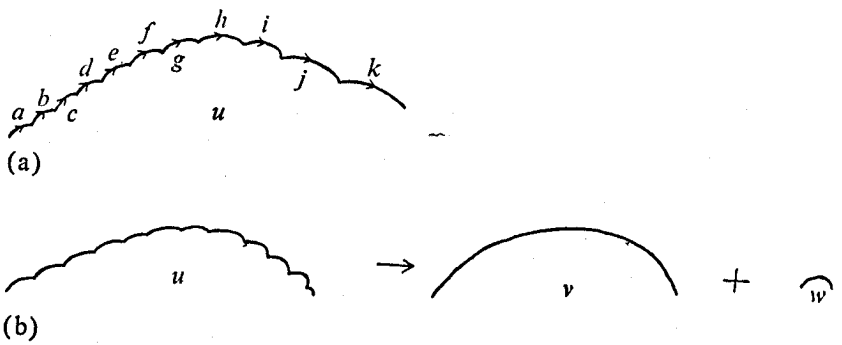


Figure 27. *u* is the name of the line; *v* the principal shape; *w* the shape of the modulation; and 0·1 the ratio size of the modulation: size of the carrier.

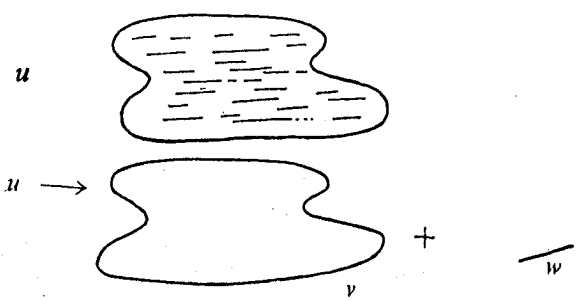


Figure 28. *u*=(CONTAINING *v w* 0·05 βββ ααα) where *v* is the shape of the container; *w* is the shape of the contained; 0·05 is the ratio, size *w*: size *v*; and α and β are other parameters (as yet imprecise), for example, how often *w* is found, in what directions, and so on.

2.6 Example of an encoded scene

The code for scene 'CUPS' (*see* figure 29 'CUPS') is

CUPS=(1 2 3 4 5 6 7 8 9)	A list of regions, plus
$1=(\bar{c}\ \bar{g}\ \bar{f}\ d\ \bar{e})$	a description of each
$2=(b\ \bar{a})$	region in terms of lines,
$3=(\bar{m}\ e\ \bar{h}\ f\ g\ i\ j)$	counterclockwise
$4=(k\ l\ \bar{j})$	(cf. section 2.4)
$5=(p\ \bar{o}\ \bar{n}\ \bar{k}\ \bar{u}\ \bar{b}\ \bar{t}\ s)$	
$6=(q\ \bar{p})$	
$7=(o\ \bar{q}\ \bar{s}\ \bar{r})$	
$8=(i\ \bar{u}\ \bar{a}\ \bar{t}\ \bar{r}\ \bar{n}\ l\ \bar{m}\ \bar{c})$	
$9=(h\ \bar{d})$	plus
$a=(\quad)$	
$b=(\quad)$	A description of each
$c=(\quad)$	line in terms of midpoints,
$d=(\quad)$	slopes, normalized axes,
.	and so on, as described
.	in section 2.3.
.	
$u=(\quad)$	

Each of these looks like the description of c or $\bar{c}$.

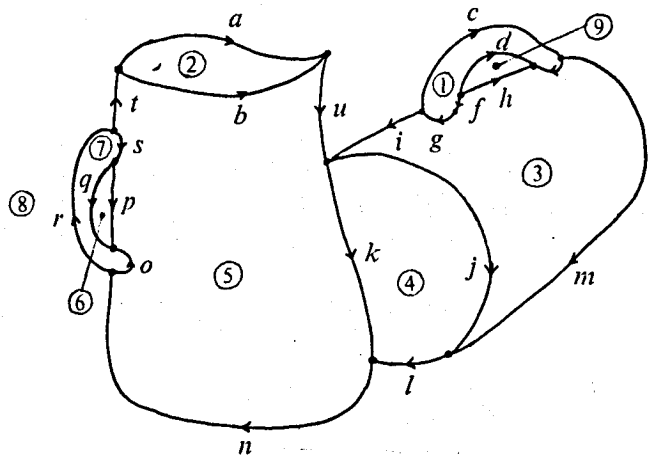


Figure 29. 'CUPS'. The code for this scene is given in section 2.6.

3. MODEL DESCRIPTION

In section 2 we have encoded the scene (input data) to a form that is presumably easier to analyze. This encoding represents the scene originating it, in the sense that you could regenerate the scene from the encoding (for instance, you could display it or plot it).

In this chapter we will encode or represent *models*, which are abstractions or representations of a class of objects (figure 30). In the first half of this chapter, models will emphasize shape, suggesting the name *shape-models* for these models; in the second half, models will emphasize relations among parts, thus the name relations-models. Both names will soon be forgotten.

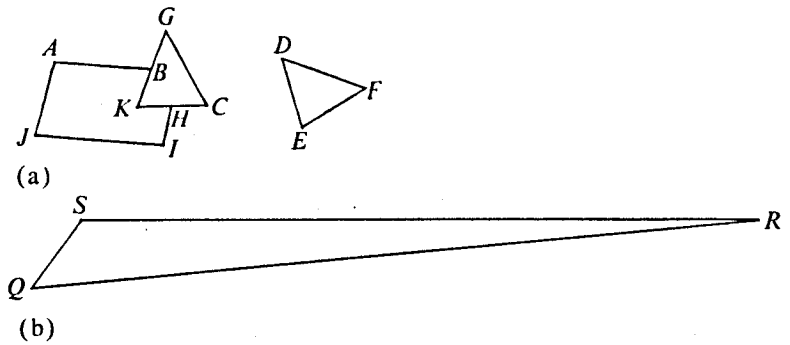


Figure 30. Objects and models. This figure refers to the FDL-1 language or notation to describe scenes and models formed with straight lines (Guzman 1967). A model is the representation of a class of objects. Given a model [such as (3)] and an object [such as (1)] or a scene, the program TD (Guzman 1967) will determine whether that object *matches* the model, which is to say, whether it belongs to the class described by the model, that is, whether it is an *instance* of the model. In this paper we follow the same philosophy, although we do not use FDL-1 notation.
Scene. The description of triangle *EFD* is (*E*(*FD*) *F*(*ED*) *D*(*EF*) (1) where *E* is (10, 6); *F*(12, 5); *D*(11, 3). In the same scene, the description of triangle *C K G* is [*G*(*BC*) *B*(*AKG*) *K*(*BH*) *H*(*KIC*) *C*(*HG*) (2) where *G*(5, 7); *B*(3, 5); *K*(2, 4); *H*(5, 4); *C*(6, 4)]. Each of these descriptions depicts only one object; for instance the last description represents triangle *CKG* in the figure *and nothing else* (and no other triangle). Also, each description *fully* describes the object, so that you could regenerate the object in the same position in space, orientation, size, and so on.
Model. The model (*Q* (*RS*) *R* (*QS*) *S* (*QR*)) represents *any* triangle. (3). It represents a *class* of objects, that is, all triangles, irrespective of position, size, and so on. As such, it represents or stands for triangle *DEF* and *KGC* above. The model ((*Q* (*RS*) *R* (*QS*) *S* (*QR*)) *where* ((LENGTH *S R X*) (LENGTH *Q R X*) (VARIABLES *X*))) represents *any* isosceles triangle. Any object belonging to the class of isosceles triangles is represented by this model. Given a model, you could draw several of the many objects it represents.

Once we have a model for a hat, we could use it to find hats in a scene, in at least two different ways:

(1) We could give it to a model-matching program, such as TD (Guzman 1967) (shape-models, first half of this chapter).

(2) We could use context and global information in addition to the model-matching program. This approach is described in this paper (relation-models).

Either way, we need a *model* for a hat. Can we isolate an object in a scene without having a model or description of it? Yes, we can, when the object is a polyhedron (flat face) [Guzman 1968].

The use of models is as follows: in order to find hats in a scene, we *describe* what a hat is, by writing a model for a hat. This model is more general than any particular hat; it represents the class of objects known as 'hats', or, at least, many hats. Ideally, it is a collection of constraints that the hats obey; it specifies the criteria to determine whether a given object belongs or not to the class hat.

This is the ideal case, when a model really represents *all* hats. In practice, our models specify geometric and topological constraints on the shape, size, and so on, of an object to be classified as hat, while the human constraints refer also to the *use* of the object (that is, covers the head); for instance, in the tropics big leaves may be used as hats in rain. Not being so smart as to take these things into account, our models are generalizations or abstractions of objects in the following manner.

3.1 Types of models

(1) *Fixed.* If necessary, a model can represent a single object, that is, a particular hat in a particular position of certain size, and so on.

Figure 30 refers to the FDL-1 language or notation to describe scenes and models formed with straight lines (cf. Guzman 1967).

A model is the representation of a class of objects.

Given a model [such as (3)] and an object [such as (1)] or a scene, the program TD (Guzman 1967) will determine whether that object *matches* the model, which is to say, whether it belongs to the class described by the model.

In this paper we follow the same philosophy, although we do not use FDL-1 notation.

Thus, it is possible to specify model M1 such that it represents hat H1, as shown in figure 31.

(2) *Rigid.* A model can represent a class of objects obtained by translation of a particular object. That is, it is possible to specify mode M2 which represents any of hats H1, H7, H8, but not the others in figure 31.

(3) *Free.* As above, but now a model can also allow for rotation. Thus M3 represents any of hats H1, H7, H8, or H9.

(4) *Arbitrary size.* A model can also allow enlargements or contractions. It is thus possible to specify model M4 to represent any of H1, H2, ..., H9.

(5) *Constraints.* Sizes and orientations of the individual components of a model (a model is composed of several regions or closed surfaces) could remain fixed or vary relative to each other in arbitrary but prespecified ways. That is, a model may specify constraints on its free variables $V1$, $V2$, ... Thus, it is possible to specify models M5 to represent any of H2, H3, H5, H7, H8, but not the others.

So far we do not have flexibility to change or specify changes or tolerances in the local shape of the model, that is to say, the shape or form of the different lines. For instance, no model could represent (so far) both H7 and H10. We remedy this in part by introducing 'OR' models and 'sloppy' models.

(6) *ORs*. A model could represent the disjunction of several models. Thus, M6 represents any of H1, ..., H10.

(7) *Sloppy*. Deviations in the shape of the lines forming the model are allowed. We do this by specifying a match tolerance or sloppiness for some metalines, that is to say, the error allowed when we compare two similar but not congruent lines. A way to implement this is by having the program that compares a line against a metaline to give as an answer, instead of 'match' or 'failure to match', a score between 0 (perfect match, congruency) and 1 (perfect mis-match), a measure of the dissimilarity in shape.

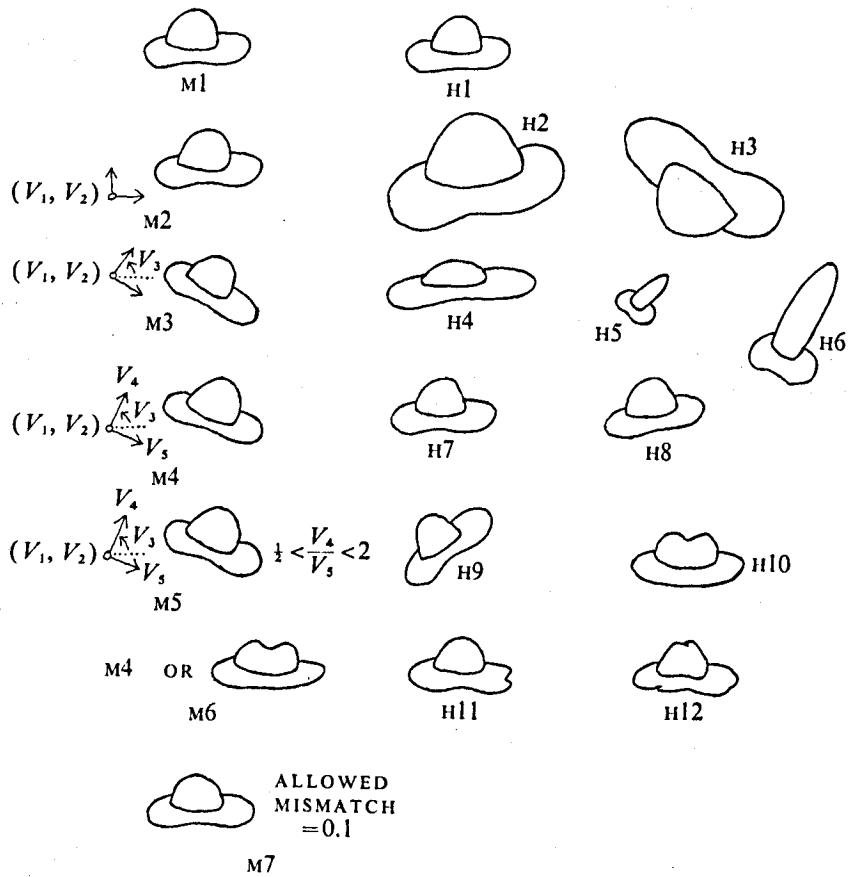


Figure 31. Models and hats. Free variables $V1$, $V2$ of model M2 allow it to match with H1, H7, or H8. Model M3 has free variables $V1$, $V2$, and $V3$.

Thus, if we obtain 0·08 when comparing a metaline with a line, but that metaline has associated a sloppiness of 0·1, we consider that it matches the line. This is because a metaline with a sloppiness of 0·1 matches with lines that are either congruent to it or differ from it by not more than 0·1. Thus, M7 represents any of H1, H2, ..., H12.

(8) *Different topology*. A way to specify changes in the topology of the model is shown in figure 32. This is modelled after CONVERT (Guzman and McIntosh 1966) and the models of my program TD (*MACTR* 37). We do *not* propose to implement (8) now. Although it will be quite useful and important to be able to specify two-dimensional patterns of such generality, we propose this as an independent study preferring to concentrate here on the Context Problem.

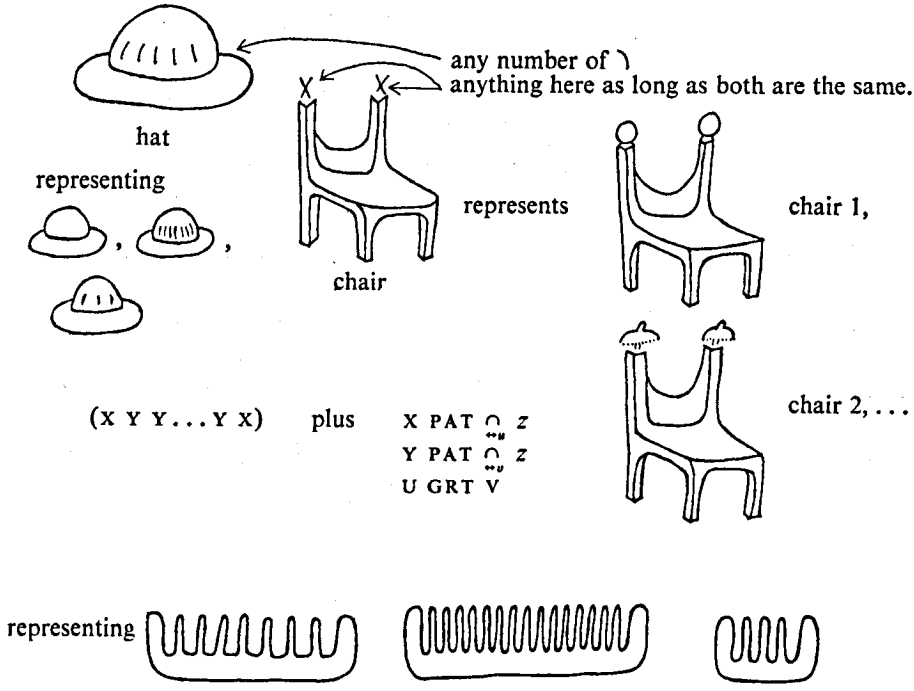


Figure 32

3.2 Producing the shape-model

We now specify the notation used to write models (called shape-models, since shape is emphasized); but we will not discuss here how to write a program that interprets this notation and really makes the comparison of a model against an object (or against a collection of regions of a scene). Such a program will answer 'yes' or 'no' to the question 'does this object match this model?', where the model is written in the notation we are about to describe. We expect that this program will follow the procedures used in TD (Guzman, CCA) and DT (*MACTR* 37), but at this stage we ask the reader to believe in such program. Later, we will discover that it is sufficient to write a program that compares individual REGIONS of the model (meta-regions) against individual regions of the scene. This simplifies the task.

Here is how we build or write models that could have properties (1) to (6) above: Let us remember that the encoding or description of a line is of the form

```
C=      ******** ******* ***  ***  ******
=((0·3, 0·2) (0·8, 0·7) 4·5  6·2  (32, 28)
  initial    final     X    Y    position
  point      point     factor factor of origin
  **
  17              P1 1 P2 0...)
  slope of main   shape
  axis            information
```

This encoding represents a *particular* line: that which has initial point (0·3, 0·2), final point (0·8, 0·7), and so on.

.1.1. In contrast to this, a metaline (model line) is a line with some of the parameters marked with *** left unspecified (we denote this by the symbol == of CONVERT), and represents the class of lines obtained by assigning arbitrary values to such unspecified parameters; for instance, the metaline C1=((0·3, 0·2) (0·8, 0·7) 4·5 6·2 == 17 *P*1 1 *P*2 0 *P*3 3 *P*4 0) where ==is the position of origin (unspecified) representing any line obtained by displacing *C* arbitrarily, that is, by choosing an arbitrary origin.

*C*1 represents any of the lines shown in figure 33(a) since we have left undefined the position of the origin.

Also, metaline *C*2=((0·3, 0·2) (0·8, 0·7) 4·5 6·2 (32, 28)==*P*1 1..) [where == is the slope of main axis (unspecified)] represents any line obtained by rotating *C*; in fact it represents 64 lines, since there are only 64 slopes. *C*2 represents any of the lines shown in figure 33(b) while metaline

```
C3=((0·3, 0·2) (0·8, 0·7) == 6·2 (32, 28) == P1 1 P2 0..)
    initial    final     X                 slope of main
    position   position  magnification     axis (unspecified)
                         (unspecified)
```

represents or describes any line obtained by tilting *C* and stretching it along the *X*-axis: *C*3 represents any of the lines shown in figure 33(c).

3.2.2. Instead of leaving these *** parameters completely unspecified, we could specify them by referring them to a variable, and 'setting the value of this variable outside the metaline', thus:

*C*4=((0·3, 0·2) (0·8, 0·7) *S S* (32, 28) 17 *P*1 1 *P*2 0 *P*3 2 *P*4 0)

(where X and Y magnification is set to be *S*) representing any line obtained from *C* by magnifying X and Y by the same amount *S*. The exact value of *S* is not specified in the metaline (but it will usually be specified 'outside' it – this will be clear in a minute), so we only know at this moment that the same magnification will be applied to both X and Y.

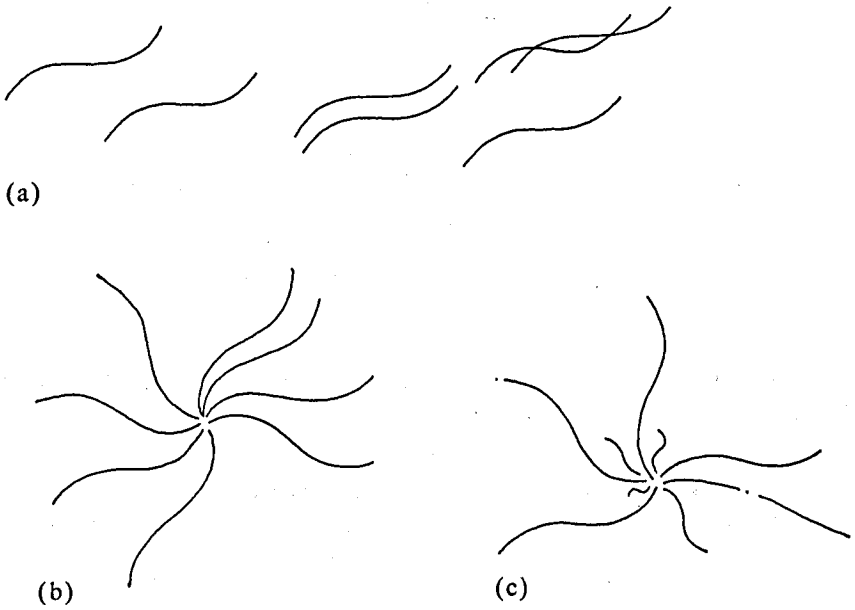


Figure 33

3.1.3. We intend models to be collections of regions (metaregions) which in turn are collections of metalines: a model M is a list of the form

$$M=(V1\ V2\ V3\ V4\ r_1\ r_2\ r_3\ldots)$$

where $V1$ is the X-magnification, $V2$ the Y magnification, $V3$ the position of origin, $V4$ the slope of main axis and $r_1, r_2, r_3\ldots$ the metaregions forming the model; plus the definition of its metaregions: $r_1=(L_1\ L_2\ L_3\ldots)$; $r_2=\ldots$; $r_3=\ldots$; and so on; plus the definition of its metalines: $L_1=\ldots.$; $L_2=\ldots.$; $L_3=(***)$; and so on; and the starred parameters are here specified to be either numbers or expressions involving $V1$, $V2$, $V3$, $V4$; that is, we are describing the initial position, X-magnification, slope of main axis, and so on, of the individual lines in terms of the X-magnification, Y-magnification, and so on, of the model.

3.1.4. The model M of section 3.1.3 has an unspecified X-magnification, Y-magnification, and so on (that is $V1$, $V2$, $V3$, and $V4$ are unspecified). We could restrict such a model by the addition of *properties* or constraints imposed on the V_i, as follows:

$M2=((V1\ V2\ V3\ V4\ r_1\ r_2\ r_3\ldots)$ *where* (greater $V1$ 5) (smaller $V1$ 10)) that is, $M2$ represents the class of objects that match model M and also have X-size between 5 and 10 (remember $V1$ is the X-magnification). The connective *where* attaches properties to models, as we previously did in DT (Guzman 1967).

These properties may be any defined function of the V_is (any function defined by a subroutine). In the example shown in figure 34, any value for

X-magnification and Y-magnification is all right. Notice that we require X-magnification=Y magnification. Any position of the mug is acceptable. Any orientation of the mug is good. Notice that we do not include *R*5 the background as part of the mug.

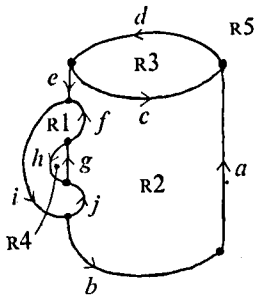


Figure 34

$$\text{MUG}=(V1\ V1\ V3\ V4\ \underbrace{R1\ R2\ R3\ R4}) \quad \text{(underbrace)}$$

where *V*1 is the X- and Y-magnification, *V*3 the original position, *V*4 the slope of main axis, and *R*1, *R*2, *R*3, *R*4 are regions of the mug.

$R1=(i\ j\ \bar{h}\ f)$
$R2=(b\ a\ \bar{c}\ e\ \bar{f}\ \bar{g}\ \bar{j})$
$R3=(d\ c)$
$R4=(h\ g)$
$a=(\quad)$
$b=(\quad)$
.....
$j=(\quad)$

where we put expressions containing *V*1, *V*3, *V*4 in the empty brackets.

We do not elaborate more, since (1) no room; (2) fine details depend much on the TD-like program that matches these models.

These models, called momentarily 'shape-models', that could describe accurately the shape of the body being described, will be used mainly to describe regions (that is, a model will contain only one metaregion), the relationships of shape and position. For example, we plan to describe a cube as shown in figure 35.

This new model of a cube (called momentarily *relation-model*) is less specific that the shape-model of a cube, because it will allow things such as figure 35 (b) to be classified as Cube, while the shape-model will reject that thing, due to the more elaborate requirements about lengths of sides, and so on. We will see later that this will cause little harm. Then, the use of shape-models will be limited to specifying each region (metaregion), such as the description of parallelogram above, and we will use relation-models to represent an object or a class of bodies, using relations between meta-

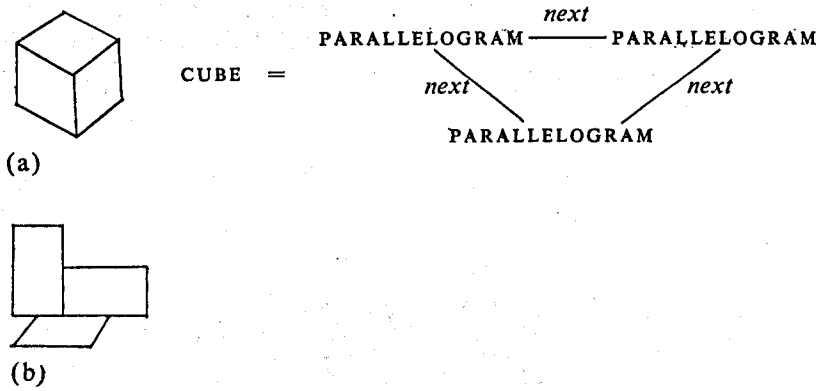


Figure 35. New model of a cube, an example of relation-model. The description PARALLELOGRAM=( ) will be described as specified above (strict description of its shape) under 'shape-models.'

regions, as in the relation-model CUBE above. We will then forget the names shape-model and relation-model.

The apparent failure of relation-models to be precise is just apparent; given more and precise relations, we could refine the relation-model of the cube so as to accept or match only cubes. But we do not choose to do so, and instead we leave the relation-model of the cube (and of most of our objects) under-specified or not sufficiently specified so that they will match with cubes but also with other things. Why do we do this? Carelessness? No; in most of our scenes things like figure 35(b) will not appear, so the insufficient specification of a cube will not hurt. What do we gain? Some speed, because a cube will be faster to check, or to match, if it contains few properties or relations.

3.2 A scene as a graph of relations between regions

3.2.1 *Synopsis*

We would like to talk of relation-models, and for this we need to establish or discover relations between parts of the scene. These parts are the *regions*.

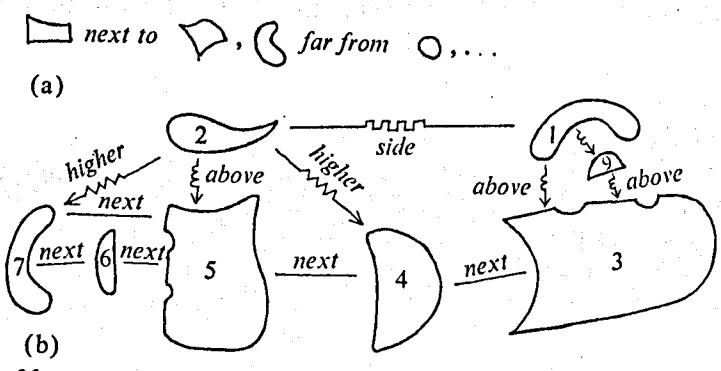


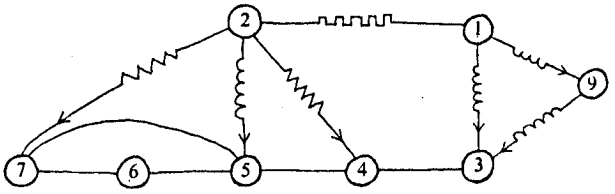
Figure 36

The relations or predicates [*see* figure 36(a)] indicate geometric properties that hold between regions, and a graph of relations is just that, for instance, for scene 'CUPS' (figure 29) we have the graph shown in figure 36(b), which could be represented as figure 37(a).

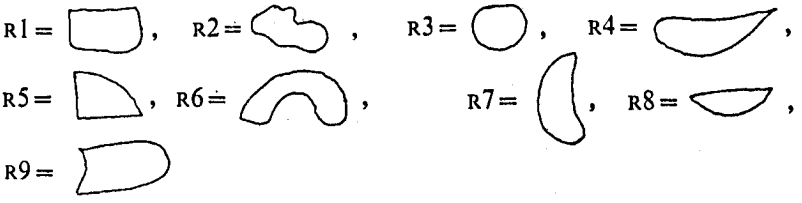
This section explains how to obtain these graphs.

What is their use? Well, we also have in memory many models (shape-models) of regions: for instance, our SHAPE memory may contain the models or descriptions shown in figure 37 and when we apply a program [ETD to the nodes 1–9 of (a)], we will find that we can label figure 37(a) as shown.

matches $1 \sim R6$ $5 \sim R1$
$2 \sim R4$ $6 \sim R7$
$3 \sim R8$ $7 \sim R6$
$4 \sim R7$ $9 \sim R7$



(a)



(b)

Figure 37

Well, we also have in memory (our model-memory) models (relation-models) that describe objects which we desire to identify in our scenes; *see*, for instance, figure 38.

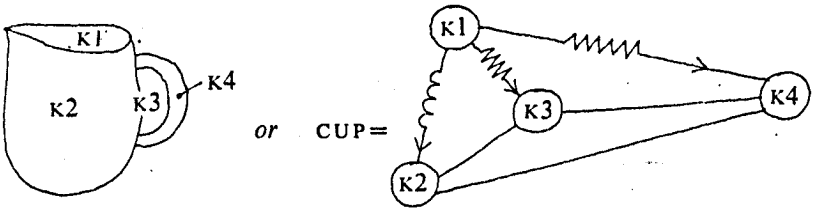


Figure 38

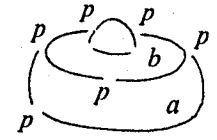
To find 'cup' in the scene, we could find whether the graph for cup is a subgraph of the graph for the scene. Our proposed method to find if a cup is present (and where it is, plus a description of the object) is not based in graph comparisons, although it could work that way by using graph isomorphism and tree search on graphs (Rastall 1969), but it is based instead on the use of global and local information.

Thus, we find *regions* in a scene, then find *relations* between these regions and finally we organize these relations into a graph, as in Barrow and Popplestone (1971).

Having done this with the scene, we could do the same with the object which we want to model, except that its regions will be described as meta-regions (model-regions, using shape-models) and that we will describe not only objects but also agglomerations of objects that occur frequently; that is, in addition to the model of hat and the model for person there may be a model for person wearing hat. (As we will see in section 6, models can have models as components.) Once we have expressed the models of the objects and the scene in this graph-oriented manner, we proceed to compare them, as explained in section 4.

3.2.2 *Regions*

Regions in the scene are closed curves (cf. section 2.4). Relations will be placed between regions. In addition, it is useful to consider the following as 'regions' for the purpose of placing or finding relations between them:
(1) 'almost closed' regions – such as *a* and *b* which will be closed if we close the little holes '*p*' as shown in figure 39(a). (Indeed, we think artists leave these little spaces to make an uncluttered and clean drawing, without joints which are blacker.)
(2) Open regions, that is, lines which are peculiar to certain objects. For instance, *R* in the fork, *S* in the shirt in figure 39(b). The trouble with these is that it is going to be difficult to find them in the scene, unless we are looking for them.



(a)

(b)

Figure 39

3.2.3 *Relations*

Each region of a scene is in certain relation(s) with others. To describe a model we are specially interested in geometric relations and in those that are peculiar or essential to the model and set it apart from other models. Thus, a relation-model will contain 'important' relations, given by the user or whoever constructed that model.

For a scene, we do not propose to find *a priori* the values of all possible relations between its regions. [My former office-mate Tom Evans (1970) finds the values that the relations (of a specified set) obtain when applied to all possible *n*-tuples of regions of the scene, in a program that studies two scenes, describes them, and then 'merges' intelligently these descriptions, giving birth to a third description which economically describes or generates both scenes.] However we will look for them as we need them (or as we think we need them), guided by the models we are trying to match. To witness, if we think figure 40(a) could match with the model in figure 40(b), only then we look whether (ABOVE K2 K1) is true. More in section 4.

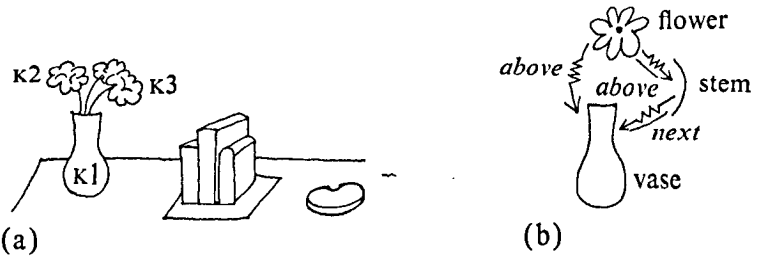
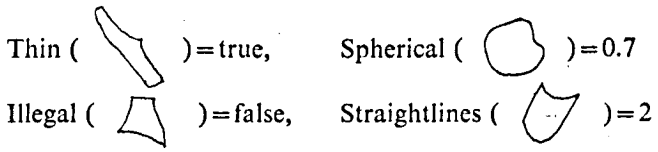


Figure 40

Instances of relations are

(1) *Predicates.* Or one-place relations. Examples are shown in figure 41. Although some of these are not predicates, we could include them here. Since these predicates refer to *one* region, they end up being a function of the shape of the region. In this sense, they do not provide 'global' information. (*Local information*: that obtained by analyzing one region alone, ignoring or disregarding its surroundings. Local information is produced by the *shape* of the region. *Global information*: that obtained by analysis of several regions together, paying attention to their disposition, relative position, similarities, and other relations among them. Global information is given by the *relation* between regions.)



Horizontal, floating, and others.

Figure 41

In the graphs of section 3.2, if the nodes are regions, then the predicates are labels or colors pasted on the nodes.

(2) *Two-place relations.* Very common. Shown in the graphs of section 3.2 by links or arcs, of different shape for different relations. Directed arcs for non-symmetric relations. Some are shown in figure 42: next, higher, left, sprouts, interleaved, non-touching, and others. Most of these relations describe the position of the two regions in two dimensions.

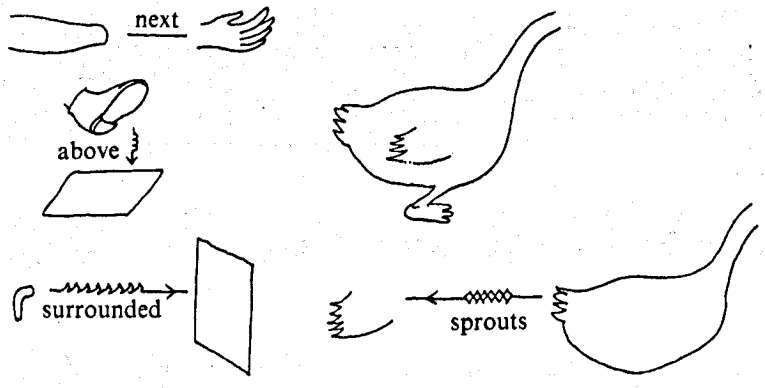


Figure 42

More difficult to compute from the 2-dimensional data are 3-dimensional relations (indeed, this is a separate problem, interesting and difficult) such as that shown in figure 43. Although we would like to, we do not propose to make use of 3-dimensional relations. Reasons: (a) our models and our data are 2-dimensional; (b) we do not know how to compute them.

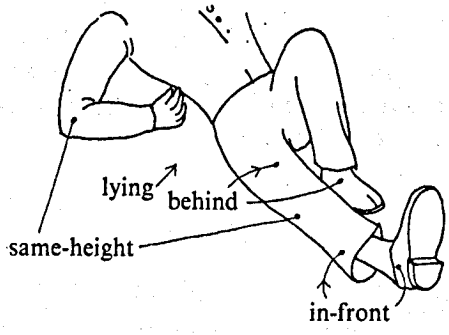


Figure 43

(3) *Three-place relations.* There are several. (*See* figure 44.) Incidentally, I have no elegant way to represent them in the graph. I do not like the notation in figure 44(b) because it adds the 'object' to the other objects in the picture.

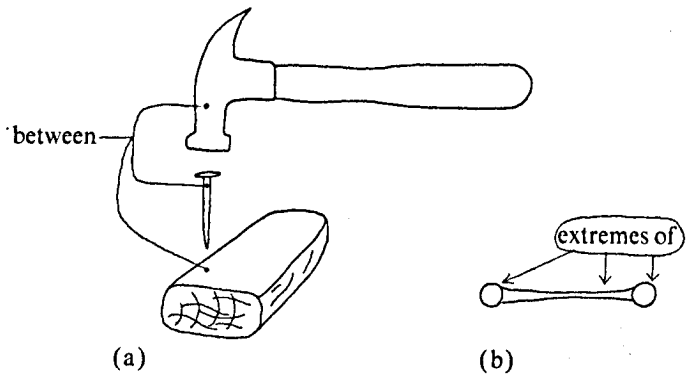


Figure 44

(4) *Many-place* relations. Some are shown in figure 45. We will handle these relations in the same manner as 2-place relations.

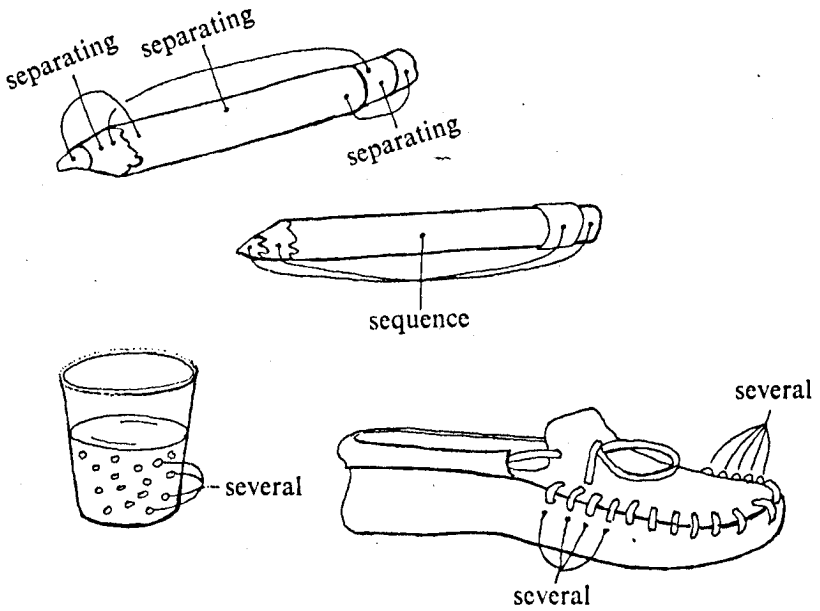


Figure 45

3.2.4 *Summary*

A model is a graph of relations between its metaregions: each metaregion is described to represent a class of regions:

$$R2=(V1\ V2\ V3\ V4\ldots)\ \text{(shape-model)}$$

Given a scene as a collection of regions we use shape-comparison (the program ETD) against our relation-models, to assign to each region a name (of a metaregion), as in figure 46.

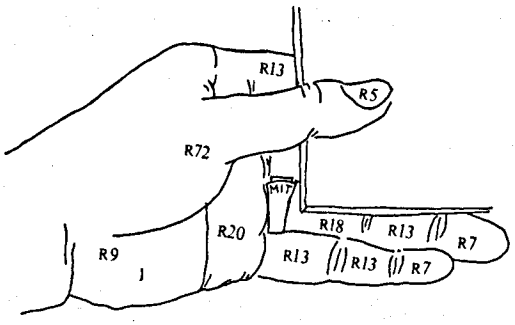


Figure 46

4. ANALYSIS

We wish to compare models with scenes, so as to ascertain, for each region in the scene, the name of the object possessing such region. There are several ways to go about this. First, there is the tree search approach, where most likely candidates are chosen first, the consequences of these choices are explored and suggest further choices; a failure motivates a revision of choices. This is a *serial* approach, where the first choices suggest the next choices. Secondly, there exists a simultaneous approach, where all possible choices are taken 'together' at the same time, wrong choices being eliminated because of contradictions sprouting from them. These two approaches are symbolic or non-numerical ways to solve the Context Problem. Thirdly, it is possible to use a statistical approach, in which shape assigns probabilities to the events that certain regions are in fact a nose, the hoof of a horse, the tail of an armadillo, and so on. Models are expressed as conditional probabilities between regions related by relations, that is, the probability of $R1$ being a tree given that $R2$ is a rock is 0·6; to obtain the best description of the scene we maximize the probability of the whole scene being consistent, once we take the models into account. In this paper we will not discuss the last two approaches, but only the first.

4.1 Tree-search approach

As we describe this procedure, a simple example will illustrate the road.

4.1.1 *Models*

Suppose we have in the memory of our computer the models appearing in table 2, coded as dictated by section 3. These models are shown in table 3 in order of inclusion or membership, to the right being the terminal models $R1$, $R2$, $R18$, . . ., $R272$, $R281$ which are defined in terms of the shape of the metaregions they represent; to the left being more complex models such as BOY, S-HEAD, or $R24$ formed by simpler models and terminal models. The terminal models themselves appear in table 4, together with the shapes they stand for. They are coded using metaregions (shape-models).

Table 2

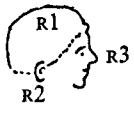
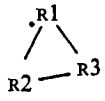
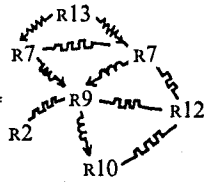
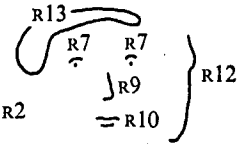
graphic explanation (not a part of the model)	MODEL (stored in memory)	Comments (not part of the model)
	S-HEAD= 	definition of side-head
	R1= 	terminal model (defined by a metaregion) (defined by its shape)
	R2= 	terminal model
	R3=R31° or R32	 or 
	R31°= 	terminal
	R32= R7–R4 / R5 / R6 } sequence	
	R4=R41 or R42	or 
	R41=	terminal
	R42= 	terminal
	R5=R51 or R52 or R53	 or or
	R6=R61 or R62	 or
	R7=R71 or R72 or R81	or  or
	R71=	terminal
	R72= 	terminal
	F-HEAD= 	

Table 2—*continued*

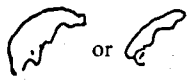
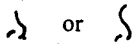
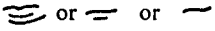
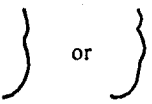
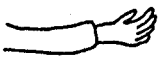
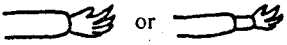
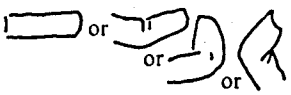
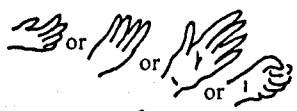
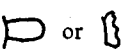
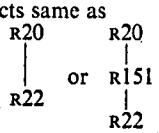
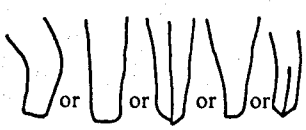
graphic explanation (not a part of the model)	MODEL (stored in memory)	Comments (not part of the model)
	R13=R1 or R132	
ellipse	R29=	terminal
	R9=R91 or R92	
	R10=R101 or R102 or R103	
	CUP= R2 R29 R62 R31	
	R12=R121 or R122	
	SUPERIOR EXTREMITY= R15–R16 or R15–R17–R16	
	R15=R151 or R152 or R153 or R154	
	R16=R161 or R162 or R163 or R164	
	R17=R151 or R172	
	R18=	terminal model
	INFERIOR EXTREMITY= R20 R151–[optional] sequence R22	acts same as 
	R20=R152 or R151 or R203 or R204 or R205	

Table 2—*continued*

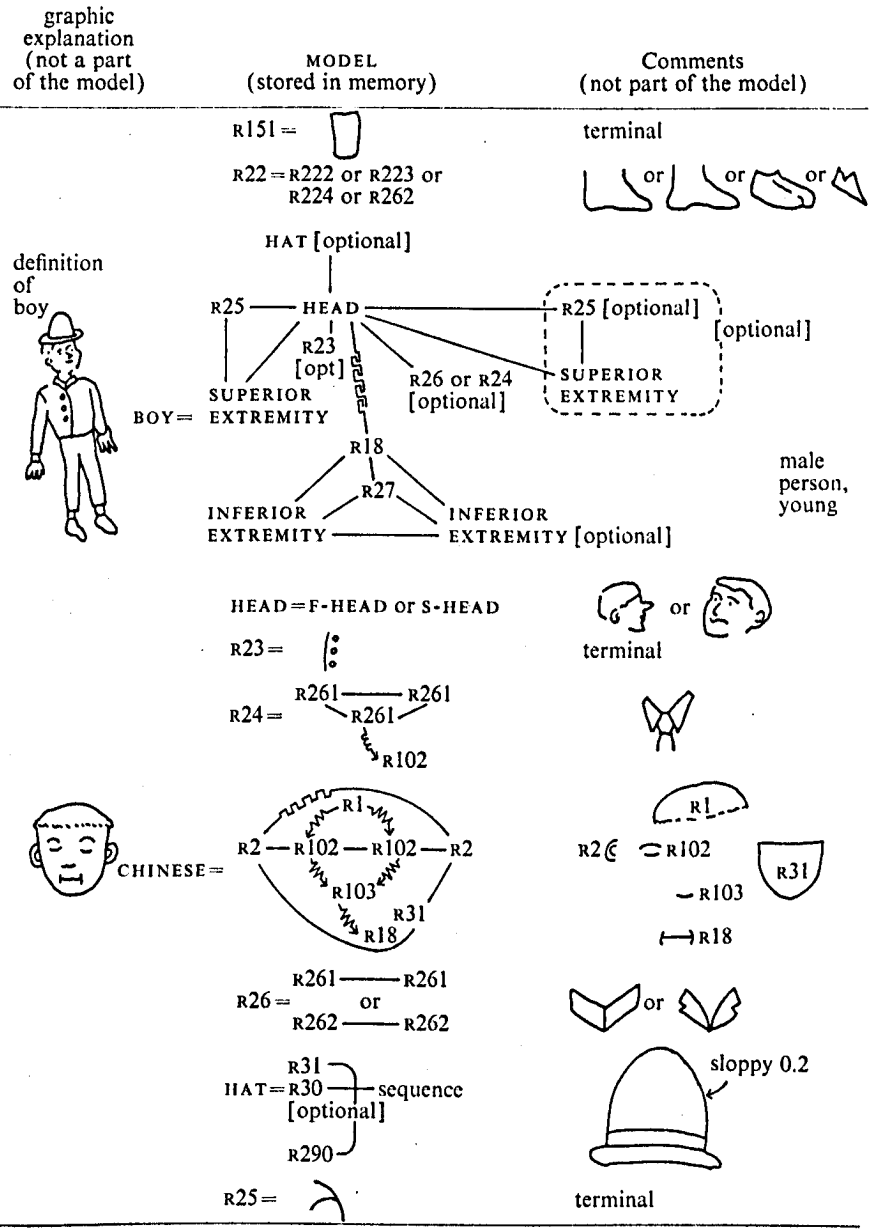


Table 2. Models. The models of this table will be used in the examples that follow. The models of this table are relation-models, and are *non-terminal*, such as R28 (defined in terms of other models), or *terminal*, such as R29 or R31, defined in terms of the shape of the metaregion (this shape is described as a shape-model) they contain. Note the similarity with production rules in a Grammar.
How can a single model, F-HEAD, for instance, represent a great variety of heads, and not only a few? This is due to the ORs that many models have, specifying several alternatives. Also, even terminal models represent many objects, as we saw in section 3.

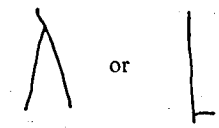
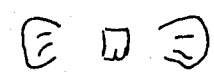
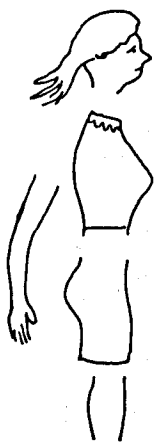
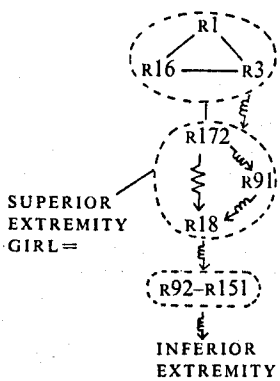
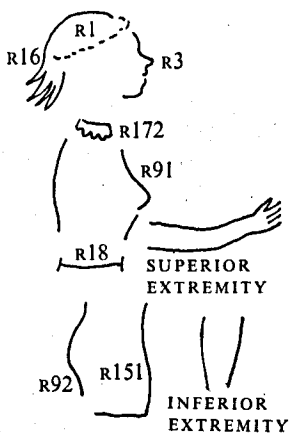
graphic explanation (not a part of the model)	MODEL (stored in memory)	Comments (not part of the model)
	R27 = R271 or R272	
	R28 = sequence R281 R151 R281	
		

Table 3. Terminal and non-terminal models

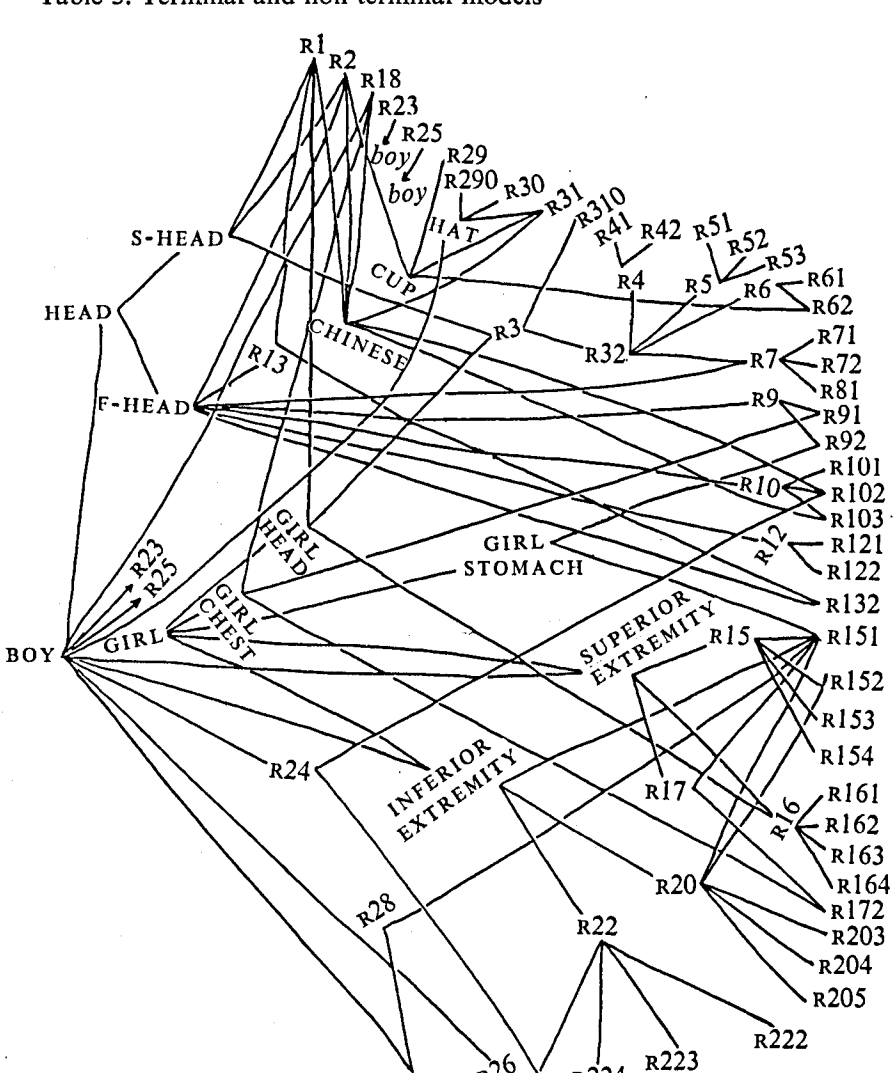


Table 4. Terminal models

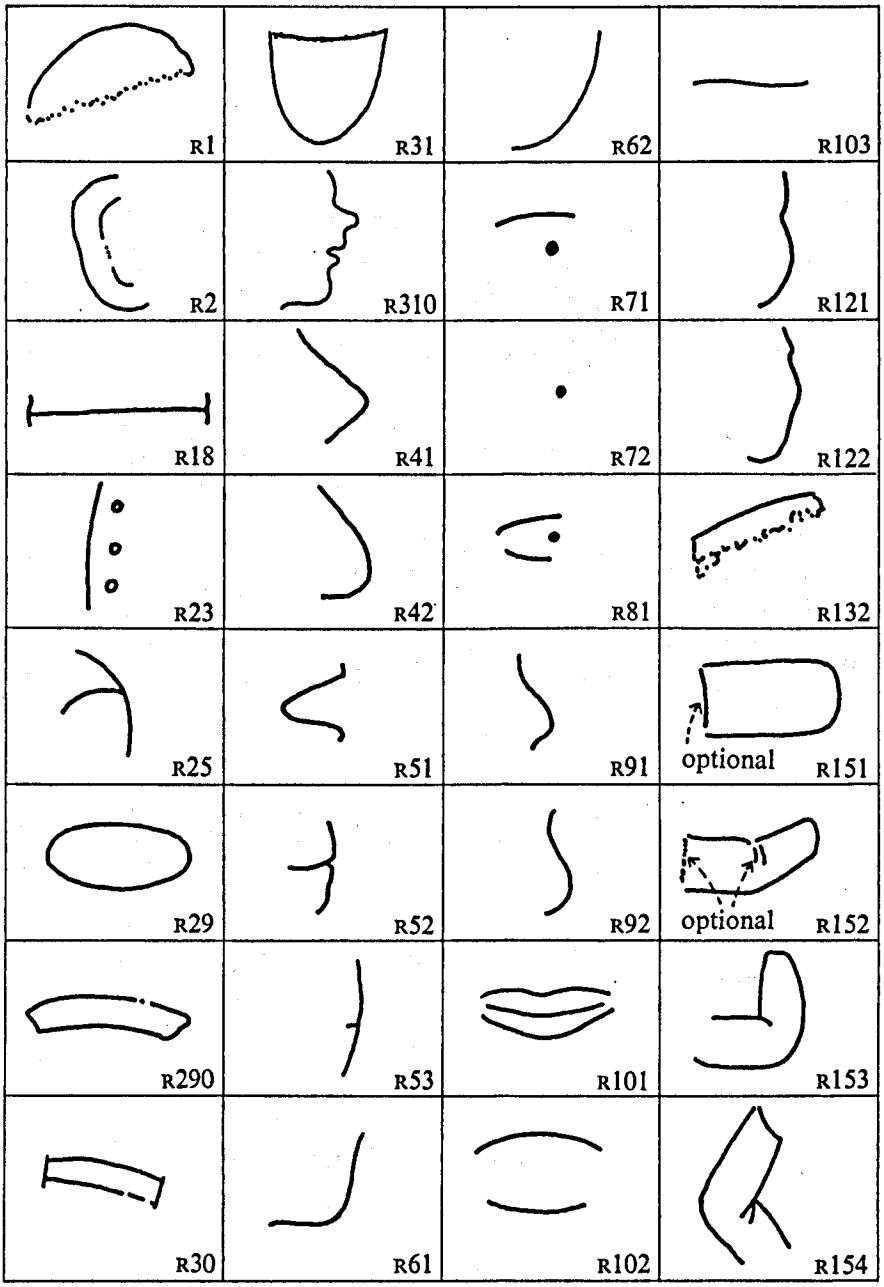
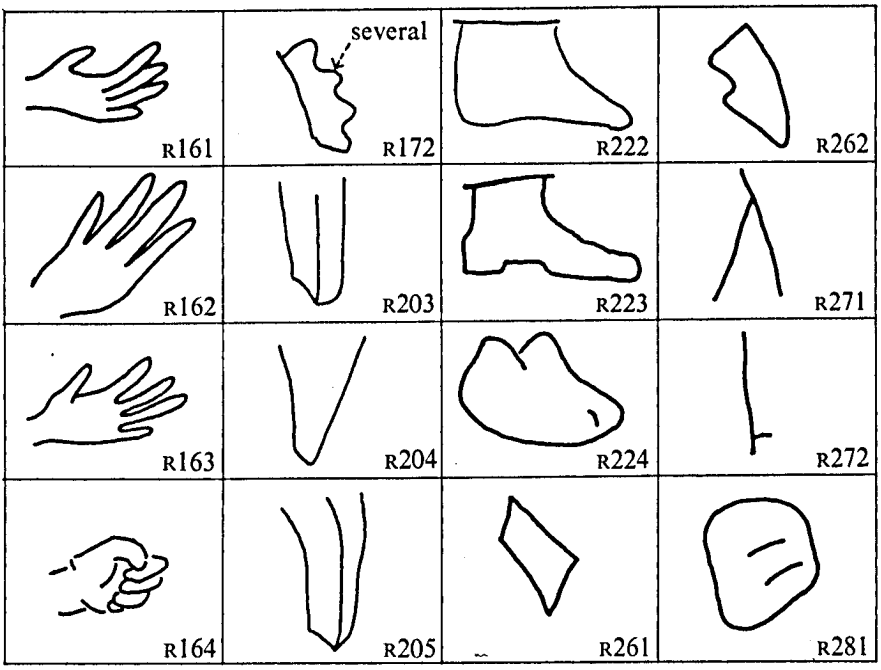


Table 4—*continued*



In general, each model (hence, each terminal model) appears as a format of other 'more complex' models; for instance, *R*18, appears as part of BOY [figure 47(a)] as part of GIRL CHEST [figure 47(b)] as part of CHINESE [figure 47(c)].

In these models the same shape plays different roles (has different names: waist, mouth) but the most we can learn from its shape is that its name is *R*18.

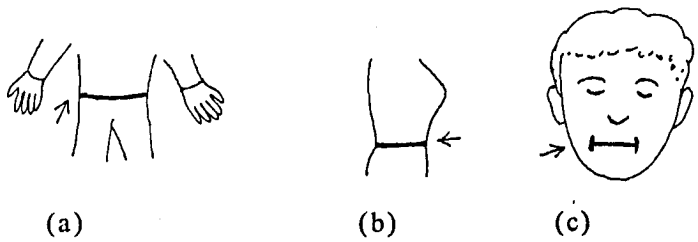


Figure 47

4.1.2 *Scene*

Suppose we want to analyze scene 'PERCHED' (*see* figure 48 'PERCHED') with the models of table 2. The scene is initially described in terms of its

regions and lines, as we saw in section 2. Thus, its regions are *a, b, c, . . . , y, z*. Our next step is to find our names for these regions, according to their shape. For instance, we want to find that region *t* should have the name *R*1, because it matches with model *R*1.

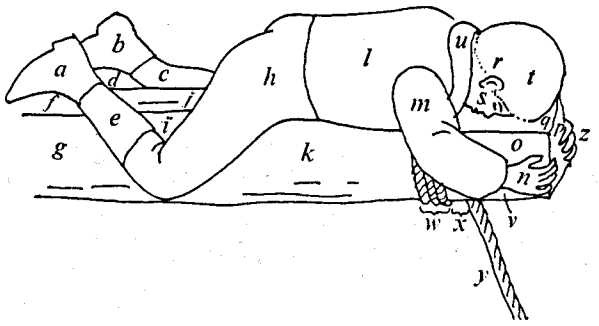


Figure 48. 'PERCHED'. Example of scene to be analyzed.

For each region *r* and each *terminal* model *m*, we could ask whether one is an instance of the other, that is, whether (ETD *m r*) is true or not (ETD is a matcher program). When it is true, we associate to region *r* the name *m*. All the possible alternatives will require too much time to compute, so we use heuristics to suggest pairs (*m, r*) likely to match, and only to these pairs we apply ETD. The paragraph below explains how this is done.

4.1.3 *Selection of likely pairs*

The preliminary search for region *r* matching terminal model *m* [$m \in$ table 3, $r \in$ 'PERCHED'] is done with the help of a vector of features extracted from each region *r* and metaregion *m*:

$$f=(f1,f2,f3,\ldots,fk)$$

where $f1$ = number of straight lines of the region
$f2$ = area of the region
$f3$ = center of gravity [computed already in section 2.3.3(3)b.i]
$f4$ = curve Total Contributions *v. j* [computed already at the end of section 2.3]
$f5$ = eccentricity of the region (cf. Krakauer 1970).
$f6$ = long and thin? yes or no
$f7$ = Fourier coefficients of curve describing boundary
$f8$ =
.
.
.

[all these features are examples only; they may not be the ones that we will eventually use]

These features should be easy (quick) to compute. It may be non-trivial to find 'important' features which will do an adequate job. Barrow and Popple-

stone (1971) have developed similar ideas and have implemented them in a program that uses this type of heuristic to achieve identification of simple isolated curved bodies as seen by a TV camera.

For our example, these features will produce the mapping in table 5. Now we use the program ETD to refine table 5 as in table 6. These matches represent the information about scene 'PERCHED' that it is possible to extract from the shape of its regions.

Table 5. Preliminary terminal model

$a \sim R222, R262$	$j \sim R30$	$s \sim R310, R103, R91$
$b \sim R41, R261$	$k \sim$nothing	$t \sim R1, R31$
$c \sim R151$	$l \sim R262$	$u \sim R30, R151$
$d \sim$nothing	$m \sim R152$	$v \sim R223, R222$
$e \sim R151, R261$	$n \sim R162, R161, R172$	$w \sim$nothing
$f \sim R102, R30$	$o \sim R223$	$x \sim R261$
$g \sim$nothing	$p \sim$nothing	$y \sim$nothing
$h \sim R152, R30, R154$	$q \sim R132$	$z \sim R172$
$i \sim R41, R31$	$r \sim R2, R42$	

Table 6. Matches obtained by shape comparison

$a \sim R222$	$r \sim R2$
$e \sim R151$	$s \sim R310$
$h \sim 152$	$t \sim R1, R31$
$j \sim R30$	$u \sim R222$
$m \sim R152$	$z \sim R172$
$o \sim R223$	

Note that ETD cannot cope with occluded regions. For example, (ETD R222 *b*) will fail because *b* is a shoe which is partially occluded. We will need the actions of the program (ISPOSSIBLE R222 *b*) which answers the question: Is it possible to occlude R222 so as to produce *b*? Is it consistent to allow *b* to be a (partially occluded, perhaps) instance of R222?

4.1.6 *The match*

Table 6 gets reduced to table 7, where we list the regions that correspond to terminal models belonging to only one non-terminal model (as deduced from table 3).

Let us analyze each correspondence in table 7.

Table 7

$a \sim R222$
$j \sim R30$
$n \sim R161$
$o \sim R223$
$s \sim R310$
$v \sim R222$

$a \sim$R222 because, from table 2, $R22 = R222$ or $R223$ or $R224$ or $R262$, thus, $a \sim R22$.

$j \sim R30$? From table 3, $R30$ belongs to HAT, which requires the sequence $R31$, $R30$, $R290$. This sequence is not found, and thus, $j \sim$nothing; thus j does not match $R30$.

$n \sim R161$ because n is part of $R16 = R161$ or $R162$ or $R163$ or $R164$, thus $n \sim R16$.

Similarly,

$o \sim R223$ implies $o \sim R22$
$s \sim R310$ implies $s \sim R3$
$v \sim R222$ implies $v \sim R22$

Our list is now

$a \sim R22$
$n \sim R16$
$o \sim R22$
$s \sim R3$
$v \sim R22$

We apply the same procedure again, but first the list is rearranged into $a \sim R22$, $o \sim R22$, $v \sim R22$, $n \sim R16$, $s \sim R3$, in order to work first with the models ($R22$) which form part of only one larger model [$R16$ and $R3$ form part of several models, hence they come at the end].

Is $a \sim R22$? Table 3 indicates that

INFERIOR EXTREMITY = sequence of $R20$, $R151$ and $R22$

So $a \sim R22$
$e \sim R151$, i.e., (a, e, h) INFERIOR EXTREMITY
$h \sim 222$
$o \sim R22$? No, because we do not find a $R20$ or a $R151$ in sequence with $o \sim R22$, as required by the model INFERIOR EXTREMITY.
$o \sim$nothing
$v \sim R22$? No, by the same reason.
$v \sim$nothing
$n \sim R16$? If so, it has to form part (by table 3) of GIRL HEAD or of

SUPERIOR EXTREMITY.

$S \sim R3$? Yes: $S \sim R3$; $r \sim R2$; $T \sim R1$ i.e., (s, r, t) S-HEAD

We have finished our second run; our list of matches is now

$(a, e, h) \sim$ INFERIOR EXTREMITY
$(m, n) \sim$ SUPERIOR EXTREMITY
$(s, r, t) \sim$ S-HEAD

We apply the same procedure again. Our list is recorded into

$(s, r, t) \sim$ S-HEAD
$(a, e, h) \sim$ INFERIOR EXTREMITY
$(m, n) \sim$ SUPERIOR EXTREMITY

in order to work first with models belonging to only a few other models: S-HEAD belongs only to HEAD (*see* table 3), while INFERIOR EXTREMITY and SUPERIOR EXTREMITY belong both to BOY and to GIRL.

$(s, r, t) \sim$ S-HEAD implies $(s, r, t) \sim$ HEAD because HEAD=S-HEAD or F-HEAD

$(a, e, h) \sim$ INFERIOR EXTREMITY?

INFERIOR EXTREMITY must be part of either GIRL or BOY. GIRL fails because there is not a GIRL STOMACH above (a, e, h) so it must be part of a BOY. This requires the presence of $R18$ and $R27$ next (*see* BOY in table 2). These models are not present in table 6. But we observe that they are unlikely to be in table 6 anyway, since they do not form closed regions. We need to look for them if we want to find them (if we suspect their presence). We do this and encounter them.

Then we need a HEAD next to $R18$. That is, $(s, r, t) \sim$ HEAD.

Then we need to find a $R25$ and a SUPERIOR EXTREMITY next to HEAD. This last is (m, n). $R25$ is one of those models that we need to look for. We look for $R25$ and we do not find it. Instead of throwing away the (large) match already found, it is time to use a rule that limits the extent of failures in a match, when many parts of the model have been explained satisfactorily. Thus, we ignore the failure to find a match for $R25$, and declare that we have found a BOY.

HEAD $\sim (s, r, t)$

SUPERIOR EXTREMITY $\sim (m, n)$

i.e., $(s, r, t, m, n, a, e, h) \sim$ BOY

INFERIOR EXTREMITY $\sim (a, e, h)$

To recapitulate, the implementation of our own approach (for which no program has been written yet) depends on the construction of the undernoted algorithms:

(1) A procedure to encode the initial data representing the scene into a more economical description akin to subsequent efficient manipulation. This is described in section 2.

(2) (a) A program that computes features that are useful for quick (preliminary) shape discrimination. Described in section 4.1.3 and table 5.

(b) ETD, a program to determine if a given region matches (is similar to, has the shape of) a given shape-model. This is described in section 4.1.2.

(c) ISPOSSIBLE, a program like ETD that handles partially occluded regions.

(3) (a) An analyzer that determines possible environments by taking notice of abundance or scarcity of certain models.

(b) A search procedure that tries to fit possible models to bigger and bigger parts of the scene, retreating in front of failures and advancing when victory crowns its efforts, until the whole scene is conquered (or relinquished). This procedure needs to have the ability of *limited failure*, that is to say, 'small' mismatches (failures match) will have limited effect.

Finally, it is possible to have line drawings with more than one meaning or consistent interpretation. The approach described in this paper should discover all meanings, provided that (1) we do not stop the tree search at the first consistent matching, and (2) the models are general enough [figure 49(a) will not serve as model of a leg if we need to turn the scene upside down, but figure 49(b) will serve] to be appropriate for each interpretation.

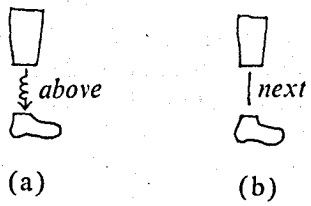


Figure 49

4.1.6 *Unsolved problems*

It is important to indicate the relative size of parts of the same model; then, the size can be used to discriminate among different models. Section 3 provides no satisfactory way to do this. The problem of segmentation is also severe: if we want to walk away from models of closed regions into models for open regions, the approach of this paper needs a manner to divide 'naturally' the scene into its different parts, if we desire to work with models of these parts. Thus we can handle figure 50(a) with models described in this paper; but we need to know how to make the segmentation, for the models of this paper to work for figure 50(b).

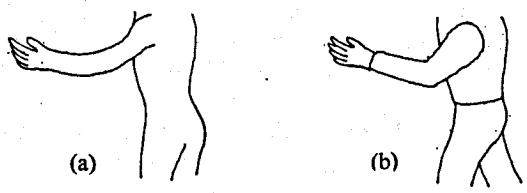


Figure 50

Acknowledgements

This work began at the Artificial Intelligence Group of MIT (Professor Marvin Minsky, Professor Seymour Papert) and has reached its present form at the Department of Machine Intelligence and Perception (Professor Donald Michie) of the University of Edinburgh, where I am spending an active summer. Both places provided a fertile habitat through interesting interactions with their members, whose help I appreciate, as well as the encouragement and sympathy for my work shown by the leaders of these two groups, and also by Dr Bernard Meltzer and his colleagues at the Edinburgh University Metamathematics Unit.

After a talk within the Computer Science Spring Colloquium at University of Illinois, Urbana, useful discussions with Professor Bruce McCormack and Mr John Schwebel indicated coloring books as a possible paradigm for the Context Problem. Some ideas were borrowed (or suggested) from a thesis by S. Ramani at Tata Institute of Fundamental Research, Bombay.

The author's present address is Centro Nacional de Calculo, Instituto Politecnico Nacional, Mexico.

REFERENCES

Barrow, H. & Popplestone, R.J. (1971) Relational descriptions in picture processing. *Machine Intelligence 6*, pp. 377–96 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Evans, T. (1970) C.O.I.N.S. Conference, Las Vegas, Nevada.

Freeman, H. (1961) On the encoding of arbitrary geometric configurations. *IRE Trans. Electronic Computers*, **EC-10**, 260–8.

Guzman, A. (1967) Some aspects of pattern recognition by computer. *Project MAC Report MAC TR* 37. Cambridge, Mass.: MIT.

Guzman, A. (1968) Computer recognition of three-dimensional objects in a visual scene. PhD. Thesis. MIT, Cambridge, Mass. Available also as a *Project MAC Report MAC TR* 59. AD 692 200.

Guzman, A. (1971) In preparation.

Guzman, A. & McIntosh, H.V. (1966) CONVERT *Comm. Ass. comput. Mach.*, **9**, 8, 604–15.

Krakauer, L. (1970) Ph.D. Thesis, MIT, Cambridge, Mass.

Montanari, U. (1968) A method of obtaining skeletons using a quasi-Euclidean distance. *J. Ass. comput. Mach.*, **15**, 600–24.

Narasimhan, R. (1969) On the description, generation and recognition of classes of pictures. *Automatic Interpretation and Classification of Images* (ed. Grasselli, A.). New York: Academic Press.

Rastall, J. (1969) Graph-family matching. *Research Memorandum MIP-R-62*, Department of Machine Intelligence, University of Edinburgh.