SHEDDING LIGHT ON SHADOWS

VISION FLASH 29

by

David L. Waltz

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

Robotics Section

JULY 1972

Abstract

This paper describes methods which allow a program to analyze and interpret a variety of scenes made up of polyhedra with trihedral vertices. Scenes may contain shadows, accidental edge alignments, and some missing lines. This work is based on ideas proposed initially by Huffman and Clowes; I have added methods which enable the program to use a number of facts about the physical world to constrain the possible interpretations of a line drawing, and have also introduced a far richer set of descriptions than previous programs have used.

This paper replaces Vision Flash 21.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0003.

0.0 INTRODUCTION

How are we able to ascertain the shapes of unfamiliar objects? Why do we so seldom confuse shadows with real objects? How are we able to "factor out" shadows when we are interpreting scenes? How are we able to see the world as having an essential identity whether it is a bright sunny day, an overcast day, or a night with only streetlights for illumination? In the terms of this paper, how can we recognize the identity of figures 0.1 and 0.2? Do we learn this identity and use our knowledge to interpret what we see, or do we somehow automatically see the world as stable and independent of lighting? Put another way, do we need to abstract global properties of a scene such as lighting in order to understand particular scene features, or is a knowledge of relatively local features alone sufficient for interpretation without the formation of global hypotheses?

Various theories have been advanced to explain how we extract three-dimensional information from scenes. For example, we can get depth and distance information from motion parallax and for objects fairly close to us from eye focus feedback and parallax. But this does not explain how we are able to understand the three-dimensional nature of photographed scenes. Perhaps we acquire knowledge of the shapes of objects by handling them and moving around them, and then use rote memory to assign shape to

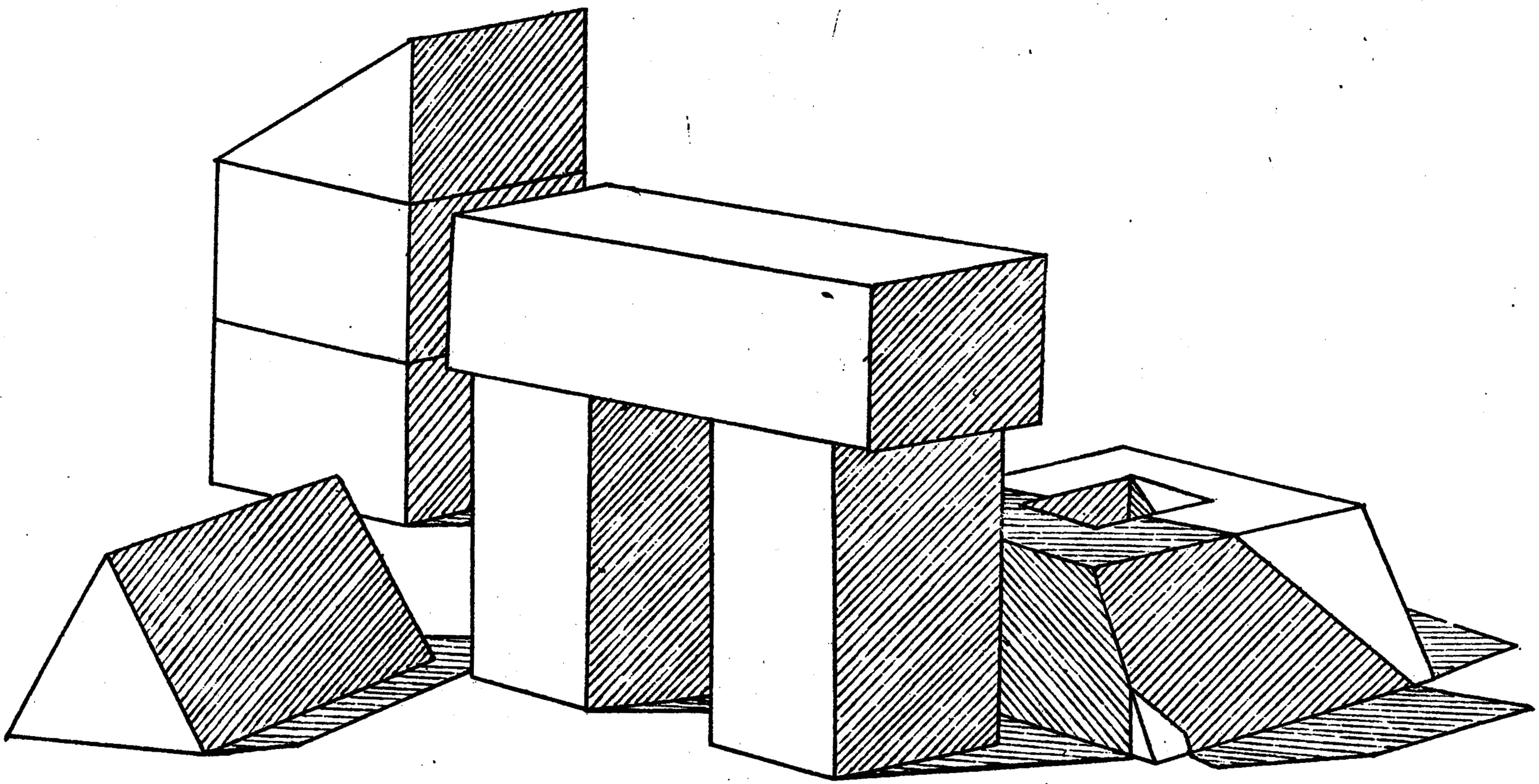


FIGURE 0.1

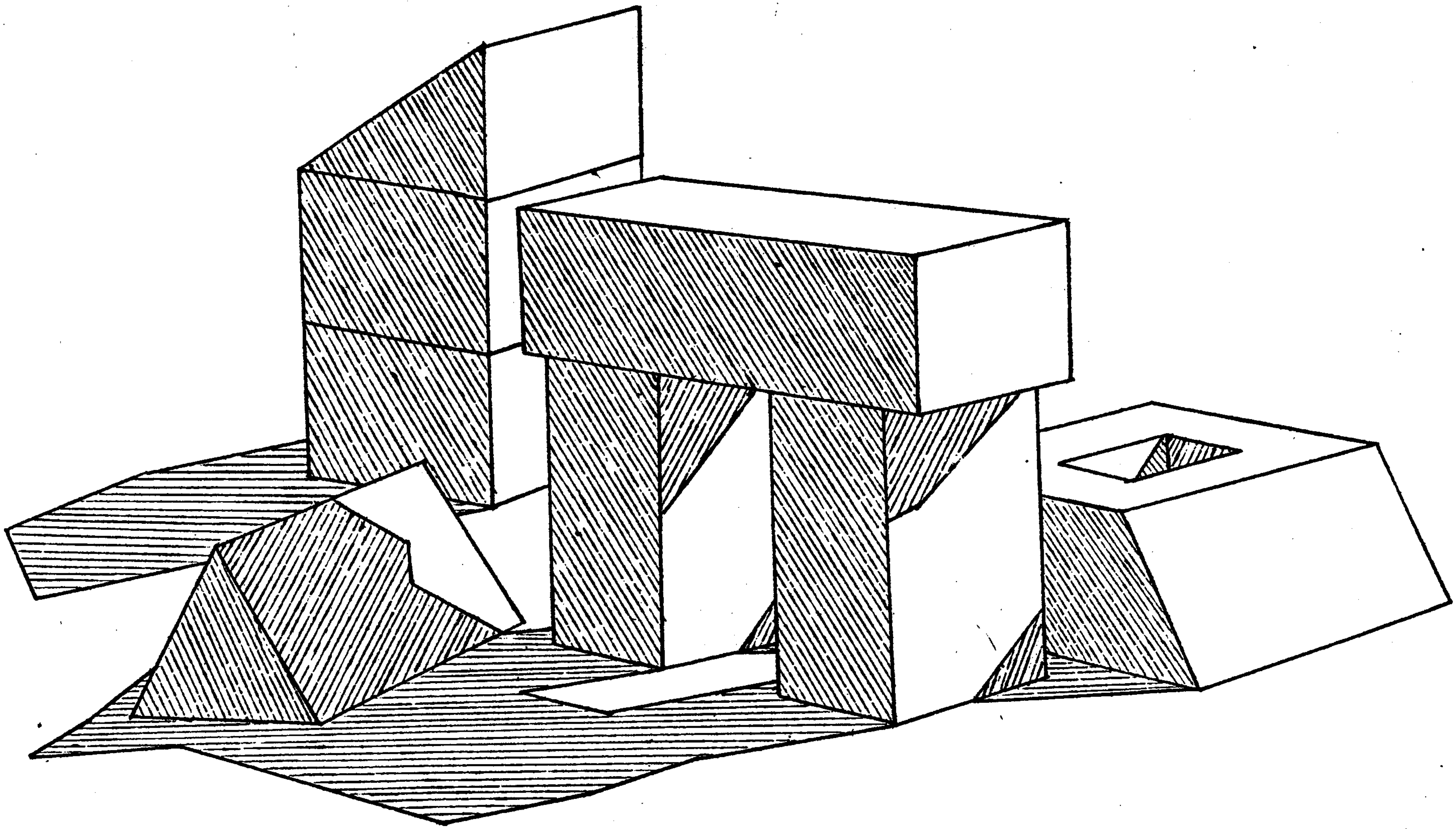


FIGURE 0.2

objects when we recognize them in scenes. But this does not explain how we can perceive the shapes of objects we have never seen before. Similarly, the fact that we can tell the shapes of many objects from as simple a representation as a line drawing precludes the possibility that we need texture or other fine details to ascertain shape, though we may of course use texture gradients and other details to define certain lines.

I undertook this research with the belief that it is possible to discover rules which will allow a program to obtain a three-dimensional model of a scene, given only a reasonably good line drawing of a scene. It seems to me that while the use of range finders, multiple light sources to help eliminate shadows, and the restriction of scenes to known objects may all prove useful for practical robots, these approaches avoid coming to grips with the nature of perception and the implicit three-dimensional information in line drawings of real scenes. While I would be very cautious about claiming parallels between the rules in my program and human visual processes, at the very least I have shown that it may be possible to write capable vision programs which use only an "eye" of some sort. This means that research can be concentrated on areas which may have direct implications on our understanding of human perception.

These are some of the issues on which I hope to shed light in this paper. In it I describe a system which assigns three-dimensional descriptions to lines and regions in line drawings which are obtained from scenes composed of plane-faced objects under various lighting conditions. This system can then identify shadow lines and regions, group regions which belong to the same object, find relations such as support and in-front-of/behind between objects, and provide information about the spacial orientation of various regions, all using the description it has generated.

0.1 DESCRIPTIONS

The overall goal of the system is to provide a precise description of the scene which gave rise to a particular line drawing. It is therefore important to have a good language in which to describe features of scenes. Since I wish to have the program operate on unfamiliar objects, the language I use must be capable of describing such objects. The language I have developed is an expansion of the labels invented independently by Huffman (7) and Clowes (1).

The language consists of labels which are assigned to line segments and regions in the scene. These labels describe the edge geometry, the connection or lack of connection between adjacent regions, the orientation of each region in three dimensions, and the nature of the illuminaton for each region (illuminated, projected shadow region, or region facing away from the light source). The goal of the program is to assign a single label value to each line and region in the line drawing, except in cases where humans also find a feature to be ambiguous.

This language allows precise definitions of such concepts as supports, supported by, in front of, behind, rests against, shadows, is shadowed by, is capable of supporting, leans on, and others. Thus, if it is possible to label each feature of a scene uniquely, then it is possible to directly extract these relations from the description of the scene provided by this labeling.

0.2 JUNCTION LABELS

The basic data of the program are lists of possible line label assignments for each type of junction in a line drawing. While a natural language analogy to these labels could be misleading, I think that it helps in explaining the basic operation of this portion of the program.

> If we think of each possible label for a line as a letter in the alphabet, then each junction must be labeled with an ordered list of "letters" to form a legal "word" in the language. Furthermore, each "word" must match the "words" for surrounding junctions in order to form a legal "phrase", and all "phrases" in the scene must agree to form a legal

"sentence" for the entire scene. The knowledge of the system is contained in (1) a dictionary containing every legal "word" for each type of junction, and (2) rules by which "words" can legally combine with other "words". The range of the dictionary entries defines the universe of the program; this universe can be expanded by adding new entries systematically to the dictionary.

In fact the "dictionary" is not necessarily a stored list. The dictionary can consist of a relatively small list of possible edge geometries for each junction type, and a set of rules which generate the complete dictionary from the original lists. Depending on the amount of computer memory available, it may either be desireable to store the complete lists as compiled knowledge or to generate the lists when they are needed. In my current program the lists are partially precompiled.

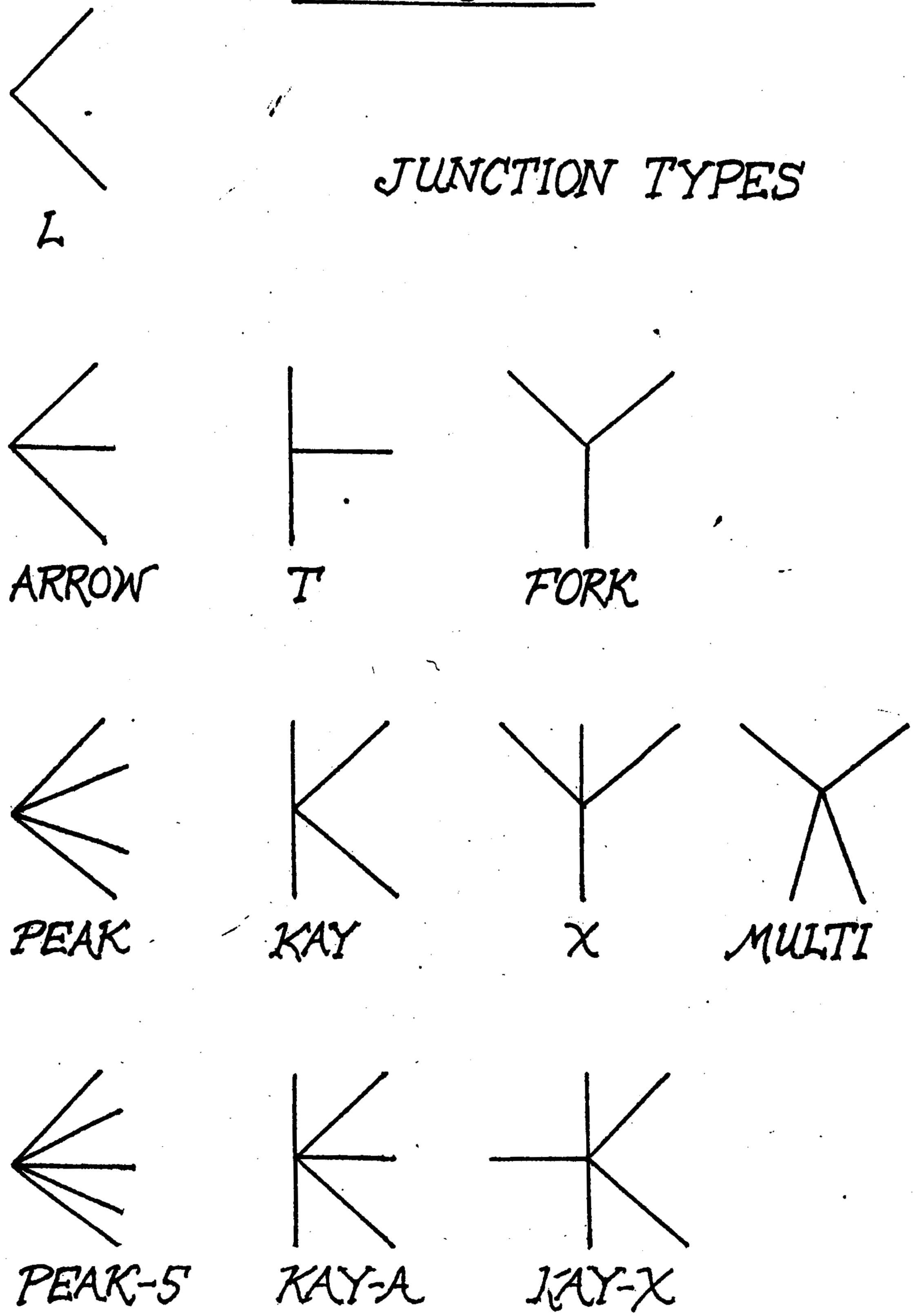
The composition of the dictionary is interesting in its own right. While some basic edge geometries give rise to many dictionary entries, some give rise to very few. The total number of entries sharing the same edge geometry can be as low as three for some ARROW junctions including shadow edges, while the number generated by some FORK junction edge geometries is over 270,000!

0.3 JUNCTION LABEL ASSIGNMENT

There is a considerable amount of local information which can be used to select a subset of the total number of dictionary entries which are appropriate for a particular junction. The first piece of information I have already included implicitly in the idea of junction type. Junctions are typed according to the number of lines which make up the junction and the two dimensional arrangement of these lines. In figure 0.3 I show all the junction types which can occur in the universe of the program. The dictionary is arranged by junction type, and a standard ordering is assigned to all the line segments which make up junctions (except FORKS and MULTIS).

We can also use local region brightness and line segment direction to preclude the assignment of certain labels to lines. For example, if we know that one region is brighter than an adjacent region, then the line which separates the regions can be labeled as a shadow region in only one way. There are other rules which relate region orientation, light placement and region illumination as well as rules which limit the number of labels which can be assigned to line segments which border the support surface for the scene. The program is able to combine all these types of information in finding a list of appropriate labels for a single junction.

FIGURE 0.3


L
JUNCTION TYPES
ARROW
T
FORK
PEAK
KAY
X
MULTI
PEAK-5
KAY-A
KAY-X

0.4 COMBINATION RULES

Combination rules are used to select from the initial assignments the label or labels which correctly describe the scene features that could have produced each junction in the given line drawing. The simplest type of combination rule merely states that a label is a possible description for a junction if and only if there is at least one label which "matches" it assigned to each adjacent junction. Two labels "match" if and only if the interpretation assigned by one junction label to the line segment which joins the two junctions is the same as the interpretation assigned to the line segment by the other junction label.

Of course each interpretation (line label) is really a shorthand code for a number of values assigned to the line and its adjoining regions. If we can show that any one of these constituent values cannot occur in the given scene context, then the whole complex of values for that line expressed implicitly in the interpretation cannot be possible either, and furthermore any junction label which assigns this interpretation to the line segment can be eliminated as well. Thus when we choose a label to describe a particular junction we constrain all the junctions which touch the regions surrounding this junction, even though the combination rules only compare adjacent junctions.

More complicated rules are needed if it is necessary to relate junctions which do not share a visible region or line segment. For example, I thought at the outset of my work that it might be necessary to construct models of hidden vertices or features which faced away from the eye in order to find unique labels for the visible features. The difficulty in this is that unless we know which lines represent obscuring edges, we do not know where to construct hidden features, but if we need the hidden features to label the lines, we may not be able to decide which lines represent obscuring edges. As it turned out, no such complicated rules and constructions are necessary in general; most of the labeling problem can be solved by a scheme which only compares adjacent junctions.

0.5 EXPERIMENTAL RESULTS

When I began to write a program to implement the system I had devised, I expected to use a tree search system to find which labels or "words" could be assigned to each junction. However, the number of dictionary entries for each type of junction is very high, (there are almost 3000 different ways to label a FORK junction before even considering the possible region orientations!) so I decided to use a sort of filtering program before doing a full tree search.

The filtering program computes the full list of dictionary entries for each junction in the scene, eliminates from the list those labels which can be precluded on the basis of local features, assigns each reduced list to its junction, and then computes the possible labels for each line, using the fact that a line label is possible only if there is at least one junction label at each end of the line which contains the line label. This list is the intersection of the two lists of possibilities computed from the junction labels at the ends of the line segment. If any junction label would assign a interpretation to the line segment which is not in this intersection list, then that label can be eliminated from consideration. The filtering program uses a network iteration scheme to systematically remove all the interpretations which are precluded by the elimination of labels at a particular junction.

When I ran this filtering program I was amazed to find that in the first few scenes I tried, this program found a unique label for each line. Even when I tried considerably more complicated scenes, there were only a few lines in general which were not uniquely specified, and some of these were essentially ambiguous, i.e. I could not decide exactly what sort of edge gave rise to the line segment myself. The other ambiguities, i.e. the ones which I could resolve myself, in general require that the program recognize lines which are parallel or collinear or regions which meet along more than one line segment, and hence require more global agreement.

I have been able to use this system to investigate a large number of line drawings, including ones with missing lines and ones with numerous accidentally aligned junctions. From these investigations I can say with some certainty which types of scene features can be handled by the filtering program and which require more complicated processing. Whether or not more processing is required, the filtering system provides a computationally cheap method for acquiring a great deal of information. For example, in most scenes a large percentage of the line segments are unambiguously labeled, and more complicated processing can be directed to the areas which remain ambiguous. As another example, if we only wish to know which lines are shadows or which lines are the outside edges of objects or how many objects there are in the scene, we may be able to get this information even though some ambiguities remain, since the ambiguity may only involve region illumination type or region orientation.

0.6 COMPARISON WITH OTHER VISION PROGRAMS

My system differs from previously proposed ones in several important ways:

First, it is able to handle a much broader range of scene types than have previous programs. The program "understands" shadows and apparent alignment of edges caused by the particular placement of the eye with respect to the scene, so that no special effort needs to be made to avoid problematic features.

Second, the design of the program facilitates its integration with line-finding programs and higher-level programs such as programs which deal with natural language or overall system goals. The system can be used to write a program which automatically requests and applies many different types of information to find the possibilities for a single feature or portion of a scene.

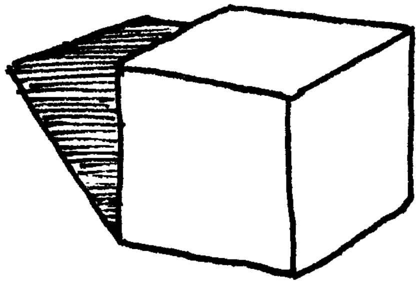
Third, the program is able to deal with ambiguity in a natural manner. Some features in a scene can be ambiguous to a person looking at the same scene and the program is able to preserve the the various possiblities. This tolerance of ambiguity is central to the philosophy of the program; rather than trying to pick the "most probable" interpretation of any features, the program operates by trying to eliminate impossible interpretations. If it has been given insufficient information to decide on a unique possibility, then it preserves all the active possibilities it knows. Of course if a single interpretation is required for some reason, one can be chosen from this list by heuristic rules.

Fourth, the program is algorithmic and does not require facilities for back-up if the filter program finds an adequate description. Heuristics have been used in all previous vision programs to approximate reality with the most likely interpretation, thereby simplifying the description of reality, but requiring sophisticated programs to patch up the cases where the approximation is wrong; in my program I have used as complete a description as I could devise with the result that the programs are particularly simple, transparent and powerful.

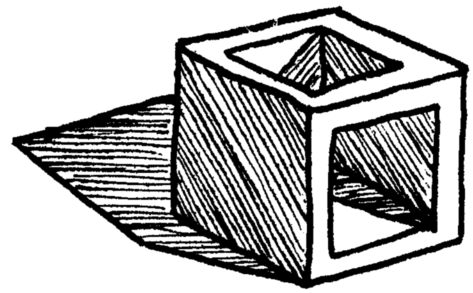
Fifth, because of this simplicity, I have been able to write a program which operates very rapidly. As a practical matter this is very useful for debugging the system, and allows modifications to be made with relative ease. Moreover, because of its speed, I have been able to test the program on many separate line drawings and have thus been able to gain a clearer understanding of the capabilities and ultimate limitations of the program. In turn, this understanding has led and should continue to lead to useful modifications and a greater understanding of the nature and complexity of procedures necessary to handle various types of scene features.

Sixth, as explained in the next section, the descriptive language provides a theoretical foundation of considerable value in explaining previous work.

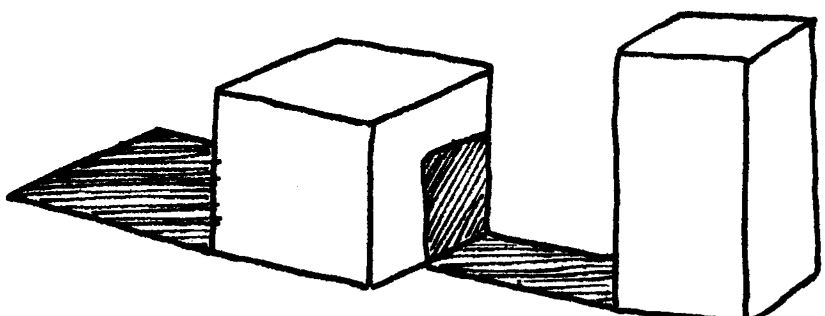
In figure 0.4 I show some of the line drawings for which the system produces unique labelings. In figure 0.5 the ambiguous line segments are marked by thicker lines, and all others are unambiguous. At this writing the system does not use line segment direction or region orientation, except that the program distinguishes between the table and all other regions. The time the program required to completely label each line drawing is noted below it.



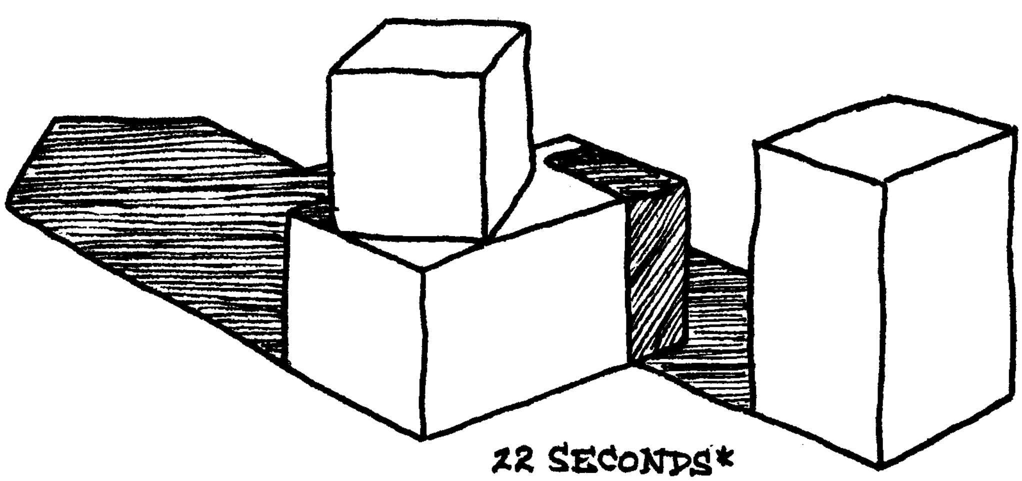
5 SECONDS*



15 SECONDS*



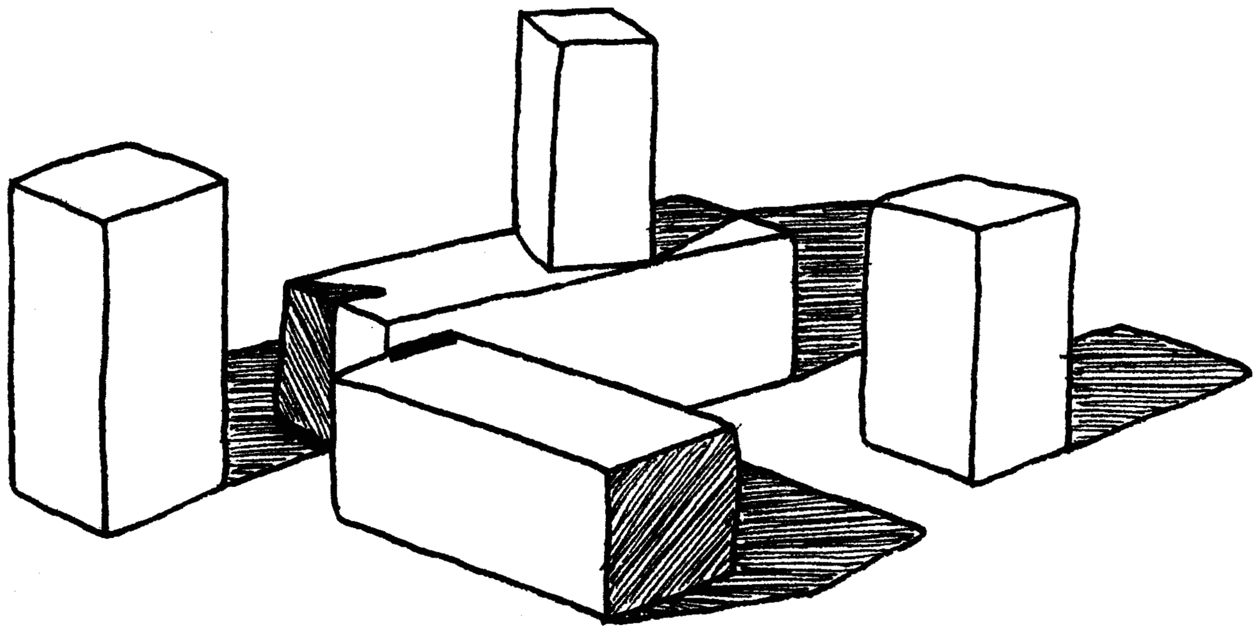
15 SECONDS*



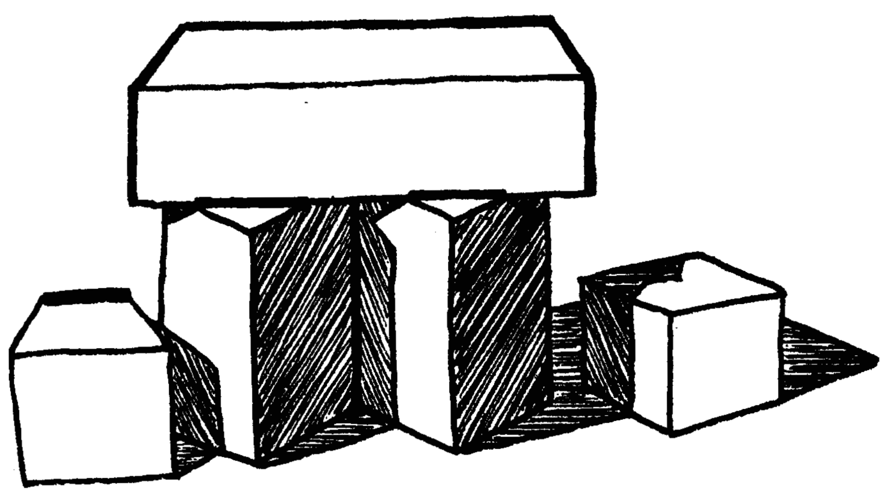
22 SECONDS*

FIGURE 0.4

*RUN ON PDP-10, PROGRAM PART MICRO-PLANNER, PART COMPILED LISP CODE.



39 seconds



48 seconds

FIGURE 0.5

0.7 HISTORICAL PERSPECTIVE

One of the great values of the extensive descriptive apparatus I have developed is its ability to explain the nature and shortcomings of past work. I will show in the body of the paper how my system can be used to clarify the programs of Guzman (6), Rattner (10), Huffman (7) and Clowes (1), Orban (9), and to explain portions of the work of Winston (13) and Finin (4,5). For example, I show how various concepts such as "support" and "skeleton" can be formalized in my descriptive language. From this historical comparison emerges a striking demonstration of the ability of good descriptions to both broaden the range of applicability of a program, and simplify the program structure.

0.8 IMPLICATIONS FOR HUMAN PERCEPTION

My belief that the rules which govern the interpretation of a line drawing should be simple is based on the subjective impression that little abstaction or processing of any type seems to be required for me to be able to recognize the shadows, object edges, etc. in such a drawing, in cases where the drawing is reasonably simple and complete. While introspection has been and should remain suspect in judging the validity of such impressions, there is no other source for judgements on which course of investigation is most likely to prove successful. I do not believe that human perceptual processes necessarily resemble the processes in my program, but there are various aspects of my solution which appeal to my intuition about the nature of perception, i.e. that portion of the problem which is independent of the type of perceiver. I think it is significant that my program is as simple as it is , and that the information stored in it is independent of particular objects. The fact that back-up is not necessary in general, the fact that the system works for picture fragments as well as for entire scenes, the fact that the processing time required is proportional to the number of line segments and not an exponential function of the number, all lead me to believe that my research has been in the right directions.

Clearly there are considerable obstacles to be overcome in extending this work to general scenes. For simple curved objects such as cylinders, spheres, cones, and conic sections, there should be no particular problem in using the type of program I have written. I also believe that it will be possible to handle somewhat more general scenes by approximating the objects in them by simplified "envelopes" which preserve the gross form of the objects but which can be described in terms like those I have used. Before this can be done successfully, the problem of reconstructing the invisible portions of the scene must be solved in my estimation. The solution to this problem is intimately connected with the problem of using the stored description of an object to guide the search for instances of this object, or similar objects in a scene. Furthermore, I believe that the ability to label a line drawing in the manner I describe greatly simplifies the specification and solution of these problems. At the end of this paper I will discuss these problems and other directions in which I believe that vision research can profitably expand.

1.0 QUICK SYNOPSIS

This portion of the paper was written separately and is included with the introduction to my thesis in order to provide a brief picture of my work. Some of the material in this chapter is a little outdated, but I decided not to revise the chapter so that I could produce this vision flash as rapidly as possible.

1.1 THE PROBLEM

In order not to confuse you, let me make some distinctions between the scene itself (objects, table, and shadows) and the retinal representation of the scene as a two-dimensional line drawing. I will use the terms vertex, edge and surface to refer to the scene features which map into junction, line and region respectively in the line drawing.

Therefore the first subproblem is to develop a language that allows us to describe the scene itself. I have done this by assigning names called labels to lines in the line drawing, after the manner of Huffman (7) and Clowes (1). Thus, for example, in figure 1.1 line segment J1-J2 is labeled as a shadow edge, line J2-J3 is labeled as a concave edge, line J3-J14 is labeled as a convex edge, line J4-J5 is labeled as an obscuring edge and line J12-J13 is labeled as a crack edge.

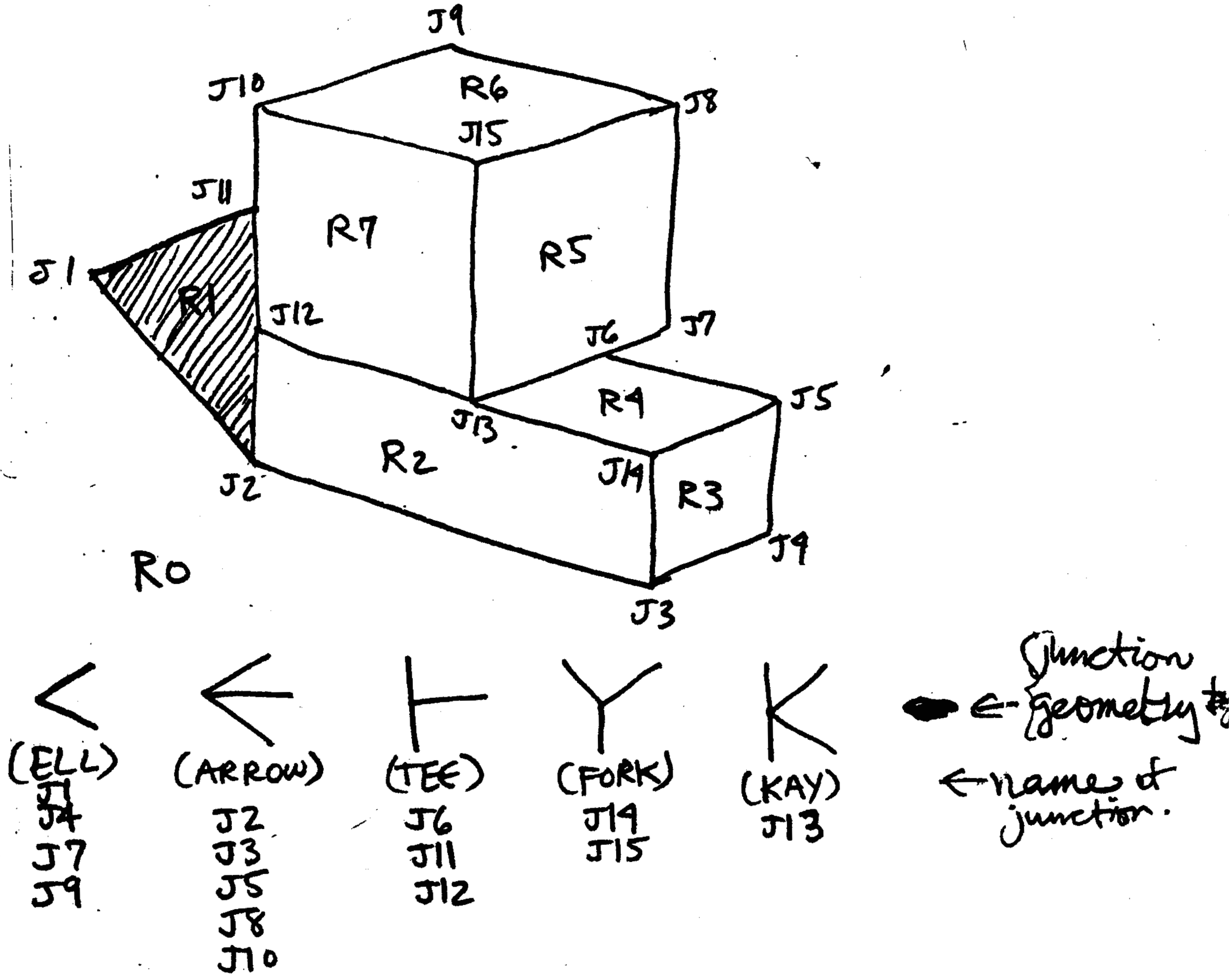

J9
J10
R6
J8
J15
J11
R7
R5
J1
R1
J12
J6
J7
R4
J5
J13
R2
J14
R3
J2
J4
R0
J3
(ELL)
J1
J4
J7
J9
(ARROW)
J2
J3
J5
J8
J10
(TEE)
J6
J11
J12
(FORK)
J14
J15
(KAY)
J13
← Junction geometry
← name of junction.

FIGURE 1.1

When we look at a line drawing of this sort, in general we can easily understand what the line drawing represents. In terms of the problem statement either (1) we are able to assign labels uniquely to each line, or (2) we can say that no such scene could exist, or (3) we can say that although we cannot decide unambiguously what the label of an edge should be, it must be labeled with one member of some specified subset of the total number of labels. What knowledge is needed to enable the program to reproduce our labeling assignments?

Huffman and Clowes provided a partial answer in their papers. They pointed out that each type of junction can only be labeled in a few ways, and that if we can say with certainty what the label of a particular line is, we can greatly constrain all the lines which intersect the line segment at its ends. As a specific example, if one branch of an L junction is labeled as a shadow edge, then the other branch must be labeled as a shadow edge as well.

Moreover, shadows are directional, i.e. in order to specify a shadow edge, it must not only be labeled "shadow" but must also be marked to indicate which side of the edge is shadowed and which side is illuminated. Therefore, not only the type of edge but the nature of the regions on each side can be constrained.

These facts can be illustrated in a jigsaw puzzle analogy, shown in figure 1.2. Given the five different edge types I have discussed so far, there are seven different ways to label any line segment. This implies that if all line labels were assigned independently there would be $7^2 = 49$ different ways to label an L, $7^3 = 343$ ways to label a three-line junction, etc. In fact there are only 9 ways in which real scene features can map into Ls on a retinal projection. See table 1.1 for a summary of the ways in which junctions can be assigned labels from this set. In figure 1.3, I show all the possible labels for each junction type, limiting myself to vertices which are formed by no more than three planes (trihedral vertices).

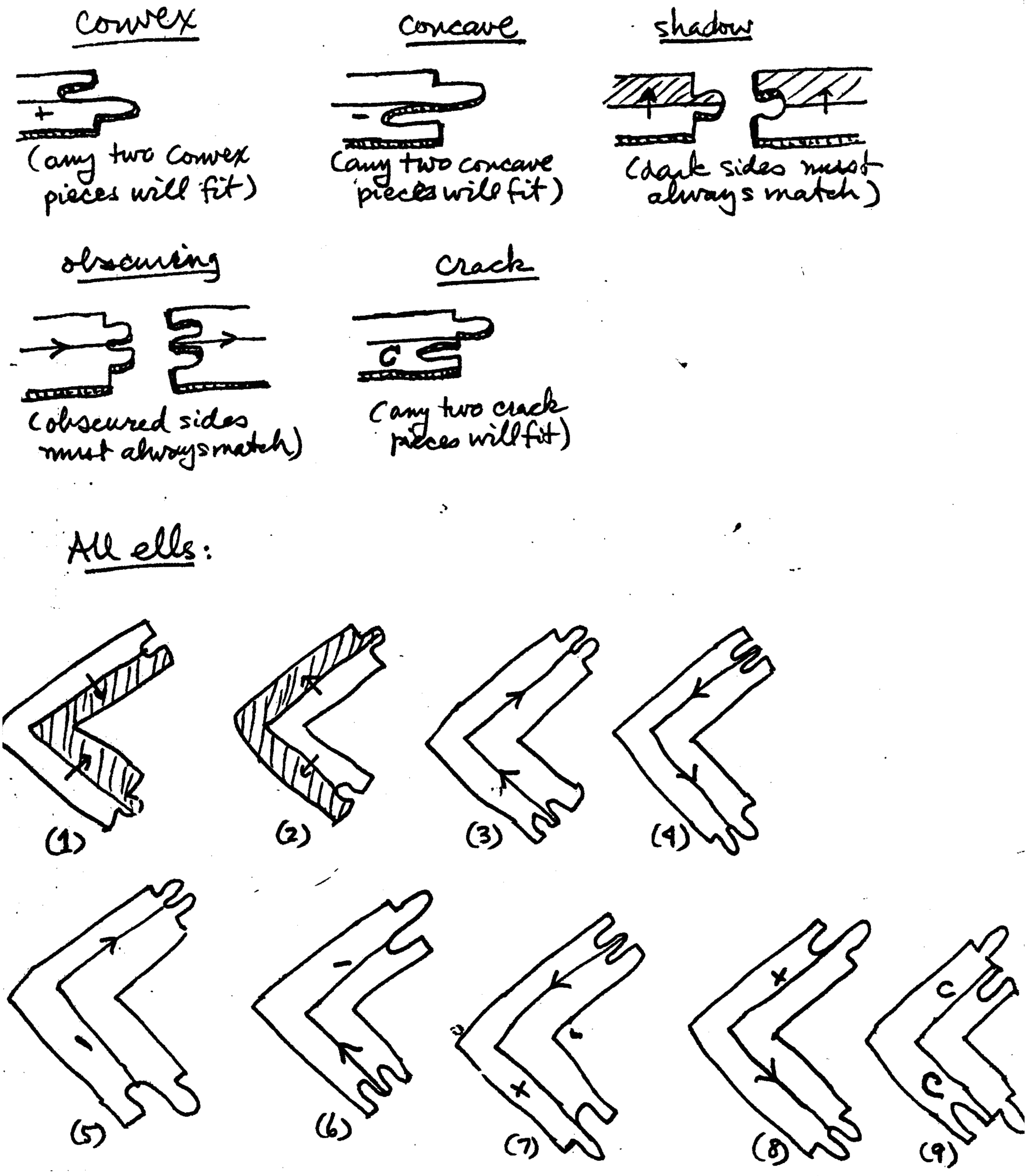
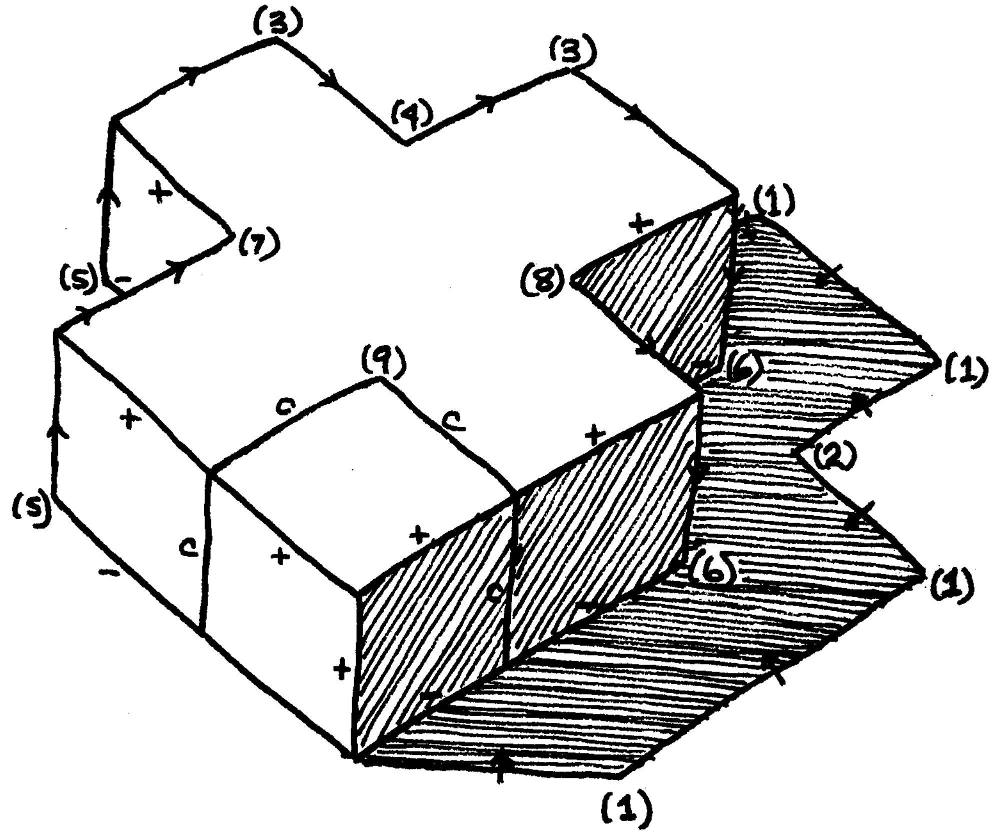


FIGURE 1.2

<PAGE ONE>

[SEE NEXT PAGE FOR EXAMPLES OF EACH "L" TYPE]



THE NUMBERS REFER BACK TO THE 'L' JUNCTIONS ON PAGE 27.

FIGURE 1.2

<PAGE TWO>

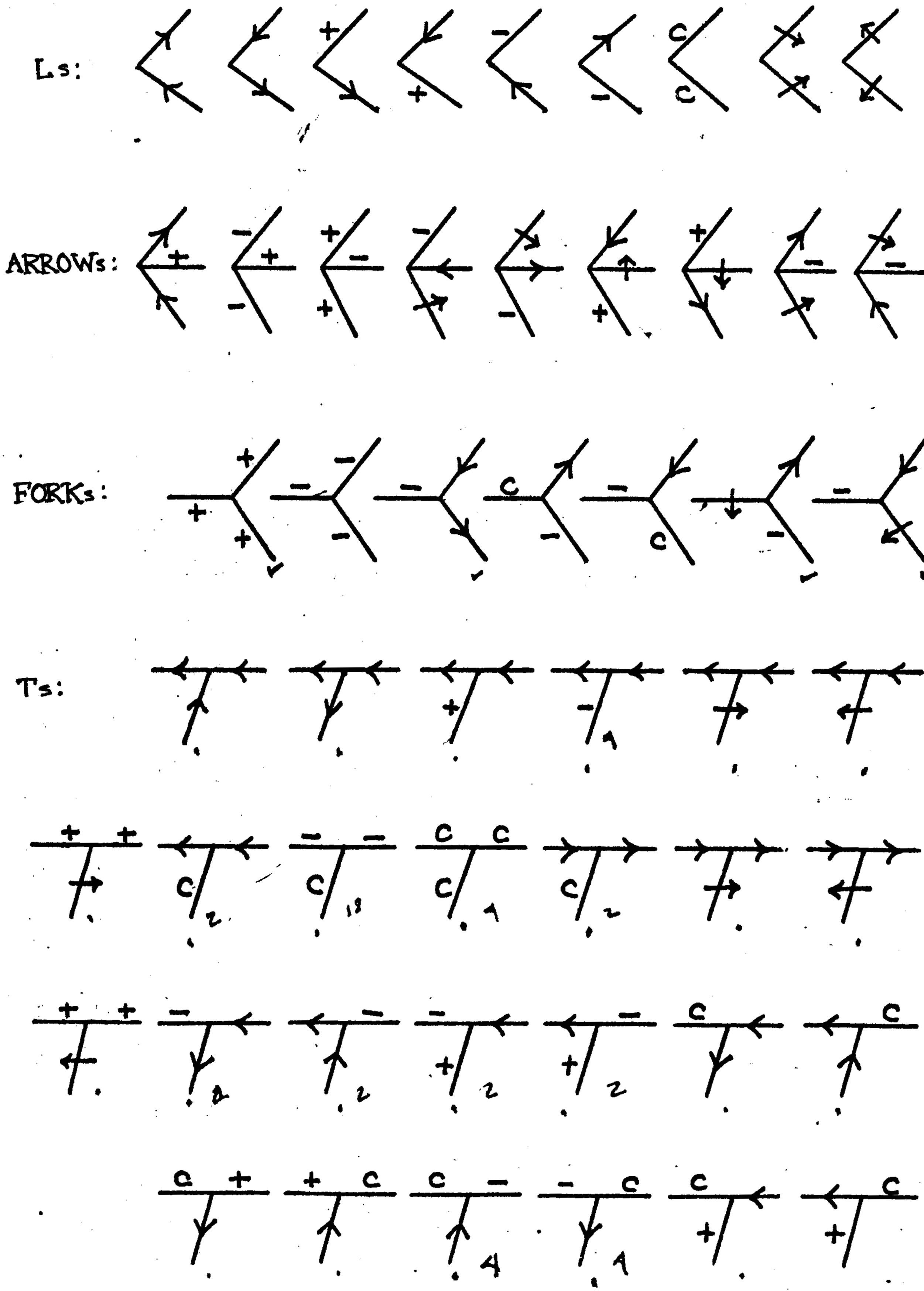
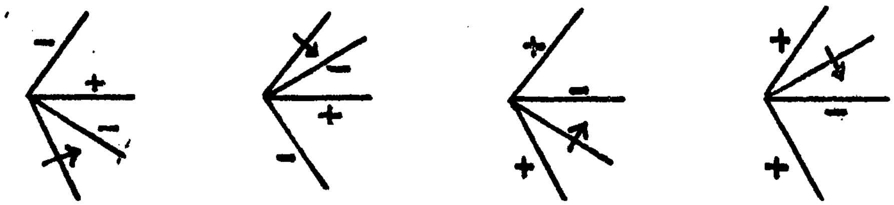
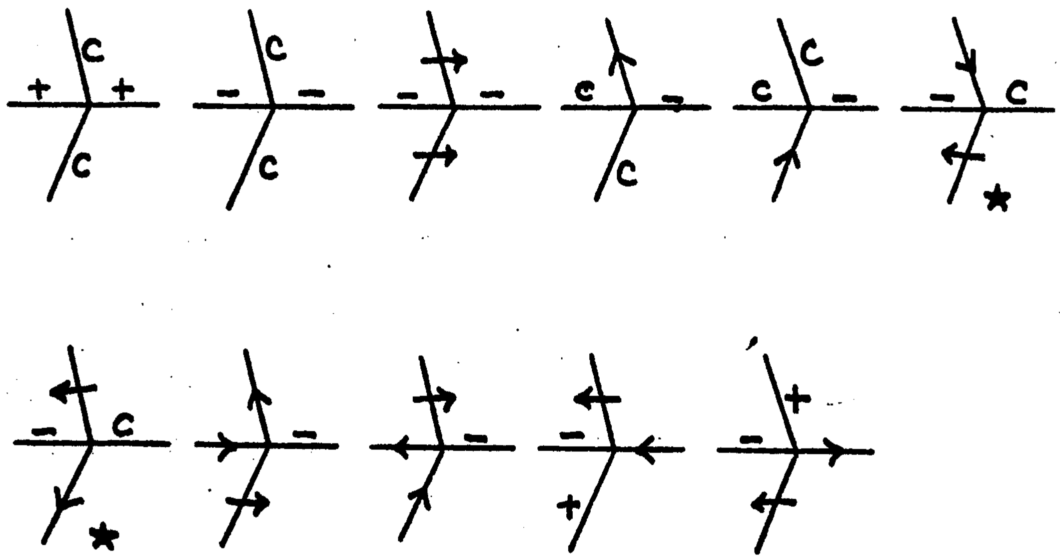


FIGURE 1.3
FIRST PAGE

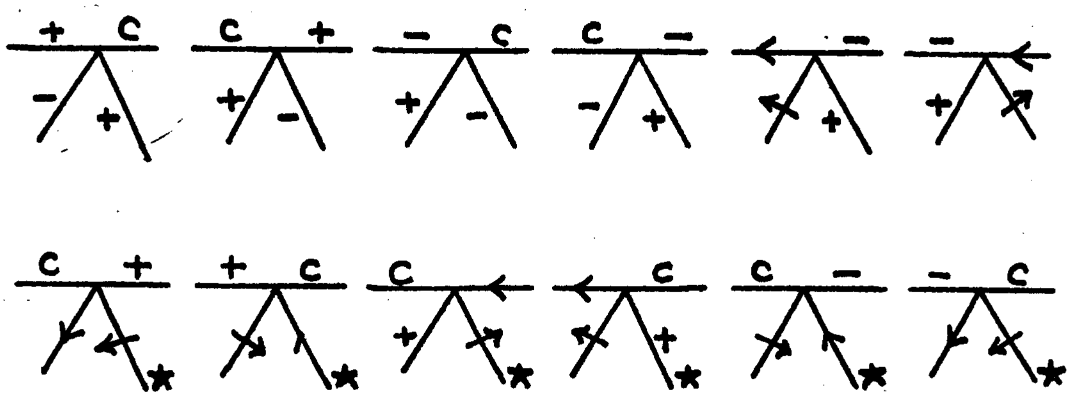
PEAKs:



Xs:



Ks:



Xs:
(SPECIAL)

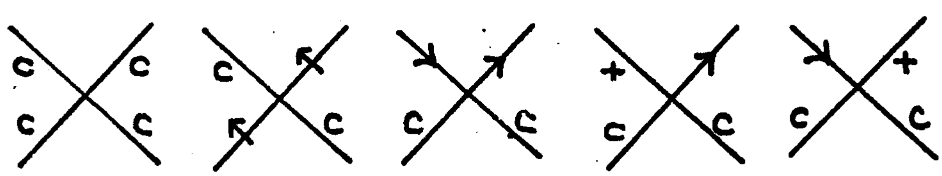
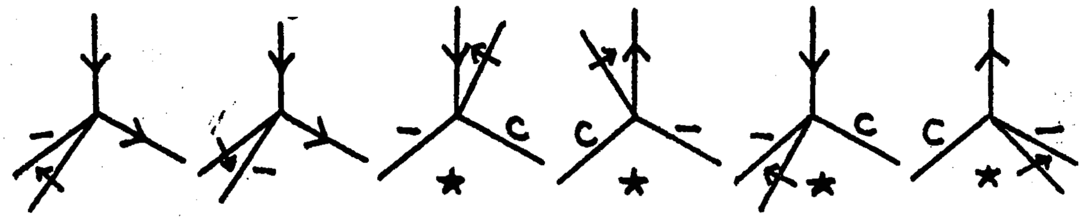
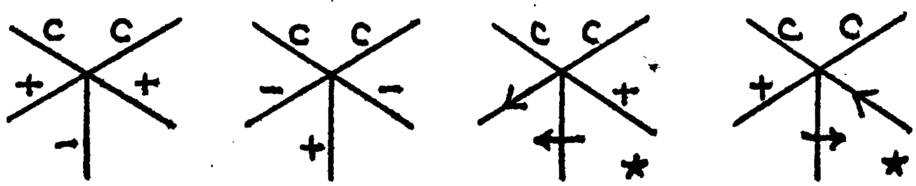


FIGURE 1.3
SECOND PAGE

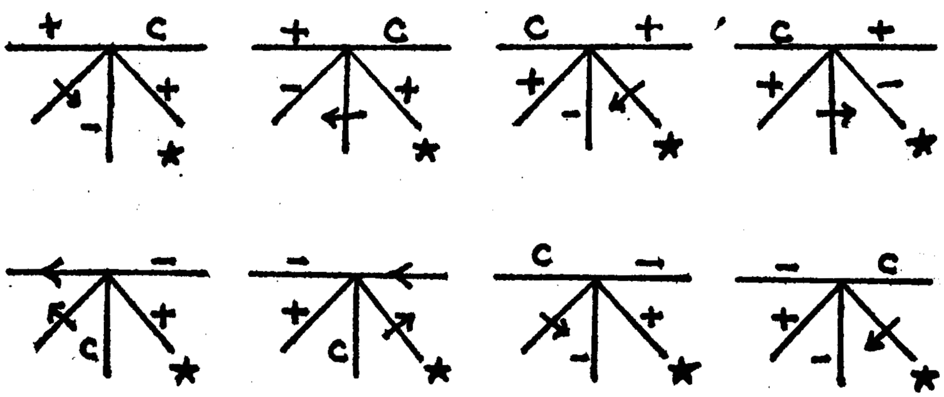
MULTIS:



K-Xs:
(SPECIAL)



K-As:



K-Xs:

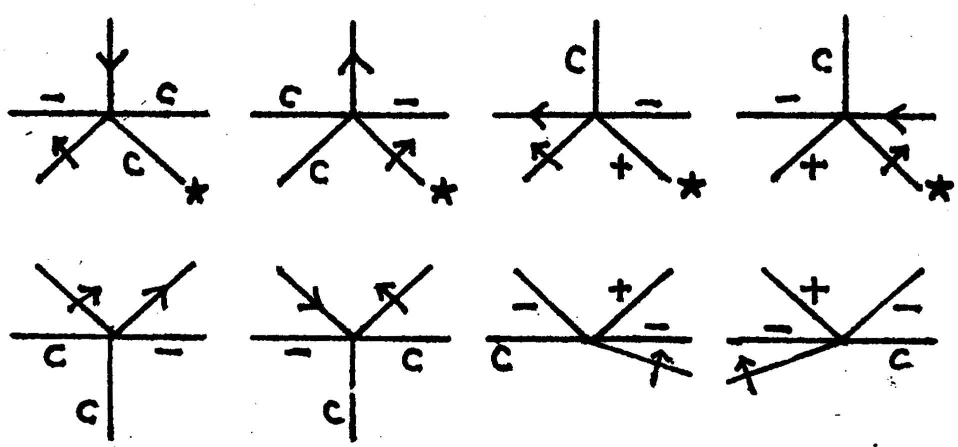


FIGURE 13
THIRD PAGE

[SPECIAL JUNCTIONS CONTAIN TWO COLINEAR RELATIONS]

JUNCTION TYPES	NUMBER OF POSSIBILITIES, LABELING INDEPENDENTLY	ACTUAL NUMBER OF WAYS JUNCTIONS CAN BE LABELED	PERCENTAGE
L	49	9	18.4
ARROW	343	9	2.6
FORK	343	17	5.0
T	343	26	7.6
PEAK	2401	4	0.2
X	2401	37	1.5
K	2401	12	0.5
MULTI	2401	24	1.0
K-A	16807	8	0.05
K-X	16807	12	0.07

TABLE 1.1

1.2 SOLVING THE LABEL ASSIGNMENT PROBLEM

Labels can be assigned to each line segment by a tree search procedure. In terms of the jigsaw puzzle analogy, imagine that we have the following items:

1. A board with channels cut to represent the line drawing; the board space can accept only L pieces at each place where the line drawing has an L, only ARROW pieces where the line drawing has an ARROW, etc. Next to each junction are three bins, marked "junction number", "untried labels", and "tried labels".

2. A full set of pieces for every space on the board. If the line drawing represented by the board has five Ls then there are five full sets of L pieces with nine pieces in each set.

3. A set of junction number tags marked J1, J2, J3, ..., Jn, where n is the number of junctions on the board.

4. A counter which can be set to any number between 1 and n.

The tree search procedure can then be visualized as follows:

Step 1: name each junction by placing a junction number tag in each bin marked "junction number".

Step 2: Place a full set of the appropriate type of pieces in the "untried labels" bin of each junction.

Step 3: Set the counter to 1. From here on in Nc will be used to refer to the current value of the counter. Thus if the counter is set to 6, then J(Nc) = 6.

Step 4: Try to place the top piece from the "untried labels" bin of junction J(Nc) in board space J(Nc). There are several possible outcomes:

A. If the piece can be placed (i.e. the piece matches all adjacent pieces already placed, if any), then

A1. If Nc < n, increase the counter by one and repeat Step 4.

A2. If Nc = n, then the pieces now on the board represent one possible labeling for the line drawing. If this is true then

i. Write down or otherwise remember the labeling, and

ii. Transfer the piece in space n back into the n-th "untried labels" bin, and

iii. go to Step 5.

B. If the piece cannot be placed, put it in the "tried labels" bin and repeat Step 4.

C. If there are no more pieces in the "untried labels" bin, then

C1. If Nc =1, we have found all (if any) possible labelings, and the procedure is DONE.

C2. Otherwise, go to Step 5.

Step 5: Do all the following steps:

i. Transfer all the pieces from the Nc-th "tried labels" bin into the Nc-th "untried labels" bin, and

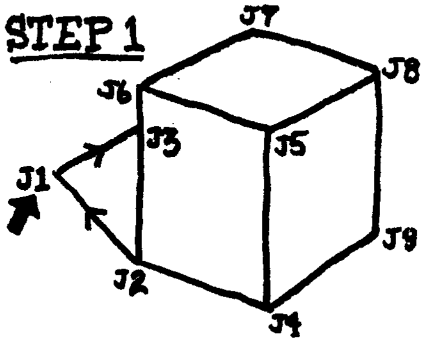
ii. transfer the piece in space Nc - 1 into its "tried labels" bin, and

iii. Set the counter to Nc - 1, and go to Step 4.

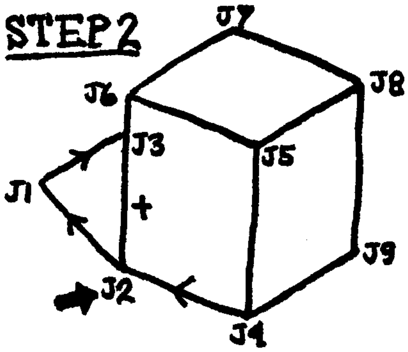
To see how this procedure works in practice, see figure 1.4. For this example I have assumed that the pieces are piled so that the order in which they are tried is the same as the order in which the pieces are listed in figure 1.3. I have only carried out the example to the first labeling obtained by the procedure. There is of course at least one other labeling, namely the one we could assign by inspection. The "false" labeling we find first

FIGURE 14

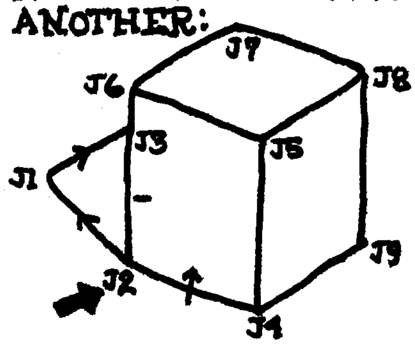
STEP 1



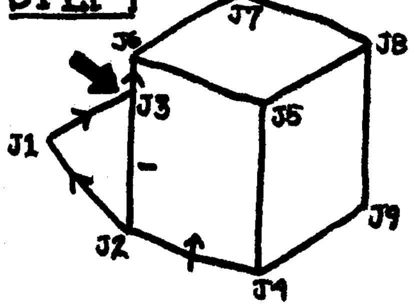
STEP 2



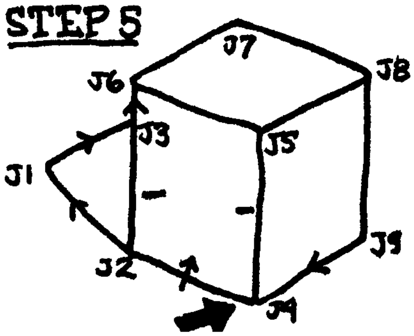
STEP 3: IMPOSSIBLE TO LABEL J3; ∴ REMOVE LABEL FROM J2 & TRY ANOTHER:



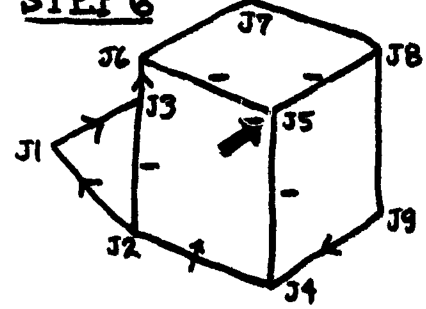
STEP 4



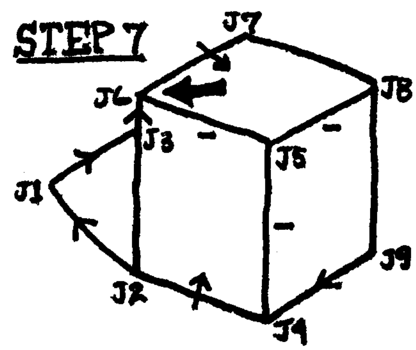
STEP 5



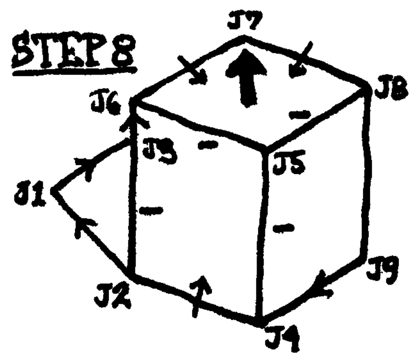
STEP 6



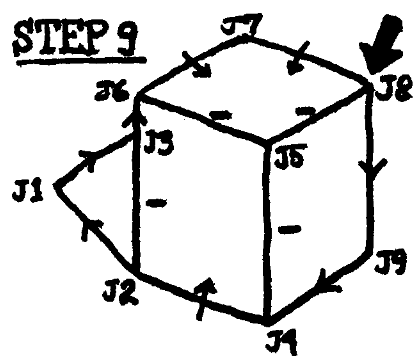
STEP 7



STEP 8



STEP 9



STEP 10

J9 HAS A LEGAL LABEL SO THE RESULT IS A POSSIBLE LABELING. NOTICE THAT IT IS <u>NOT</u> IN FACT A VALID INTERPRETATION; OUR INFORMATION IS TOO LOCAL OR NOT PRECISE ENOUGH.

could be eliminated in this case if we knew that R3 is brighter than R1 or that R2 is brighter than R1. We could then use heuristics which only allow us to fit a shadow edge in one orientation, given the relative illumination on both sides of a line. However, if the object happened to have a darker surface than the table, this heuristic would not be help.

Clearly this procedure leaves many unsolved problems. In general there will be a number of possible labelings from which we must still choose one. What rules can we use to make the choice? Even after choosing a labeling, if we wish to answer questions about the number of objects in the scene, about which edges are shadows, about whether or not any objects support other objects, etc. we must use rules of some sort to deduce the answers from the information we have.

There must be good reasons why we see only one interpretation of a line drawing in most cases. I will argue that what is needed is not a more clever set of rules or theorems to relate various features of the line drawing, but merely a better description of the scene features. In fact it turns out that we can use a parsing procedure which involves less computation than the tree search procedure!

1.3 BETTER EDGE DESCRIPTION

So far I have classed together all edges only on the basis of geometry (concave, convex, obscuring or planar) and have subdivided the planar class into crack and shadow sub-classes. Suppose that I further break down each class according to whether or not each edge can be the bounding edge of an object. Objects can be bounded by obscuring edges, concave edges, and crack edges. In figure 1.5 I show the results of appending a label analogous to the "obscuring edge" mark to crack and concave edges. In addition, the obscuring edge class is divided into subclasses according to whether the edge obscures part of the object to which the edge belongs or whether the edge is an outside edge of an object (see figure 1.6).

I can also label each region as belonging to one of the three following classes:

I - Illuminated directly by the light source.

SP - A projected shadow region; such a region would be illuminated if no object were between it and the light source.

SS - A self-shadowed region; such a region is oriented away from the light source.

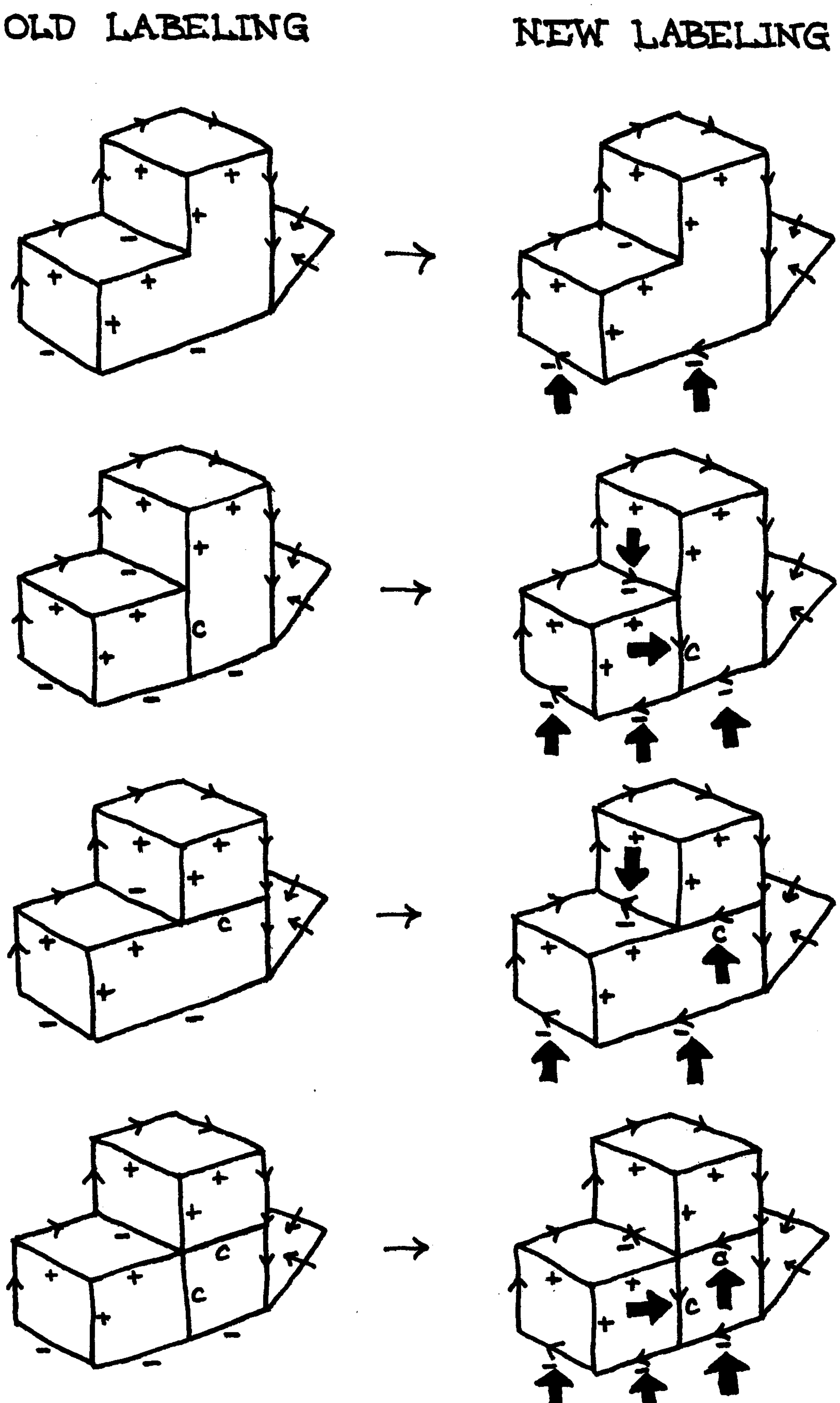
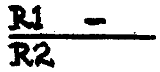
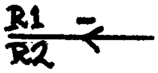
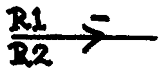


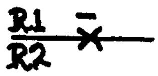
FIGURE 1.5
<FIRST PAGE>

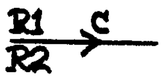
INTERPRETATION:

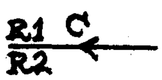

AN INSEPARABLE CONCAVE EDGE; THE OBJECT OF WHICH R1 IS A PART [OB(R1)] IS THE SAME AS THE OBJECT OF WHICH R2 IS A PART [OB(R1) = OB(R2)].


A SEPARABLE CONCAVE EDGE; IF R1 IS ABOVE R2, THEN OB(R2) <u>SUPPORTS</u> OB(R1).

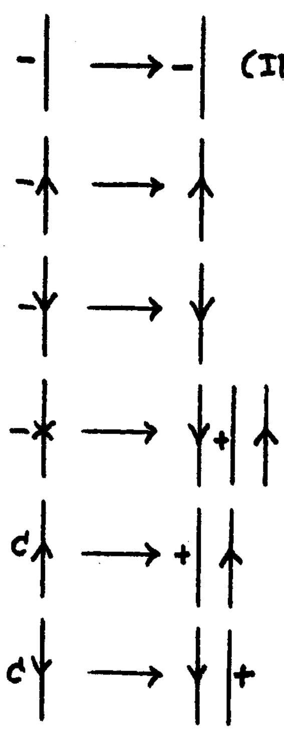

SAME AS ABOVE; IF R1 IS ABOVE R2, THEN EITHER OB(R1) <u>SUPPORTS</u> OB(R2) OR OB(R2) IS <u>IN FRONT OF</u> OB(R1).


A 3-WAY SEPARABLE CONCAVE EDGE; NEITHER OBJECT SUPPORTS THE OTHER.


A CRACK EDGE; OB(R2) IS <u>IN FRONT OF</u> OB(R1) IF R1 IS ABOVE R2.

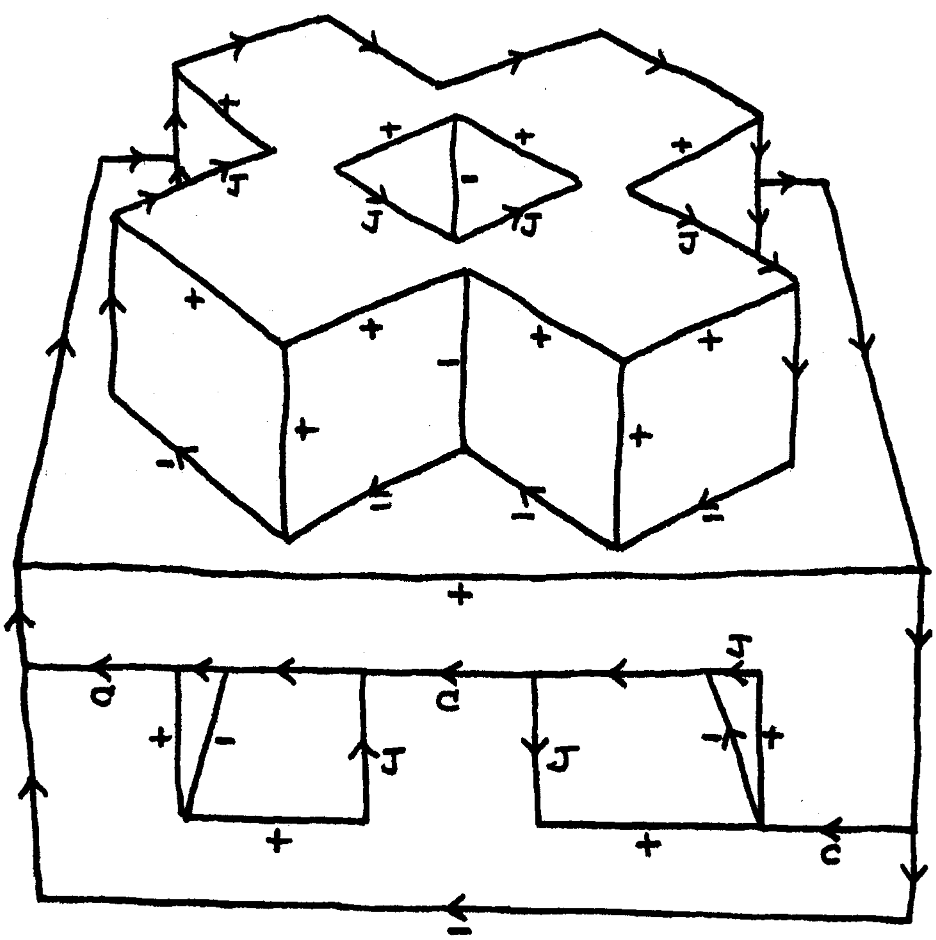

A CRACK EDGE; OB(R2) <u>SUPPORTS</u> OB(R1) IF R1 IS ABOVE R2.

SEPARATIONS:



(INSEPARABLE)

FIGURE 15
<SECOND PAGE>



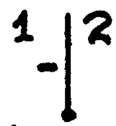
DISTINCTION BETWEEN →— AND →—J .

FIGURE 1.6

Given these classes, I can define new edge labels which also include information about the lighting on both sides of the edge. Notice that in this way I can include at the edge level, a very local level, information which constrains all edges bounding the same two regions. Figure 1.7 is made up of tables which relate the region illumination types which can occur on both sides of each edge type. For example, if either side of a concave or crack edge is illuminated, both sides of the edge must be illuminated.

We can use these tables to expand our set of allowable junction labels; the new set of labels may have a number of entries which have the same edge geometries but which have different region illumination. It is very easy to write a program to expand the set of labelings; the principles of its operation are (1) each region in a given junction labeling can have only one illumination value of the three, and (2) the values on either side of each line of the junction must satisfy the restrictions in the tables of figure 1.7.

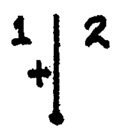
CONCAVE



(ANY OF FOUR-FIG.1.5)

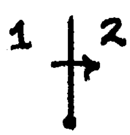
1 \ 2	I	SP	SS
I	YES	NO	NO
SP	NO	YES	YES
SS	NO	YES	YES

CONVEX



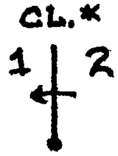
1 \ 2	I	SP	SS
I	YES	NO	YES
SP	NO	YES	YES
SS	YES	YES	YES

SHADOW C.C.*



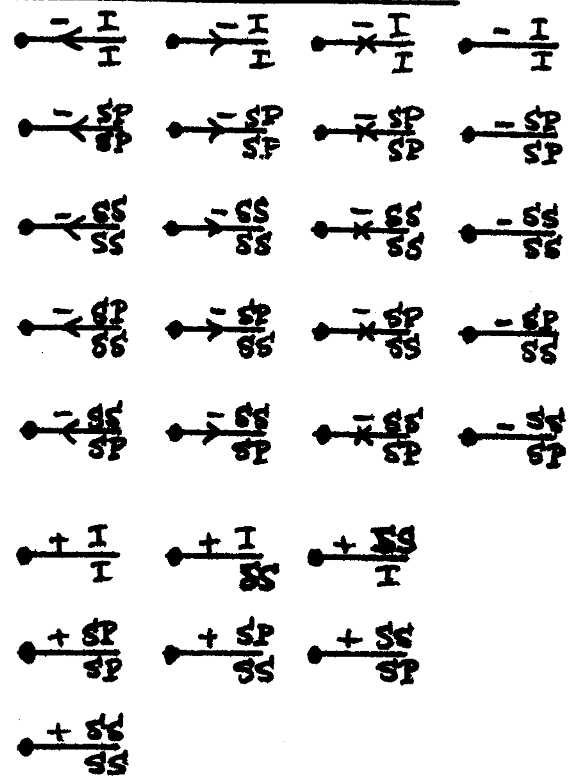
1 \ 2	I	SP	SS
I	NO	YES	NO
SP	NO	NO	NO
SS	NO	NO	NO

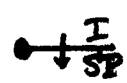
SHADOW CL.*

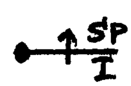


1 \ 2	I	SP	SS
I	NO	NO	NO
SP	YES	NO	NO
SS	NO	NO	NO

SET OF ALLOWABLE LABELS:



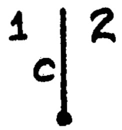




*C.C.= COUNTERCLOCKWISE;
CL.= CLOCKWISE

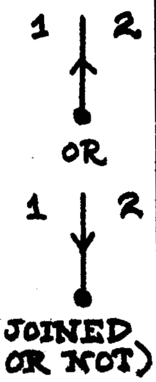
FIGURE 1.7
<FIRST PAGE>

CRACK



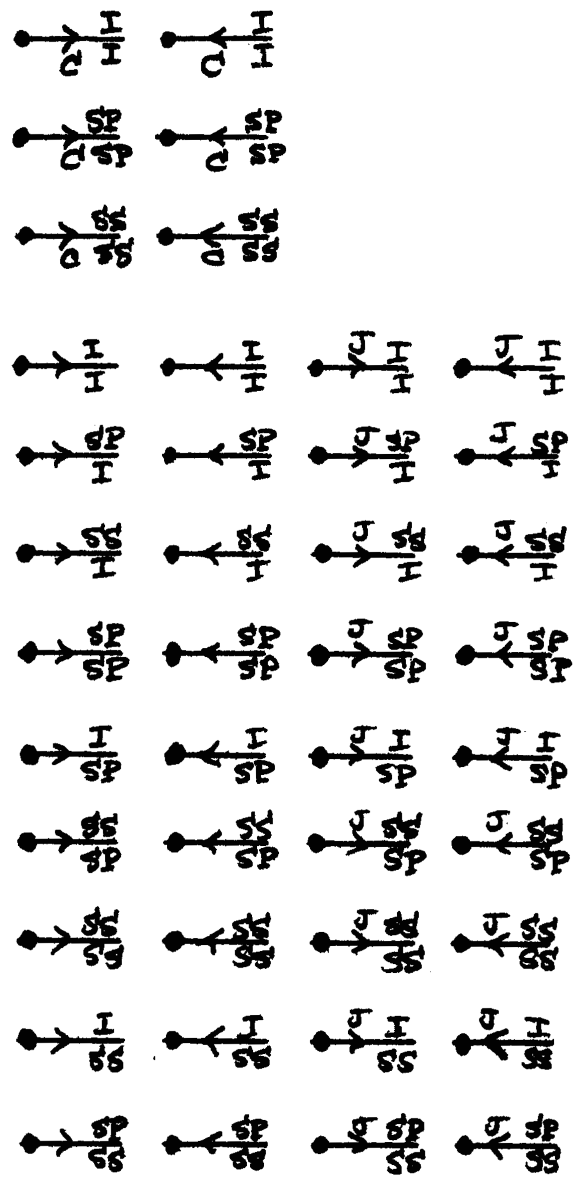
1 \ 2	I	SP	SS
I	<u>YES</u>	NO	NO
SP	NO	<u>YES</u>	NO
SS	NO	NO	<u>YES</u>

OBSCURE



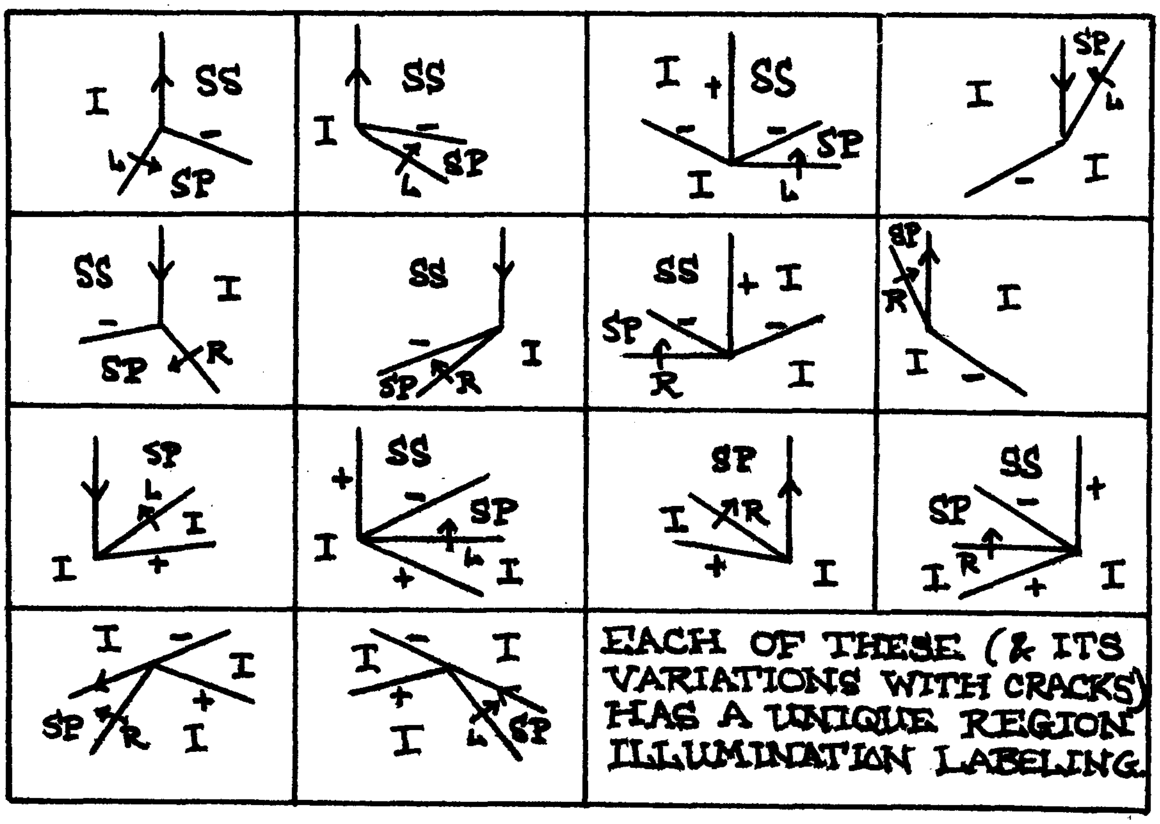
1 \ 2	I	SP	SS
I	<u>YES</u>	<u>YES</u>	<u>YES</u>
SP	<u>YES</u>	<u>YES</u>	<u>YES</u>
SS	<u>YES</u>	<u>YES</u>	<u>YES</u>

<u>SET OF ALLOWABLE LABELS (CON'T):</u>



<u>FIGURE 1.7</u>
<SECOND PAGE>

An interesting result of this further subdivision of the line labels is that, with four exceptions, each shadow causing junction has only one possible illumination parsing, as shown in figure 1.8. Thus whenever a scene has shadows and whenever we can find a shadow causing junction in such a scene, we can greatly constrain all the lines and regions which make up this junction. In figure 1.8 I have also marked each shadow edge which is part of a shadow causing junction with an "L" if the arrow on the shadow edge points counter-clockwise and an "R" if the arrow points clockwise. No "L" shadow edge can match an "R" shadow edge, corresponding to the physical fact that it is impossible for a shadow edge to be caused from both of its ends.


EACH OF THESE (& ITS VARIATIONS WITH CRACKS) HAS A UNIQUE REGION ILLUMINATION LABELING.

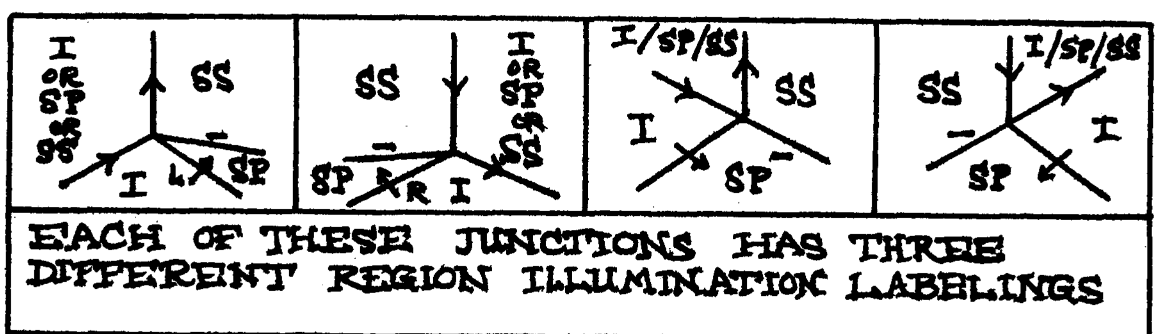

EACH OF THESE JUNCTIONS HAS THREE DIFFERENT REGION ILLUMINATION LABELINGS

FIGURE 1.8

There are two extreme possibilities that this partitioning may have on the number of junction labelings now needed to describe all real vertices:

(1) Each old junction label which has n concave edges, m crack edges, p clockwise shadow edges, q counterclockwise shadow edges, and s obscuring edges and t convex edges will have to be replaced by $(20)^n(6)^m(3)^p(3)^q(18)^s(8)^t$ new junctions.

(2) Each old junction will give rise to only one new junction (as in the shadow causing junction cases).

If (1) were true then the partition would be worthless, since no new information could be gained. If (2) were true we would have greatly improved the situation, since in a sense all the much more precise information was implicitly included in the original junctions but was not explicitly stated. Because the information is now more explicitly stated, many matches between junctions can be precluded; for example, if in the old scheme some line segment L1 of junction label Q1 could have been labeled concave, as could line segment L2 of junction label Q2, a line joining these two junctions could have been labeled concave. But in the new scheme, if each junction label gives rise to a single new label, both L1 and L2 would take on one of the twenty possible values for a concave edge. Unless both L1 and L2 gave rise to the same new label, the line segment could not be labeled concave using Q1 and Q2. In fact the truth lies somewhere between the two extremes, but the fact that it is not at the extreme of (1) means that we have a net improvement. In table 1.2 I compare the situation now to cases (1) and (2) above and also to the situation depicted in table 1.1.

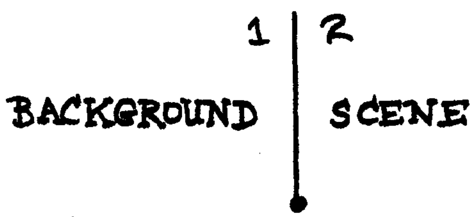
JUNCTION TYPE	NUMBER OF POSSIBILITIES IF EACH BRANCH WERE LABELED AS INDEPENDENT	NUMBER IF OLD JUNCTIONS EXPANDED INDEPENDENTLY $=\sum(20^{n_i} \cdot 6^{m_i} \cdot 3^{p_i+q_i} \cdot 8^{r_i} \cdot 18^{s_i})$*	OLD NUMBER OF JUNCTIONS	ACTUAL NUMBER OF NEW JUNCTIONS (APPROX.)	OLD PERCENTAGE	NEW PERCENTAGE
		CASE (1)	CASE (2)		(%)	(%)
L	4624	1710	9	80	18.4	1.7
ARROW	314,432	12352	9	73	2.6	0.02
FORK	314,432	47382	17	~500	5.0	0.16
T	314,432	61896	26	~1000	7.6	0.32
PEAK	21,381,376	27280	4	8	0.2	$\sim 4\times10^{-5}$
X	21,381,376	?	37	414	1.5	$\sim 2\times10^{-3}$
MULTI	21,381,376	?	24	96	1.0	$\sim 5\times10^{-4}$
K	21,381,376	?	12	~100	0.5	$\sim 5\times10^{-4}$
K-A	1.45×10^{9}	?	8	30	0.05	$\sim 2\times10^{-6}$
K-X	1.45×10^{9}	?	12	38	0.07	$\sim 2.5\times10^{-6}$

TABLE 1.2

*SEE TEXT, PAGE 46

I have also used the better descriptions to express the restriction that each scene is assumed to be on a horizontal table which has no holes in it, and which is large enough to fill the retina. This means that any line segment which separates the background (table) from the rest of the scene can only be labeled as shown in figure 1.9. Because of this fact I can greatly restrict the number of junction labels which could be used to label junctions on the scene/background boundary.

The value of a better description should be immediately apparent. In the old classification scheme three out of the seven line labels could appear on the scene/background boundary, whereas in the new classification, only seven out of fifty labels can occur. Moreover, since each junction must have two of its line segments bounding any region, the fraction of junctions which can be on the scene/background boundary has improved roughly from (3/7)(3/7) = 9/49 to (7/50)(7/50) = 49/2500, which is less than one in fifty. The results of these improvements will become obvious in the next section.



CAN ONLY BE LABELED IN ONE OF THE FOLLOWING WAYS:

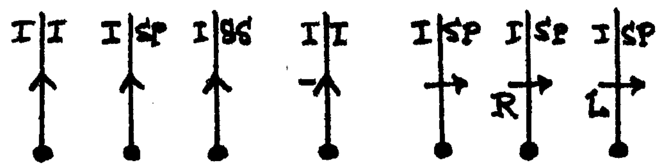


FIGURE 1.9

1.4 PROGRAMMING CONSEQUENCES

There are so many possible labels for each type of junction that I decided to begin programming a labeling system by writing a sort of filtering program to eliminate as many junction labels as possible before beginning a tree search procedure.

The filter procedure depends on the following observation, given in terms of the jigsaw puzzle analogy:

> Suppose that we have two junctions, J1 and J2 which are joined by a line segment L-J1-J2. J1 and J2 are represented by adjacent spaces on the board and the possible labels for each junction by two stacks of pieces. Now for any piece M in J1's stack either (1) there is a matching piece N in J2's stack or (2) there is no such piece. If there is no matching piece for M then M can be thrown away and need never be considered again as a possible junction label.

The filter procedure below is a method for systematically eliminating all junction labels for which there can never be a match. All the equipment is the same as that used in the tree search example, except that this time I have added a card marked "junction modified" on one side and "no junction modified" on the other.

Step 1: Put a junction number tag between 1 and n in each "junction number" bin. Place a full set of pieces in the "untried labels" bin of each junction.

Step 2: Set the counter to Nc = 1, and place the card so that it reads "no junction modified".

Step 3: Check the value of Nc:

A. If Nc = n + 1, and the card reads "no junction modified" then go to SUCCEED.

B. If Nc = n + 1, and the card reads "junction modified" then go to Step 2. (At least one piece was thrown away on the last pass, and therefore it is possible that other pieces which were kept only because this piece was present will now have to be thrown away also.)

C. Otherwise, go to Step 4.

Step 4: Check the "untried labels" bin of junction J(Nc):

A. If there are no pieces left in the Nc-th "untried labels" bin, then

A1. If there are no pieces in the Nc-th "tried labels" bin, go to FAILURE.

A2. Otherwise, transfer the pieces from the Nc-th "tried labels" bin back into the Nc-th "untried labels" bin, add 1 to the counter (Nc) and go to Step 3.

B. If there are pieces left in the Nc-th "untried labels" bin, take the top piece from the bin and place it in the board, and Go to Step 5.

Step 5: Check the spaces adjacent to space Nc:

A. If the piece in the Nc-th space has matching pieces in each neighboring junction space, transfer the piece from space Nc into the Nc-th "tried labels" bin, and transfer the pieces from the neighboring spaces and the neighboring "tried labels" bins back into their "untried labels" bins.

B. If there are empty neighboring spaces, then

B1. If there are no more junctions in the neighboring "untried labels" bins which could fit with the piece in space Nc, then that piece is not a possible label. Throw it away, and arrange the card to read "junction modified" if it doesn't already.

B2. Try pieces from the neighboring "untried labels" piles until either a piece fits or the pile is exhausted, and then go to Step 5 again.

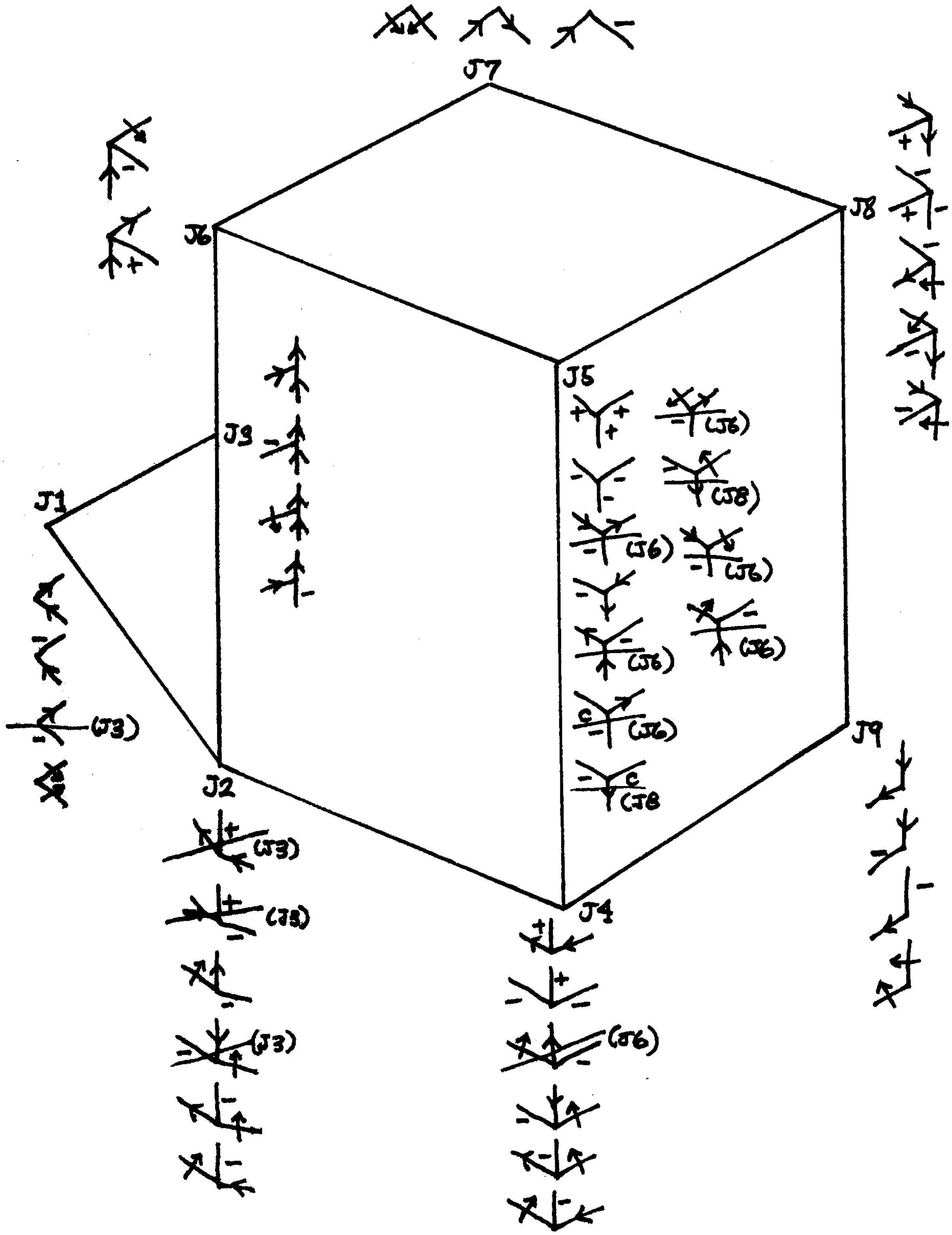
SUCCEED: The pieces in the "untried labels" bins of each junction have passed the filtering routine and constitute the output of this procedure.

FAILURE: There is no way to label the scene given the current set of pieces.

In the program I wrote, I used a somewhat more complex variaton of this procedure which only requires one pass throught the junctions. This procedure is similar to the one used to generate figure 1.9, and is described below.

When I ran the filter program on some simple line drawings, I found to my amazement that the filter procedure yielded unique labels for each junction in most cases! In fact in every case I have tried, the results of this filtering program are the same results which would be obtained by running a tree search procedure, saving all the labelings produced, and combining all the resulting possibilities for each junction. In other words, the filter program in general eliminates all labels except those which are part of some tree search labeling for the entire scene.

FIGURE 1.9



It is not obvious that this should be the case. For example, if we apply this filter procedure to the simple line drawing shown in figure 1.4 using the old set of labels given in figure 1.3, we get the results shown in figure 1.9. In this figure, each junction has labels attached which would not be part of any total labeling produced by a tree search. This figure is obtained by going through the junctions in numerical order and:

(1) Attaching to a junction all labels which do not conflict with junctions previously assigned; i.e. if we know that a branch must be labeled from the set S, do not attach any junction labels which would require that the branch be labeled with an element not in S.

(2) Looking at the neighbors of this junction which have already been labeled; if any label does not have a corresponding assignment for the same branch, then eliminate it.

(3) Whenever any label is deleted from a junction, look at all its neighbors in turn, and see if any of their labels can be eliminated. If they can, continue this process iteratively until no more changes can be made. Then go on to the next junction (numerically).

The junction which was being labeled (as in step (1)) at the time a label was eliminated (struck out in the figure) is noted next to each eliminated label in figure 1.9.

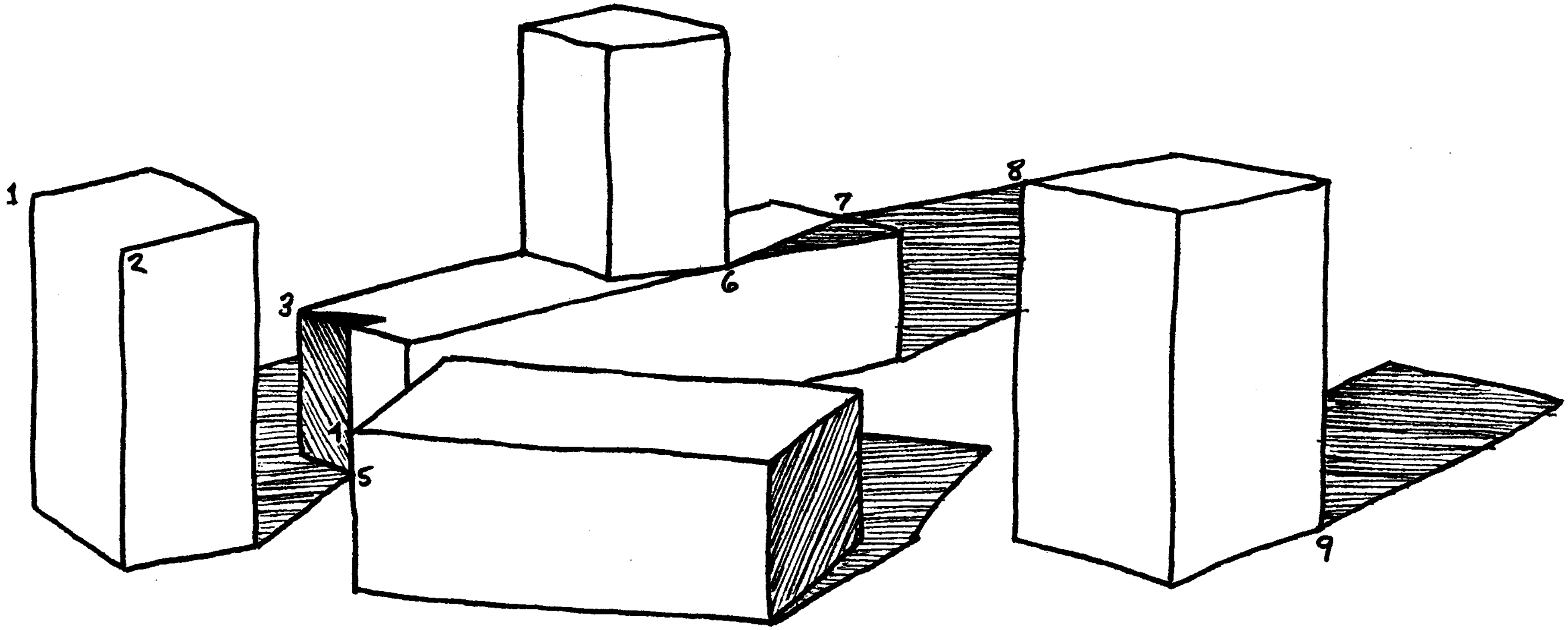
The fact that these results can be produced by the filtering program says a great deal about line drawings generated by real scenes and also about the value of precise descriptions. There is sufficient information in a line drawing so that we can use a procedure which requires far less computation that does a tree search procedure. To see why this is so, notice that if the description we use is good enough, then many junctions must always be given the same unique label in each tree search solution; in this case we need to find such a label only once, while in a tree search procedure, we must find the same solution on each pass through the tree.

Quite remarkably, all these results are obtained using only the topology of line drawings plus knowledge about which region is the table and about the relative brightness of each region. No use is made (yet) of the direction of line segments (except that some directional information is used to classify the junctions as ARROWs, FORKs, etc.), nor is any use made of the length of line segments, microstructure of edges, lighting direction or other potentially useful cues.

1.5 HANDLING BAD DATA

So far I have treated this subject as though I would always be given perfect data. In fact there are many types of bad data which frequently occur. Some can be corrected through use of better line finding programs and some can be eliminated by using stereo information, but I would like to show that the program can handle various problems by simple extensions of the list of junction labels. In no case do I expect the program to be able to sort out scenes that people cannot easily understand.

Two of the most common types of bad data are (1) edges missed entirely due to equal region brightness on both sides of the edge, and (2) accidental alignment of vertices and lines. Figure 1.10 shows a scene containing instances of both type of problem.



1,2 LINE MISSING

3 SHADOW EDGE IN ACCIDENTAL ALIGNMENT; APPEARS TO BE PART OF REGULAR PEAK JUNCTION.

4,5 ACCIDENTAL ALIGNMENT

6 NON-TRIHEDRAL JUNCTION FORMED BY ACCIDENTAL ALIGNMENT.

7,8 ACCIDENTAL ALIGNMENT.

9 CLOSE ALIGNMENT OF LINE SEGMENTS LEADS TO INTERPRETATION AS T JUNCTION INSTEAD OF ARROW.

FIGURE 1.10

I handle these problems by simply generating labels for "bad" junctions as well as "good" ones. It is important to be able to do this, since it is in general very difficult to identify the particular junction which causes the program to fail to find a parsing of the scene. Even worse, the program may find a way of interpreting the scene as though the data were perfect and we would then not even get an indication that the program should look for other interpretations. I would line to be reasonably certain that the parsing I desire is among those produced by the program.

1.5A ACCIDENTAL ALIGNMENT

I consider three kinds of accidental alignment:

(1) cases where a vertex apparently has an extra line because an edge obscured by the vertex appears to be part of the vertex (see figure 1.11a);

(2) cases where an edge which is closer to the eye than eye than a vertex appears to intersect the vertex (see figure 1.11b); and

(3) cases where a shadow is projected so that is actually does intersect a vertex (see figure 1.11c).

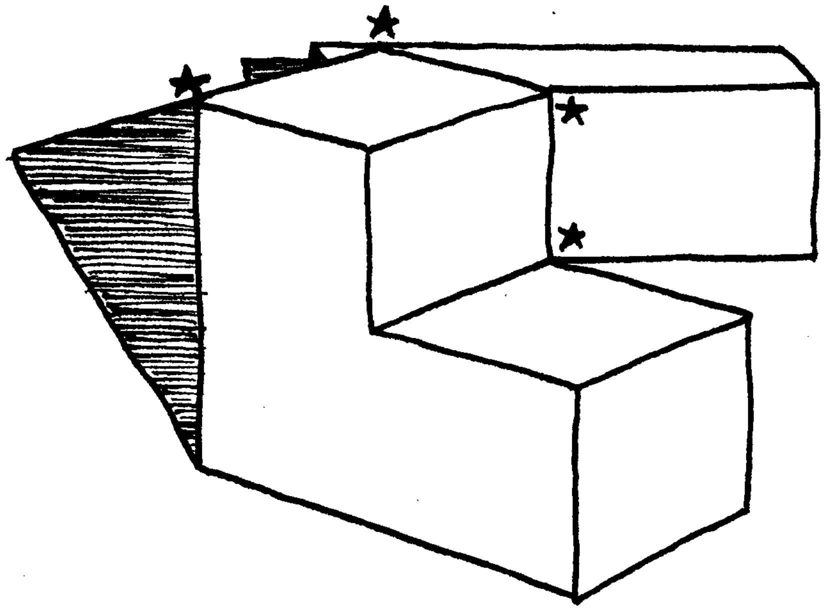


FIGURE 1.11a.

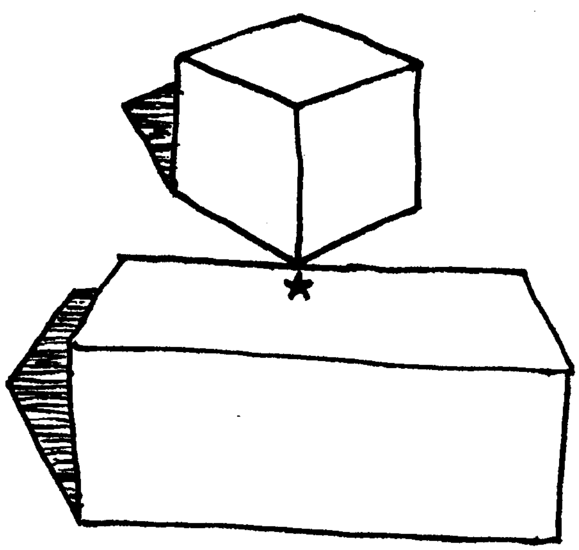


FIGURE 1.11b

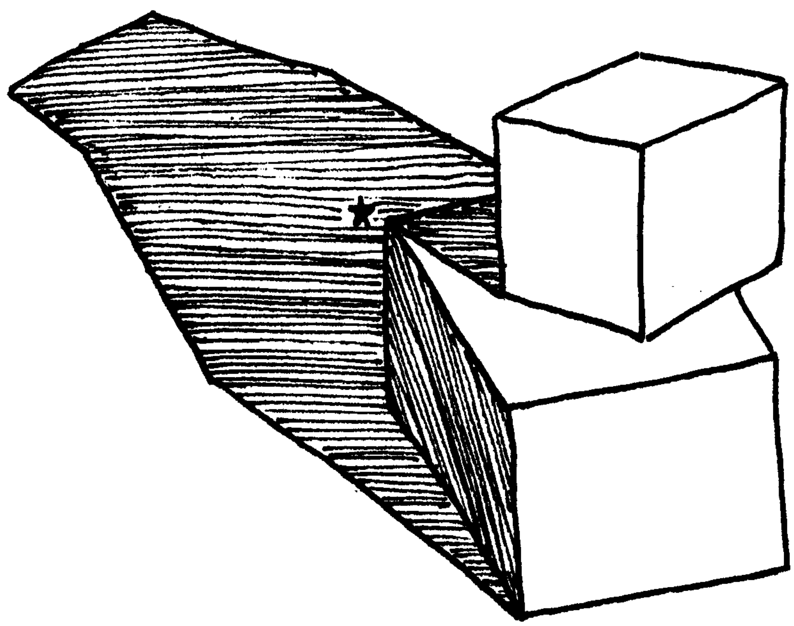


FIGURE 1.11c.

1.5B MISSING LINES

I have not attempted to systematically include all missing line possibilities, but have only included labels for the most common types of missing lines. I require that any missing line be in the interior of the scene; no line on the scene/background boundary can be missing. I also assume that all objects have approximately the same reflectivity on all surfaces. Therefore, if a convex line is missing, I assume that either both sides of the edge were illuminated or that both were shadowed. I have not really treated missing lines in a complete enough way to say too much about them. I feel that in general, there will have to be greater facilities in the program for filling in hidden surfaces and back faces of objects before the missing lines can be treated satisfactorily.

In general the program will fail if more than a few lines are missing and the missing line labels are not included in the set of possible junction labels. This is really a sign of the power of the program, since if the appropriate labels for the missing line junctions were included, the program would find them uniquely. As an example, the simple scene in figure 1.12 cannot be labeled at all unless the missing line junctions are included.

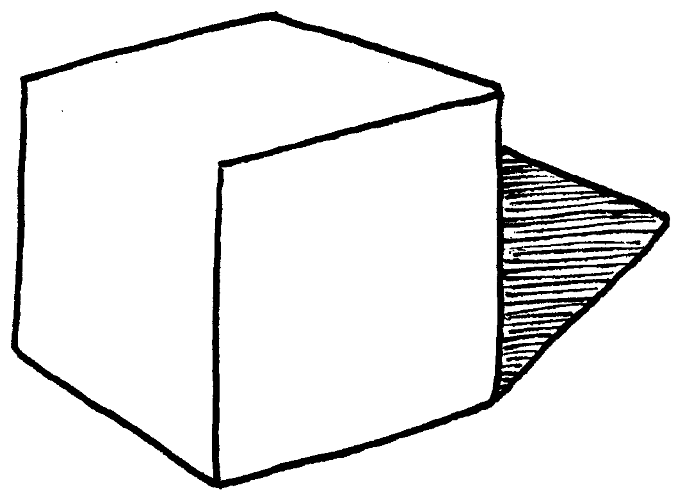


FIGURE 1.12

1.6 REGION ORIENTATIONS

Regions can be assigned labels which give quantized values for region orientatin in three dimensions. These labels can be added to the junction labels in very much the same way that the region illumination values were added. For a full treatment of these labels you will have to wait for my thesis.

BIBLIOGRAPHY

1 Clowes M B — On Seeing Things
AI Journal
Spring 1971

2 Dowson M — What Corners Look Like
MIT AI Vision Flash 13
June 1971

3 Dowson M & Waltz D L
Shadows and Cracks
MIT AI Vision Flash 14
June 1971

4 Finin T — Two Problems in Analyzing Scenes
MIT AI Vision Flash 12
June 1971

5 Finin T — Finding the Skeleton of a Brick
MIT AI Vision Flash 19
August 1971

6 Guzman A — Computer recognition of Three-dimensional Objects in a Visual Scene
MIT Project MAC Technical Report MAC-TR-59
December 1968

7 Huffman D A — Impossible Objects as Nonesense Sentences
Machine Intelligence 6
1970

8 Mahabala H N V
Preprocessor for Programs Which Recognize Scenes
MIT Project MAC AI Memo 177
August 1969

9 Orban R — Removing Shadows in a Scene
MIT Project MAC AI Memo 192
August 1970

10 Rattner M H — Extending Guzman's SEE Program
MIT Project MAC AI Memo 204
July 1970

11 Shirai Y — Extraction of the Line Drawing of 3-Dimensional Objects by Sequential Lighting from Multidirections
Electrotechnical Lab
Tokyo

12 Waltz D — Understanding Scenes with Shadows
MIT AI Vision Flash 21
November 1971

13 Winston P — Learning Structural Descriptions from Examples
MIT Project MAC Technical Report MAC-TR-76
September 1970